RL-TR-97-219
Final Technical Report
February 1998

RAPID OBJECT APPLICATION DEVELOPMENT (ROAD) CONSORTIUM

Sponsored by
Advanced Research Projects Agency
ARPA Order No. C539

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-219 has been reviewed and is approved for publication.

APPROVED:

DOUGLAS A. WHITE
Project Engineer

FOR THE DIRECTOR:

WARREN H. DEBANY, JR.
Technical Advisor
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ALTHOUGH THIS REPORT IS BEING PUBLISHED BY AFRL, THE RESEARCH WAS ACCOMPLISHED BY THE FORMER ROME LABORATORY AND, AS SUCH, APPROVAL SIGNATURES/TITLES REFLECT APPROPRIATE AUTHORITY FOR PUBLICATION AT THAT TIME.

RAPID OBJECT APPLICATION DEVELOPMENT (ROAD) CONSORTIUM

Contractor: Andersen Consulting
Contract Number: F30602-95-2-0005
Effective Date of Contract: 12 May 1995
Contract Expiration Date: 31 August 1997
Program Code Number: 5U10
Short Title of Work: Rapid Object Application Development (ROAD)
Period of Work Covered: May 95 - May 97

Principal Investigator: Colin T. Scott
Phone: (312) 507-2243
AFRL Project Engineer: Douglas A. White
Phone: (315) 330-2129

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Douglas A. White, AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1998	Final May 95 - May 97

4. TITLE AND SUBTITLE

RAPID OBJECT APPLICATION DEVELOPMENT (ROAD) CONSORTIUM

5. FUNDING NUMBERS

C - F30602-95-2-0005
PE - 63570E
PR - C539
TA - 00
WU - 01

6. AUTHOR(S)

Andersen Consulting: Colin T. Scott, James Adamczyk, Thomas Moldauer, Jin Chang, Donna Kelly, Jeffrey Mackay, and John Shiner
(Continued on Reverse)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Andersen Consulting
3773 Willow Road
Northbrook IL 60062
(Continued on Reverse)

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-219

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Douglas A. White/IFTD/(315) 330-2129

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the work carried out by the Rapid Object Application Development (ROAD) consortium under the auspices of the Technology Reinvestment Project (TRP). This DARPA (Defense Advanced Research Projects Agency) funded, Rome Laboratory managed initiative addressed the following problems related to building large-scale distributed software systems in the defense software industry: (1) Increasing difficulty (increasing functionality increases complexity); (2) Increasing cost (every system seems to be "new" with a high associated price tag); (3) Increasing uncertainty (nothing "tried and tested" that can be used). The TRP objectives were to accelerate the adoption of rapid application development via object technology and reuse, advance the state-of-the-art, and show the application of these techniques by means of demonstration applications. The work of the ROAD consortium has been to develop and demonstrate the use of tools and processes that meet the above objectives.

14. SUBJECT TERMS

Software, Object Technology, Software Development Environments, Computer aided Software, Engineering

15. NUMBER OF PAGES

136

16. PRICE CODE

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

6. Authors (Continued).

Raytheon: Cathy Bilotta-Lucia, Joseph Branciforte, Bernie Bussiere, Karl Gardner, Diane Seltz, Terri SooHoo, and Barry Hantman

ExperSoft: Tom Greene and Rich Llewellyn

CoGen Tex: Tanya Korelsky, Owen Rambow, Michael White, David E. Caldwell, and Benoit Lavoie

7. Performing Organization Names and Addresses (Continued).

Raytheon
50 Apple Hill Drive
Tewksbury MA 01876-0901

Expersoft
5825 Oberlin Drive
San Diego CA 92121

CoGen Tex
840 Hanshaw Road
Ithaca NY 14850

Executive Introduction

This report describes the work carried out by the Rapid Object Application Development (ROAD) consortium under the auspices of the Technology Reinvestment Project (TRP). This DARPA (Defense Advanced Research Projects Agency) funded initiative has sought to address the following problems related to building large-scale distributed software systems in the defense software industry:

- Increasing difficulty: increasing functionality increases complexity
- Increasing cost: every system seems to be "new" with a high associated price-tag
- Increasing uncertainty: nothing "tried and tested" that can be reused

DARPA realized that these concerns were endemic throughout the software industry and sought to apply a wider-range of commercial software development expertise to the problem.

Specifically, this TRP has the following objectives:

- Accelerate the adoption of rapid application development via object technology and reuse based on open systems standards
- Advance the state-of-the-art in object-oriented software development
- Enable interoperability of reusable assets
- Show the application of these techniques by means of demonstration applications

The ROAD consortium comprises:

- Andersen Consulting: responsible for process definition, tool development and integration
- CoGenTex: responsible for natural language generation
- Expersoft: responsible for development of CORBA-compliant middleware, and
- Raytheon: responsible for the development of two applications, one commercial and one military, that demonstrate the application of these tools and techniques.

The work of the ROAD consortium has been to develop, and demonstrate the use of, tools and processes that meet the above objectives. Specifically, the vision for the capability-based process is one of a supportive, asset-rich environment where analysts and developers distributed across both space and time collaborate to produce high-quality, client-focused solutions by assembling them from extensible, re-usable components.

The collaboration has been successful to the extent that all of the above objectives have been met:

- Raytheon has taken the technology developed as part of TRP and are in the process of applying it to a number of internal projects in both the defense and commercial

sides of their business. The ROAD desktop (TSE - TRP Support Environment) has been demonstrated to a number of Andersen Consulting's clients and some are interested in the application of this technology to their own process and tool suite.

- Andersen Consulting have been successful in integrating knowledge engineering and object-oriented development techniques. This is proving attractive to a number of their clients. Additionally, Andersen Consulting have also successfully demonstrated the integration of mobile agents with distributed object technology. As a result they are taking a lead role in the development of the OMG's (Object Management Group) standard in this area.

- The Component Specification Language (CSL) and the Asset Catalog provide a sophisticated, semantically-rich mechanism for defining, accessing, and re-using assets of the component-based development process. This goes beyond the capabilities of current standards and environments.

- Raytheon has successfully completed the development of the demonstration applications.

- The ongoing commercialization objectives have been met by involving third-party software product organizations in much of the TRP development activities. This provides a path for the TRP technology to reach a wide audience.

Participation in the ROAD TRP consortium, with the opportunity to advance the state-of-the-art in software development, has been a very stimulating and rewarding experience for all participants. The consortium format with its emphasis on mutual benefit and commitment has been particularly successful and, in the view of all participants, is a model that should be re-applied in future ventures of this kind. DARPA funding for the TRP project (a 50/50 cost split between consortium members and the government) enabled consortium members to accelerate development efforts in areas key to the accomplishment of TRP objectives, with the following results:

- Andersen - the innovative integration of knowledge-based business modeling tools with commercial object-oriented software development tools. These tools are now being applied on Andersen client engagements. The use of software agents to support process quality initiatives within the firm. This is viewed as a potential breakthrough technology and application.

- Expersoft - developed a CORBA-compliant Object Request Broker (ORB) execution infrastructure, not only to support the TRP consortium, but as a complete commercial product, one to two years earlier than could have been accomplished without DARPA funding.

- CoGenTex - validated their emerging new natural language generation technology in multiple applications (commercial and defense-oriented systems), and took advantage of the opportunity to mature and enhance this technology based on input and feedback from consortium members. CoGenTex believes their commercialization opportunities have been enhanced by participation in the TRP project.

- Raytheon - achieved a paradigm shift in the methodology used internally to develop software solutions by successfully demonstrating the use of distributed, component-based, object-oriented technology in both a defense and a manufacturing application. In addition, developed a core group of competency to seed future deployment of this methodology/ technology internally within Raytheon.

ROAD Consortium Members

The Rapid Object Application Development (ROAD) consortium under the auspices of the Technology Reinvestment Project (TRP) consists of the following members:

Andersen Consulting
3773 Willow Road
Northbrook, IL 60062

Dr. Colin T. Scott, Consortium Program Manager
Tel: 847-714-2655
FAX: 847-714-5812
Email: colin.t.scott @ ac.com

Raytheon
Missile Systems Division
50 Apple Hill Drive
Tewksbury, MA 01876-0901

Joseph C. Hintz, Ph.D.
Tel: 508-858-5907
FAX: 508-858-9380
Email: jxh @ msd.ray.com

Barry Hantman
Tel: 508-858-5778
FAX: 508-858-5976
Email: barry_hantman @ caemac15.msd.ray.com

Expersoft
5825 Oberlin Drive
San Diego, CA 92121

Tom Greene
Tel: 619-824-4100
FAX: 619-824-4110
Email: tgreene @ expersoft.com

CoGenTex
840 Hanshaw Road
Ithaca, NY 14850

Tanya Korelsky, Ph.D.
Tel: 607-266-0363
FAX: 607-266-0364
Email: tanya @ cogentex.com

Table of Contents

Chapter 1 - Andersen Consulting

1. Overview 1-1

2. Methodology 1-1
2.1 Capability-based Process Model 1-4
2.2 Capability-based Development Process 1-5

3. Process 1-9
3.1 Initial Objectives 1-9
3.2 Design Rationale 1-10
3.3 Process Specification 1-11
3.4 Overall Architecture 1-12
3.4.1 Process (Agent) 1-13
3.4.2 Personal Agents 1-13
3.4.3 Process Manager 1-13
3.4.4 Plan Manager 1-13
3.4.5 User Interface 1-14
3.4.6 Scenarios 1-16
3.4.6.1 Scenario 1 (TSE) 1-16
3.4.6.2 Scenario 2 (Domain Application) 1-16
3.5 IPS (Integrated Performance Support) 1-17
3.5.1 Initial IPS Objectives 1-17
3.5.2 IPS Design Rationale 1-17
3.5.3 IPS Specification 1-18
3.5.4 IPS Overall Architecture 1-18
3.5.4.1 Informal Process Methodology 1-19
3.5.4.2 Process Specification Tool (CST) 1-19
3.5.5 IPS Module 1-20
3.5.6 Instance Data 1-20
3.5.7 Execution Platform for Process 1-20

4. Tools 1-21
4.1 Business Modeling Knowledge Engineering Tools 1-21
4.2 Object Technology Visualization 1-23
4.3 REMAP 1-23
4.4 Component Specification Tool 1-23
4.4.1 Design by Contract 1-24
4.4.2 Event-based Communication 1-24
4.4.3 States: Simplifying Dynamic & Multiple Classification 1-25
4.4.4 Progress 1-26
4.5 Component Integration Tool 1-27
4.6 Asset Catalog Tool 1-28
4.7 Transformation Architecture 1-29
4.8 Unix and Windows Component Development Environments 1-30

5. Infrastructure 1-31
5.1 Introduction 1-31
5.2 Component Specification Goals 1-31
5.3 TRP Specification Models 1-33

5.3.1 Solution Component Characteristics 1-34
5.3.1.1 Key Ingredients of CSL 1-35
5.3.1.2 Advantages for Developing Components 1-36
5.3.1.3 Key Ingredients of a Solution Component 1-36
5.3.2 Component & Component-based Solution Models 1-37
5.3.2.1 Notation Conventions 1-37
5.3.2.2 Solution Specifications 1-37
5.3.2.3 Structure of a Solution Component 1-38
5.3.2.3.1 Provided Interface Specifications 1-39
5.3.2.3.1.1 Concepts 1-39
5.3.2.3.1.2 States 1-40
5.3.2.3.1.2.1 Events 1-40
5.3.2.3.1.2.2 Operation Extensions 1-40
5.3.2.3.1.3 Required Interfaces 1-41
5.3.2.3.1.4 Component Level Operations 1-41
5.3.2.3.1.5 Properties 1-41
5.3.2.3.2 Component Types 1-41
5.3.2.3.2.1 Domain Components 1-41
5.3.2.3.2.2 Process Components 1-41
5.3.2.3.2.2.1 Process Component Overview 1-41
5.3.2.3.2.3 Rule Components 1-42
5.3.2.3.2.3.1 Actor Components 1-43
5.4 The Component Specification Language (CSL) 1-44
5.5 Component Architecture Services and Facilities 1-45
5.5.1 Event Management 1-45
5.5.2 Process Management 1-45
5.5.3 Component Implementation Services 1-45
5.5.3.1 Interface Semantics 1-45
5.5.3.2 Service Request Stubs & State Mgmt Frameworks 1-46

6. Technology Transfer 1-46
6.1 External Knowledge Transfer 1-46

Chapter 2 - Raytheon

1. Introduction 2-1

2. Methodology 2-2
2.1 Introduction 2-2
2.2 Business Modeling 2-2
2.3 Solution Strategy 2-2
2.4 Conceptual Design 2-3
2.5 Discovery Model 2-4
2.6 Working Model 2-5

3. Development Environment 2-6
3.1 Introduction 2-6
3.2 Support Tools 2-7
3.2.1 HTML Editors 2-7
3.2.2 Groupware 2-7
3.2.3 Utilities 2-8
3.3 Analysis Tools 2-8
3.3.1 Domain Requirements Analysis Tools 2-8

3.3.2 Object Modeling Tool 2-9
3.3.3 Component Specification Tool (CST) 2-9
3.3.4 Component Integration Tool (CIT) 2-10
3.3.5 ModelExplainer 2-10
3.3.6 CogentHelp 2-11
3.3.7 Two-Dimensional Graphical User Interface Tool 2-11
3.4 Implementation Tools 2-12
3.4.1 Configuration Management Tool 2-12
3.4.2 Memory Leak and Error Detection Tool 2-13
3.4.3 Graphical User Interface Test Tool 2-13
3.4.4 C++ Compiler 2-13
3.4.5 Object-Oriented Database Management System 2-14
3.4.6 Object Request Broker (ORB) 2-14
3.5 Network Environment 2-15
3.5.1 Hardware 2-15
3.5.2 Secure Identification 2-16
3.6 Auto Code Generation 2-17

4. Applications 2-18
4.1 Capabilities-based Battle Management System (CBMS) 2-18
4.2 Work Center Management System (WCMS) 2-20

5. Technical Architecture 2-23
5.1 Introduction 2-23
5.2 Common Component Infrastructure 2-25
5.3 Object Distribution 2-26
5.4 Event Notification 2-27
5.5 Transaction Serialization 2-28
5.6 Exception Handling 2-28
5.7 Access Control 2-29
5.8 Legacy Architecture 2-29

6. Conclusions 2-29

Chapter 3 - Expersoft

1. Introduction 3-1
1.1 Expersoft Role 3-1
1.2 Methodology 3-2

2. Program "Build" Phases 3-2
2.1 Execution Environment Build 1 3-2
2.1.1 B1.EE.1 Initial Environment 3-3
2.1.2 B1.EE.2 GUI Integration 3-3
2.1.3 B1.EE.3 CORBA IDL 3-4
2.1.4 B1.EE.4 Interoperability Plan 3-4
2.1.5 B1.EE.5 OLE Integration Plan 3-4
2.2 Execution Environment Build 2 3-4
2.2.1 B2.EE.1 Interface Repository 3-5
2.2.2 B2.EE.2 Dynamic Invocation Interface 3-5
2.2.3 B2.EE.3 Distributed Namespace 3-6
2.2.4 B2.EE.4 Smalltalk Bindings 3-6
2.2.5 B2.EE.5 Multi-Threaded Support 3-6

2.2.6 B2.EE.6 CORBA 2.0 Interoperability 3-6
2.3 Execution Environment Build 3 3-7
2.3.1 B3.EE.1 ORB Interoperability 3-7
2.3.2 B3.EE.2 Transactions 3-8
2.3.3 B3.EE.3 Lifecycle 3-8
2.3.4 B3.EE.4 Naming 3-8
2.3.5 B3.EE.5 Events 3-9
2.3.6 B3.EE.6 OLE Integration 3-9
2.4 Execution Environment Build 4 3-9

3. Program Summary/Conclusions 3-10

Chapter 4 - CoGenTex

1. Introduction 4-1

2. ModelExplainder: Customizable Descriptions of Object-Oriented Models 4-2
2.1 Introduction: Object Models 4-2
2.2 Features of ModEx for Ptech 4-2
2.3 A ModEx Scenario 4-3
2.4 How ModEx Works 4-7
2.5 ModEx for PCPACK 4-8
2.6 Lessons Learned 4-8

3. LIDA: Linguistic assistant for Domain Analysis 4-9

4. CogentHelp: A Tool for Authoring Dynamically Generated On-line Help 4-10
4.1 Introduction 4-10
4.2 Design Goals 4-10
4.3 Automated Documentation 4-11
4.4 System Overview 4-11
4.5 NLG Techniques 4-12
4.5.1 Knowledge Representation 4-12
4.5.1.1 Phrasal Lexicon 4-13
4.5.1.2 IKRS 4-13
4.5.2 Text Planning 4-15
4.5.2.1 Capitalizing on Domain Structure 4-15
4.5.2.2 Grouping 4-16
4.6 Authoring 4-16
4.7 Lessons Learned 4-17

5. CogentTIPS 4-18
5.1 Introduction 4-18
5.2 Personalized Task Summaries 4-18
5.3 System Overview 4-20
5.4 Lessons Learned 4-21

6. Conclusions 4-21

Bibliography 4-22

List of Figures

Chapter 1 - Andersen Consulting

Figure 1 - Eagle Phases 1-2
Figure 2 - General Process Model 1-4
Figure 3 - ROAD Process 1-6
Figure 4 - Domain Model Concepts v. Solution Concepts 1-7
Figure 5- ROAD Business/Domain Model / Solution Structure 1-8
Figure 6 - Integrated Modules of Process Architecture 1-12
Figure 7- Novice User Interface View I 1-14
Figure 8 - Novice User Interface View II 1-15
Figure 9- Expert User Interface 1-15
Figure 10 - Overall IPS Architecture 1-19
Figure 11 - Tool Framework 1-21
Figure 12 - Transformation Architecture 1-30
Figure 13 - TRP Component Specification Approach 1-33
Figure 14 - Component Semantics 1-34
Figure 15 - Generalized, High-Level View of a TRP Solution 1-36
Figure 16 - Notational Conventions 1-37
Figure 17 - Solution Specifications 1-38
Figure 18 - A Solution Component 1-39
Figure 19 - Process Component Overview 1-42
Figure 20 - Rule Component Example 1-43

Chapter 2 - Raytheon

Figure 1 - TRP Network 2-16
Figure 2 - Code Generation Process 2-17
Figure 3 - CBMS Components 2-18
Figure 4 - CBMS Main Screen 2-19
Figure 5 - WCMS Business Practices 2-22
Figure 6 - High Level Architecture 2-25
Figure 7 - Dynamic Reference Passing 2-26
Figure 8 - Singleton Approach 2-27

Chapter 4 - CoGenTex

Figure 1 - The University O-O Diagram 4-3
Figure 2 - Description Used for Validation 4-4
Figure 3 - Text Plan for Validation 4-5
Figure 4 - Text Plan for Documentation 4-6
Figure 5 - Description Used for Documentation 4-6
Figure 6 - ModEx Server Architecture 4-7
Figure 7 - Sample Application Window 4-14
Figure 8 - Sample Help Page 4-16
Figure 9 - CogentHelp in Authoring Mode 4-17
Figure 10 - Sample CogentTIPS Page 4-18

Chapter 1

Andersen Consulting

1. Overview

Prior to the start of the TRP, Andersen developed an object-oriented, component-based solution construction approach - Eagle/ODM (Object Development Methods) - that was supported by a full life-cycle methodology and a corresponding development process. As part of the TRP, we applied that approach within Raytheon and, based on feedback from the TRP initiative, we defined an Asset-Centric Process [Andersen, 96], which is a refinement and specialization of the original overall ODM.

The vision for this asset-centric process is an environment where analysts and developers distributed across both space and time collaborate to produce high-quality, client-focused solutions by assembling them from extensible, re-usable components.

Among its objectives are:

- support of the distributed, collaborative development of component-based solutions that match client requirements
- reduction of the lifecycle costs of solutions by encouraging the creation, re-use, and evolution of software development assets
- embodying the process in an extensible and variable set of tools.

The asset-centric process is supported by a complementary tool model. This model is a generic representation of the types of tool that are needed to support the process, along with a characterization of the assets produced as a result of applying the process. Any given development or development location can then choose to specialize or ground this general model in specific tools and specific asset representations. In addition to this tool model, developers have the benefit of a collaborative environment that includes Integrated Performance Support (IPS) as "intelligent help" to support them during the process.

2. Methodology

During the first year of the TRP, we worked with Raytheon application teams to transfer to them the methodology and process captured in AC/Methods, Eagle, and early versions of the ODM.

An outline of this process is shown below in Figure 1:

Through the course of the project, we monitored the use of the process and learned several lessons concerning the introduction and application of an object-oriented development process:

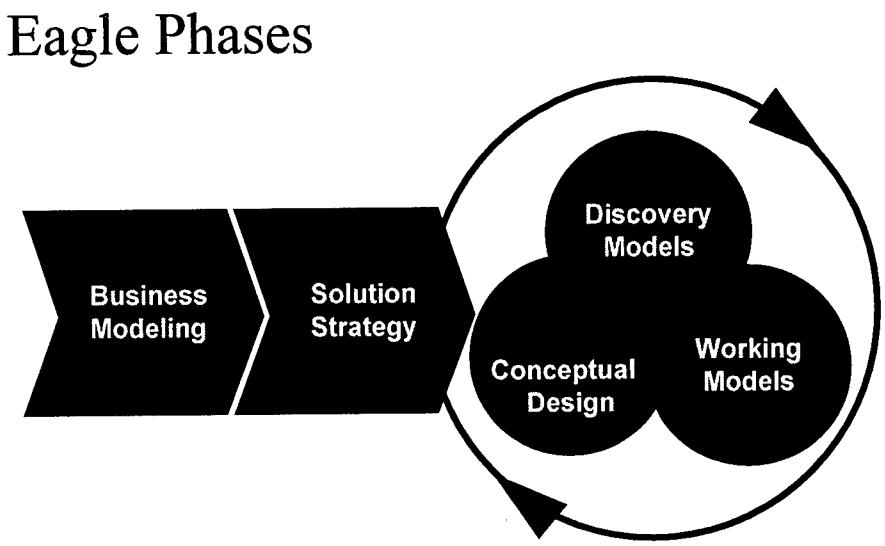

Figure 1 - Eagle Phases

The purpose of each of the above phases (in business domain terms) is:

- **Business Modeling:** to understand the industry and the enterprise by focusing on identifying current and future variability; to model the business as a set of large-grained objects that allow for flexibility to meet that variability; and to produce an early version of the business component model.

- **Solution Strategy:** to identify and detail the business practices, or how the processes will work in the future.

- **Conceptual Design:** to gather requirements detailing specifically how the business practices will work, both from a business process and user perspective; in the UI (User Interface) 'layer', to design the user interaction model; to detail the system logic, and also create a first version of the class model; to refine and then design the business components in detail, then to partition them into implementation components.

- **Discovery Models:** to find solutions to areas of uncertainty early on. Then the solutions (working code, patterns, standard, etc.) are fed into the Working Models phase.

- **Working Models:** to fully specify each "thing" that will be to the level required by the developers; to begin iterations on the frameworks that provide for consistency and reuse when the application is built; to build iterations of the application itself, using the frameworks developed earlier; to perform a component test on each thing built; to fully

and formally test the integration of all of the pieces; to fully and formally test the business requirements

Applying the activities in each of these phases to the defense domain provided valuable feedback and raised a number of issues and concerns that were addressed as part of the work in Builds 3 and 4.

One issue related to the iterative nature of the object-oriented development process. We believe that a process of this type is essential to successful object-oriented development projects. However, many current software projects in the Defense industry are constrained by contracts with fixed delivery schedules and delivered functionality. In these cases, the iterative process must utilize appropriate risk management techniques. The evolving nature of DoD (Department of Defense) software standards should allow freer use of iterative development in future contracts.

Raytheon has a sophisticated software development process in place that scores highly on the SEI CMM (Software Engineering Institute Capability Maturity Model). This process follows traditional (waterfall) lines. The challenge was to encourage application teams' management to suspend their concerns about the repeatability and reliability of an iterative process and thereby reap the hard-to-quantify benefits of object technology.

A second, yet related, point concerned the level of detail in the methodological or process descriptions. The lack of prescriptiveness that gives Andersen Consulting the flexibility to tailor the process to many different client environments could be viewed as fuzzy or inexact from a single client's or development's perspective. Raytheon application teams expected a process to outline in some detail each task package, task, and activity necessary to achieve the goal. One of the achievements of TRP was the tailoring the methodology for Raytheon's two application domains to provide the appropriate level of detail for these OO (object-oriented) developments.

Another point concerned the volume of the methodological descriptions. These books certainly passed the "weight test", but tended, almost as a direct consequence, to fail at the usability level. Although much useful information is included in these works, if it is hard to find they are next to useless. As a consequence, a substantial amount of work in the latter builds was focused on delivering information about the process in a manner and at a time that was directly useful to the developer.

Some of Raytheon's additional concerns gave us starting points for substantial improvements in the initial process:

1. *Too many deliverables of unconfirmed value:* This caused us to examine the relationship of the process deliverables and the evolvable assets that would form the knowledge capital of a project.

2. *Need to have more quality check points embedded in the process:* We made progress on this issue by considering the difference between a project where quality is "inspected-in" and one where quality is "built-in" as part of the process support environment. Taking the latter path has led us to an innovative approach based on the notion of *active quality agents.*

3. *Address the issue of traceability of requirements through the development process:* This was partially addressed by a feature of the transformation technology used to pass domain knowledge between tools.

We, therefore, refined the original process used by Raytheon application teams, but at the same time maintained consistency with the overall frameworks of AC/Methods and ODM and took account of the above comments. In its final form, the ROAD (Rapid Object Application Development) process is a very specific instance of the overall Andersen Consulting methodology and process, and whose deliverables are the defining characteristics of a TRP-based project.

2.1 Capability-based Process Model

We took, as a starting point for the new process model, a capability-based perspective on process definition. A *capability* is defined as a set of resources or products which are:

- in a particular required state
- required by a process or organization to meet its objectives, or
- delivered to a customer, another process, or another organization.

In military terms, a capability is, for example, a division in a particular state of readiness: serviced vehicles, trained personnel, available supplies, etc. It is the role of a process to deliver a capability. However a process can only deliver a capability if this represents a value added to the resources for that customer.

In order to deliver a capability, a process typically needs to be provided with capabilities of its own. This gives rise to a general process model. An example is shown below in Figure 2:

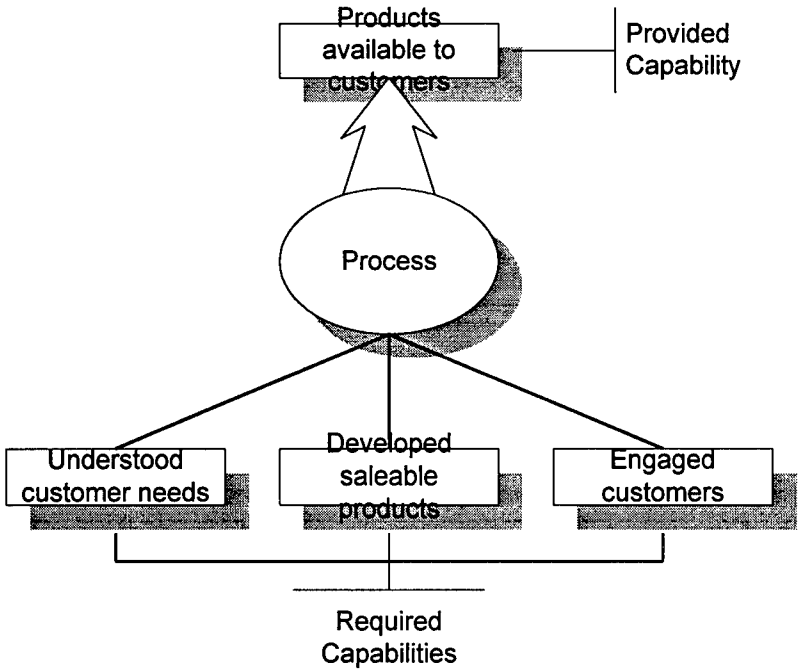

Figure 2 - General Process Model

There are two other features of the base model that are not clearly highlighted by the above example. Firstly, the notion that there are external influences on any system which we do not choose to model *per se*, preferring to model only their interaction with the system. In the above example, customer may fall into this category. We would not necessarily model all of the systems that make customers behave as they do, but we would model the relevant observable events or operations caused by them. We call these features *actors* (see below).

Secondly, the interaction between processes and capabilities is often governed by *rules*. These rules are often qualitative and are indicated by concepts like "saleable" or "engaged" in the above model. Rules are focal points for variability in systems. For example, the above model will hold for many different definitions of "saleable."

2.2 A Capability-based Development Process

Following this approach led us to identify the primary capability of the ROAD process:

- *solutions* assembled from re-usable components

In order to deliver such a capability, three additional (provided) capabilities are required:

- re-usable, extensible *components*
- a *component architecture* into which the components could be set, and
- a consistent, evolvable *domain model* which would serve as the logical blueprint for the construction of solutions.

These capabilities are themselves the product of three separate processes, whose required capability is (as in the above diagram):

- understood customer needs and requirements

This capability is the result of a process that refines the provided requirements by producing a complete and consistent picture.

Graphically, the ROAD process is shown below in Figure 3:

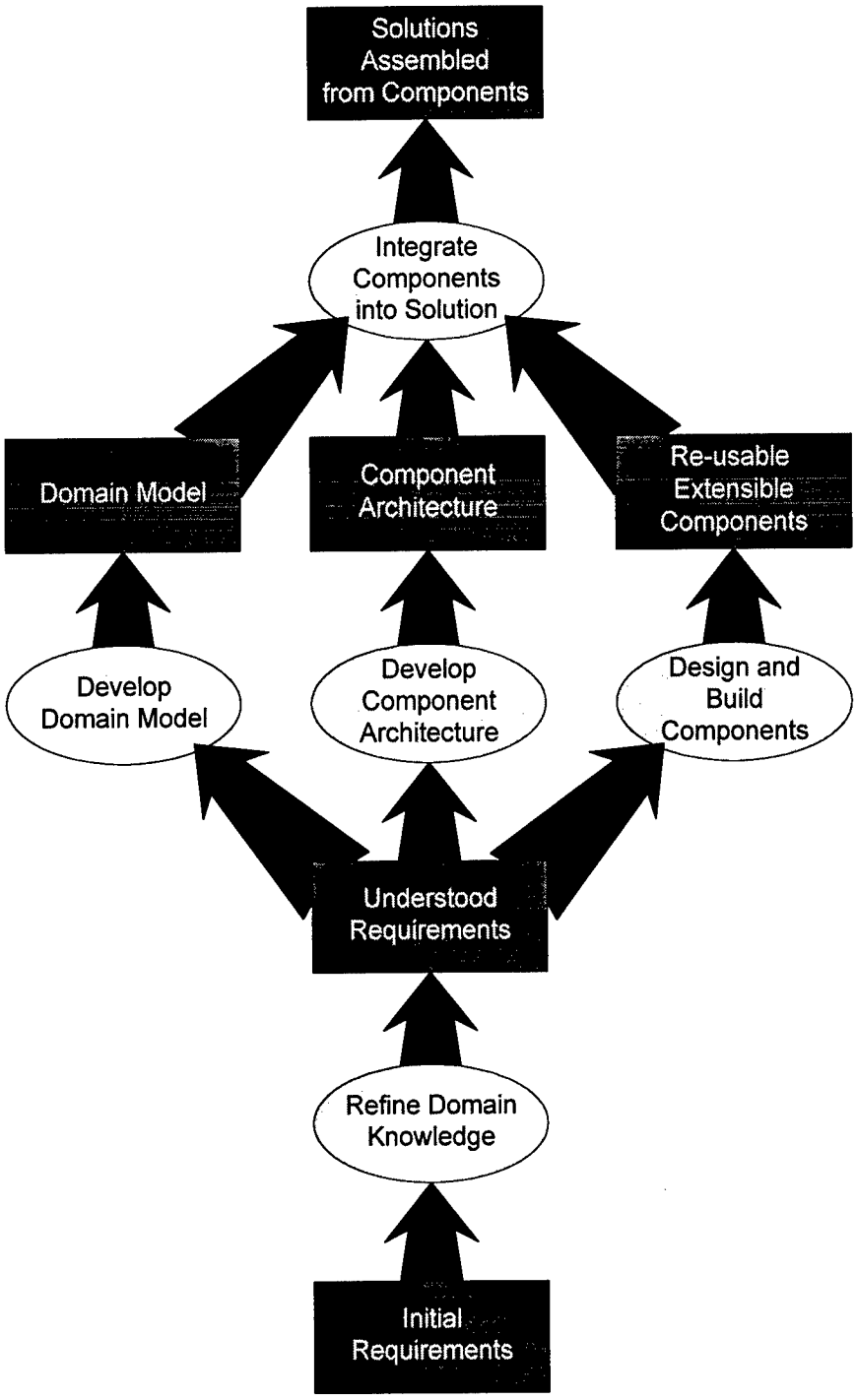

Figure 3 - ROAD Process

The delivered capability of each of the above processes is itself a collection of products called *assets*. The ability to re-use any or all of these assets (rather than simply code elements) is one of the major benefits of the ROAD process. The principal assets produced during the execution of the ROAD process are:

- *Understood requirements:* These represent the result of the application of knowledge engineering techniques to the original requirements. Thus they represent a qualitative

refinement of the originals – ambiguities have been removed, terms are used consistently, incomplete areas have been filled in etc. In some domains, and defense is often one, complete and formal requirements are (assumed to be) given at the outset. In this case the initial refinement process can be skipped.

Even once the requirements are well-understood for a given system, another hurdle remains. There is often a mismatch between the conceptual framework used by the domain modeler and that, eventually, implemented by the solution designer. This can affect the long-term viability of a solution in a number of ways:

- There is a lack of traceability between requirements/domain model features and elements of the eventual solution.
- A minor change in requirements may have a disproportionately large impact on the delivered system.
- The cost to effect a change may be proportional to the size of the system rather than the size of the required change.

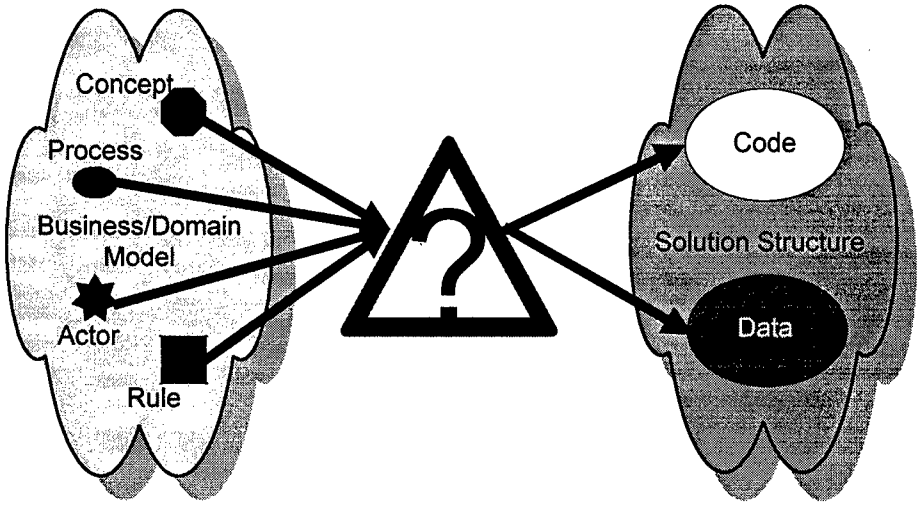

Figure 4 - Domain Model Concepts v. Solution Concepts

Within the ROAD project we decided to try to maintain a direct link between concepts in the delivered system and those in the domain model. In this way we maintain a direct link between the concepts in each model and can localize the impact of changes in requirements.

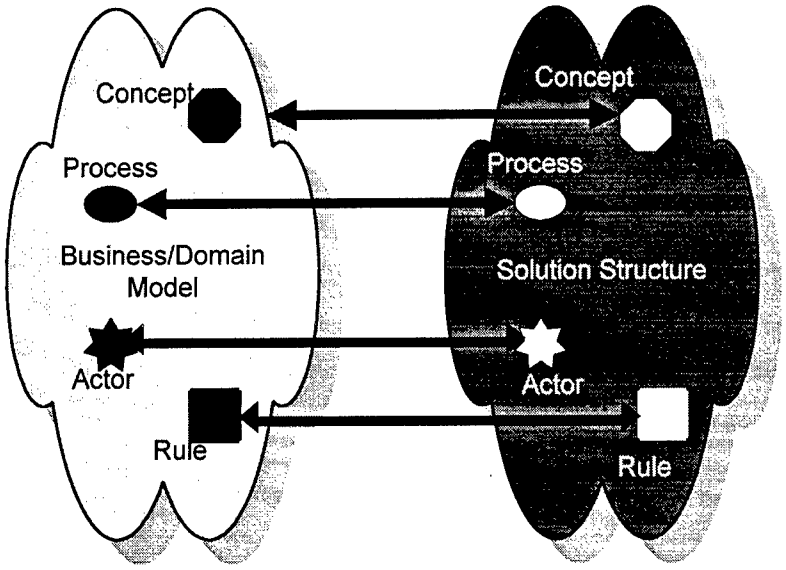

Figure 5 - ROAD Business-Domain Model / Solution Structure

This approach gives rise to the following key analysis assets in the ROAD process:

- *Process:* a partially ordered set of activities with some specific business/domain goal. The functionality of a system can be described in terms of a small number of processes. Depending on the scope of the system, these can be very high-level indeed. For example, a complete enterprise can typically be decomposed into 10 - 20 such processes. Smaller systems yield the same number of (smaller-scale) processes.

- *Domain concept:* an initial class candidate. From this perspective, we are trying to determine the domain level concepts, a vocabulary, that will help us describe the solution. This should go as far as the definition of responsibilities and some attributes for each concept as well as the relationships between them. This last aspect is particularly important in anticipation of some of the clustering activities that are carried out as part of the process.

- *Rule:* a policy that governs the interaction between processes and domain concepts. Each solution implements rules that are inherent in the domain. In some cases these rules are closely related to another element (e.g. a process or a concept). For example, there might be a rule that a Division Manager must approve any purchase order for items costing more than $10,000. This rule could be attached either to the Purchase Order object or to the process that drives this functionality.

  However, there are cases where these rules span multiple elements. In these situations, we seek a way to implement the rules *once* in order to localize (and make predictable) the impact of changes.

- *Actor:* Actors are representations of external influences on, or interactions with, our solutions: typically, users and external systems. In the case of the ROAD process itself, these external systems are, for example, the COTS (Commercial-Off-The-Shelf) tools used to carry out parts of the overall process. Because they are external, we

can choose to ignore the internals of any actors in a system, and focus only on the impact they have on the system.

The approach in the ROAD process is to view solutions as being constructed from these four "atomic" elements (process, domain concept, rule, and actor). We view systems as interacting collections of molecules formed by clustering such atoms. Each molecule provides and requires some of the features that are useful in the context of the whole solution.

Thus we distinguish the following assets arising from the synthesis activities in the ROAD process:

- *Cluster:* a logical grouping a collection of the more fundamental elements based on some criterion (e.g. cohesiveness, performance requirements, geographical location)
- *Interface:* a logically associated set of operations with associated signatures and semantics.
- *Component:* a packaging or system delivery element that implements the functionality required by one or more interfaces
- *Solution:* the element that implements the requirements in their entirety.

3. Process

Our technical deliverables are divided into three sections: Process, Tools and Infrastructure. The process team delivered process management and monitoring tools, the tools team delivered tools to support the ROAD process, and the infrastructure team delivered tools used in the execution of component-based solutions.

3.1 Initial Objectives

Efforts in the Process area are intended to support a variety of processes, such as the ROAD process or Raytheon's own methodology, known as the BlueBook process[1], by providing a collaborative environment among participants allowing them to transcend the requirements of being in the same place and working together at the same time. Unlike some of the well known groupware products such as Lotus Notes or WebForum, which simply provide a passive information space, the goal of TRP's process architecture is to enable active collaborative work among participants working in TRP's "component based software engineering environment". This active collaboration implies that the system knows the importance of individual work items and dynamically organizes a person's agenda based on that knowledge. The priority of work items is largely determined based on the temporal relationships (such as work item A has to be done before B). In addition, the system also monitors any changes in the status of relevant deliverables and deliberations involved with work processes in order to notify the appropriate participants.

[1] The BlueBook process describes Raytheon's software engineering standards. For details, please see Software Engineering Standards Practices Manual from Raytheon.

In short, the main objectives considered for development of the overall process architecture are:

- Define precise semantics for the general concept of a process
- Define a consistent programming interface for executing and managing processes
- Define a clear method of coordinating and collaborating other types of components such as domain, actor or rule components
- Allow independent process implementations such as workflow, agents, or customized application of the process specification
- Support for active notifications and plan integration
- Support for the customization of various processes such as the ROAD or BlueBook processes
- Provide a portable user interface for various different platforms
- Support for existing standards such as WfMC (Workflow Management Coalition) or OMG's (Object Management Group) Agent Transfer Facility
- Support for distributed and 'occasionally attached' processes

3.2 Design Rationale

From the beginning of the TRP project, the basic notion of workflow, groupware and computer supported cooperative work (CSCW) support for a variety of processes, such as ROAD or BlueBook, has been investigated. In particular, a typical workflow tool, since it supports coordination between multiple roles through a well defined structured process, seemed suitable for the basis of the TRP process architecture.

It has been clearly noted that the "next generation" of workflow tools has to be able to reconcile workflow process models and software with a rich variety of activities and behaviors that comprise "real" work. This implies that the workflow system not only has to be able to integrate other software with standardized interfaces and interchange formats (such as our distributed components), but also has to provide capabilities to handle distributed processes and "occasionally attached" processes.

Unfortunately, current workflow systems lack most of the above capabilities. Current systems largely assume a closed network where all software is available on a homogeneous platform and all participants are locally linked together at the same time.

On the other hand, the concept of an agent, in particular that of a mobile agent which travels around the network on behalf of its owner, has gained significant interest in various areas of computation including artificial intelligence, distributed computing and communications. Consequently, there have been multiple publications regarding mobile agents in general, and the benefits seemed obvious for a collaborative environment. Indeed, the issues for mobile agents have been discussed and the requirements for an agent infrastructure (so called agent meeting point) were investigated by many different researchers.

Unfortunately, these research efforts have largely ignored the details of workflow processes involved with mobile agents. For example, a typical organization usually has its own workflow processes defining business activities and overall control of the work flow. It seems reasonable to assume that mobile agents in an active collaborative environment should have process information embedded within themselves in order to find a target destination once an activity has been initiated. Additionally, a mobile agent must know the various characteristics of the actual roles involved with workflow processes. For example, a mobile agent at a project leader's machine may have complete access privileges to a deliverable schedule, while a mobile agent at a programmer's may not. Obviously, this difference is not bound to an actual machine but by the roles participants play. In fact, if a programmer is allowed to act as a project leader (for example, because a project leader is sick), a mobile agent must change its characteristics dynamically in order to avoid any disruptions in the workflow process.

For these reasons, a significant amount of work was required to develop an innovative overall process architecture to support the above main objectives and to overcome the disadvantages of the existing implementations.

As a result, the core concept of the TRP process architecture is based on the WWW (World-Wide Web) and a distributed set of agents which facilitate the workflow process, in order to improve efficiency among participants and for the overall process. Although the original concept of an agent, "acting on behalf of someone else" is simply lost in current research on agents, it is precisely this notion which our process architecture adopts. In other words, the agents in our architecture largely act on behalf of participants, organizing their agenda and looking for important events (such as completion of a deliverable).

3.3 Process Specification

After analyzing the TRP ROAD methodology (or BlueBook), the process is further specified using the Component Specification Tool (CST). Once specified, the process from CST may be exported to a project planning tool such as Microsoft Project.

When the project plan is finalized, the information from the project plan is exported into the TRP desktop, which will manage each individual's agenda list. The tasks in each agenda are generated based on the tasks assigned from the project planning tool. When individuals update the tasks in their agenda list, the progress status is updated back into the activities in the project planning tool. Periodically, a project manager may review the status of various tasks by simply invoking the project planning tool.

As the project progresses, the plan begins to change. For example, new tasks may need to be added and some obsolete tasks may need to be removed. When the addition / deletion of the tasks does not alter the overall definition of the process, they can simply be added into / deleted from the project planning tool.

When the actual definition of process changes due to the addition / deletion of a whole summary task with multiple activities, a project manager is required to re-specify the process in the CST. Once a new process is defined using CST, a new project plan is generated. Since this assumes no consistency with existing instances of the project plan and the TRP desktop agenda, this may only be suitable for major planning efforts, such as planning a new build based on input from the previous plan.

The processes specified in CST also generates the high level run time process specifications which represent high level run time agent specifications.

3.4 Overall Architecture

The overall process architecture consists of a process manager, process system, process, plan manager and a personal agent for each user, as shown in Figure 6 below. These modules in the architecture publish their names and properties with the Naming and Trader Services from CORBA (Common Object Request Broker Architecture). In addition, a process and process manager also interact with TRP Event Services (shown as subscribe / publish in Figure 6) in order to subscribe and publish the events of interest.

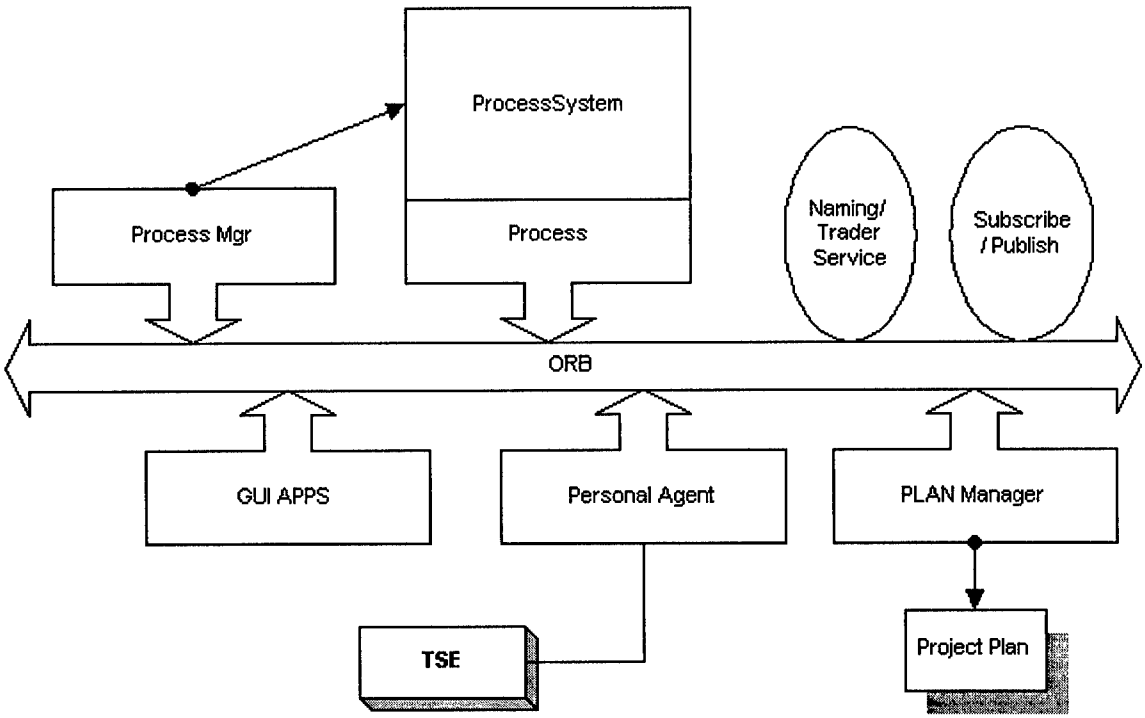

Figure 6 - Integrated Modules of Process Architecture

The personal agent is responsible for organizing a user's agenda and for communicating with multiple process agents and an actual participant. The personal agents are stationary agents which act on behalf of a specific user with certain roles, such as object modeler or application developer. The processes are implemented in agents. Therefore, the processes (agents) are workflow agents which support a specific process, such as ROAD or BlueBook. The process agents are mobile, traveling from one node of the network to another. The process system is responsible for the actual execution, serialization and de-serialization of process instances. Since the process system is dependent on the specific implementation of a process, there may be multiple different process systems such as Java, Tcl and Python.

3.4.1 Process (Agent)

A process agent may be viewed as a specific instance of a workflow (process). A process agent contains a list of goals, activities and a process in which various roles are involved. Process agents are mobile agents because they may travel around a network with an already encoded itinerary. This implies the process can support some level of disconnected situations, unlike centralized workflow. The process agents already know, or are able to find out if they do not already know, where to go and who to talk to. A similar analogy can be made to individuals working in any business organization - for example, a manager knows where to go and who to talk to in order to get his budget approved.

Process agents are instantiated when the associated process is created. When the execution is complete, the status of the tasks in the process are updated. Each instance of a process agent has a unique ID and its behaviors may be changed by simply re-defining the default goals and the definition of the associated process.

3.4.2 Personal Agents

Personal agents act on behalf of actual participants. A personal agent represents an actual individual in a business organization with multiple roles (actor agents). Upon login, the personal agent consults a user profile which contains a list of preferred tools, local host name and address, name of the HTTP (Hypertext Transfer Protocol) server and port number, user skills, WWW browser of choice, etc. The personal agent registers the user's profile with the Trader Service and communicates with the local process manager in order to retrieve the participant's agenda. Based on this agenda list and a user profile, the personal agent constructs HTML (Hypertext Markup Language) pages and invokes the WWW browser for those pages for the user interactions.

3.4.3 Process Manager

The process manager is responsible for providing a basic infrastructure for "guaranteed delivery" for transporting processes from one computer to another. It provides a number of services to process (agents), such as creating specific agents as well as locating already existing agents. For example, an external application (such as GUI - Graphical User Interface) can interface with the process manager to create an instance of an agent. Furthermore when the agent moves, the application can simply ask the local process manager to find the location.

3.4.4 Plan Manager

The plan manager is responsible for maintaining and updating a plan generated from a project planning tool – currently, Microsoft Project. Each activity within a plan represents a generic process (TSE - TRP Support Environment - process) with a single activity. Therefore, a project plan represents many instances of the TSE process. As a user interacts with the process agents and updates the status of his activities, the appropriate activities within the project plan are updated.

In this regard, the plan manager may be considered to be a specific instance of a simplified process manager. The plan manager only knows about one type of process – a generic TSE process which sets the context for the activity and updates the project plan.

3.4.5 User Interface

The user interfaces for process are implemented on Java and run on any Java enabled browser. In the case of a specific domain application such as CBMS (Raytheon's Capabilities-based Battle Management System), the user interface will be provided by a customized GUI application. For support of the ROAD process, however, two different user interfaces have been tested – one for a novice and another for an expert user.

The novice user interface was based on a "room" metaphor, where each room represented meta-level patterns of activities that can occur in the process. Upon entry into a specific room, the room displayed an agenda, as well as tools appropriate for the current role (Figures 7 & 8). Although use of the novice user interface provided useful feedback, it proved to be impractical for daily use. As a result, a more efficient expert user interface has been implemented.

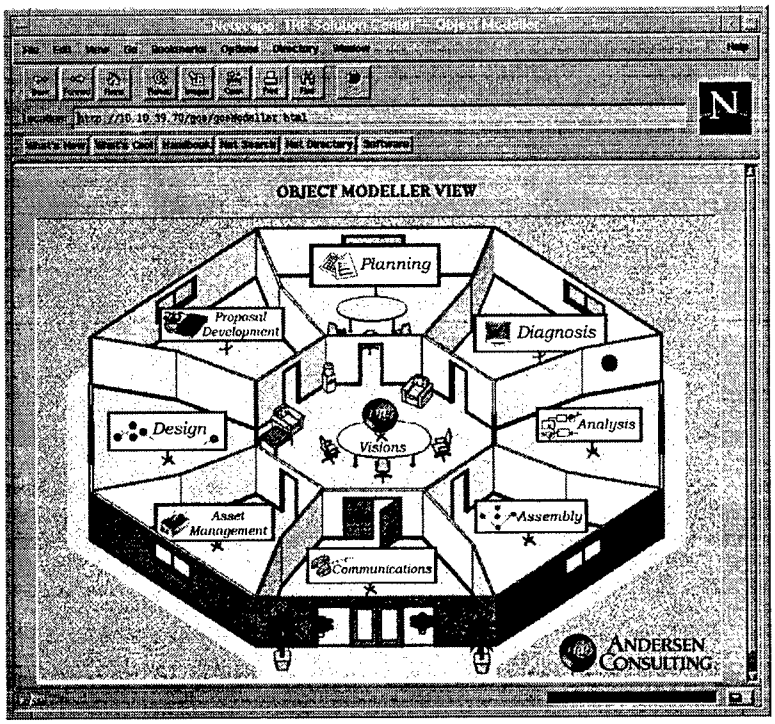

Figure 7 – Novice User Interface View I

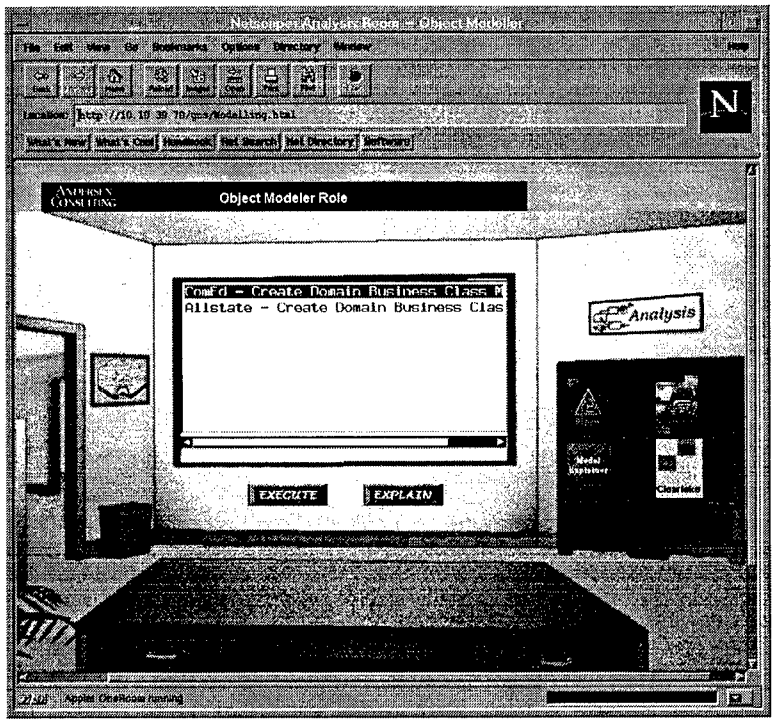

Figure 8 – Novice User Interface View II

The expert user interface displays 3 different parts (Figure 9) – a list of tasks assigned to the current user in a given project, available tools that can be launched and any outputs from those tools.

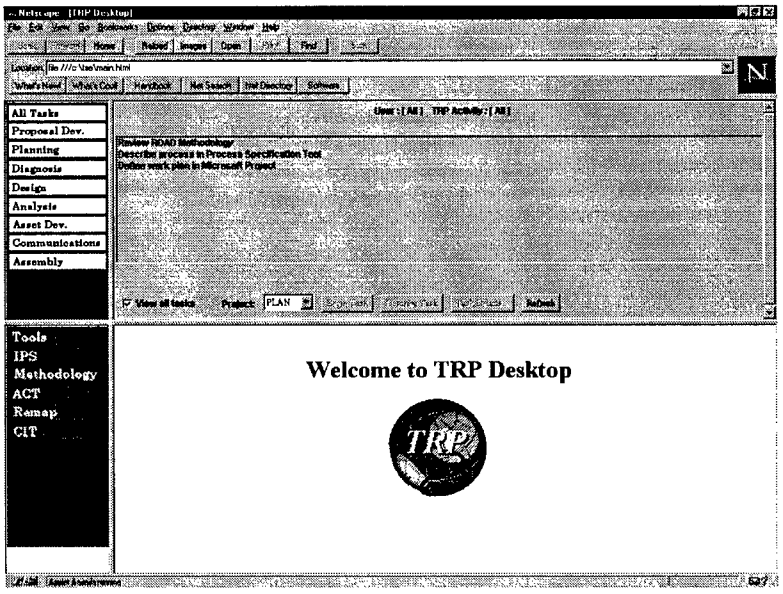

Figure 9 – Expert User Interface

3.4.6 Scenarios

There are basically two different scenarios using the process architecture. One is based on TSE, following either the ROAD or the BlueBook process. The other scenario supports customized domain applications such as CBMS.

3.4.6.1 Scenario 1 (TSE):

1. The Plan Manager (PM) is created by the project manager.

2. When PM is invoked, it consults the current project plan. The entire plan represents many instances of a single workflow with a single activity.

3. As a user starts TSE, the Personal Agent (PA) registers itself and a number of its properties with the Naming Service. These properties include name of the user, his current role, network information for his current machine, etc.

4. As TSE comes up, it asks the PA to retrieve current activities for the user. The PA then sends a request to the PM to retrieve all activities assigned to the user. Some of the activities are immediately executable, while others are waiting for events, for example, completion of previous activities. For those activities which are immediately executable, the PM sends requests to the Agent Factory (not shown in the above diagram) to create agents. After all agents are created, the list of activities is returned back to the PA from the PM.

5. The PA passes a list of activities back to TSE. The TSE simply displays those activities.

6. The user starts an activity from TSE. The TSE then sends an 'execute' request for the activity (with agent) to the PA. The PA forwards the request to the actual agent and the agent executes the activity. For TSE, the agent will be generic and may already have a pre-defined user interface (in order to specify whether it has been completed or the percentage completed and so on; deliverables; inputs, previous history; duration; etc.).

7. When the current activity is complete, the agent sends a request to update back to the PM and terminates.

3.4.6.2 Scenario 2 (Domain Application):

1. The Process Manager (PrM) is created by the project manager.

2. As a user starts the GUI, the Personal Agent (PA) registers itself and a number of its properties with the Naming Service. These properties include name of a user, his current role, network information for his current machine, etc.

3. When the GUI comes up, it asks the PA to retrieve current activities for the user (passing the parameter specifying which process components to retrieve). The application specific agents such as Defense Design Review (DDR) process agents may be located at remote locations and be managed by a remote PrM. The PA then sends a request to the PrM (located at the same machine as in PA) to retrieve all activities assigned to the user. As a result, the local PrM knows where to go and ask about these agents. Some of the activities are immediately executable, while others

are waiting for some events, such as completion of previous activities. The PrM indicates which activities are immediately executable. After all agents are retrieved, the list of activities is returned back to the PA.

4. The PA passes a list of activities back to the GUI. The GUI simply displays those activities.

5. The user starts the activity from the GUI. The GUI then sends an 'execute' request for the activity to the PA. The PA forwards the request to the actual agent and the agent executes the activity.

6. When the current activity is complete, the agent updates its own state.

3.5 IPS (Integrated Performance Support)

3.5.1 Initial IPS Objectives

The goal of Integrated Performance Support (IPS) is to "communicate the information people need to do their job well, at the moment they need it". A full IPS system has been initially defined to consist of the following key conceptual components :

- Worker - identify and maintain background and skill of the user
- Advice - provide experience-based advice or guidance
- Work - identify and maintain requirements, inputs, and outputs for a given task
- Tools - aids which help the user to accomplish a specific task, with little or no supervision

Ideally, these components are focused on, and organized around, a problem domain workspace, so that what is actually provided is a human-problem domain interface rather than a human-computer interface provided by the traditional help system.

3.5.2 IPS Design Rationale

From early on in the TRP project, two distinct types of IPS have been identified. The first type, which is called Taskware IPS, is the support that is provided within a particular tool in order to use that tool effectively. Given TRP's intention to meet its goals by both building internal tools and integrating COTS tools, it is clear that the various TRP tools will have varying degrees of this kind of IPS. The most notable Taskware IPS in the TRP project is Model Explainer for the Component Specification Tool (for more detailed description, see Chapter 4, Section 2 on Model Explainer). Unfortunately, extending Taskware IPS requires that open architectures will be available for those COTS tools. For this reason, during the TRP project, Taskware IPS has not been emphasized.

The second type of IPS, termed Teamware IPS, is the support that is provided for the overall TRP tool suite, and as such focuses on coordination and collaboration among people, processes, tools and deliverables. Teamware IPS has been the main focus of our efforts, which is supported through the tool called Process Explainer (for more detailed description, see Chapter 4, Section 5 on CogentTIPS).

3.5.3 IPS Specification

As expected, overall IPS specifications may come from many different sources. In the case of existing process such as ROAD or BlueBook, descriptive information is available largely from informal documents in various formats, such as Microsoft Word or HTML. This information, in general, provides the basis for high level methodology, explaining what must be done to achieve specific goals.

After reviewing overall methodology, the detailed process is encoded using the Process Specification Tool (PST, which is a part of the Component Specification Tool). Once encoded, the PST generates Process Component Specifications describing each activity with preconditions, post conditions, transitions, roles and the tools involved. This information provides the IPS with the definition of a structured process which is supported by the TRP process architecture.

Once the definition is retrieved, the IPS may be able to extract the run time instance data information from a Plan Manager which maintains the project plan. The instance data contains information such as task status, user assigned to a task, and dependencies.

3.5.4 IPS Overall Architecture

The overall IPS architecture consists of an informal process methodology, process specification tool (CST), IPS module, process instance data and a process architecture, as shown below in Figure 10. The bubbles in the diagram represent significant entities (not necessarily components) involved with overall IPS and the arcs represent the interaction between those entities (not necessarily interfaces within components). The detailed explanations for each of the entities and their interactions are described below.

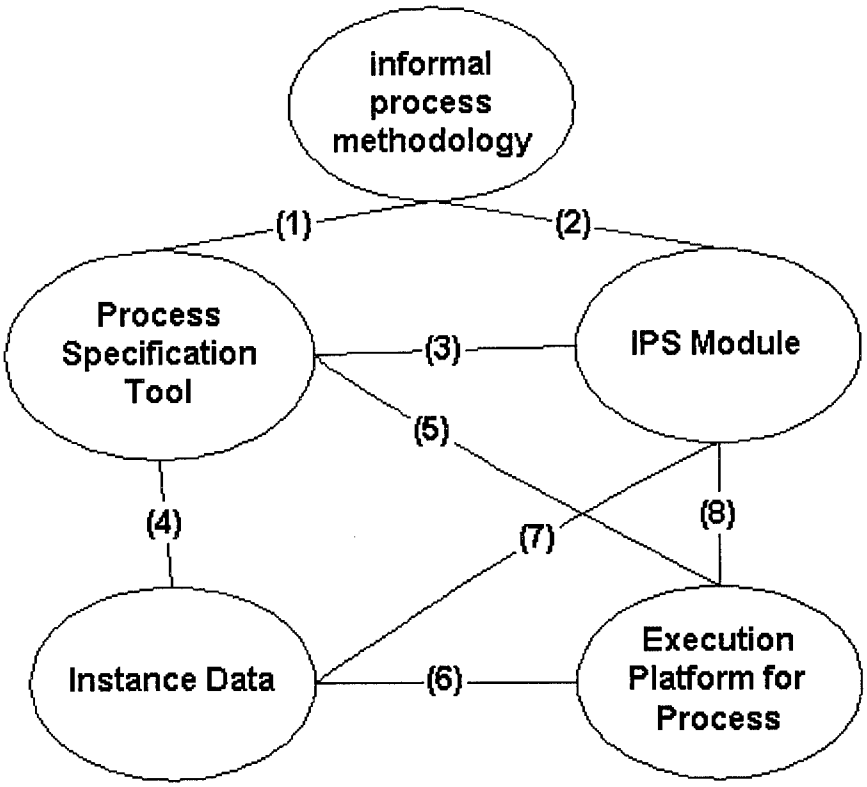

Figure 10 - Overall IPS Architecture

3.5.4.1 Informal Process Methodology

This document may be in various forms, such as HTML or Microsoft Word, with arbitrary structures. It describes basic process concepts informally. For example, in TRP, we have the ROAD process in HTML form, which consists of "form", "function", "guidelines" and "evolution". On the other hand, the BlueBook process is structured differently, introducing concepts such as "overview" or "planning". The purpose of this module is largely for high level description and review by team members.

3.5.4.2 Process Specification Tool (Component Specification Tool)

PST allows an individual to formally define what the process should be (in terms of components). Various characteristics are defined in the process specification tool (PST), such as preconditions, post conditions, events, deliverables and roles. The tool is used to generate a process component specification which is a subset of the overall component specification. The basic information for constructing a process (using the process component specification tool) initially comes from the above informal process methodology.

For each process, the analyst studies (reviews) the above informal process methodology and decides which part of it needs to be encoded in PST. This is because not all of the informal process methodology is appropriate for formal encoding in PST. Therefore, the link shown as (1) from the above diagram is largely a manual process.

3.5.5 IPS Module

The IPS module provides the explanation for the overall process defined in PST. In addition, the IPS incorporates the instance data in order to display the current status (shown as interface (7)).

The interface (3) suggests the process specification will be parsed by the IPS module. Once IPS retrieves the process definitions from PST via the interface (3), an IPS person (possibly a business architect) reviews those processes and analyzes the informal process methodology to extract detailed descriptions for each process. After sufficient review, the individual encodes the detailed descriptions of each process.

The interface shown as (2) from the above diagram is mostly manual. The interface (2) is used to extract the IPS description from informal process methodology while the interface (1) is used to extract the process definition information.

The interface (8) is primarily used to retrieve actual IPS information from the execution environment. For example, the TRP desktop may ask questions such as what am I suppose to do for this activity or what is the current status (who are currently on)?

When the process is modified (customized) in PST, for example a new activity is introduced, the IPS for it will initially be empty. The interface (2) from IPS will be used by a person to encode the description of this activity. Once the encoding is complete, the IPS can extract the relations between the newly created activity and the other existing activities from PST (via interface (3)).

When the existing activity is deleted, the IPS simply notices its absence from PST (via interface (3)) and will ignore IPS information for this activity. All other previous relations between activities will be recalculated from PST. When the customization occurs at a high level process, a similar scenario occurs, which should not cause any problems

3.5.6 Instance Data

Instance data represents the actual status of each instance of a process. In the case of the TRP desktop, it represents a project plan and its current progress. The definitions for each process come from PST via an interface (4), and these definitions should be identical with those from interface (3) for consistency. A project manager may augment the instance level data (such as adding multiple activities) and this information will be made available through another interface (7).

The interface (6) is used to retrieve the instance data for each process execution. It provides detailed information regarding each activity (such as the resource assigned; scheduled start date; duration, etc.).

3.5.7 Execution Platform for Process

This is an actual execution platform for process instances. Currently, it is based on a mobile agent architecture.

The interface (5) is used to generate the definition of the process (component) for the execution environment. There are a series of transformations before the actual process component can be created with the appropriate instance data from interface (6).

4. Tools

The tools team concentrated on creating an extensible framework for incorporation of third party and internally-developed tools that support the ROAD process. The tool framework is illustrated below in Figure 11:

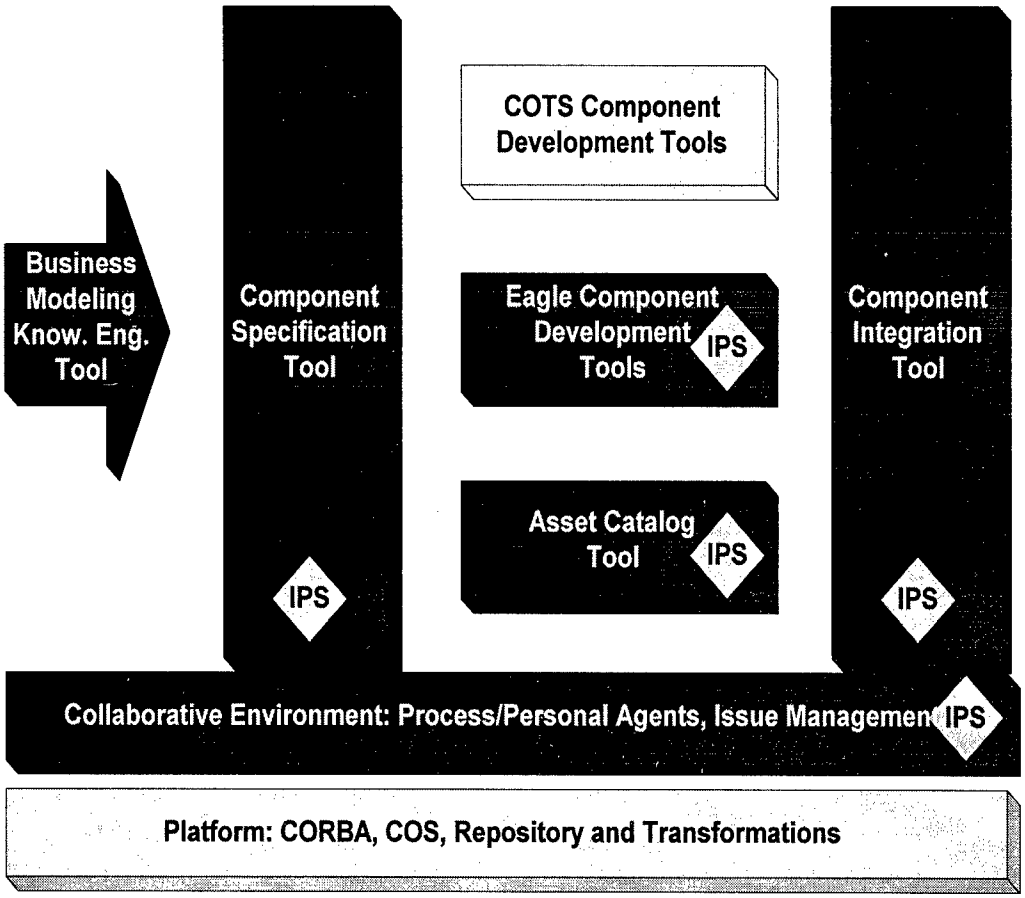

Figure 11 - Tool Framework

4.1 Business Modeling Knowledge Engineering Tools

One of the original goals of the TRP project was to incorporate knowledge engineering techniques into the component-based solution development process. We chose to incorporate a suite of tools named PC-PACK from a third-party vendor named Epistemics, based in the UK, as the TRP VITAL toolset.

The VITAL tools help a domain analyst extract as much information as possible from the managers, users, experts and legacy documentation in the system domain. The goal of an analyst is to produce a requirements and domain model which adequately describes the proposed system in terms that its purchasers and/or users can fully understand.

The VITAL tools incorporate established knowledge engineering techniques into the requirements elicitation and analysis with the end result being significantly more accurate analysis models. The VITAL tools include the following:

Protocol Editor	Provides markup tools to begin structuring the knowledge contained in a requirements document or interview transcript.
Laddering Tool	Structures the concepts, attributes, processes, and relationships identified by the protocol editor into a classification hierarchy.
Card Sort Tool	A tool used to differentiate between concepts by sorting them into "stacks" based on common attributes.
Matrix Tool	Represents concepts, attributes, and values in a matrix form that simplifies editing of the VITAL repository.
Repertory Grid Tool	Shows "ratings" of the attributes owned by concepts and relates concepts based on those ratings. This simplifies tasks of eliminating redundant concepts and, with advanced techniques, eliciting new concepts
GDM Workbench	A collection of abstract process patterns that can be applied to virtually any problem. GDM (Generalized Directive Model) patterns can be "grounded" in a domain model through the use of the control editor.
Control Editor	Specifies control and data flow between processes and concepts
Relationship Tool	Specifies relationships between concepts, attributes, and processes.

The VITAL tools implement cognitive psychology techniques to elicit knowledge based on theories about mental models. We have found these techniques to be more effective and efficient at eliciting knowledge than simple interviewing and note taking. They allow the analyst to discover domain elements that would otherwise be missed or surface as issues during implementation, when it is too late.

The VITAL tools also provide the ability to take a new approach to development of a solution model. In traditional development, analysts start with domain scenarios, abstract the policies out of them and compose a solution. Because reliable, standard solution models do not exist, every solution is essentially developed "from scratch." The Generalized Directive Model (GDM) technique implemented by the VITAL tools provides the structure and rules necessary to start instead with a similar (canonical) problem, take its solution model and adapt it to the problem. Bearing a great deal of similarity between GDM and the work of the patterns community, the result is solutions that take characteristics of previous work.

The tools team expended significant effort into integrating the VITAL tools into the TRP tool suite. We funded major enhancements of the tools to include development of the relationship tool, and a REMAP-like capability to gather rationale-based concepts like "requirements", "issues", "positions", "arguments", "assumptions", etc. in order to explain better the design choices made as solution construction progresses. (See Section 4.3 for a description of REMAP.)

We also developed utilities that enable the VITAL tools to participate in the TRP transformation architecture. These utilities essentially export the content of a VITAL model into CDIF (CASE Data Interchange Format) format, and import CDIF data from other tools into a VITAL model. These utilities lessen the need for manual transformations between different activities in the development process.

The tools team expended significant effort implementing business rule editing functionalities in VITAL, in line with the description of rule assets in the ROAD process described in Section 2.2. These include graphical representations of dependencies between domain elements, based on data mining and induction capabilities already available in the VITAL toolset.

4.2 Object Technology Visualization

An important aspect of the validation of the Requirements and Domain model is the verification of the dynamic behavior of the system (interactions based on business case scenarios). Animation and simulation can play a significant role here and we have found that the use of interaction diagrams and 3-D animation are a natural choice. We incorporated the Object Technology Visualization (OTV) tool, originally developed by Andersen's Education Division, into our tool set.

OTV supports capture of scenarios and interaction diagrams, and uses three-dimensional techniques to visual those interactions. While many dismiss the 3D techniques as unnecessary, we have found the visualization techniques to be especially helpful in explaining dynamic interactions between components to non-technical process participants.

The Tools team efforts in integrating OTV into the tool set involved enhancing the tool's integration diagram editor to collapse the interactions between clusters of objects, and in integrating the tool with the TRP transformation architecture.

4.3 REMAP

The REMAP tool was a modification of Andersen Consulting's REMAP tool for the ADM project. REMAP is used in recording requirements, issues, arguments, and positions and is especially useful in the early parts of the process. We re-implemented the Andersen REMAP model using the TRP technical architecture, including both an ObjectStore database and a NeuronData front-end.

The first release of the REMAP tool was not seen as especially helpful in recording issues, arguments, and positions. It also became apparent that REMAP duplicated some of the functionality of the VITAL protocol editor and laddering tool.

We dropped support for REMAP early in the project because it was not adding any apparent value to the process. We did, however, keep many of the core concepts. Issues, positions, arguments, and assumptions were integrated as core concepts by Epistemics into the VITAL tool suite. Issues resolution features were also added to the TRP desktop and the asset catalog to take a more proactive approach to the reactive REMAP model.

4.4 Component Specification Tool

The component specification tool (CST) is used to specify the structure and behavior of components in a solution. The most important concept behind the component specification tool is separation of interface from implementation. Interfaces specified in CST serve as a contract between analysts, designers, and coders. An implementation that conforms to a well-specified interface leads to greater consistency throughout the development process.

Ensuring that components deliver their required capabilities is a significant task requiring detailed knowledge of both the requirements and the expected behavior of components. TRP developed the Component Specification Language (CSL) to capture the semantics of components and their interfaces. CSL extends the Object Management Group (OMG) Interface Definition Language (IDL) to extend in terms of semantic description.

In CSL, components are defined by the *interfaces* they support. The description provided for the interface should be sufficient to enable clients to use the component effectively and correctly. In order to enable location independence and dynamic configuration, implementations can register that they support a given interface at run time. The details of an implementation are generally hidden from its clients.

The component specification forms a kind of *contract* between the component building team and the rest of the solution. The logical separation between the areas of functionality being addressed by each component also ensures that teams can progress comparatively independently of each other.

For description of domain objects, CSL extends IDL in three major areas: design-by-contract, states, and event specification.

4.4.1 Design by Contract

The idea of software contracts originally appeared in [Meyer88]. It presents a design metaphor based on contracts between objects and clients, which aids in creating robust and correct software. One of the goals of component-based software engineering is to improve the reliability of systems. The use of software contracts enables TRP components to realize this increased reliability in much the same way that legal contracts guarantee satisfactory business transactions. They specify the relationship between consumer and supplier (or client and server), describing the obligations of both parties to guarantee a successful relationship.

Interfaces implement software contracts through the use of *preconditions*, *post-conditions*, and *invariants*. Preconditions and post-conditions constrain the behaviors provided by an interface. A precondition expresses the conditions that must be met for a behavior to be called correctly, representing the client's side of the contract. A post-condition characterizes the situation that occurs when a behavior executes successfully. It represents the component's side of the contract. So a precondition describes an obligation to be met by a client before invoking a behavior, while a post-condition specifies an obligation to be met by the behavior. Invariants specify conditions that must be maintained by all operations in an interface.

The design-by-contract features found in CSL increase the semantic level of interface description well beyond the capabilities of IDL. An interface specified with CSL forms a kind of *contract* between the component building team and the rest of the solution. The logical separation between the areas of functionality being addressed by each component also ensures that teams can progress comparatively independently of each other.

4.4.2 Event-based Communication

Distributing an object-oriented solution across a number of nodes in a network adds complexity to the more common single-node or single-process implementation model. In the single-process environment, there is often only a single thread of control passing through the execution of a program. The message-passing inherent in the object-oriented

approach is essentially synchronous: the sender waits until the receiver of the message returns a response before continuing. In a distributed environment, such a single-threaded approach would not take advantage of the concurrency possible when multiple nodes operate simultaneously to solve a problem.

The current communication model in distributed object systems corresponds most closely to the non-distributed message passing model. That is to say, the requester sends a message to the receiver and waits for a response. Only once the response has been received does the requester continue. This kind of communication does not foster parallelism in a distributed system. Nevertheless, it is appropriate and useful in many situations.

Synchronous communication is by nature *tightly-coupled*. The requester and often the receiver have explicit knowledge of the identity and type of the other object. However, there are circumstances where the requester does not (or should not) know the identity of the receiver. There are two common cases. Firstly, when there are multiple equivalent receivers for a message. The requester does not care who processes it (as long as *someone* does). In this circumstance the message should be addressed to the pool of receivers. The second case is where receivers register interest in particular messages being sent. These usually indicate occurrences of interest in a given component (major state changes are the primary example). In this case it is not appropriate for the sender of the message to identify the receivers. Indeed their number and identities may change without the sender being aware of it.

Both of these cases are covered by a publish/subscribe model. One component specifies that it will publish events (special types of messages) in given circumstances. These messages will not be directed to any individual receiver. Rather all potential receivers indicate interest in the event by subscribing to it. This ensures that when the event is published each subscriber will receive a copy. This type of interaction where the sender is unaware of the identities of the potential receivers is known as *loosely-coupled*.

Events are specified in CSL in a manner similar to data structures. An event has a type, with every event of a certain type carrying the same type of information. Events are published and subscribed by components and interfaces. Integration features of CSL allow matching the events published by one component with those subscribed to by another.

4.4.3 States: Simplifying Dynamic and Multiple Classification

In a capability-based model, the features provided by an object determine its capabilities. However, over the lifecycle of an object, those features may change. An employee may start as a mail room clerk, then be promoted to supervisor, and later be appointed to an executive position. As a mail room clerk, mail delivery is the employee's main responsibility. As a supervisor, the employee's responsibilities also include scheduling and counseling. As an executive, planning and budget preparation are the main responsibilities. The responsibilities of that employee change over time, but the employee does not change identity (personality and identity are separate concepts!). The concept of dynamic classification, as described in [Martin 1992] provides an elegant solution to problems of this type.

An object may play multiple roles at any given time. Through participation in a stock ownership program, an Employee may also be a Stockholder. The concept of multiple classification, described also in [Martin 1992] enables description of this type of problem. Unfortunately, few object-oriented languages directly implement either dynamic or multiple

classification: the transition from reality to implementation becomes especially problematic in situations benefiting from these modeling techniques.

We address this problem in CSL through the use of states. An interface can contain zero or more states. Each state offers a set of operations and attributes that are valid only when the object is in that state. So in the problem illustrated above, an Employee interface has four states: MailRoomClerk, Supervisor, Executive, and StockHolder. Each state offers its own set of operations and attributes. The first three (MailRoomClerk, Supervisor, and Executive) are mutually exclusive: an employee may only be in one of these states at any given time. The fourth (Stockholder) is not. An employee in any of the three employment states can also be in a Stockholder state.

In translating from CSL to IDL and implementation languages, the operations and attributes enclosed within a state are promoted to an interface level. Any violation of the state specification (for example, calling the PrepareBudget operation on an Employee object in the MailRoomClerk state) is signaled by an exception.

4.4.4 Progress

The Tools team initially developed CST using a third party meta-case tool. The first version of CST supported specification of the structure of a domain component. The second version added design-by-contract capabilities. Version three added process specification capabilities. Version four replaced the third party tool as the foundation of the tool, replacing it with an HTML-based user interface and a persistence mechanism based on the asset catalog's storage model.

Results in using the component specification tool were mixed. Developers found value in the concepts behind CST, but the third party CASE tool offered very poor usability and hindered attempts to specify components. Developers found little additional benefit in graphically describing components, and bypassed CST by creating specifications directly in IDL.

This situation led us to examine the use of a third-party CASE tool as a foundation for the Component Specification Tool. We decided to replace the third party tool with internally-developed technology, removing dependencies for component specification. We chose to integrate the interface of the component specification tool with that of the component integration tool and asset catalog.

Reaction to the final tool was again mixed. The HTML interface we chose made display of existing specifications more intuitive. However, using HTML to capture specifications served to be less useful. It appears that our original method of specification was too graphical, while HTML was not graphical enough.

Earlier versions of the component specification tool were seen as taking priority over the asset catalog. In the final version, this situation was reversed. The asset catalog served as the main focal point for specifying and integrating components. The component and solution specification tools took on a support role. The switch in focus was in keeping with TRP's goals. From the analysts and designers' vantages, we moved from a component and implementation-centric view to a reuse-centric view.

4.5 Component Integration Tool

The component integration tool is used to assemble solutions from existing components. The key concept behind the component integration tool is late binding. The later the binding, the better the chances for reuse. Delaying the decision as to which components cooperate to implement a solution increases the ability to "plug-and-play" with different components.

When implementing a component during solution modeling, developers concentrate only on the semantics of a single component. All interaction outside the component itself occurs through the component's provided and required interfaces. The reason behind this strict partitioning is to eliminate the need to know which components provide the interfaces required by another component.

A component providing interfaces should not need to know which components are using those interfaces. Likewise, a component requiring interfaces should not need to know which components are providing those interfaces. That is, until it is absolutely necessary.

We have found four different methods for integrating components:

Static Integration	Connecting components whose type, semantics, and location are known at build time. Static integration is the simplest form of integration and is most suitable for classic client-server connections.
Dynamic Integration	Integration based on a "trading" mechanism where components whose location and semantics are not necessarily known at build-time. Dynamic integration is suitable for "peer-to-peer" communication models
Event Integration	Connecting components via event publication and subscription. Event integration builds on dynamic integration capabilities, and is most suitable for very loosely-coupled solutions.
Unconstrained Integration	The most infrequently-used type of integration that uses features of the CORBA interface and implementation repositories to connect components whose type are unknown at build-time. Unconstrained integration is useful mainly for development tools (browsers, automated testing tools, etc.)

The main difference between the different integration methods is the time at which the actual binding between the components in a solution occurs. Static integration occurs at build time. Before a solution can be packaged, all static integration must be completed. Dynamic integration occurs after packaging, at creation time. Dynamic connections between components are bound when the components are created. Event integration occurs even later, at execution time. A subscribing component will bind to a publishing component when the latter publishes an event. During analysis and design of the first version of the component integration tool, we identified four distinct methods of integration: static, dynamic, event, and unconstrained. Static integration binds components together at build time, and was therefore not of primary interest to the tools team. Unconstrained integration was useful in very limited circumstances. We therefore concentrated on dynamic and event-based integration.

The first version of the component integration tool dealt with matching required and provided interfaces. Interfaces required by one component were matched with those of the same type provided by another component. The bindings were captured by the

component integration tool. An integration service produced by the infrastructure team processed the bindings, providing a matching service available to components at runtime.

The first version of the component integration tool had drawbacks. It ignored the fact that multiple instances of a component may be available at runtime. We addressed this problem in the second release of the component integration tool by implementing a property and constraint based trading mechanism. The component integration tool supported specification of the properties of provided interfaces and the constraints on required interfaces. These properties and constraints were processed by the integration service at runtime, with the assistance of a minimal trader service produced by the infrastructure team.

The second version of the component integration tool served to be more flexible than the first. However, it made the simple dynamic integration bindings enabled by the first version impossible. The third version of the component integration tool combined these two capabilities with event publication and subscription support.

Response to the concepts underlying the integration tool was basically positive. After training and workshops related to component integration, Raytheon application team developers saw the benefit of delaying binding decisions. The benefits became especially clear with their implementation of the agent-based Defense Design Review process.

4.6 Asset Catalog Tool

The asset catalog tool stores descriptions of reusable assets to foster a widespread reuse program. The key idea behind the asset catalog is that it contains descriptions of assets, not the assets themselves. While examining the state of the repository market, we realized that creation of another proprietary repository architecture would provide little benefit. We had no desire to duplicate solutions to problems such as configuration and version management already solved by existing commercial repositories. We switched focus from the more common model of storing physical components to the storage of reusable asset descriptions.

The first version of the asset catalog tool was named the component catalog tool (CCT) and was essentially a proof of concept. We demonstrated the ability to store the content of component specifications in an object-oriented database. Specifications created by the component specification tool were parsed and stored in the asset catalog. A rudimentary Java-based user interface was used to browse the content of those specifications. The focus of the first version of the asset catalog was portability and technical integration between CORBAPlus, HP Softbench C++, and the Versant object-oriented database.

The second version of the asset catalog added significantly more functionality. During evaluation of the ROAD process and analysis of the repository, we realized that a truly effective reuse program does not limit itself to component implementations. Requirement documents, analysis models, design models, and other assets are also candidates for reuse.

For the second version, we modified the asset catalog to store descriptions of multiple types of assets. The catalog could store semantic descriptions of assets created by each of high-level tools in the TRP tool suite: VITAL, OTV, CST, and Ptech. Version two of the asset catalog also served as the foundation for the beginnings of the TRP transformation architecture.

The third version of the asset catalog focused on two major requirements: traceability and usability. We implemented dependency links between assets produced using the TRP transformation architecture. These links allow analysts and designers to evaluate the impact of changes in one asset to dependent assets. We also made significant modifications to the user interface for the catalog, making it resemble commercial, consumer-oriented catalogs on the World-Wide Web. Reaction by Raytheon application team personnel to both changes was very positive.

4.7 Transformation Architecture

One of the early stated goals for the ROAD project was the reduction in the manual transformations necessary during the life of a development project. We realized that goal through the TRP transformation architecture.

The ROAD process is characterized by distinct phases: Domain Modeling, Solution Strategy, Component Construction and Integration. To support construction of quality solutions, each phase must be well-supported by development tools, but no single tool is sufficient to support the entire development process. This requires a "best-of-breed" approach to development process automation. The problem with this approach is that

- the techniques and tools required by each phase of the process are quite different, and
- the information the tools manage is dissimilar and it is difficult to propagate that information between tools.

The root cause of this problem is differing points of view. The perspective of a system for an analyst may be different from the one for a technical architect, and consequently, the meta-models for tools used by an analyst are different from those used by technical architects. Specifically, VITAL deals with "concepts" or "business processes"; OTV deals with "interactions"; CST deals with "components" and "provided" and "required" "interfaces"; and OO CASE tools deal with "classes" and "operations".

However, it is very likely that "classes" are based on "concepts", "operations" are based on "business processes" and "provided" or "required" "interfaces are based on "interactions". While the ontology or meta-model for each tool is distinct, the meaning behind the ontologies is similar. In current development processes, the transformation from analysis models to design models or from design models to implementation models is performed manually, relying on the skill and understanding of the person performing the transformation. The tools team focused on automating the transformation of domain content from the meta-model in one tool to the meta-model in another.

Our transformation architecture is based on three principles:

- Different meta-models and repositories would be used by different tools to provide different perspectives of the system
- Translation and transformation rules would support the mapping of elements between meta-models

- An Asset Catalog (not a common repository) where assets from the different activities can be found, manages access to both the tool meta-models, and the content of individual assets.

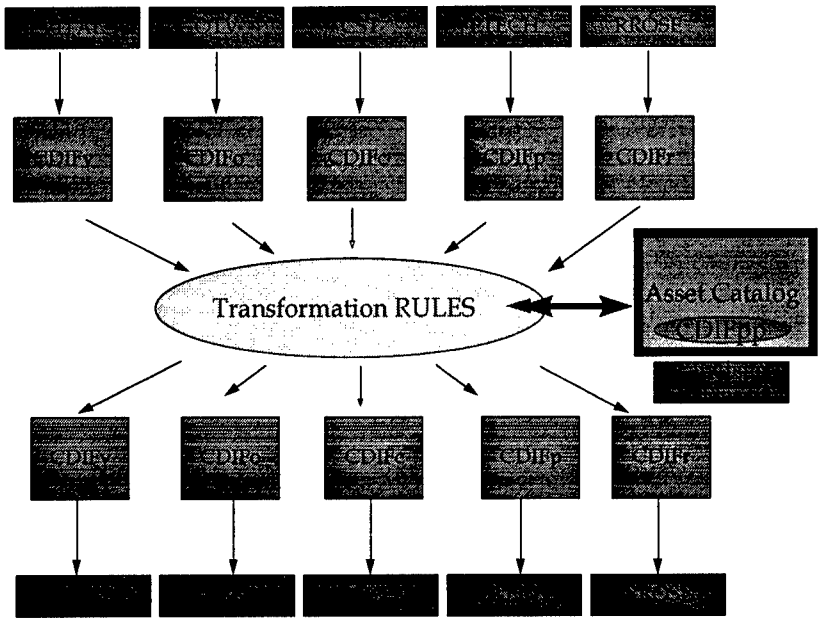

Figure 12 - Transformation Architecture

We adopted the CASE Data Interchange Format (CDIF) as a transfer and translation mechanism between tools. CDIF specifies both a transfer format and common meta-models, but because the common meta-models have not been widely accepted, we chose to adopt only the transfer format. The transformation architecture assumes that all tools participating in the TRP tool suite can import and export native content in CDIF format. The transformation architecture can then transform the content of an asset produced by one tool directly into the format needed by another tool, using translation and transformation rules to implement the actual transformation of content.

We implemented the transformation architecture using CORBAPlus, C++, and the public domain CLIPS inference engine. Inference rules are used in translation to and from a native CDIF representation into a common ontology. The content is then transformed by a different set of rules from one state in this common ontology to another.

The transformation architecture has been very well received, especially by third party CASE tool vendors. There is a good possibility that the TRP transformation architecture will be incorporated directly into commercial object-oriented analysis and design tools.

4.8 Unix and Windows Component Development Environments

The tools used to implement components on Windows and UNIX were selected by a joint team of Andersen Consulting and Raytheon application team personnel. The tools team assisted in the early selection and deployment of third-party commercial-off-the-shelf

(COTS) tools for use in developing components. Our efforts in this area were limited to recommendation, selection, and initial integration.

5. Infrastructure

5.1 Introduction

The Andersen infrastructure team established a development environment and execution environment for enabling TRP solutions. Our contributions spanned many areas, including:

- providing expertise to the Andersen team in key technologies, including CORBA (Powerbroker), object databases (ODI and Versant), and language specification, parsing, and code generation (using the Purdue Compiler Construction Tool Set, PCCTS, language tool suite),
- facilitating the selection of COTS tools by bringing in commercial tool vendors for group evaluation,
- establishing success criteria for integrating a COTS tool into the TRP tool suite,
- overseeing the development of component specification models,
- implementing a component specification language derived from the models,
- implementing runtime architecture to support component semantics,
- implementing code generators to convert CSL specifications into code and IDL interfaces,
- and, providing conversion utilities to allow proprietary models to be exchanged between specification tools.

The following sections illuminate some of the highlights of our primary deliverables: the Component Specification Language, and the TRP runtime component architecture services and facilities.

5.2 Component Specification Goals

In large, distributed software systems, the implementation landscape is potentially infinite and the entirety of the capabilities that a system may represent over time is *typically* beyond the comprehension of any one system analyst. One reason for this is that the semantics of the components of a system are not apparent from the models and specification techniques used to represent them. Often, the semantics which convey the capabilities and limitations of a component lies in a separate description detached from the specifications used to create executable code.

Object-oriented (OO) techniques offer much improvement over previous techniques in implementing and managing the complexity of a software application. OO languages incorporate language constructs that directly support the best implementation practices used by programmers, namely encapsulation and inheritance. Consequently, with corresponding training, general purpose OO languages offer modeling abstractions and developer support for delivering better understood (and hence, maintainable) software implementations.

Component architectures, such as CORBA and OLE, provide standard integration mechanisms for isolating implementation environments from a common integration environment. They provide benefits in integration that gives an organization more implementation options. Component architecture technology is useful and desirable at a design/implementation level.

Although OO technology and commercial component architecture technologies improve on previous technologies, we questioned whether they were enough to break the software development bottleneck facing virtually all organizations. Both allow complex systems to be implemented, but neither provide higher-order semantic constructs to allow components and interfaces to be browsed by their suitability to some developer's purpose.

From this perspective, the problem with general purpose OO languages and commercial component technologies is that they are specifications based on general purpose semantics. We found that the specifications for these systems were too low-level and implementation dependent to capture, in a concise, expressive representation, the semantics of the solution areas we were trying to model. More precisely, we found that by the time our technical architects get through with a business object specification, it has so many inherited interfaces and technology semantics in addition to its original purpose, that its original purpose becomes mitigated away by its technology roles. From an implementation standpoint, this is understandable and desirable. From a specification viewpoint, however, it becomes very difficult to separate the semantics of a component which should be ported from environment to environment from the implementation semantics which may or may not require the same technology characteristics.

For instance, a simulation tool to validate the semantics of a use-case across a set of components would not necessarily need all of the technology interfaces present to demonstrate the validity if the tool is seeded with the same specifications provided to the development team for creating the semantic skeletons of their implementations.

We wanted to provide notation and concepts for solution component interfaces that would illuminate, at the specification level, the *intent* of the components. We imagined treating these components (or interface packages) as reusable specifications, which could target implementations in a variety of implementation languages, integration architectures, and proprietary architecture services environments. We feel this approach allows a component in our component-based solutions to be:

1. *Stable* in terms of its integrated capabilities which will be valued and used in solutions,

2. *Adaptive* to accommodating deeper and richer functionality and features over time,

3. *Independent,* identifying its needs from a solution environment without building-in dependencies, and

4. *Well-defined,* allowing specification semantics for a context and describing how simple interface types can be assembled, constrained, and coordinated in the larger component context.

Our CSL language is one result of this goal. The CSL syntax is derived from a set of specification models we used to characterize the semantics of a component interface. The specification model approach we adopted is another significant result. The specification models underwent substantial change during the project as the purpose and potential of component specifications became more understood. The following sections illuminate some details of these results.

5.3 TRP Specification Models

When we started the project and decided to provide a semantically rich, heterogeneous specification language for our component specifications, it seemed pragmatic and a path of least resistance to base such a language on CORBA's Interface Definition Language (IDL). While the similarities between IDL and CSL are apparent, and CSL base types are based on IDL, this approach caused a great deal of confusion across the consortium members as the semantic models of what we were trying to accomplish and what IDL accomplishes are substantially incompatible beyond surface similarities. To address this, we spent significant time and effort developing specification models for each of our component types.

Our conceptual approach for each component type was the same, evolving from both direct experience and from the effort of simplifying Andersen's ODM methodology for component development. The following illustration (Figure 13) show's the high-level ingredients of the TRP component specification approach.

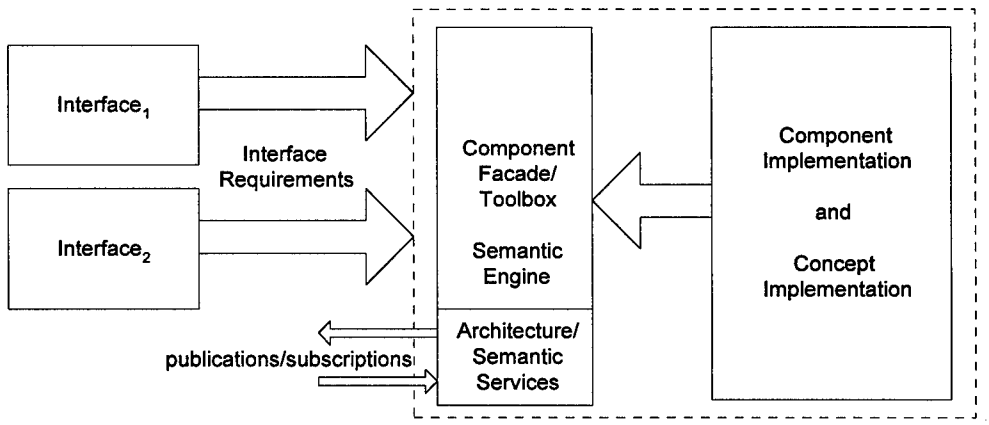

Figure 13 - TRP Component Specification Approach

Solution components represent semantics and capabilities that are not all directly accessible to an external client. The heart of a component is the reusable set of executable capabilities that may be extended and evolved to adapt a component to different solution environments. This set of capabilities provides the semantic engine to support constraints and state management of an interface, and provides the unstructured potential capabilities of a software toolbox for extending and evolving the component.

The elements comprising the component facade represent specification elements derived from both analysis and design. The following shaded picture (Figure 14) shows how the component semantics are contributed to.

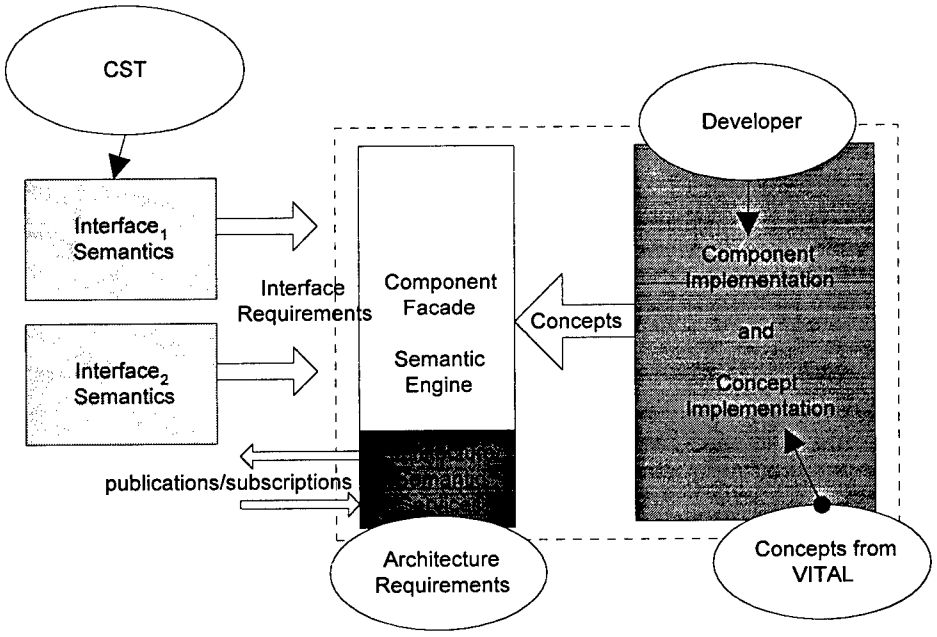

Figure 14 - Component Semantics

5.3.1 Solution Component Characteristics

A solution component is a domain of functionality that may evolve independently of other solution components so long as its provided and required solution interfaces are maintained. While we can implement solution components using OO and/or component technologies (and in fact, we do) we have more semantic leverage and traceability across time (maintenance teams) and space (distributed development teams) by defining semantic constructs that relate the internal and external characteristics of a component explicitly.

Object-oriented modeling techniques and component integration technologies offer leverage to realize different objectives for advanced system development. However, while both technologies improved the state of software engineering, neither alone provides a semantic specification model to allow high quality evolvable solutions to be specified by domain experts. We address this goal by mapping semantic constructs in our component specification language to mechanisms controlled by a semantic engine. When a client accesses an interface on a TRP component and invokes an operation, a semantic engine,

driven by information contained in the component specification, applies whatever semantic elements have been defined in their proper sequence.

Our approach builds on the strengths of both OO technology and component integration technologies. Our interfaces extend CORBA and Java interface semantics by allowing the declaration of operation lifecycle states, operation preconditions and post-conditions, application-level events subscribed to and published, and the declaration of concepts accessed by the interface. This last declaration is unique in that concepts, as mentioned above, are semantic, domain-level abstractions which help explain the *logical* behavior of an interface's operations, but which may or may not be directly accessible to a client outside of the component. They are used to illuminate the internal dynamics of the component as presented to a solution.

5.3.1.1 Key Ingredients of CSL

To realize these semantics, especially in today's technologies, our component specification language (CSL) integrates three key ingredients:

- Component technology for integration,
- Object technology for implementation, and
- Architecture-enabled semantic engines supporting semantic component specifications.

The purpose of CSL, specifically, is to provide a heterogeneous specification language that can, by automated transformations, contribute to architecture-enabled, functionally rich, evolvable software systems. CSL constructs for each component type relate specifically to the type of component being specified. In other words, we have chosen *not* to reuse general-purpose specification mechanisms to specify domain, process, actor, and rule components. Instead, the *intent* or purpose of a component, as characterized by its type, is the first reduction in complexity for a subsequent developer tasked with maintaining or reusing a given component specification.

Consider the following figure which illustrates a generalized, high-level view of a TRP solution:

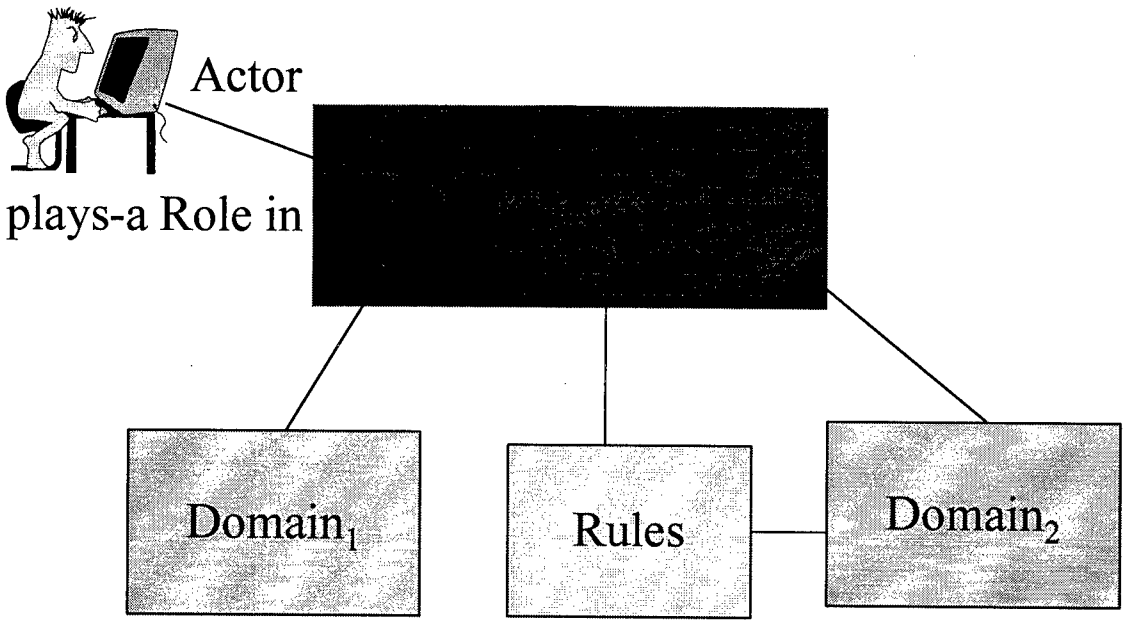

Figure 15 - Generalized, High-Level View of a TRP Solution

This high-level picture of how our component element types may be combined to create a solution is both generic and lacking in details. Still, the amount of *general* information about how the solution was produced is significant when an understanding of the semantic model for each component type is assumed.

5.3.1.2 Advantages for Developing Components

A component is an organizing concept which consists of provided and required interfaces. The interfaces for a component have semantics beyond type and signature. These semantics, specified in CSL, provide a set of requirements for implementation to the development environment. Using transformation technology, we generate stubs for semantic elements and architectural services (such as event services) that have to be defined by the component developer. This not only improves traceability between specification and implementation, it also represents a very quick path for delving into the internals of a component domain from a known entry point.

For a developer, the actual development required to implement a semantically valid component interface is analogous to using a framework to program by difference. While we recognize that the implementation behind an interface may be quite sophisticated, we believe our technique provides a sure-footed path to rapid solution development. Interfaces between components may be satisfied minimally at first, and their implementation made more reliable and robust during subsequent iteration.

The ROAD development process identifies assets created at different stages by different roles and/or tools which contribute to the final component specification and its implementation.

5.3.1.3 Key Ingredients of a Solution Component

So a TRP solution component generally has the following elements:

- Semantic Engine and Services (architecture)

- Semantic Specification Model (CSL Specs)
- Well defined Implementation Hooks (implementation requirements)

The following sections present models of the semantic structure for each of the four solution component types and for TRP solutions specifications. These models correspond directly with what can be expressed with CSL.

5.3.2 Component and Component-Based Solution Models

5.3.2.1 Notation Conventions

The notation conventions shown in Figure 16 are used within the attached models.

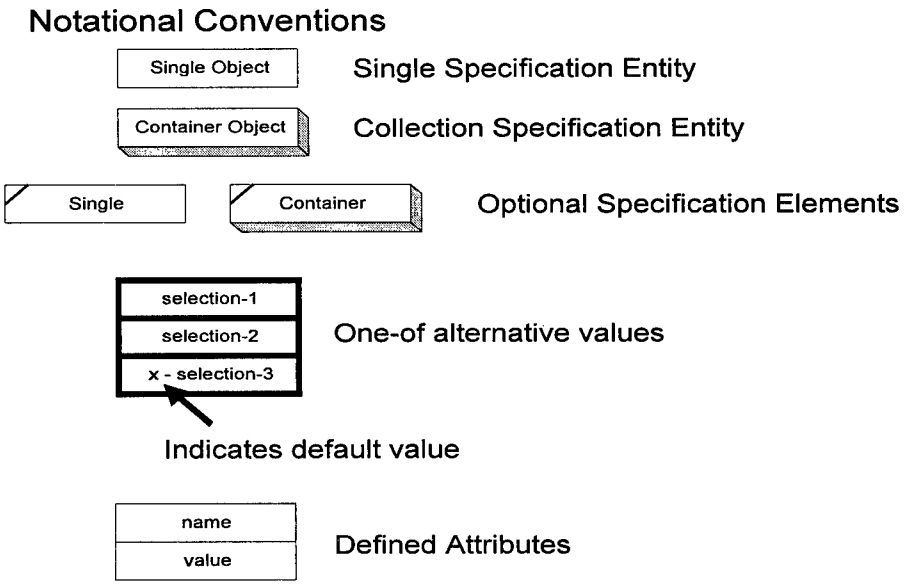

Figure 16 - Notational Conventions

These symbols are used to represent a schema of specification concepts.

Diagrams are generally read from left to right and generally illustrate a hierarchical decomposition of the template for a solution component type or for solution specifications.

5.3.2.2 Solution Specifications

Static bindings among the components comprising a solution can be represented by identifying the participant components, describing which required interfaces are bound to which provided interfaces, and describing event bindings between published and subscribed declarations within the participant components. The following picture (Figure 17) details the specification model.

Figure 17 - Solution Specifications

5.3.2.3 Structure of a Solution Component

All components share structure to support their participation and specification within a solution. In general, a solution component represents a set of contracts which allow a component to be analyzed for its capabilities, understood, and implemented in semantically consistent ways. The high-level view of a solution component is represented in Figure 18.

The following model defines the elements of a solution component and represents the common specification elements captured by domain, process, actor, and rule components.

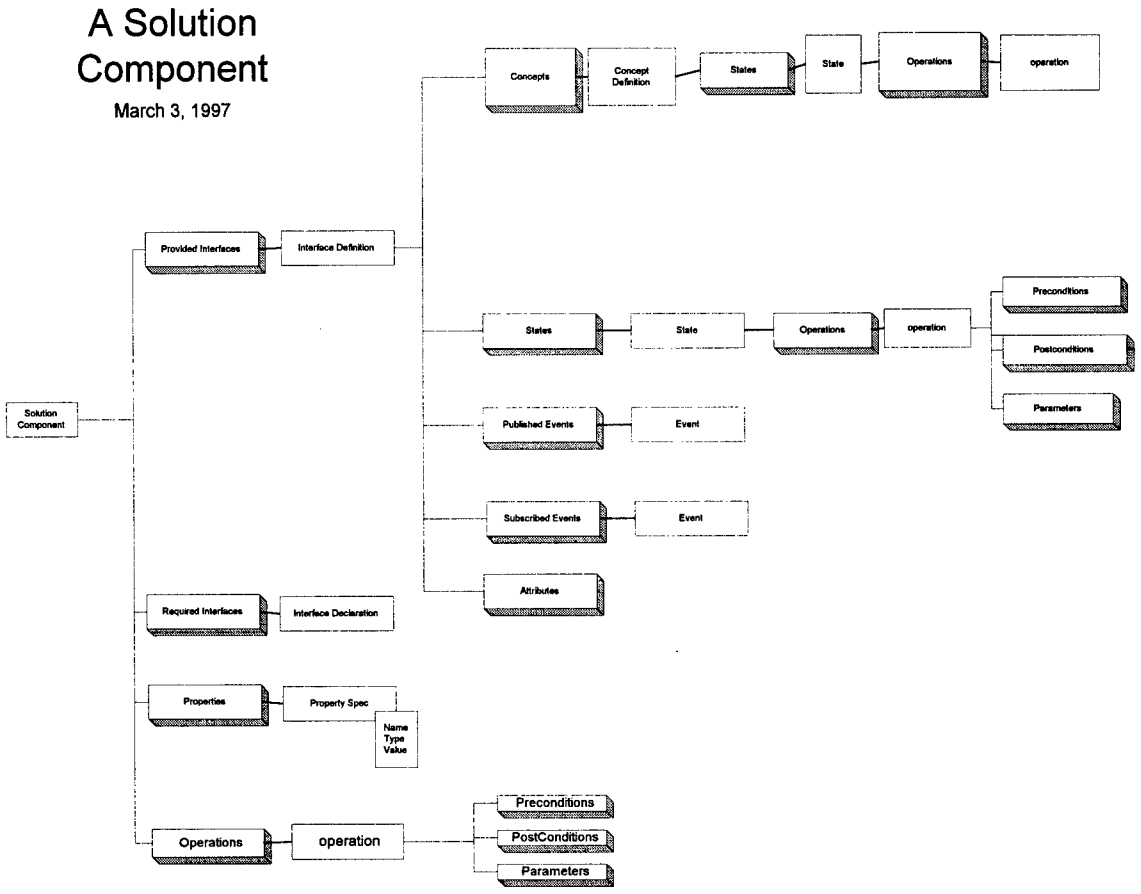

Figure 18 - A Solution Component

A solution component brings together three kinds of specifications: provided interface specifications, required interface specifications, and property specifications.

5.3.2.3.1 Provided Interface Specifications

Provided interface specifications extend the Object Management Group (OMG) Interface Definition Language (IDL) in ways to allow semantics to be associated with the interface that are not possible in current IDL. In addition to being able to declare signatures for attribute and operations, as IDL provides, we have extended the notion of interface to include declarations for published and subscribed events which occur while the interface is active, declarations for interface states which may limit govern the visibility of operations provided by the interface, and declarations for internal concepts which represent the cluster of information entities managed by the interface.

5.3.2.3.1.1 Concepts

Of these, concepts are perhaps the most unique. Concept specifications are a direct result of clustering activities which occur during the partitioning of business analysis entities into packaging units. Concepts are not necessarily directly accessible by clients of the component, but serve as a domain representation in which the operations on the interface can be defined and described. Because concepts are not automatically visible, their implementation may vary significantly from their concept representation. How the

interface is bound to the concept implementations is a result of the best suitable combination of automated translation, experienced developer, and available architecture capabilities.

With the notion of concept explicit, operations on interfaces can be treated as external system interfaces to the concepts declared in an interface. Explaining the behavior of an operation can be specified using object interaction diagrams detailing the sequence of actions between an interface operation invocation and the resulting collaborations amongst the concepts.

For a concept to be both an object for explaining the behaviors provided by the interface and accessible by clients, the concept would also have to be declared as a provided interface.

Concepts are similar in specification to interfaces. They differ primarily in their visibility. Because concepts are first and foremost a means of documenting and tracing the evolution of analysis activities and products (concept definitions) to final packaged realizations (interfaces and components), their implementation is intentionally left to criteria of the project at hand. Because of this, things that appear at the solution specification level (interface and event bindings, for instance), do not appear in concept specifications. These relationships are made typically made on a CDE (Component Development Environment) basis.

5.3.2.3.1.2 States

States are provided to allow interface operations to be scoped by interface life cycle states. Note, in this use of interface, we are discussing its particular relationship to the providing component, not its plug-and-play-ability for being plugged onto different implementations. States represent constraints on the implementation that must be satisfied by the implementation.

5.3.2.3.1.2.1 Events

Interfaces provide event declarations for events that may be published or subscribed from within the lifecycle of the interface.

In CSL, event types are declared outside of a component or solution model by event name and an optional structure detailing any event-specific content (all events are presumed to carry generally available information, such as sender).

5.3.2.3.1.2.2 Operation Extensions

IDL operation declarations are extended with additional qualifiers to indicate preconditions and postconditions invoked upon entering or leaving the operation scope. Preconditions and postconditions may be expressed in CSL expression syntax, or may declare the calling of a boolean operation defined at the component level. Such an implementation predicate may be presumed to be provided by the implementation environment to validate conditions outside of the direct scope of the interface. For example, checking to see that a customer exists before extending his/her credit might be such an operation.

This utility operation would be declared at the component level as:

```
boolean customerExists(string customerId);
```

Its call would be declared as a precondition on an operation to extend credit as follows:

```
operation extendCredit(string custId)

   precondition customerExists(custId);
```

5.3.2.3.1.3 Required Interfaces

Required interfaces are merely declarations of interfaces that the component will be accessing from its implementation to realize its objectives.

5.3.2.3.1.4 Component Level Operations

To realize the facade nature of solution components, utility operations may be defined at the component level which may be used to *build up* the capabilities of the component's facade. Different component types may place different demands upon such a convenience. These specifications are intended to support invocations which are visible in some way at the specification level. For domain components, a boolean operation representing a predicate which may be used in a precondition might utilize this mechanism. For process components, activities can invoke an operation. If the operation is not resolved to an explicit required interface, then it is presumed that there is a component level operation defined that can satisfy the request.

5.3.2.3.1.5 Properties

Component properties may be specified that can allow a specification to be more easily and elegantly adapted to different execution environments. Properties may be used at run time for any number of strategies to determine technical and logical compatibility of a component with a solution's demands. Properties are often orthogonal to the domain of the component but highly relevant to the particular execution platform, operating system, and integration architecture the component resides within.

5.3.2.3.2 Component Types

5.3.2.3.2.1 Domain Components

All components represent some domain. Domain components incorporate the solution component model to indicate that its purpose is to represent a business object that can be processed (created, persisted, modified, and deleted) and accessed by other components. Domain components are represented by the generic solution component model.

5.3.2.3.2.2 Process Components

Process components share the general characteristics of a solution component, but in addition may specify additional elements representing process semantics.

5.3.2.3.2.2.1 Process Component Overview

The following diagram (Figure 19) shows a process component overview. We show explicitly, at this level, that it shares solution component characteristics. We see, in addition, a set of specification elements representing the process characteristics of this component.

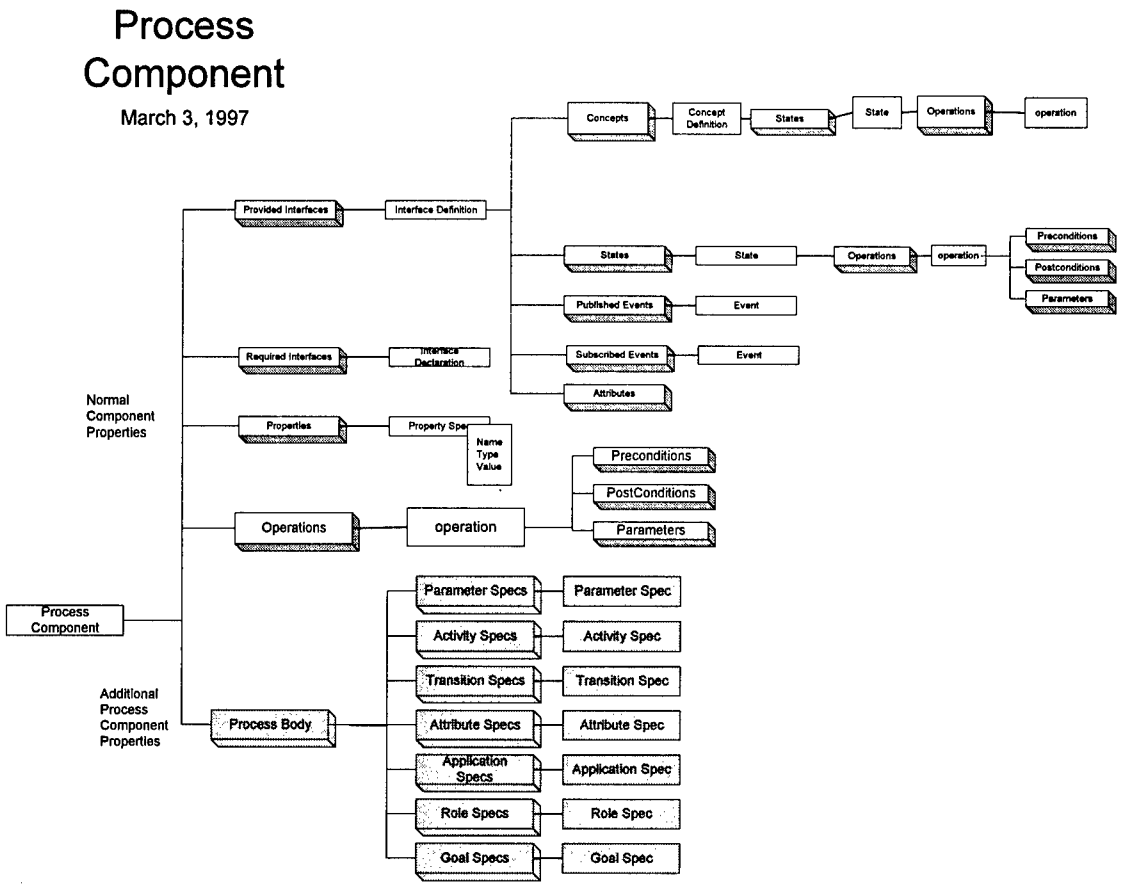

Figure 19 - Process Component Overview

The elements within the process body are further developed in similar models.

5.3.2.3.2.3 Rule Components

A rule component represents an entity that:

- Can provide policy judgments requested from other solution components, and
- Can provide external interface mechanisms to allow policies to change without influencing solution interface bindings.

The following picture (Figure 20) shows a typical sequence flow for the high-level elements of a rule component. The example is to evaluate the policy driven money adjustments applicable to a given purchase order, and return the adjustments in some itemized form. Not shown are the required steps for someone to populate the mapping schema with rules for mapping domain interface structures to rule facts representations.

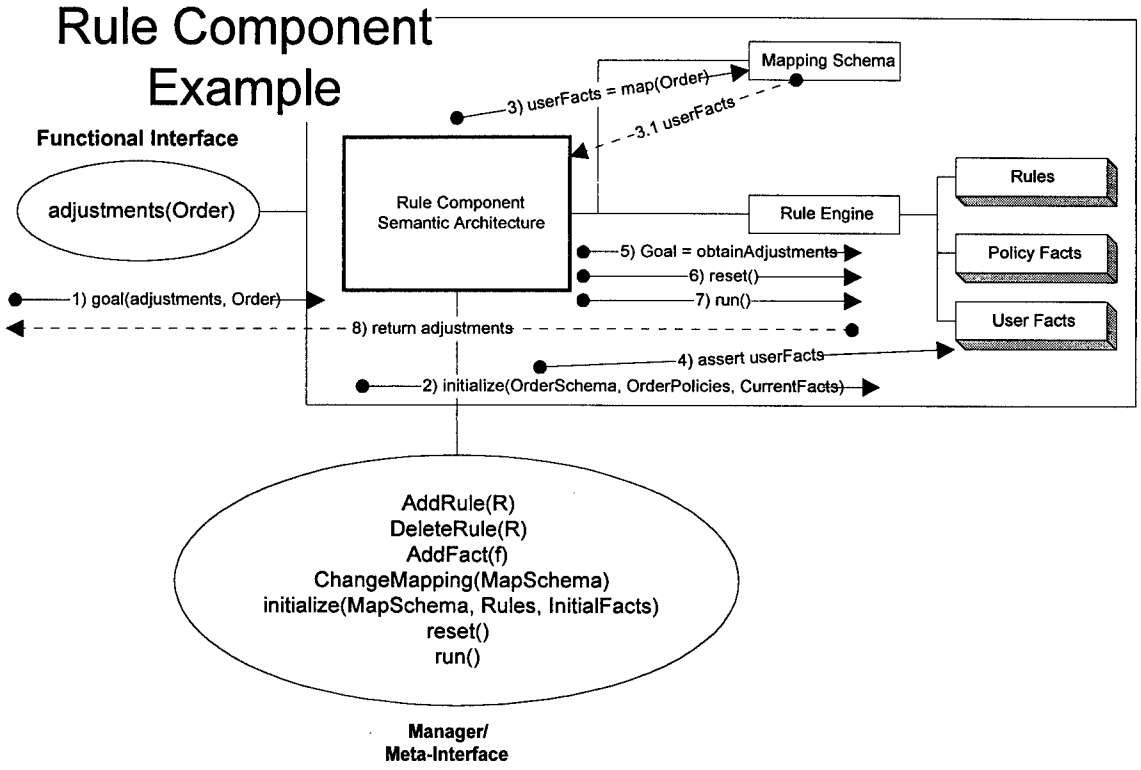

Figure 20 - Rule Component Example

While this is a reasonable model for rule components, it is by no means the only model. We think that the mechanisms and details of *how* rules are specified should be differentiated from the specifications for how it will satisfy its responsibilities to a solution. Further, we think that it is important to maintain this flexibility.

However, we also realize that, for certain classes of problems, that well defined rule engine and rule management mechanisms can be assumed and CSL for rule components could evolve to assume more process component structure (i.e. there are generic solution component structures for a rule component, and a rule body specification that details the knowledge structure for the component). We have not demonstrated this explicitly in CSL. Our intent is to maintain the advantages of both approaches by letting such a rule body characteristic for a rule component be *optional,* and use it only for a well defined category of rules. In hindsight, we should probably addressed process specification in the same way – providing one syntax for workflow, one for mobile agents, one for automated interface coordination, and so on.

Functional interfaces on rule components can be represented using normal domain component modeling techniques, however the rule keyword classification allows it to *qualify* for particular interfaces and/or properties in an implementation and execution environment to support the maintenance of varying solution policies.

5.3.2.3.2.3.1 Actor Components

Actor components represent external control interface boundaries to the solution. Typically, they embody a GUI, but there is no inherent reason why they can not

encapsulate a system interface (message distributor from an EDI source, for instance). Actor components can be used for two types of modeling situations.

The first occurs when you want to identify external points of control. The actor component passes control for the process into the hands of a user role driving the actor component. You might consider these pre-planned points of non-determinism.

The second situation occurs when you wish to model an activity and show its relationship to your business process, but the interface to that activity is completely outside the scope of the solution. Here, you rely on the user to indicate how the process should proceed, but there is very little if any solution interaction while work is being performed in the external environment.

Actor components can be represented using normal domain component modeling techniques, however the actor classification allows it to *qualify* for particular interfaces and/or properties in an implementation environment to be treated as a system access point to the solution.

5.4 The Component Specification Language (CSL)

The Component Specification Language (CSL) allows a direct representation of the specification models to be represented in textual form. CSL is a descendent of CORBA IDL in terms of core base types syntax but has specific extensions to support our specification characteristics goals described above.

The first semantic category CSL addresses is categorizing components by their primary purpose. We identify four fundamental component types:

- domain components,
- process components,
- rule components, and
- actor components.

We provide keywords in CSL to distinguish components implementing these types. Each type has additional keywords, as required, to distinguish specific semantics which may be applied to that type.

CSL has syntax for specifying public interfaces to a component. These descriptions are detailed enough to deduce implementation requirements for satisfying the semantic specifications within the CSL interface descriptions. In addition, CSL has notation for relating externally visible operations at the interface level to internal concepts that are (logically, at least) managed by the component implementation. For instance, a Library component may have an interface for checking out a book. Pre-and-post conditions on the check-out operation may indicate that an internal concept representing the instance of the book changes state from available to unavailable. A client may never see the instance of a book through the library interface, but we would have, at the specification level, an

indication that something managed by this component is undergoing a change as a result of one of its public interfaces.

5.5 Component Architecture Services and Facilities

The TRP runtime environment provides some convenience facilities and services to enable the CSL specification semantics for the CORBA environment. Due to the immaturity and general unavailability of CORBA COSS (Common Object Services Specification) services, and more importantly, to simplify the developer's task, we have standardized how certain mechanisms in the CSL semantic specifications are implemented in our CORBA execution environment. This standardization allows more code generation opportunities for enabling the semantics between CSL and our implementation environments.

5.5.1 Event Management

All components may declare publications and subscriptions to external events. The TRP event management facility allows an event subscription to be mapped to any compatible CORBA event by specifying mapping rules independent of a component's implementation. This allows semantically similar components to be *adapted* to a particular solution environment more easily.

5.5.2 Process Management

Facility interfaces for process management exist at both configuration and runtime. At configuration time, a process facility component acts as a server to specify process maps between process instances and process types, actors and process roles, activity descriptions and implementations, and process control mechanisms to sequencing rules. At runtime, the process management facility allows an instance of a process to be located and queried for status, and allows one of a set of pre-defined ad-hoc processes to be launched from within a running process activity.

5.5.3 Component Implementation Services

Each component type represents a kind of virtual machine with instructions in the CSL syntax to specify implementation elements which will collaborate in well-defined ways to realize the component semantics.

5.5.3.1 Interface Semantics

All components can provide interfaces. CSL code generators provide stubs and/or control mechanisms for the following interface semantics:

- An interface may define a set of logical states in which it may transition. Operations defined on an interface may be assigned to one or more logical states in which the operation may legally be invoked.

- An operation within an interface may be assigned a pre-condition (or post-condition) to evaluate prior to (or after) execution of the operation.
- Invariants may be assigned to an interface which will be validated as an implicit pre-condition and post-condition on all operations defined on the interface.
- Pre-conditions and post-conditions may reference interface attributes, operation arguments, or component concept states for validation.
- Interfaces may declare events that may be subscribed to and published within its lifecycle. CSL translators provide standard mapping for subscribing to named events and for registering an event-handling function with the subscriptions.

5.5.3.2 Service Request Stubs and State Management Frameworks

Implementation stubs are provided for subscribing to and publishing events. In addition, state management code is generated to assist the developer in adhering state semantics captured in CSL.

6. Technology Transfer

The Andersen Chicago team committed significant resources to technology transfer with Raytheon application team analysts and designers. Early in the project our efforts focused on training and mentoring in the ROAD process.

Raytheon application teams encountered problems in integrating several implementation tools. They performed a "discovery model" to minimize the risk of that integration, but the discovery model took much longer than expected to complete. To prevent this problem from recurring, the ROAD consortium agreed to form SWAT teams of appropriate Andersen, Raytheon, CoGenTex, and Expersoft people to path-find through the more significant technical risks. The SWAT teams assisted the application teams in installing and upgrading tools and architecture, and in creating process components. The result was a less painful experience in integrating new concepts and technology into their development and execution environments.

6.1 External Knowledge Transfer

The Andersen Chicago team participated in several events to transfer knowledge of our tools and techniques outside the ROAD consortium.

We demonstrated our tools and desktop at ObjectWeek '96, a commercial trade show; Andersen Consulting's ObjectWeek '96, a conference for Andersen Consulting employees and clients; and at Andersen's Global Consulting Seminar. The result was increased awareness of the ROAD consortium activities.

Members of the Andersen Chicago team have also been active in authoring papers for submission to industry and academic conferences and workshops. Those papers include:

- *The role of Knowledge Acquisition in Component Based System Construction (CBSC)*, Luis Montero and Colin T. Scott, 1996 Software Engineering and Knowledge Engineering (SEKE) Conference

- *Using Knowledge Transformation to Improve the Software Development Process*, Luis Montero and Colin T. Scott, 1997 Software Engineering and Knowledge Engineering (SEKE) Conference

- Agent-based Workflow : TRP Support Environment (TSE) , Jin W. Chang and Colin T. Scott, 1996, 5th International WWW Conference

- Integrating Components with WWW to Support Workflow, Jin W. Chang and Colin T. Scott, 1996, OOPSLA 1996 Workshop on Toward the Integration of WWW and Distributed Object Technology.

- Dynamic Collaboration for Agents and Objects, Jin W. Chang and Colin T. Scott, November/December Issue of First Class from OMG.

- TRP Support Environment (TSE), Jin W. Chang and Colin T. Scott, 1996, International workshop on CSCW and the Web.

- Mobile Agents Scenario for evaluation of Mobile Agent Facility, Jin W. Chang, 1996, OMG cf/96-10-05.

Chapter 2

Raytheon Application Teams

1. Introduction

Validation of the CORBA-compliant execution infrastructure, component-based software development methodology, and supporting toolset was performed by developing two applications in the areas of defense planning and commercial factory scheduling. Two application teams were formed. The military application was developed by RES (Raytheon Electronic Systems) System Design and Software Engineering Laboratories. The manufacturing application was jointly developed by Raytheon Computer Services Department and Andersen Consulting.

This section of the Final Report documents our practical usage of the methodology and toolset, describes the application systems, and discusses the issues surrounding our ORB-based technical architecture. The focus of each area is described below.

The methodology was directed to producing component-based object-oriented software solutions. It was also a user-centered spiral-development approach which promotes reuse through the design of reusable components. One of our goals was to adapt this methodology to developing defense-related software systems.

The development environment entailed a combination of consortium-developed tools and commercial-off-the-shelf (COTS) tools. Another goal was to investigate through the use of state-of-the-art tools, automatic code generation, and automatic document generation, the reality of rapid application development.

Two applications were built to address current problems in the manufacturing and defense planning sectors. Efficient management of factory scheduling is stifled by the large volumes of information the foreman must summarize to acquire an accurate picture of work center activities. The Work Center Management System (WCMS) provides the foreman with a view of the impacting factors within a work center and supports the prioritization and assignment of workloads to the appropriate operators. Defense planning is a time-critical cycle of analysis and refinement of defense effectiveness by collaborating staff officers to generate a set of deployment strategies to be used in real-time battle situations. The Capabilities-based Battle Management System (CBMS) provides the Operations Officer with the ability to conduct Force Operations defense planning and provides a three-dimensional view of combined weapon system capabilities.

The basic technical architecture supporting both applications consisted of an Object Request Broker (ORB) developed by Expersoft, an object-oriented database, Graphical User Interface (GUI) components, and processing components. Our goal was to exercise distributed object-oriented technology via this technical architecture.

2. Methodology

2.1 Introduction

Both Raytheon development teams used the TRP Methodology to guide application development. This methodology, based on Andersen's Eagle Methodology, was tailored specifically for use on the TRP project. This section of the Final Report is organized by methodology phase and briefly describes, for each phase, each development team's experiences and the lessons learned from exercising the methodology.

2.2 Business Modeling

The Business Modeling process provides important contextual information for applications under development so that they have the flexibility to meet both current and future needs. During the Business Modeling phase, the Raytheon teams studied the enterprises that would use the systems being developed (as well as the industry of which these enterprises are members). The teams created an Abstract Industry Model of the Aerospace and Defense Industry, using data from Andersen Consulting's Knowledge Exchange and DoD data as a baseline. Then, Concrete and Composite Models were created which documented the current state, benchmarks, trends and best practices for the enterprises which would use the applications.

Lessons Learned:

- Business Modeling aids in identifying best practices. It also documents the current and future needs of the enterprise, thus providing a baseline against which the system under development can be measured.

- The Concrete Modeling phase requires a great deal of time and resources to complete. Therefore, it is most cost-effective when integrated with a process re-engineering effort.

- It is difficult to schedule time with high level management in order to interview them on their views of the current status and future direction of the organization. Therefore, "substitutes" were used for some interviews (which can, unfortunately, alter the results of the analysis), or project due dates were missed in order to accommodate the schedule of an important company official.

- The models help team members understand the internal and external issues that drive the business processes. This is especially helpful for analysts/developers who are unfamiliar with the domain.

- A project scope should be defined in order to reduce the number of domain processes to be modeled. It needs to be focused quickly to the domains of critical interest.

2.3 Solution Strategy

During the Solution Strategy process, team members select the specific functionality to be developed and determined the implementation strategy. Solution Strategy documents were created for both applications. In preparing these documents, the teams analyzed the cost, benefit and risk of developing various business practices. The methodology helped

to determine the best solutions possible, within the project constraints, and to plan the iterations for developing the selected functionality.

Lessons Learned:

- The Solution Strategy document provides a "road map" for the project team, so that all team members understand what functionality will be developed and why. The document should be evaluated at key points in the project, and the implementation plan should be modified, as appropriate.

- It is important that end-users participate with information systems personnel in the Solution Strategy sessions. It helps ensure that the proper functionality is developed and that the end-users "buy-in" to the project early.

- The cost/benefit/risk data that was collected during this methodology phase helped to objectively prioritize the implementation of the business practices. It can also be used to help evaluate the feasibility of future application functionality.

- Using group "brainstorming" sessions worked well as a means to draft the cost/benefit/risk details.

2.4 Conceptual Design

The Conceptual Design process aids in gathering detailed requirements, designing a proposed solution, and then reviewing that design. Both functional and usability requirements were gathered from end-users and documented in the form of business scenarios and task analysis documents. The solution design was documented as an object model and a user interface prototype - either paper-based (Lo-fi) or on-line (Hi-fi). Then, the prototypes were used as the primary vehicle by which to test the design. A high-degree of user input is encouraged during Conceptual Design in order to accurately capture the problem domain and design a flexible solution.

Both teams also attended the Andersen Consulting (AC) training sessions on Process and Domain Components and worked jointly with AC-Northbrook developers to partition the object models into various domain and process components.

Lessons Learned:

- It is challenging to find end-users who are able to communicate both how they would like to improve a business process and how a process currently works; most users can describe the current business process well, but provide less reliable suggestions for major redesigns. Interviewing users at different levels of the organization (when collecting the scenarios) can help solve this dilemma.

- Using Lo-fi (paper-based) prototypes was a fast, inexpensive way to validate designs early. Using this technique forced the teams into an "iterative" design mode. It allowed developers to quickly test (and re-work) a number of designs before building the actual application.

- Although the Lo-fi sessions were intended to test the usability of the application, most users wanted to talk about the functionality of the application. In fact, the first few Lo-fi sessions were often used to detail the correct application functionality. In order to

have a Lo-fi session that is focused on usability topics, you must first "nail down" the functionality. Otherwise, you are testing the usability of an inappropriate application.

- The component-based design methodology worked reasonably well in helping the teams to break down their object models into individual domain and process components that made sense. However, a key concern about the Conceptual Design phase is the fact that it does not take into account any implementation issues. Although this "de-coupling" of the design from the implementation may result in a more flexible design, the danger is that, technically, the design may not be able to be implemented. This is especially true when the resulting design has multiple domain and process components, where the immaturity of the tools precludes the implementation of some of our designs.

- The size of the object model should be taken into account when trying to split it into multiple components. The WCMS object model was too small to realistically break down into separate domain components. After the design was complete, some components contained only one object. Although the resulting design made sense functionally, we felt that the extra effort required to build the extra domain and process components would not be beneficial until the application object model grew to a larger size. We anticipate that this will happen with the design and implementation of our Resource Availability business practice.

- A concise diagrammatic notation is required to help the developers keep track of how the components connect to form a solution. For this, a modified Ada structure graph from the Booch notation was used.

- It would be practical to include more diagrammatic notation in the methodology that specifically supports component-based design. For example, a notation that concisely summarizes the activities within a business process would be especially helpful. This is an area for further study using information from the Workflow Management Coalition interface specifications and using the process specific extension to the Unified Modeling Language (UML).

- One must design interfaces to maintain a component's internal integrity. When designing a component there was a natural tendency to expose the public interfaces to the component's internal classes. This posed a problem when exposing operations like constructors, destructors, or other operations that could leave the object model in a state that was inconsistent (i.e. dangling pointers, garbage objects, etc.). Exposing these operations at the component level meant that a client component (not under the control of the development team) could exercise a sequence of operations that could leave the component's internals in a state that could cause the component to fail. Since a client component is outside the control of the team developing the server component, the interface to a server component needs to be more robust so that a client can not corrupt the internal state of a server component.

2.5 Discovery Model

The Discovery Model process helped the development teams to pathfind solutions in areas of high complexity or uncertainty. These pathfinding activities helped to manage risk areas and confirm functional designs through the use of iterative development techniques. Due to the immaturity of our development tools/technology, the CBMS and WCMS teams jointly worked on many technical discovery models. Both teams re-used many of these technical models as the foundation on which they built their technical architectures.

Lessons Learned:

- The Discovery Model phase was an excellent way of forcing the team to identify risk areas and test alternative solutions to manage the risks.

- It was difficult to determine how much iteration/discovery is sufficient. We determined that it is essential to establish termination criteria for each pathfinding activity.

- Lessons learned and deliverables from the Discovery Model can be reused in the Working Model. It is a big time-saver when code from the Discovery Model can be reused in the Working Model. However, it should be made clear to the developers that this is not the objective of the exercise. The objective is to find a quick answer to a difficult question.

2.6 Working Model

The Working Modeling process supports the implementation and testing of the design in compliance with the requirements gathered during Conceptual Design. Working Modeling consists of two phases: Solution Modeling and Solution Integration. In Solution Modeling, the components were individually constructed and unit tested. These completed components were then integrated and system tested during Solution Integration.

Lessons Learned:

- For the most part, the iterative approach to development worked well. However we struggled with finding the appropriate level of granularity for each six month iteration. It was helpful for us to establish incremental builds within the six month periods, each of which reflected increasing levels of application functionality. The smaller the builds, the easier they were to control.

- As stated earlier, there were problems trying to implement the multiple component designs that came out of the Conceptual Design process. Our tool suite lacked two important features that would have made it possible to implement more components: 1) persistent references across components, and 2) a transaction mechanism across components. We were trying to implement component-based designs using COTS tools that had little technical support for such designs. The CBMS group was able to implement a design that included process components. They were able to successfully add process components that required only minimal changes to the existing domain components. However, the process components that were created were not "pure" process components. They contained some domain entities that were related specifically to the business process.

- In order to manage the development effort, both the CBMS and WCMS teams instantiated the methodology and documented detailed development steps. The development processes for each team were fundamentally the same. However, it was necessary for the teams to design separate processes for a few reasons:

    1. A few development tools/languages were not used by both teams (e.g. 3D visualization tool, HTML, JavaScript)

    2. The CBMS developers relied more on the generated code from the object modeling tool than did the WCMS developers. The extra "hand-coding" required

for WCMS led to different development steps. (The reasons why each team took a different approach are discussed in section 3.6 Auto Code Generation.)

3. The source code configuration management procedures were slightly different for each team.

- The usefulness of Dynamic Integration of components became apparent as more components were built for the CBMS application. This is an area that should get further attention: as more components become available, different solutions could be built with those components, but the current difficulty is that each component must know about its containing solution.

- When a component X has a reference to a CORBA object A in another component Y, it is possible for object A to become unavailable even though component X is still active. When performing straight forward OO design for one executable, the designer assumes that the lifecycle of a referenced object is under his programmatic control during the time that the executable is running. This false assumption must be taken into consideration when building a distributed component solution.

3. Development Environment

3.1 Introduction

The goal of the Technology Reinvestment Program (TRP) was to develop commercially viable technology to support a new software development approach based on rapid, iterative, user-centered techniques. To evaluate this approach, developers used emerging tools and infrastructure to develop both manufacturing and military applications. Development tools included those provided by Andersen Consulting, CoGenTex and Expersoft. COTS tools which were not critical to the evolutionary development process were selected by the CBMS and WCMS application development teams.

Lessons Learned:

We learned some important general lessons :

- Using state-of-the-art tools was a challenge. Most tools developed had excellent concepts, but were immature as an end-user product, making full use for development difficult. However, the automatic generation of documents was beneficial to design review activities.

- System administration of the Raytheon TRP network was also a challenge. Porting to the CORBA 2.0 ORB required multiple platforms for application development introducing issues regarding the support of heterogeneous environments. We found compatibility issues across software releases and hardware platforms. Much time and effort was devoted to vendor contact to acquire technical support and clarify licensing issues. The stand-alone network with SecurID cards supported the collaboration and communication among consortium members.

3.2 Support Tools

3.2.1 HTML Editors

Initially, the WCMS team used HTML pages to capture/organize technical documentation. An HTML editor was used to quickly create these "static" HTML pages. However, during Build 4, WCMS migrated its user interface to a web-based implementation, thereby using HTML as a core development language. The team searched for an HTML editor which could support the development of an HTML user interface (via the use of dynamic HTML). Dynamic HTML is composition of HTML pages dynamically at the moment of invocation. No HTML editor was found that could support our needs (see Lessons Learned below), so the team wrote their own "helper" classes which generated dynamic HTML.

Lessons Learned:

- WCMS felt that HTML documents were the easiest way to communicate technical information among the various team members. It worked out so well that the team also started to collect functional information on HTML pages as well.

- COTS web-development tools seem to fall into one of the following categories:

    1. simple tools that let a developer paint a window and generate static HTML pages

    2. high-end tools which aid developers in generating dynamic HTML pages (including database access)

    Tools which fall into category 1 are similar to a word processing tool. They are fine for creating documentation and other static pages (such as demos), however they lack the functionality (such as dynamic HTML generation) to develop applications. Category 2 tools aid in creating a robust user interface, with dynamic HTML. However, they are typically expensive because they come bundled with expensive middleware, such as ORBs and "pseudo ORBs." WCMS was looking for (and could not find) a tool that would fall in between the two categories - something that would help us develop a robust interface using our existing ORB infrastructure.

3.2.2 Groupware

A tool classified as "groupware" was used on the Hewlett-Packard (HP) workstations to serve a couple of different purposes: 1) to record and track issues, and 2) to collect time tracking data for metrics analysis.

Lessons Learned:

- For the most part, the groupware tool performed well. It was a mature tool with a large user base, so we did not anticipate many problems. The problems that did occur were due to the fact that the product was designed for a PC or Mac environment. Although we used the version that ran on UNIX, we could tell that the product was architected for a PC platform. The initial installation consumed a lot of disk space, as did the

individual user accounts. However, through the use of UNIX links (i.e. file sharing), unnecessary files were deleted and disk space usage was reduced.

- Ideally, both the system administrator and groupware administrator should be knowledgeable about both the groupware and the target operating system/hardware platform. The initial installation and set-up of the tool was cumbersome because the Raytheon groupware administrator was unfamiliar with UNIX, and the UNIX administrator was unfamiliar with the groupware.

3.2.3 Utilities

Many home-grown utilities, macros, and scripts were written to help developers with tasks such as makefile generation, string manipulation, performance measurement, debugging and testing.

Lessons Learned:

- These types of utilities take relatively little time to write as compared with the alternative, which is to allow each developer to code their own functionality by hand. They are a big time-saver, because they facilitate the implementation of repeatable, mundane functionality (such as string manipulation, database calls, etc.).

- In addition to being a time-saver, they also help to enforce coding standards. This, in turn, makes the code easier to read and understand.

3.3 Analysis Tools

3.3.1 Domain Requirements Analysis Tool

A domain requirements analysis tool was used to identify and analyze the problem domain entities like classes, attributes, processes, etc. This tool allowed the user to highlight the requirements text in different colors where the color represented the type of entities. The tool allowed the analyst to view the entities in different formats such as hierarchy charts, tables, cards, relationship diagrams, etc. This allowed the analyst different ways of checking/challenging the requirements text, while coming up with an initial set of entities to drive the object and behavioral modeling.

Lessons Learned:

- Conceptually the domain requirements analysis tool appeared to be a very good toolset for organizing domain knowledge. This tool suite should facilitate communication between developers and domain experts. However, the tool supports individual analysts and does not directly support group analysis techniques such as JAD (Joint Application Development).

- It needs to be more reliable (easy to cause system crash), stable (product is continuously being improved), and compatible with development environment hardware and operating system (currently only a PC/Windows version).

- It needs improved printing capabilities to facilitate documentation.

- Using the requirements analysis tool through a windows emulator (Softwindows) was very slow and unstable. Use on a PC proved to be a better solution.

3.3.2 Object Modeling Tool

The object modeling tool used by the application developers provided automatic C++ code generation from the object models. It became an integral part of the TRP tool suite as it was used as the foundation for one of Andersen Consulting's component-development tools (Component Specification Tool).

Lessons Learned:

- Documentation: A robust Object Modeling tool should have automated documentation features. Many Object Modeling tools provide the developer with the ability to partition their Object Model onto multiple pages. Most mature tools today can easily produce an organized output of data for inclusion into formal documentation.

- Code Generation: Many Object Modeling tools provide some degree of code generation. In addition to finding the tool that produces the coding language of choice, the tool should produce file headers and attribute/relationship accessor functions.

- Tailorability: Code Generation features provided by the Object Modeling tool should be tailorable by the developer. This will allow project standards to be enforced at an early stage of code development.

- Most Object Modeling tools support a single Object Modeling Methodology. Several on the market today support multiple methodologies. Most important is the selection of an Object Modeling tool that supports a mainstream modeling methodology. This will help ensure that the Object Model produced will be understood by others, since there is a better chance of previous exposure to the methodology and modeling notation.

- In addition to providing documentation support, it is important that an Object Modeling tool provide the developer with the means to produce customized reports from the Object Model.

- Validation: It is critical that the Object Modeling tool support interaction diagrams with automated checking against the object model.

3.3.3 Component Specification Tool (CST)

The Component Specification Tool (CST) is a component design tool developed by Andersen Consulting. CST was intended to assist the developer in the design of components for a solution.

Lessons Learned:

- The use of a common format (CDIF in this case) for describing a model is only part of the solution design models through phases and through various tools; translators are also required to map concepts across tool clients. AC's approach to this shows promise.

- CST generated Component Specification Language (CSL) and skeleton C++ interface code stubs.

- CST did not allow entry of descriptive text on model elements: interfaces, operations, etc.

- The process component portion of CST was used for diagrams only because it wasn't ready for full development use.

- The generated interface specifications required substantial manual modification to handle ordering of attributes and support for singleton interface. The tool does not directly support a singleton interface.

3.3.4 Component Integration Tool (CIT)

The Component Integration Tool (CIT) is an integration tool developed by Andersen Consulting. Integrated with the Asset Catalog Tool (ACT), the CIT tool was designed to allow the assembly of a solution based on pre-existing components. The CIT was evaluated through hi-fi usability testing by developers. Use of the CIT tool was limited due to the fact that the tool was based on the assumption that a populated repository of components existed. The components designed by the application teams were still under development.

Lessons Learned:

- There needs to be a mechanism for finding components. The purpose of component development is to allow solutions to be built from already developed components. To support this feature a means of dynamically finding and connecting to the other components is required. AC is working on an *integrator* that works with the COSS lifecycle and trader services to supply this functionality. With AC's approach the Component Integration Tool (CIT) is where one component's required interfaces are matched up to another component's provided interfaces in order to build a specific solution. CIT would then generate a solution file that would be used by the integrator to assemble the solution without requiring the components to be modified. Pending the outcome of the pathfinding of the CORBAplus ORB product and availability of trader services by the ORB, we plan to exercise the CIT and integrator tools. Therefore our current component's required interfaces are matched to another component's provided interfaces through code (static integration). This means that our current components need to know which solution they are participating in (at this time).

3.3.5 ModelExplainer

ModelExplainer is a natural language document generator developed by CoGenTex. ModelExplainer works explicitly with the object modeling tool used by the application developers. ModelExplainer parses the object model and generates the data dictionary for all classes. It documents all attributes and methods, and provides a natural language definition of every relationship participated in by each class. The developer must then add descriptive text (using HTML) for all attributes and methods. For more information, see the CoGenTex chapter.

Lessons Learned:

- The natural language text produced by ModelExplainer was a useful addition to our development tool suite. The text output helped application team members to gain a common understanding of the object model.

- The tool facilitated design reviews and provided data dictionary templates for documentation of object models.

- For some of the developers, a filtering process was established to convert the HTML-based files into an Interleaf format for importing to design documents.

- The "data dictionary" features of ModEx were used to augment our Object Modeling Tool which did not support documentation.

- ModelExplainer offered a unique feature of putting into words how a class relates to other classes; this was useful in helping people unfamiliar with object-oriented notation to check the model. Using this feature suggested some useful and interesting requirements on how to choose relationship and class names.

3.3.6 CogentHelp

CogentHelp is a tool developed by CoGenTex which supports the generation of on-line help (natural language-based) for software applications. CogentHelp was compatible with the two-dimensional graphical user interface (GUI) development tool used by the application developers. CoGentHelp dynamically produces help-screens for the user during an application session. For more information, see the CoGenTex chapter.

Lessons Learned:

- The text-based help windows were useful, but could be improved by including "screen shots" and then allowing end-users to use the screen shots to navigate through the help information.

- The authoring interface was cumbersome for developers to use. A new interface (which hides the details of the integration with the two-dimensional GUI tool) would be very helpful.

3.3.7 Two-Dimensional Graphical User Interface Tool

The two-dimensional GUI tool used by the application developers had features to support the design of user interface screens and it also generated C++ source code. The GUI tool was used to create Hi-Fidelity GUI prototypes of the applications to retrieve feedback from end-users during usability testing. Following usability testing, source code was added to create a fully functional user interface.

Lessons Learned:

- Robust Graphical User Interface (GUI) development tools provide a "What you see is what you get" (WYSIWIG) window drawing tool.

- The ability to create sub-classes from the widget classes provided by the GUI builder is critical in order to add additional application specific functionality to the widgets.

- A GUI builder should provide a generic mechanism to link all types of widgets to non-GUI structures or class instances, resulting in links of the data display to its storage.

- A GUI builder should support automatic code regeneration features to allow for easy modification of existing windows.

- In addition to basic widgets, GUI builds should provide table, tree and graph widget generation.

- The scripting language provided by the tool did not have all of the features required for Hi-Fi prototyping. Much C++ code had to be written to supplement the scripting language.

- Classical GUI builder tools require the complete interface to be defined and generated as part of the executable at code compilation time, resulting in the necessity to recode and recompile to support new interfaces. We recommend pursuing another approach which would ease the effort to change user interfaces: allow the GUI definition to be imported into a component at runtime, much the way Web browsers dynamically present new interfaces in the form of Web pages.

3.4 Implementation Tools

3.4.1 Configuration Management Tool

The configuration management tool used by the application developers was robust and provided features for version management, workspace management, build management and process management. Although the configuration management tool was primarily used to track versions of source code during incremental build releases, the tool was also used to archive supporting design documents.

Lessons Learned:

- A configuration management (CM) process and directory structure should be established before development begins in order to optimize coordination of implementation and integration activities.

- The CM process should involve establishing the directory structure, defining branching and merging rules, specifying development rules, and defining roles and responsibilities.

- The Buildmeister (integration lead) should be responsible for creating all master branch views and baseline views, labeling all builds and development baselines, coordinating all merges and final build releases, check-out/check-in access to the main-line and integration branches.

- The Developer should be responsible for creating a private development view from the master branch view, performing all development strictly on a development branch, notifying the Buildmeister when the source code is ready for merging, and check-out/check-in access on only the development branch.

- Deep knowledge and understanding of the CM tool was isolated to the Buildmeister providing developers the freedom to focus on implementation issues.

- The task lead needs to factor into the schedule some lag time between builds for set-up prior to start of a new build and for clean-up of views/branches at the end of each build.

3.4.2 Memory Leak and Error Detection Tool

The application developers employed a memory leak and memory error detection tool during integration activities. This tool located memory problems that could lead to system failures if left undetected.

Lessons Learned:

- Memory Detection tools are instrumental in resolving memory handling problems as well as complex logic errors, that normal debuggers do not assist the developer in resolving.

- Need for such tools lessen as developers become more expert in the programming language.

- These tools are not needed for the Java programming language because these capabilities are inherent in the language itself.

3.4.3 Graphical User Interface Test Tool

The application developers used a tool to test the graphical user interface of both applications. It provides capture/replay capabilities for producing repeatable user interface tests.

Lessons Learned:

- A testing infrastructure (including formal test plans) must be established in order to make full use of the tool.

- A means of organizing the test scripts is also highly recommended. Some vendors may provide this feature as a separate product.

3.4.4 C++ Compiler

The C++ compiler used on TRP came, by default, with our operating system and uses the C-Front utility.

Lessons Learned:

- Critical to any application development is a reliable and well tested compiler.

- Many Third Party Tools automatically generate code, which sometimes requires specific versions of a compiler or operating system.

- For TRP support of template and exception handling was critical. Since many compilers are directly tied to a specific version of an operating system, upgrading a compiler is not always a simple task.

3.4.5 Object-Oriented Database Management System

The application developers used an Object-Oriented Database Management System (OODBMS) which had a C++ API. The OODBMS provided the following database capabilities: transaction consistency recovery, distributed on-line backup, archive logging, referential integrity through relationships, and two-phase commits. It also had a unique feature called Virtual Memory Mapping Architecture (VMMA). VMMA allows developers to reference information as local data. The OODBMS also supported distributed and multi-user environments.

Lessons Learned:

- Robust OODBMS should support referential integrity and delete propagation.

- Most application developers take advantage of Third Party Standard Template Libraries (STLs) in order to provide standard utilities, such as basic string and math functions. Many OODBMS vendors support an integration with STLs. Some integrations may require writing classes to make the STL objects persistent.

- OODBMS support for C++ exception handling may be at several levels. In most cases, support for C++ exceptions is directly related to the operating system and compiler being used to develop the application. At one level, proprietary exceptions and C++ exceptions could be intermingled, at another level, the two types of exceptions could be used independently, and at the lowest level, only the proprietary exceptions could be used. Support for these exceptions could involve defining separate inheritance structures for proper destruction of transient objects.

- OODBMS are relatively new to the market place, so training and technical support are almost always required. The first line of technical support should be able to answer simple installation and usage questions. There should be documentation with examples readily available. Reasonable default parameters when the tool is installed also help start-up.

- The OODBMS C++ Application Programmers Interface (API) should be robust. Many tools require a separate inheritance structure for persistence.

3.4.6 Object Request Broker (ORB)

TRP utilized a CORBA-compliant object request broker (ORB) to implement the application infrastructure. Application interfaces are modeled using Interface Definition Language (IDL). The IDL compilers generate client and server C++ stubs from the IDL.

Lessons Learned:

- An ORB provides the infrastructure for a component-based application so that its separate interworking components can be distributed over several computers. The IDL provides an adequate specification of the services offered by a component. Although we didn't use this feature, IDL also allows implementations of components to be in different languages.

- It is more difficult to investigate and solve integration problems with a distributed application. This makes standalone testing of a component very important.

- An ORB product should comply fully with the CORBA 2.0 specification; we used a partially compliant product which limited us to static invocations of defined interfaces.

- The additional object services offered by an ORB vendor should be used judiciously. If the service does not comply with an existing Object Management Group (OMG) object service specification, it should be used only as a prototyping capability until such compliant services become available; we used this approach with the event services offered by our ORB product. If the service has no corresponding approved or pending OMG service specification, it can be used as part of the delivered application.

- An ORB product needs to demonstrate its interoperability with different ORB products using the Internet Interoperability Protocol (IIOP). Although IIOP is part of the CORBA 2.0 specification, a demonstration is necessary to validate the ORB's interoperability with ORBs which may likely be used in adjacent systems.

- The application developers migrated the CBMS application from the vendor's proprietary hybrid-CORBA product to a CORBA 2.0 compliant ORB. This proved to be a time-consuming tasks (four man-months in duration) with much effort devoted to splitting the application components across platforms and to resolving issues regarding other development tools and the operating system to support C++ exceptions.

- The CBMS development team converted the CBMS application to using a CORBA 2.0 compliant ORB from previously using the vendor's proprietary hybrid-CORBA product. This proved to be a time-consuming tasks of four man-months, with much effort devoted to rippling effects on other development tools and the operating system in order to support C++ exceptions.

3.5 Network Environment

3.5.1 Hardware

The following diagram depicts the Raytheon TRP network. The specific software development platforms and corresponding operating systems have evolved throughout the TRP program. The final configuration is comprised of 4 Hewlett-Packard (HP) 715/100 machines running HPUX 10.01, 4 HP 715/100 machines running HPUX 10.10, 8 NCD X-Display terminals, 1 Sun Solaris 2.5 machine and an HP 9000 server running HPUX 9.04.

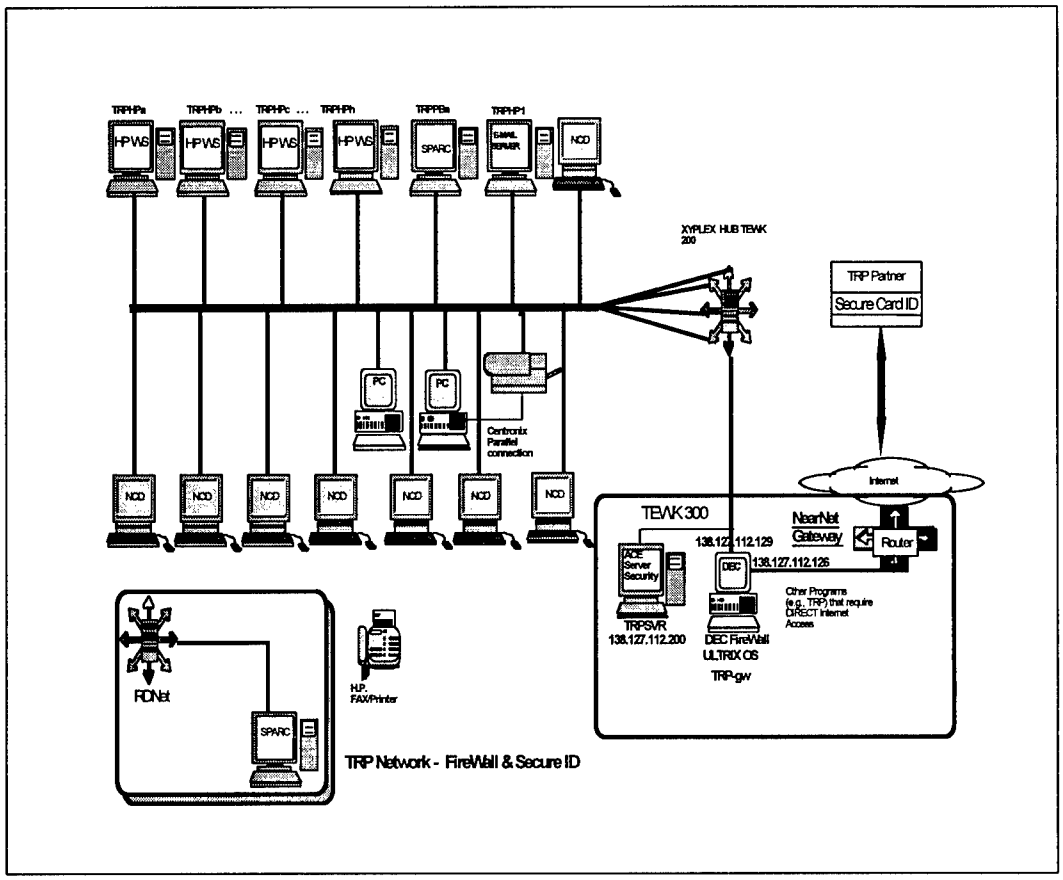

Figure 1 - TRP Network

3.5.2 Secure Identification

The Raytheon TRP network was established as a self contained network, separate from the main Raytheon network. The Raytheon TRP network was set up with its own firewall and direct access to the Internet to provide a trusted working environment and protect against computer vandalism or viruses. Anyone accessing the TRP network from the Internet was required to carry a secure identification card called SecurID (manufactured by Security Dynamics). The SecurID card generated random numbers in synchronization with the master Ace software server supporting the network's firewall. The log-in procedure required the user to enter the generated passcode within a 60-second time window. Without the proper code, the user was refused access. The availability of

SecurID cards gave fellow-consortium members access to the Raytheon TRP network allowing them to install and test beta-releases of their tools remotely.

3.6 Auto Code Generation

The code generation process employed by the CBMS team is depicted in the following diagram. Automatic code generation was provided by: the object modeling tool, 2-D GUI tools, 3-D GUI tool, and the ORB. The code generated by the 3rd-party tools required the engineer to add hand crafted code where logic/implementation specific code was required. The object modeling tool generated C++ code with embedded database logic. Although the generated code was not ideal, for the scope of TRP, it allowed the CBMS team to concentrate on functional issues instead of designing and implementing a database schema by hand. All the auto code generation enabled the developers to readily implement functional features of the application.

WCMS relied on the code generated from the ORB and the GUI. However, the code generated from the object modeling tool was too inefficient to be used in a "live" application. This decision resulted in WCMS developers writing most of the C++ code for the business classes, thus, slowing down the development process.

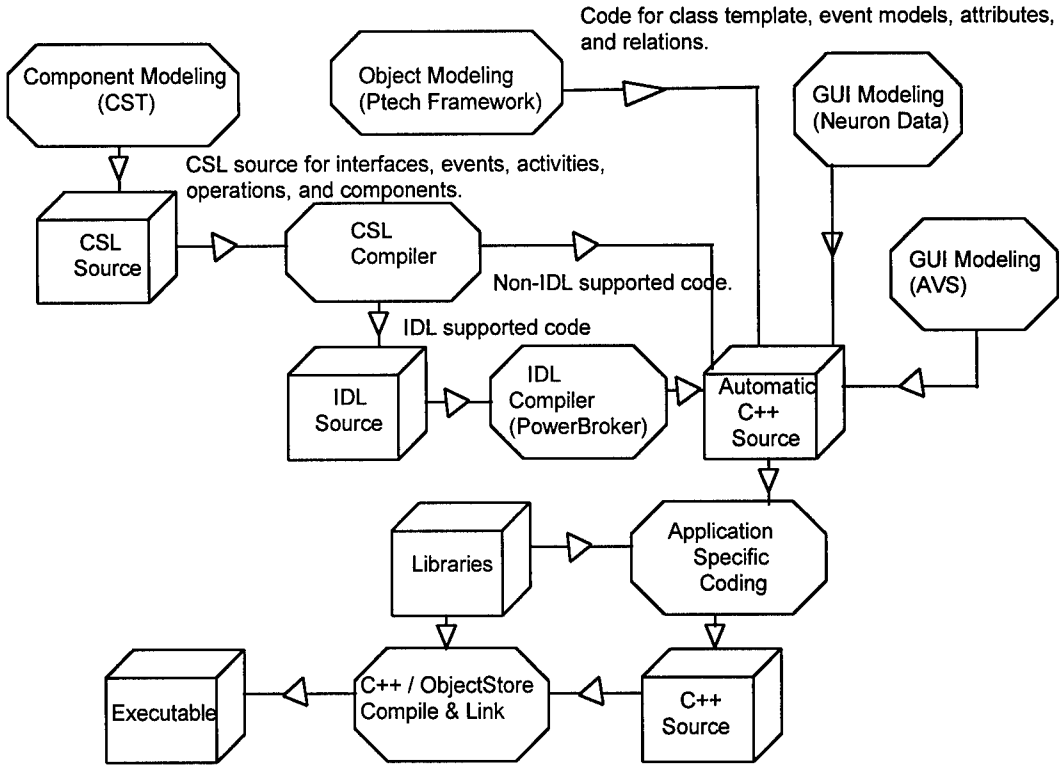

Figure 2 - Code Generation Process

4. Applications

4.1 Capabilities-based Battle Management System (CBMS)

The released capabilities of CBMS are designed to assist an Army Operations Officer in building and implementing defense plans during force operations and managing engagement operation capabilities for a variety of existing and new combat systems, based on a core set of battle management and weapon control algorithms. The system consists of an integrated 3D visualization capability of defense planning and battle management functions with decision support tools for the commander along with reactive re-planning capabilities. CBMS was constructed using a component-based software engineering methodology and is architected to allow functional extensions in any follow-on work.

The CBMS System is designed to assist an Operations Officer in four different business practices : Defense Design, Defense Design Review, Parameter Initialization and Reactive Replanning. It is composed of several components which comprise the Operations Officer solution. (A solution is a set of components which become one user's application.) A Communications Officer solution was created to take part in the Defense Design Review and Parameter Initialization processes but only implements these tasks (and not an entire solution for the Communications Officer).

The Operations Officer solution includes the following components : Operations Application (OP), Operations Graphical User Interface (OPGUI), Operations 3D GUI (OP3DGUI), Defense Design Review (DDR), Parameter Initialization (PI), Process Architecture Control (PAC) and Reactive Replanning (RR). The solutions with their required components are shown in Figure 3.

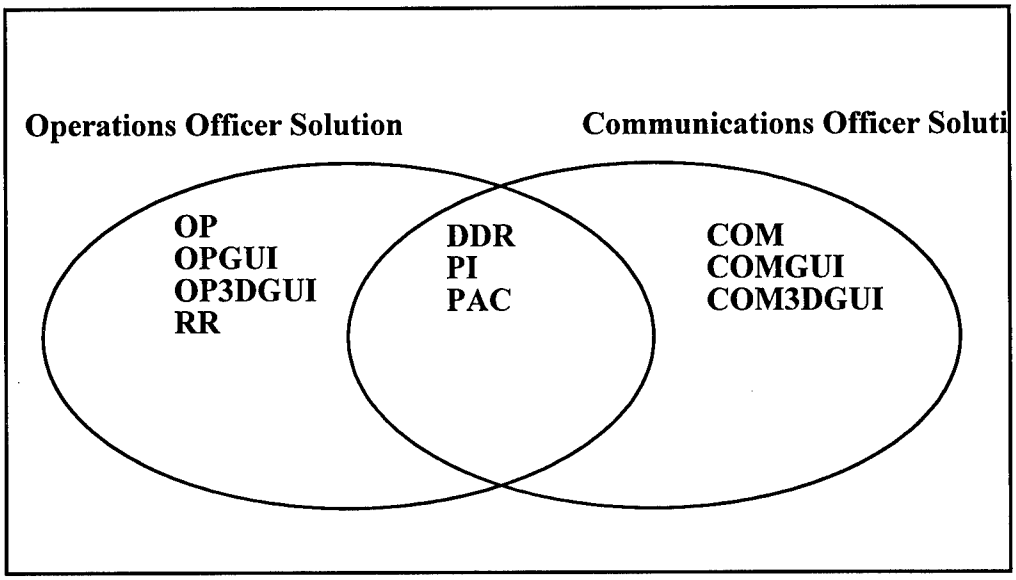

Figure 3 - CBMS Components

The Operations Officer uses the OPGUI Component to enter the inputs for CBMS. The Operations Officer can create defense designs from scratch or copy another design to

update. The defense design inputs consist primarily of the Operations Order and the Mission, Enemy, Terrain, Troops and Time-available (METT-T) report. The METT-T consists of detailed information about the commanded units, threats, defended asset and terrain. The officer can add, remove or change the defended assets, threats, commanded units as well as the hierarchy of the commanded units. The OPGUI also displays the textual or graphical information that is available from the OP Component which has been previously entered and stored in a database.

The 3-dimensional map (OP3DGUI) component allows the Operations Officer to move a commanded unit, threat or defended asset using the mouse (drag-and-drop). The OP3DGUI component was developed using a 3-dimensional (3D) visualization tool. This component displays the 3D map of the terrain along with the threats, assets, troops (Figure 4). The OP3DGUI also displays weapon system capabilities which are calculated from a set of compensation curves.

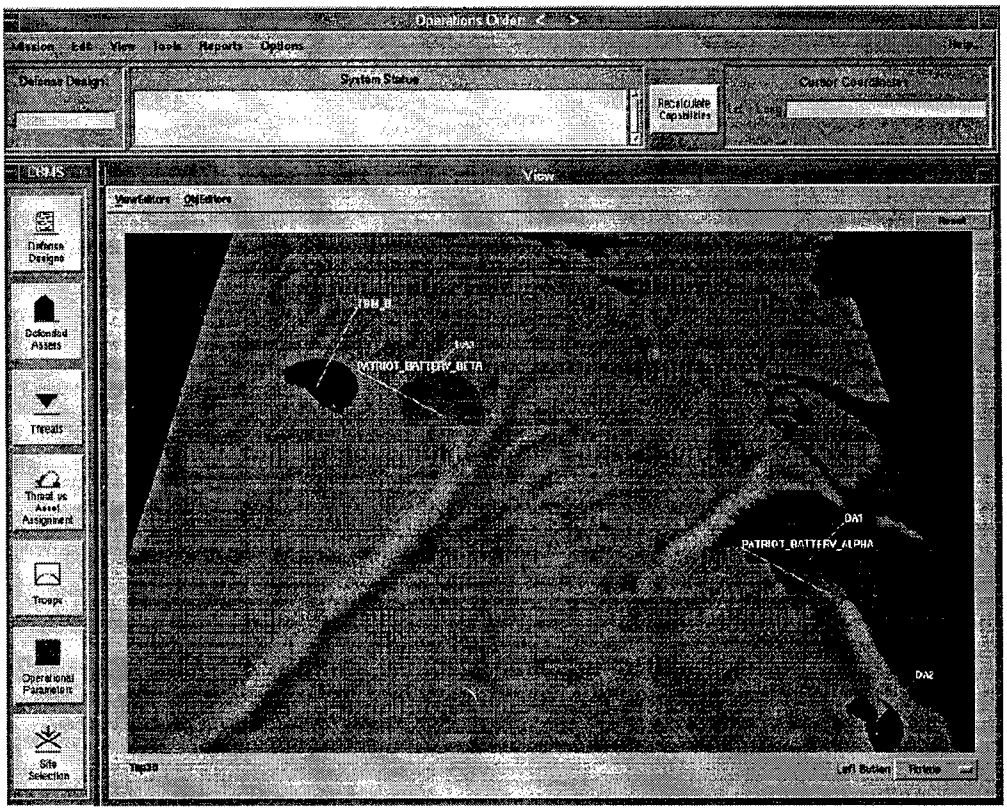

Figure 4 - CBMS Main Screen

The Defense Design Review (DDR) process is initiated by the Operations Officer when a review of a defense design is desired. The Operations Officer determines who shall receive the request for the defense design review, whether or not the recipient is required to respond, the time that the review needs to be completed, the priority of the review and the instructions to be sent with the request. The DDR component keeps track of the state of the review process. Another component, the Process Architecture Control (PAC) Component was developed to keep track of the processes for each solution (Operations

and Communications). The PAC Component keeps track of both the DDR Processes and the PI Processes.

The Parameter Initialization (PI) process is initiated by the Operations Officer when the final Defense Design is completed. The request for parameters is sent to a default distribution list which includes the Communications Officer. In the current system, the Communications Officer is required to generate the communication network for the defense plan. The Operations Officer is responsible for generating the firing doctrine, site specific information and the mission statements. The firing doctrine and the site specific information have already been defined during the defense planning but may be changed during this process. The mission statements must be generated during this process. The Operations Officer is responsible for reviewing all of the parameters before distributing them to the Staff officers and the subordinate's commanded units.

The Reactive Replanning (RR) functionality addresses the monitoring, assessing, and replanning of a previous defense design during the course of a battle (in our case a simulation). External messages about the state of the current battle situation are monitored in relation to the selected defense design (i.e. specific commanded units, threats, and defended assets).

The state of the current battle situation is used by the Operations Officer who monitors and assesses the current situation. The Operations Officer assesses the current battle situation in light of his intended mission. He is also responsible for determining when to return to the mode for updating the defense design. When switching back to the defense design process, the system uses the currently known battle situation as a baseline.

Lessons Learned:

- We believe that the use of object-oriented concepts with component-based software engineering (CBSE) were well-suited to the design and implementation of the CBMS application.
- Primary benefits of the CBSE approach are in the following areas: impact of change, extensibility, and reuse.
- Enhancements or modifications to the application were isolated to specific components minimizing side-effects to other areas of the application.
- We were able to extend the CBMS solution to incorporate new business practices with minimal impact to existing components.
- We plan to store a set of artifacts such as the scenarios, design documents, object models, component models, and source code into the Software Engineering Laboratory's repository for reuse by future projects tasked to design component-based solutions for the battle management domain.

4.2 Work Center Management System (WCMS)

The Work Center Management System (WCMS) provides shop floor personnel with a summarized view of factors which impact manufacturing schedules. The main users of the system are the factory Foremen and Production Control personnel. This system is designed to help these users track the status of work-in-process, manage work center

priorities, assign jobs to operators, and assess the impact of new work coming into their area.

Current factory floor personnel are inundated with various reports and on-line screens (mainframe based) which provide information on high-level factory schedules, etc. However, the data is often not organized in a manner that is appropriate for work center personnel. The primary functional goal of WCMS is to be an application for the work center. Data that is important to the work center will be collected, updated and presented in a manner that will help the factory floor personnel make informed decisions.

WCMS will be deployed as a pilot application in the Cables work center of Raytheon Electronic System's Andover plant. Originally, WCMS was comprised of an X-Windows user interface. However, in the last build, the decision was made to migrate the user interface to a web-browser. This positions WCMS to more easily integrate with some other web-based factory applications that Raytheon plans to design. WCMS is the first attempt at installing in the Andover plant a web-based, distributed object-oriented system that was developed using a component-based methodology. Therefore, this pilot will serve to test not only the appropriateness of the WCMS functionality, but the strength and flexibility of its technical architecture and development process.

WCMS was designed to support shop floor personnel in two main business practices: Prioritize Workload and Workload Allocation/Operator Assignment. However, in order to fulfill the requirements of these practices, we would need to collect input data from supporting activities. Listed below are the business practices/functionality WCMS is designed to support.

- Workload Allocation/Operator Assignment - Initially, provide a simple view of a work area's workload and available resources. This allows foremen and production control personnel to make better decisions on prioritizing the workload and assigning the workload to operators. Allow for the future integration with a Finite Capacity Scheduling system to provide more robust features, such as reactive reprioritization and automated assignments to operators.

- Material Availability - Provide a summarized, consolidated view of the material status for both work-in-process and new work.

- Resource Availability - Provide a summarized view of available labor and machines. Employee data would include skill set information, thus allowing foremen to compare available skills vs. the skills required to work on a particular job.

- Tooling Availability - Provide tool status for the workload (i.e. tool requirements vs. their availability). Highlight any current or future jobs whose tools are not readily available.

- Documentation Availability - Highlight any jobs coming into the work area which do not have the appropriate documentation.

- Prioritize Workload - Coordinate the input received from the previously mentioned business practices and provide production control personnel and foremen with an easy method for reprioritizing work. This may include interfacing with a COTS Finite Capacity Scheduling tool.

The figure below depicts how these areas of functionality interact.

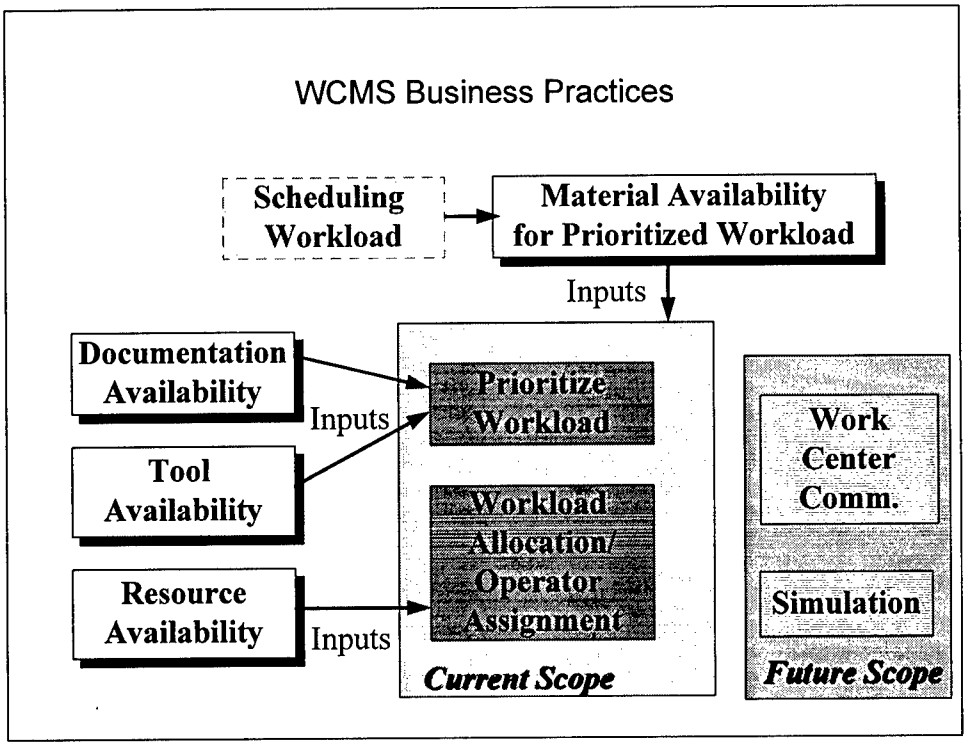

Figure 5 - WCMS Business Practices

The first two business practices - Workload Allocation and Material Availability - will be completed during the TRP project timeframe. Implementation for the remaining business practices is planned for post-TRP.

In order to provide this "decision support" functionality to the factory personnel, WCMS makes heavy use of the vast amounts of legacy data available on current mainframe systems. This data includes MRPII schedules, Time and Attendance, and Shop Floor Data Collection information. Both batch and on-line (real-time) interfaces were developed to access the legacy data, which is only available on an IBM mainframe platform.

Each legacy interface is a component in WCMS and communicates with the WCMS application server component. WCMS' fourth component is the user interface component. As stated earlier, the current WCMS object model was deemed to be too small to split up into multiple domain components. However, as WCMS grows there will be opportunities to add additional domain components (especially in the area of Resource Availability).

Lessons Learned

- Overall, the WCMS application seems to be well-suited to the object technology paradigm. Based on some of the technical problems we encountered, however, we feel fortunate that WCMS was not a "transaction heavy" system. The tools available to support robust transaction processing are just making their way into the marketplace.

- Designing the system, from a functional perspective, was difficult for the WCMS team, because we did not have as much access to our end-users as we would have liked. We were geographically located in different facilities. This was compounded by the

fact that since it is our user's function to be on the factory floor, it was difficult to get them out of the factory for meetings, design reviews, etc. We would have liked to have been in the same facility so that we could interact with the users more and try out ideas in a more spontaneous manner.

- This project was the first exposure to Object Technology for the Raytheon WCMS team members. In fact, all of them were coming from a mainframe background. The transition to the technology used on this project was easier for some than for others. A common problem was the difficulty of learning the C++ language. This was a key factor which contributed to a slow transition for some team members. However, those team members who were excited about the component-based technology, were able to get beyond the difficulties of C++ and transition more easily.

- For many team members, this was also their first time working with a scenario-based development methodology. Some could not see the benefits at first, because we only developed scenarios at the end-user level. Once we determined that we could write scenarios at all levels of development (from high-level user scenarios to detailed scenarios which aided in coding business logic), team members realized their value.

- Setting up a development infrastructure and publishing a clear set of development steps/guidelines was the most important thing we did to ensure we could meet project schedules. Setting up this infrastructure was a painful process, however, because both the tools and the methodology were new to everyone.

5. Technical Architecture

5.1 Introduction

This section describes the elements of the technical architecture designed and used on TRP for both the WCMS and CBMS applications. Each application is composed of components which interact to provide the necessary application functionality. A component provides a coherent set of services through one or more well-defined interfaces defined in CORBA IDL (a subset of CSL). Although the interface is normally organized in an object-oriented manner, a component's implementation may or may not actually be object-oriented; for newly built components, the implementation will be object-oriented consisting of a number of classes and objects. Usually then, it is reasonable to think of a component as a cluster of classes working together to provide the component's services.

Undergirding these components is the technical architecture which provides the general capabilities required by each component so it can deliver its services. For the WCMS and CBMS applications, it includes the following capabilities which are outlined in the subsequent subsections:

- Common Component Infrastructure -- support for component initialization and run-time linking of components.

- Object distribution -- a layer which allows business objects to be accessed remotely while insulating the business code from the intricacies of the ORB.

- Event notification -- supports asynchronous notifications from server objects to clients (and other server objects).

- Transaction serialization -- addresses concurrency problems brought about by the combination of PowerBroker and ObjectStore.

- Exception handling -- integrates and supplements the exception support from PowerBroker and ObjectStore. WCMS implemented an approach to support exception handling while CBMS deferred this activity until the tools support exception handling natively.

- Access control -- controls user access to the application server.

- Legacy interface -- provides a batch and on-line interfaces from the mainframe to the application server for WCMS. (CBMS does not have a legacy requirement.)

It should be noted that, although the technical architectures for the WCMS and CBMS applications are very similar, there are important differences. These differences arise from two sources. One is the difference in the domains themselves which require a few different capabilities from the technical architecture; for example, WCMS requires capabilities for access control and legacy interfaces not needed by CBMS. The second source is the conscious decision springing from the spirit of TRP to understand variations in application character; thus, we emphasized different application qualities (robustness vs. functionality) to understand the solution trade-offs. The resulting technical architectures reflect this emphasis. Hence, the WCMS team explored a more robust technical architecture which would better support a production system. Conversely, the CBMS team expended its efforts expanding the application functionality on top of the available architecture.

At a high-level, both the CBMS and WCMS technical architectures follow a common structuring of components, as shown in the following figure. In this approach, one component (the GUI) communicates with another component (an application component) via CORBA IDL interfaces. That same GUI component also registers (or subscribes) for events from the application component. This approach de-couples the application component from those components that wish to monitor it, and is similar to the model-view-controller (MVC) approach promoted by Smalltalk. Another fundamental aspect of the architecture is how a component's persistent data is kept private. The only way a component can get access to the persistent data of another component is through the published IDL interfaces. There are a number of different approaches to implementing both the IDL interfaces and the events between the components which we explore in section 5.4. (Refer to Figure 6.)

The component-based methodology allowed easy migration from an X-windows implementation to a web-based implementation for the user interface component; the WCMS server component was barely affected by this change.

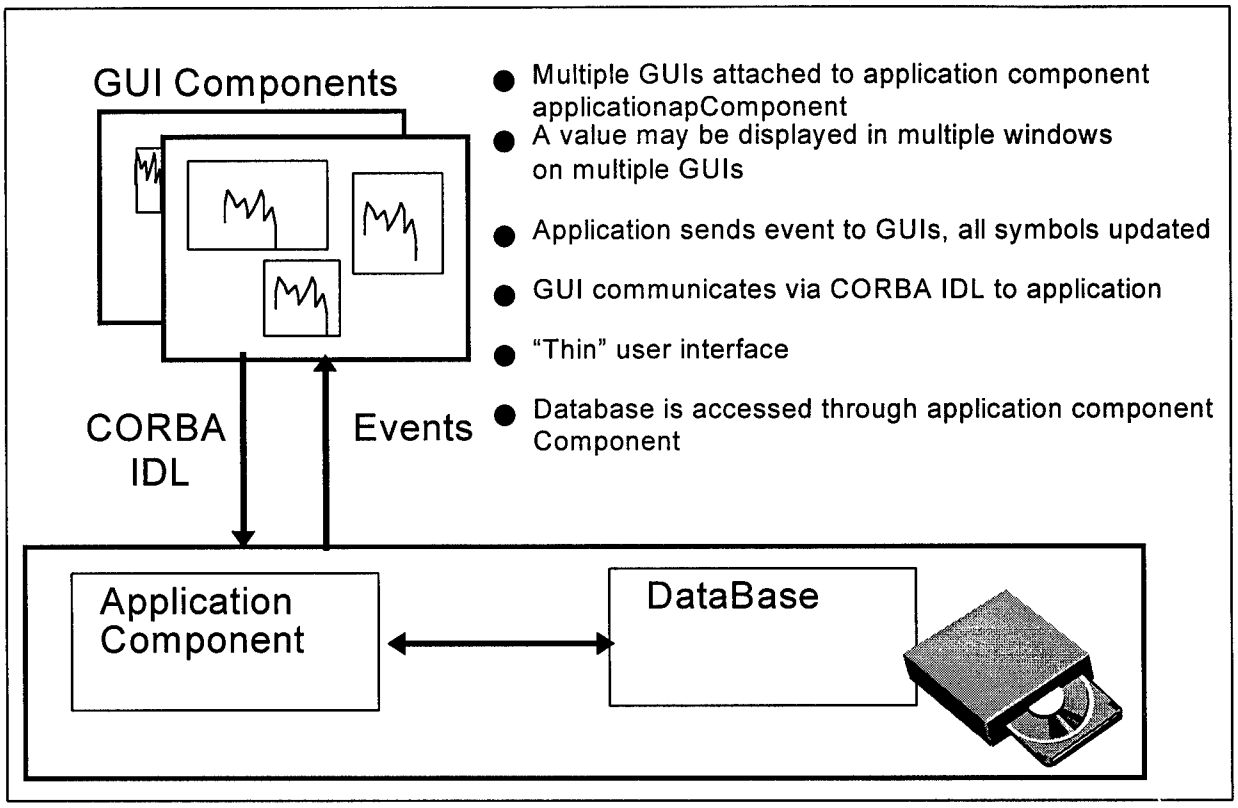

Figure 6 - High Level Architecture

5.2 Common Component Infrastructure

The essential concepts of this infrastructure are:

- Each component must go through an initialization process in order to be able to communicate with other components and to connect with resources such as the ORB and the database.
- All components must initialize the ORB run-time library. Most components must also start an event loop. WCMS CGI clients do not require an event loop because the web server has its own.
- Each server component needs to initialize the database run-time, connect to the correct database, and register itself with the ORB's naming service.
- Each client component needs to obtain references to server components that it depends upon.
- Each GUI component needs to integrate the handling of window events into the event loop.
- The Common Component Infrastructure consists of a set of architecture classes which provide most of the initialization code required by components.

5.3 Object Distribution

A critical architectural mechanism developed during TRP addresses the following problem: How do we write the business logic without having to worry about object distribution issues, and subsequently make the business objects accessible via the ORB?

One approach is to design classes which combine the business logic with hooks into the ORB. We consider this option highly undesirable, however, as it would unnecessarily complicate the development of the business logic and would violate a fundamental layering principle.

Another approach is to use an ORB which provides the necessary object adapters for object-oriented databases. These adapters greatly facilitate the separation of business logic and distribution logic.

Two architectural approaches we developed to provide object distribution for persistent objects are:

- Dynamic reference approach -- Under this approach, an IDL interface is defined for each business class whose objects need to be accessed remotely. The operations on the IDL interface map directly to the methods in the class, and a CORBA object instance is dynamically created and "tied" to each business object that needs to be accessed remotely. (Refer to Figure 7.)

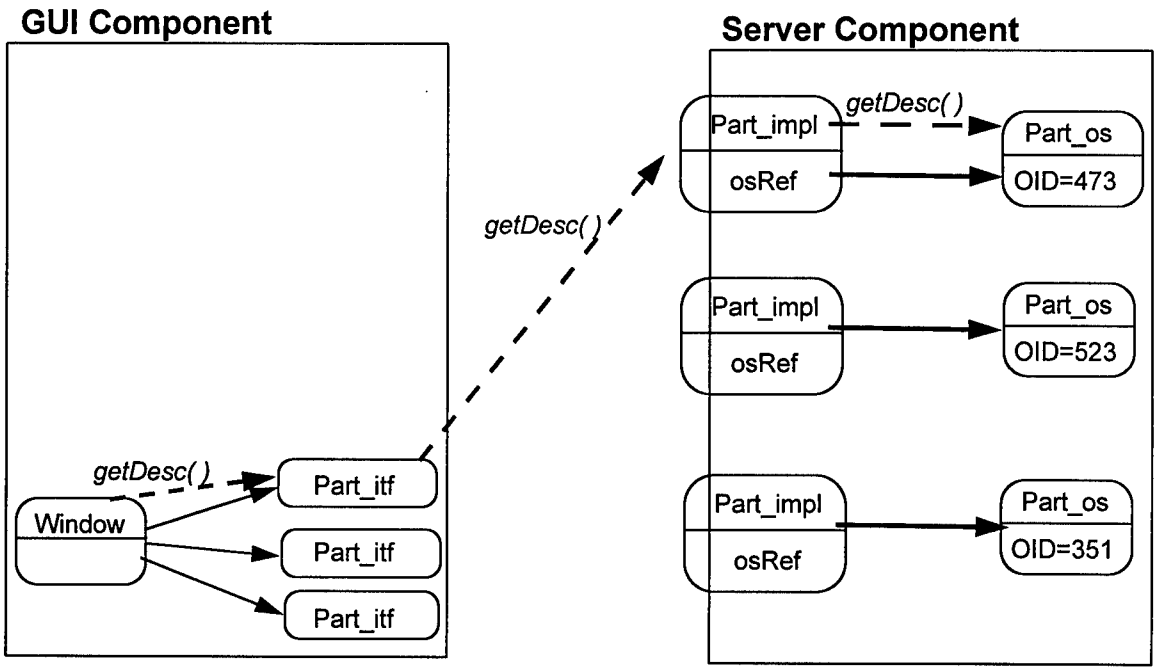

Figure 7- Dynamic Reference Passing

- Singleton approach -- Under this approach, the mapping between business objects and CORBA references essentially occurs at the class level. Methods in the business class map to operations on an IDL interface, but all the objects of the business class are represented by a single CORBA object instance. In order

to get to a particular business object instance, a remote client must pass a unique object identifier as an additional parameter on every operation invocation. (Refer to Figure 8.)

The choice of either architectural approach results in an impact on the effort and resources required to build and support the underlying technical architecture. The singleton approach avoids system resource issues, but was implemented using a partially-compliant CORBA 1.0 product; the dynamic reference approach deserves further study with the availability of fully-compliant CORBA 2.0 products.

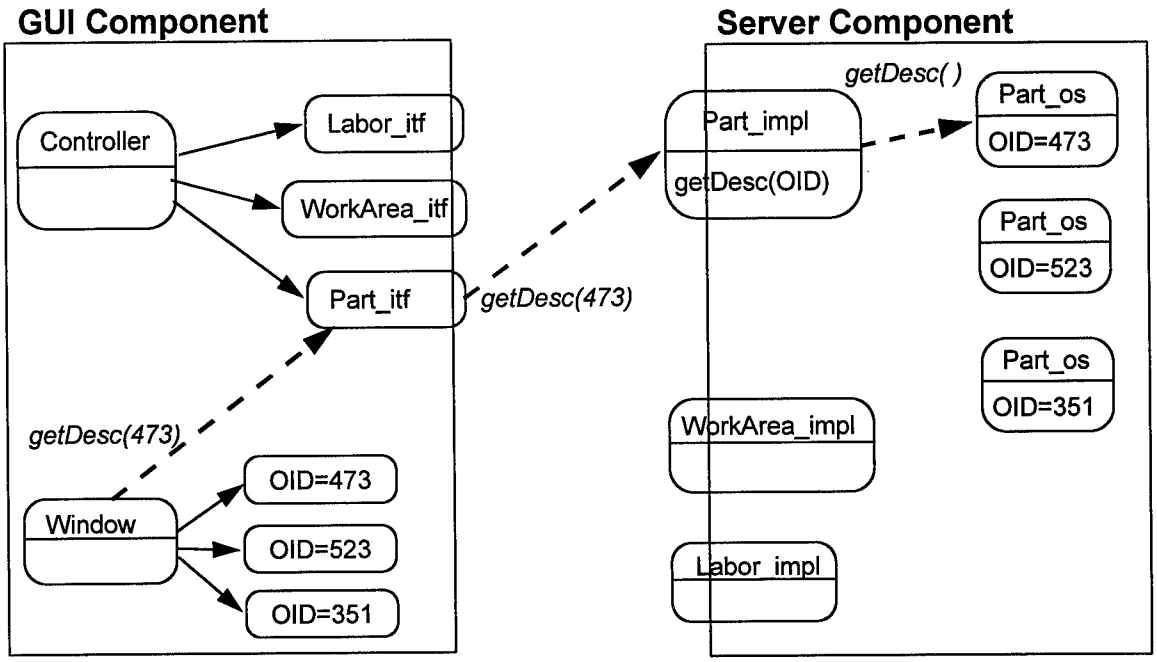

Figure 8 - Singleton Approach

5.4 Event Notification

A general problem in distributed component systems (and client-server systems in general) is the need to notify clients of significant state changes which may have been caused by other clients. We developed two approaches.

WCMS developed a publish-subscribe architecture which addresses both local (i.e., within the server) and remote (server-client) event notification. For remote events, this architecture provides functions roughly comparable to the basic services defined in the OMG's COSS Event Services. The architecture is, however, simpler to use than COSS Events Services and supports event subscription-notification on ORBs (such as PowerBroker) which do not support the COSS Event Services. The event notification mechanism also provides for the queuing of event notifications at the client, so as to prevent problems associated with reentrant processing.

The remote event notification mechanism is not currently used in WCMS' web-based version. This is mostly due to the different usability expectations within a browser metaphor and the limitations of the HTML language.

CBMS uses an event notification mechanism which is based on a global event transmitter/receiver per component. A client subscribes to specific events (on the client side). The server side will broadcast all events to all client side receivers that are listening and the client will then filter out which events to use. This approach was simple to implement, but used proprietary PowerBroker features. Also, because every change is broadcast to every client, this approach has the disadvantage of being not very scaleable for a large number of components. However, the solution has provided the necessary functionality and performance for the current CBMS design.

Ultimately, because the choice of event mechanisms has an impact on performance, the solution may be a combination of the two approaches (a server-side subscription solution) so that events are filtered before being broadcast.

Also, combining the event loops of different tools (GUI Builder, ORB, 3D Mapping) into a single working solution proved to be quite challenging and required considerable vendor assistance.

5.5 Transaction Serialization

The use of PowerBroker in combination with our current OODBMS gives rise to the possibility of the loss of transactional integrity. This is due to the database's reliance on threads to support transaction isolation together with PowerBroker's reentrancy features and lack of support for threads.

The WCMS architecture includes a mechanism to prevent this integrity problem by serializing client accesses to a server component. CBMS has not yet implemented transaction serialization as their current plans do not require this solution.

5.6 Exception Handling

In a state-of-the-art CORBA/C++ environment, C++ standard exception handling should be seamlessly integrated with CORBA exceptions. In such a scenario, all the architect needs to do is define a set of exception classes and CORBA exception types, and define mechanisms for default exception mapping and handling.

However, in the case of WCMS and CBMS, the development and execution tools precluded the use of standard C++ exceptions. The team was forced to reconcile disparate proprietary exception handling mechanisms provided by PowerBroker and our database, and plug remaining gaps with the aid of a public domain exceptions library. This was done through the creation of a number of macros and exception mapping functions. With our combination of exception mechanisms, unlike when standard C++ exceptions are used, it is not practical to prevent resource leaks when exceptions are thrown.

The WCMS and CBMS architectures also include support for assertions, which are liberally used to enforce method pre-conditions. Unlike traditional applications, a component-oriented application server needs to be especially careful about ensuring that potentially buggy clients do not cause data corruption or crashes on the server. Therefore, three different assertion macros were developed to provide appropriate responses to pre-condition violations (contingent on where the violation originated and where it was caught).

The long-term solution is for the tools supplying generated code to catch up with the ANSII C++ standards.

5.7 Access Control

In order to support the production pilot, it was necessary for the WCMS application to include some form of access control (i.e. user security). The WCMS architecture includes support for controlling user access at two levels of granularity: class-function (i.e. can the user access a particular function in a particular class) and instance-function (i.e. can the user invoke a particular function on a particular instance of a given class).

In the new browser-based version of WCMS, end-users cannot be identified based on their UNIX logins. Therefore, the web-server's own user identification features are used to recognize specific users. Class-function access control is driven by a table which contains a list of valid userids and their access to various classes. For instance-function access control, the architecture defines a common "access checking" interface. However, the specific access control functionality must be custom designed/implemented for each class requiring it.

CBMS did not require access control.

5.8 Legacy Architecture

WCMS includes batch and on-line interfaces to a legacy mainframe system. For the batch interface, the mainframe generates extract files and transfers them to the UNIX server on a scheduled basis. On-line access to the mainframe is achieved via 3270-terminal emulation software.

- Each legacy interface consists of portions of the application server component, together with a separate legacy interface control component.

- The legacy interface control component is responsible for: (i) initiating a new update whenever new extract files are available; and (ii) pacing the update transactions so they can be processed concurrently with the GUI client transactions.

- The code within the application server is responsible for the actual updates to the database with data from the extract files.

CBMS does not need to support any legacy code at this time.

6. Conclusions

We conclude that the methodology applied by both teams and the resulting components hold considerable promise for the future. The methodology successfully encompassed marketing analysis and system engineering, activities traditionally outside the boundaries of the software engineering development process, and it also stressed good engineering practices such as prototyping and pathfinding technical risk areas. As the result of applying this methodology, both application teams have an extensible-open architecture where additional components can be independently developed and attached to create alternative software solutions or enhanced software solutions.

Because of our CBMS efforts, we are adapting the methodology on new projects such as Medium Extended Air Defense System (MEADS) and Command and Control Product

Lines (CCPL); the CBMS components are also prime candidates for reuse on these projects. Additionally, our international air traffic control projects are evaluating the TRP methodology for their next-generation product line. Finally, the TRP methodology is an important element in the next revision of our internal practices book which is the basis for the software development process of all future Raytheon Electronics System projects.

As stated earlier, WCMS is our Andover plant's first distributed component-based factory system. Raytheon is now investigating replacing some of its aging factory systems. As a result of TRP, we now have a component-based architecture foundation on which to build these future factory applications. That, along with the general component/OO development experience that we've gained along the way, will give us a tremendous head start as we embark on these new projects.

In summary, initial impressions of our applications by users are positive because the user-driven approach kept the user involved throughout the development process and committed to the end-product. The artifacts such as the scenarios, design documents, components, and source code are being stored in our reuse repository and are already identified as key assets for certain new projects. By leveraging the skills and experience of both teams as well as the lessons learned documented in this report, we will significantly benefit future projects.

Chapter
3

Expersoft

1. Introduction

This chapter covers Expersoft's participation as a member of the Rapid Object Application Development (ROAD) Consortium led by Andersen Consulting. The other members included Raytheon, CoGenTex and Rome Laboratory. In order to develop usable systems more rapidly, technology was developed to assemble systems from re-usable components and describe new components via high-level languages and integrated graphic interfaces. In evaluating this approach, developers used the emerging tools and infrastructure to develop a battle management system and a factory work center management system. The overall goal of this project was to help the DOD and US industry meet the ever more pressing demands for complex defense and commercial applications in areas such as business re-engineering, knowledge-based systems, multi-media and the National Information Infrastructure.

1.1 Expersoft Role

The overall project was structured to be accomplished in three parallel activities. The first activity was Tool Development performed by Andersen Consulting and CoGenTex. The second was Application Development which was further sub-divided into a DoD Application performed by Raytheon and a Commercial Application performed by both Raytheon and Andersen. The third activity was the development of the Execution Environment performed by Expersoft.

The Execution Environment was built for Consortium member use and follows the requirements identified in the Object Management Group (OMG) CORBA 2.0 specification and the related Common Object Services (COS) specifications. In addition the Execution Environment included the integration of a GUI product, first Galaxy and then Neuron Data, provided for Smalltalk bindings, OLE integration, and an ability to inter-operate, in accordance with CORBA specifications, with other similar Consortia Execution Environments.

All work products developed by Expersoft were provided to consortium members as they were developed. Consortium members then provided feedback on the various pieces of the Execution Environment. All feedback was evaluated by Expersoft and incorporated back into the Execution Environment development in an iterative fashion. The net result was the development of a CORBA 2.0 execution environment useable by consortium application builders. This completed environment was then provided back to consortium members as enhancements. It was then further developed into a commercial product suitable to assist industry to meet the ever more pressing demands for development of

complex defense and commercial applications in areas of distributed computation systems and the National Information Infrastructure.

1.2 Methodology

The development of the Execution Environment followed the overall program methodology, that is it followed an iterative process model. Every 6 months a Build was performed that extended and revised the execution environment. Expersoft's own development process within a build cycle was also iterative in that it first built basic capability, refined this capability and added increased functionality in regular, measured steps. At appropriate intervals completed portions of the Execution Environment were provided to consortium members for evaluation. Evaluation feedback from Andersen and other consortium members was used to refine the previously delivered Execution Environment increments and to guide development of the next increment.

To allow application development to proceed early in the program while the Execution Environment was under development, Expersoft provided its proprietary infrastructure product (PowerBroker Extended C++) to the program with a CORBA 2.0 compliant IDL product as a front end to interface with the application code. This strategy provided a fully functional product to the consortium application builders allowing both tool development and applications development to proceed unencumbered by the development schedule of the final Execution Environment. A change over to the final Execution environment occurred in the final build.

All development was structured to allow easy migration into a commercial product. As such, full requirement documents were prepared along with test plans and automated code test routines to assure proper operation of resulting code. All development was controlled under a configuration management system (Clearcase). Lastly, full commercial quality documentation, including users manuals and reference manuals were prepared and provided to the Consortium.

2. Program "Build" Phases

As described in the ROAD Statement of Work, the project is divided into four (4) major phases. These phases occur every six (6) calendar months during the project. They are defined as "builds" of the execution environment, tools, and applications. The builds correspond roughly to phases defined in the JNIDS (Joint National Services Specification) Development Cycle. Build 1 corresponded to an "Initial Prototype". Build 2 consisted of "Iterative Refinement". Builds 3 and 4 are considered to be "Operational Prototypes". As laid out in the Statement of Work, Expersoft responsibilities for these builds has been for the "execution environment".

2.1 Execution Environment Build 1

Overall, the objectives for the execution environment for Build 1 were to make the initial tools available to other consortia members and plan several key interoperability objectives for the project and how they would be achieved. In particular, Expersoft was to provide:

initial development tools and training; an integration with Galaxy; support for CORBA IDL; and plans for interoperability and OLE integration.

Below are the specific tasks/milestones called out in the original Statement of Work:

B1.EE.1	5/1/95	Expersoft will provide the current version of XSHell for Training, early prototyping, and development.
B1.EE.2	5/15/95 - 7/13/95	Expersoft will release a version of Xshell that seamlessly integrates the Galaxy GUI event loop with Xshell's event loop. This will provide developers access to services provided in the Galaxy Application Environment via the Xshell execution environment.
B1.EE.3	8/12/95 - 10/4/95	A version of Xshell will be released which supports an IDL compiler with a C++ binding.
B1.EE.4	6/17/95 - 7/26/95	A development plan and schedule for interoperability among consortia as it relates to the execution environment will be completed.
B1.EE.5	8/5/95 - 9/7/95	The OLE integration strategy and development plan will be provided.

2.1.1 B1.EE.1 Initial Environment

The program proposed for this TRP included the initial development of the two applications in the first build. A preliminary Execution Environment was required from Expersoft to support these early activities. The initial environment provided to the consortium was our standard ORB (Object Request Broker) product, XShell 3.5. This product is a fully featured ORB, but its internal construction and interfaces did not conform to the requirements of the CORBA 2.0 specifications as required in our Statement of Work. To provide an IDL interface, we provided a CORBA 2.0 compliant IDL product as a front end to the XShell product. This allowed each application development team to write their applications using IDL and a robust ORB while waiting for the final CORBA compliant Execution Environment to be completed by Build 3. Note that during the course of the TRP Expersoft changed the name of XShell to PowerBroker Extended C++.

2.1.2 B1.EE.2 GUI Integration

An initial integration of Visix's Galaxy GUI tools was developed by Expersoft as part of the initial execution environment. However, use of this integration was suspended in July,

1995 due to the consortium's decision to replace the use of Galaxy with similar products from Neuron Data. An integration between Neuron Data tools and the initial XShell-based execution environment was designed and delivered to the consortium members in November, 1995

2.1.3 B1.EE.3 CORBA IDL

A CORBA IDL-to-C++ compiler was built and available to consortium members by July, 1995 on Solaris, Windows/NT, and AIX platforms. Defects in the HP-UX C++ compiler delayed availability on HP (Hewlett Packard) platforms by about 90 days (October, 1995).

2.1.4 B1.EE.4 Interoperability Plan

The initial efforts on designing the means of interoperability and planning its implementation were suspended by September/October of 1995. This was largely due to the complexity and coordination required among the various consortia execution environments. Subsequent to that, all interoperability concerns focused on the Internet Inter-ORB Protocol (IIOP). That protocol became the connection point between consortia execution environments. IIOP is defined and regulated by the Object Management Group (OMG). Compliance with this protocol is guaranteed by the execution environments used by other TRP consortia. As will be discussed elsewhere, this IIOP support was implemented as part of the actual ROAD-TRP execution environment.

2.1.5 B1.EE.5 OLE Integration Plan

A plan to integrate OLE with the Execution Environment was defined by Expersoft. This was delivered to the consortium in September, 1995. In a nutshell, this plan described the technical feature functionality of the OLE integration and a strawman schedule. This integration would be based on the OMG's COM/CORBA mapping specification and delivered to the Consortium as a part of the Execution Environment in April 1996.

2.2 Execution Environment Build 2

Build 2 objectives for the execution environment consist mainly of putting in place the basic facilities for CORBA-based application development.

B2.EE.1	10/1/95 - 11/23/95	Develop and provide for use by other consortium members interface repository functionality for Xshell.
B2.EE.2	12/9/95 - 3/5/95	Develop and provide for use by other consortium members dynamic invocation interface functionality in Xshell.

B2.EE.3	11/18/95 - 3/5/96	Develop and provide for use by other consortium members distributed namespace support in Xshell.
B2.EE.4	8/26/95 - 1/17/96	Develop and provide for use by other consortium members Smalltalk IDL binding for Xshell.
B2.EE.5	11/4/95 - 3/5/96	Provide multi-threaded application support in XShell
B1.EE.6	8/19/95 - 2/28/96	Develop and provide for use by other consortium members CORBA 2.0 interoperability support in XShell

Lessons Learned

Among the more significant aspects of the Build 2 experience was that Expersoft encountered a number of significant technical problems in porting the Execution Environment developed technologies to HP-UX. These difficulties were caused by a number of crucial defects in the C++ language support on that platform which were expected to be resolved by HP by the end of 1995. Unfortunately, HP compiler improvements lagged by almost a year, causing substantial rewrite of the Execution Environment code to work around compiler deficiencies. Due to these problems, there was a substantial delay between availability of execution environment functionality on the PowerBroker CORBAplus reference platforms (Solaris & Windows/NT/95) and the platform chosen for consortium application development (HP-UX).

Below is a discussion of each of these Build 2 items.

2.2.1 B2.EE.1 Interface Repository

A CORBA 2.0 compliant Interface Repository is included in the Execution Environment. The Interface Repository feature was available in a "beta-test" version to ROAD-TRP consortium members in February, 1996 (on Windows/NT). General availability of the Execution Environment (on Solaris & Windows/NT/95) occurred by June, 1996. A preliminary version on HP-UX was available to consortium members at the beginning of August, 1996. General availability on HP-UX occurred in September, 1996

2.2.2 B2.EE.2 Dynamic Invocation Interface

The Dynamic Invocation Interface specified by CORBA 2.0 is included in the base Execution Environment. This interface (DII) was available in a "beta-test" version to ROAD-TRP consortium members February, 1996 (on Solaris & Windows/NT platforms). General availability of the complete version (on Solaris & Windows/NT/95) occurred by June, 1996. A preliminary version on HP-UX was available to consortium members at the beginning of August, 1996. General availability on HP-UX occurred in September, 1996

2.2.3 B2.EE.3 Distributed Namespace

The distributed namespace functionality was provided by the COSS Naming Service implemented as part of the base Execution Environment. The service was delivered initially to consortium members in "beta-test" form (on Windows/NT & Solaris) in February, 1996. A final version, including the Naming service, was made available by June, 1996 on the Solaris and Windows/NT/95 platforms. A preliminary version on HP-UX was available to consortium members at the beginning of August, 1996. Complete availability on HP-UX occurred in September, 1996

2.2.4 B2.EE.4 Smalltalk Bindings

Expersoft has provided CORBA-compliant Smalltalk ORB functionality in the form of the PowerBroker CORBAplus for Smalltalk product. This was made available, first in a "beta-test" form, by April, 1996. Commercial, general availability of a Smalltalk binding occurred (for Solaris & Windows/NT) by June, 1996. A version of this Smalltalk support was validated for use on HP-UX in conjunction with the port of the base C++ product. General availability for HP occurred in September, 1996.

Lessons Learned

It should be noted that a significant delay occurred in the development and production of Smalltalk bindings. The initial design path of adding Smalltalk bindings as a layer above the base C++ ORB was unsuccessful. A significant design change was required, involving introduction of a pure Smalltalk ORB. This accounted for the substantial delay in making the Smalltalk bindings available. Since Smalltalk was not employed in the ROAD-TRP applications, this did not impact consortium activities.

2.2.5 B2.EE.5 Multi-Threaded Support

The base Execution Environment supports implementation of multi-threaded applications. This support was first made available to consortium members in a "beta-test" version in December, 1995 (on Windows/NT). General Availability (on Solaris & Windows/NT/95) occurred by June, 1996. As is described elsewhere, difficulties porting to HP-UX resulted in a delay of availability on the HP platform. Initial deliveries on HP did not include thread-safe libraries. However release 2.0.2 in December, 1996 included support for multi-threaded application development on HP.

2.2.6 B2.EE.6 CORBA 2.0 Interoperability

Interoperability specified by the CORBA 2.0 standard is achieved via the Internet Inter-ORB Protocol (IIOP). Support for this protocol is at the core of the PowerBroker CORBAplus 2.0 product. An initial "beta-test" version of CORBAplus supporting this protocol was made available in February, 1996 (on Windows/NT). General availability on Solaris & Windows/NT/95 occurred by June, 1996. A preliminary version on HP-UX was

available to consortium members at the beginning of August, 1996. Complete availability on HP-UX was available in September, 1996.

2.3 Execution Environment Build 3

From the standpoint of the execution environment, Build 3 was intended to fill out the layer of key application services above the basic ORB. In this build, Expersoft was to deliver several CORBA-specified application services along with facilities for integrating with OLE and on-line transaction processing. Below is a summary of the objectives for the execution environment for Build 3. This includes a brief description from the Statement of Work, and original time-frames.

B3.EE.1	2/26/96 - 5/15/96	Demonstrate ORB interoperability with other consortia
B3.EE.2	6/15/96 - 9/27/96	Develop and provide for use by other consortium members on-line transaction services
B3.EE.3	4/26/96 - 8/14/96	Develop and provide for use by other consortium members COSS Lifecycle Services for XShell
B3.EE.4	2/26/96 - 5/15/96	Develop and provide for use by other consortium members COSS Naming Services for XShell
B3.EE.5	5/11/96 - 9/26/96	Develop and provide for use by other consortium members COSS Event Service and publish/subscribe mechanism for XShell
B3.EE.6	2/26/96 - 9/26/96	Develop and provide for use by other consortium members OLE integration with XShell

Below is a discussion, item-by-item, of the status of the Build 3 objectives for the execution environment. It should be noted that, as was described earlier, the execution environment became based on the PowerBroker CORBAplus product, not XShell. In addition, availability of items discussed below may have depended on the delivery platform. Where that was the case, it is noted.

2.3.1 B3.EE.1 ORB Interoperability

As was mentioned earlier, interoperability of the ORB in the execution environment was to be achieved using the Internet Inter-ORB Protocol (IIOP). That protocol is fully supported by the Execution Environment. The ability of the base ORB to inter-operate with other commercial ORB's was demonstrated at the Object World East trade show / conference in

March 1996. Since this interoperability is a standard feature of IIOP, and thus a standard feature of the CORBAplus product, it is supported in the Execution Environment as any other product feature.

2.3.2 B3.EE.2 Transactions

The ROAD-TRP applications, in the end, did not require the use of CORBA transaction services in the Execution Environment. However, design and implementation work still continued at Expersoft in order to support potential late use of transactions. This work was also used as the foundation of eventual incorporation in CORBAplus. A design and implementation plan was determined by Expersoft by October of 1996, and is reflected in Expersoft functional and design specifications. Implementation of the extensions necessary to the base Execution Environment have been implemented and available as Release 2.1 on 15 April, 1997. This implementation will support the integration of a commercially available On-Line Transaction Service (OTS) and Transaction Processing Monitor. That integration is currently being performed by the TP vendor. An initial version of the integration should be available to consortium members by the end of the ROAD-TRP project.

2.3.3 B3.EE.3 Lifecycle

The Execution Environment supports standard object activation policies defined by the CORBA 2.0 specification as required by the Consortium. This includes support for Basic Object Adapter activatable objects. Standard implementation and executable repositories are included as well. These object Lifecycle support features were included in the Execution Environment and were available commercially in June, 1996 (Solaris & NT). Initial availability to consortium members on HP-UX came at the beginning of August, 1996 (commercially available in September, 1996). The support for these object Lifecycle management features, though providing standard capabilities, is not done through the COSS defined interface. Further development is expected to refine the life Cycle service to include COSS interfaces.

2.3.4 B3.EE.4 Naming

A COSS Naming Service was implemented as part of the Execution Environment. This service provides a mechanism for objects on an ORB to locate other objects. This service was delivered initially to consortium members in "beta-test" form (on Windows/NT & Solaris) in February, 1996. An updated version of the Execution Environment, including the Naming service, were made available by June, 1996. As is discussed elsewhere, technical difficulties in porting the Execution Environment to HP-UX delayed availability on HP. Initial availability to consortium members of Naming on HP-UX was delayed until the beginning of August, 1996.

2.3.5 B3.EE.3 Events

The initial Execution Environment did not include either a COSS Event Service or a Publish & Subscribe service. During research and design at Expersoft, the design decision was made to support the publish & subscribe paradigm by integrating message-oriented middleware. Work to that end has been on-going at Expersoft, and continues at this writing. Since that approach would not provide the basic event services in the time-frame required for the ROAD-TRP application development, Expersoft has built an additional, simpler COSS-compliant Event Service. This Event Service was constructed specifically to meet the needs of the ROAD-TRP application development activities and was delivered to consortium members in November, 1996. The publish and subscribe work continues at Expersoft. Functional and design specifications exist, and an initial version of the messaging substrate was available at the end of April, 1997. It should be noted that Expersoft has submitted its approach to messaging, developed under this program, to the OMG for standardization. In addition, Expersoft is working with others to define a CORBA standard for publish & subscribe style services.

2.3.6 B3.EE.4 OLE Integration

As was discussed earlier, the approach taken to integrating OLE with the ROAD-TRP execution environment was to adopt the CORBA/COM mapping approach defined by the OMG. This mapping was implemented by Expersoft for the Execution Environment. This effort formed the basis of a commercial product called CORBAplus for OLE. A "beta-test" version of this integration was made available to consortium members in April, 1996. It should be noted that due to the nature of this product, CORBAplus for OLE only exists for Windows/NT and Windows/95 platforms. It supports interaction between Windows-based OLE applications and CORBA-compliant ORB's on Windows, UNIX, or other platforms.

2.4 Execution Environment Build 4

Build 4 was not expected to include new functionality in the execution environment. Instead, from the standpoint of Expersoft, this phase should consist of updates and fixes to the Execution Environment as required to support the application development teams. The Statement of Work captured this in its single Build 4 execution environment task:

B4.EE.1	10/26/96 - 4/30/97	Support and maintain Xshell for application and tool developers

Support and maintenance of the Execution Environment and resulting PowerBroker CORBAplus products is on-going at Expersoft. Support has been provided to consortium members ranging from informational to detailed technical troubleshooting. In addition, Expersoft engineering staff and consultants have been involved in on-site support, training, and development services. As part of Build 4, Expersoft has recently completed the

software development activities of converting one of the target applications from XShell to new Execution Environment to aid the Raytheon engineering team.

It should be noted that the PowerBroker CORBAplus products based on the Execution Environment are actively used at a number of customer sites beyond the consortium. All functionality employed by consortium tool and application developers is in the standard product. This assures the continuation of support for the execution environment components from Expersoft.

3. Program Summary / Conclusions

As a direct result of this program, Expersoft was able to develop and deliver a complete CORBA 2.0 compliant Object Request Broker (ORB) to the consortium members as a major component of the Execution Environment. This development effort was further extended into a complete commercial product, PowerBroker CORBAplus for C++ and OLE and included a Smalltalk binding. This product was first introduced at Object World East on May 8, 1996. At this industry show the CORBAplus product was demonstrated and subsequently won two major awards for Excellence; the first was the Industry Judges Best New CORBA-based Product, the second was the Object World Attendees award, voted by all the people attending the show. These awards were based on both technical excellence, standards compliance, and ease of use. This program enabled the development of this product some 1 to 2 years earlier than could be accomplished without program funding.

Certain CORBA Services were also sponsored, including Naming, Life Cycle, Events, Object Transactions and one element not specified by the OMG, Publish and Subscribe, as an extension of the Events Service. Of these the Naming Service and Events service (push model) and a partial Life Cycle service were delivered to the Consortium. Naming and the partial Life Cycle service were first included in the standard product as a part of our May product announcement and continues to meet the needs of both the consortium and the commercial marketplace. Events was first provided to the consortium members in November 1996 and subsequently included into the standard product in March 1997. As discussed in Section 2 above, the Object Transactions Service was delayed in development in favor of other program required Services. However, the development of the Transaction Service is a major priority and remains under active development even though program funding has ceased. This Service and follow-on expansion of the other Services will be completed in the near future and included in our standard product.

Interoperability between Consortiums was an early requirement. Expersoft demonstrated its interoperability with the consortium ORBs at the Object World Shows.

Chapter
4

CoGenTex

1. Introduction

CoGenTex, Inc. participated in the TRP ROAD project as a Small Business Partner (SBIR). According to the DARPA (Defense Advanced Research Projects Agency) provisions for the TRP small business partners, CoGenTex was supposed to transfer to the partners of the TRP ROAD consortium the technologies being developed under an on-going Small Business Innovation Research project. Cost sharing was accomplished by supporting the TRP work being done on the SBIR project in question, "Linguistically-Based Software Documentation Generation", awarded by AF Rome Laboratory, contract number F30602-94-C-0124.

Participation in the TRP ROAD program has presented new opportunities and challenges for CoGenTex. As a small company with a rapidly emerging technology, the program provided us an opportunity to validate the technology with industry leaders, Andersen Consulting and Raytheon, for use in both commercial and defense-oriented systems. At the same time, we were challenged by our exposure to two different styles of managing the software development process and its products. The valuable feedback obtained from both these consortium partners provided an opportunity to mature the technology and better situate it in the software development process.

Under the SBIR contract, CoGenTex developed text generation applications in two areas of the general software engineering domain: ModelExplainer, for object-oriented software models explanation and documentation, and CogentHelp, for authoring dynamically generated on-line help. During the TRP contract, these applications were ported to the tools in the TRP tool suite and substantially enhanced in response to the feedback from the application team users at Raytheon. In particular, ModelExplainer was ported to the Ptech CASE tool, and to the Epistemics PC PACK requirements analysis and engineering tool. CogentHelp was initially ported to the Neuron Data GUI building tool, and then to the Java AWT (Abstract Windowing Toolkit). In addition, CoGenTex developed a first version of LIDA (LInguistic assistant for Domain Analysis), an add-on to PC PACK that pre-processes texts for the human analyst, and CogentTIPS, an application that generates Teamware Integrated Performance Support (TIPS).

The next four sections describe these applications in detail, with overall program conclusions following in the final section.

2. ModelExplainer: Customizable Descriptions of Object-Oriented Models

2.1 Introduction: Object Models

With the emergence of object-oriented technology and user-centered software engineering paradigms, the requirements analysis phase has changed in two important ways: it has become an iterative activity, and it has become more closely linked to the design phase of software engineering (Davis, 1993). A requirements analyst builds a formal object oriented (OO) domain model. A user (domain expert) validates the domain model. Then, the domain model undergoes subsequent evolution (modification or adjustment) by a (perhaps different) analyst. Finally, the domain model is passed to the designer (system analyst), who refines the model into a design model (also object-oriented, of course). This design model is used as the basis for implementation. Thus, we can see that the OO model forms the basis of many important flows of information in OO software engineering methodologies. How can this information best be communicated?

It is widely believed that graphical representations are easy to learn and use, both for modeling and for communication among the engineers and domain experts who together develop the OO domain model. This belief is reflected by the large number of graphical OO modeling tools currently in research labs and on the market. However, this belief is a fallacy, as some recent empirical studies show. For example, (Kim, 1990) simulated a modeling task with experienced analysts and a validation task with sophisticated users not familiar with the particular graphical language. Both user groups showed semantic error rates between 25% and 70% for the separately scored areas of entities, attributes, and relations. Relations were particularly troublesome to both analysts and users. (Petre, 1995) compares diagrams with textual representations of nested conditional structures (which can be compared to OO modeling in the complexity of the "paths" through the system). He finds that "the intrinsic difficulty of the graphics mode [alone] was the strongest effect observed" (p.35). We therefore conclude that graphics, in order to assure maximum communicative efficiency, needs to be complemented by an alternate view of the data. We claim that an explanation tool that represents the data in the form of a fluent English text should provide the alternate view. This chapter presents such a tool, the ModelExplainer, or ModEx for short, and focuses on the customizability of the system.

Automatically generating natural-language descriptions of software models and specifications is not a new idea. The first such system was Swartout's GIST Paraphraser (Swartout, 1982). More recent projects include the paraphraser in ARIES (Johnson, Feather, and Harris, 1992); the GEMA data-flow diagram describer (Scott and de Souza, 1989); and Gulla's paraphraser for the PPP system (Gulla, 1993). ModEx certainly belongs in the tradition of these specification paraphrasers, but the combination of features that we will describe in the next section (and in particular the customizability) is, to our knowledge, unique.

2.2 Features of ModEx for Ptech

Our design of ModEx for Ptech is based on initial interviews with software engineers working on a project at Raytheon, and was modified in response to subsequent feedback during an iterative prototyping approach when these software engineers were using our system.

- ModEx output integrates tables, text generated automatically, and text entered freely by the user. Automatically generated text includes paragraphs describing the relations between classes, and paragraphs describing examples. The human-authored text can capture information not deducible from the model (such as high-level descriptions of purpose associated with the classes).

- ModEx lets the user customize the text plans at run-time, so that the text can reflect individual user or organizational preferences regarding the content and/or layout of the output.

- ModEx uses an interactive hypertext interface (based on standard html-based WWW technology) to allow users to browse through the model. The output can of course also be printed on paper.

- Input to ModEx is based on the ODL (Object Definition Language) standard developed by the Object Database Management Group (OMG) (Cattell, 1994). This allows for integration with most existing COTS OO modeling tools. Some previous systems have paraphrased complex modeling languages that are not widely used outside the research community (GIST, PPP).

- ModEx does not have access to knowledge about the domain of the OO model (beyond the OO model itself) and is therefore portable to new domains.

The availability of different types of output (automatically generated, tabular, human-authored) and the ability to customize the output means that ModEx can be used in a flexible manner to address all the communication needs outlined above.

2.3 A ModEx Scenario

Suppose that a university has hired a consulting company to build an information system for its administration. Figure 1 shows an example of object model (adapted from [p.56] (Cattell, 1994), using the notation for cardinality of (Martin and Odell, 1992)) that could be designed by the requirements analyst from the consulting company.

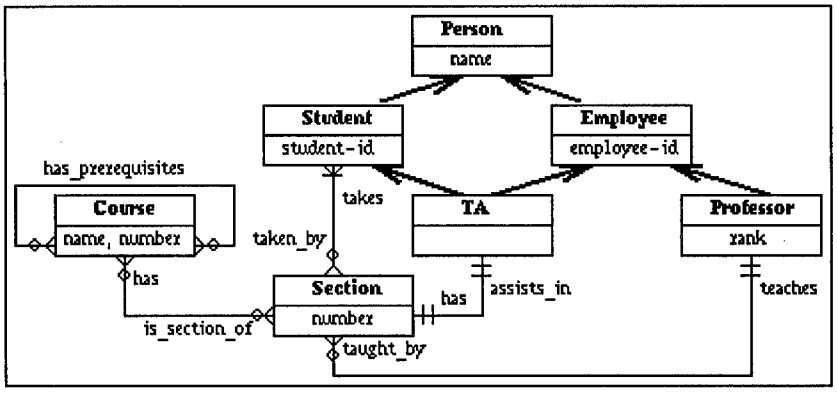

Figure 1. The University O-O Diagram

Once the object model is specified, the requirements analyst must validate her model with a university administrator (and maybe other university personnel, such as data-entry

clerks); as domain expert, the university administrator may find semantic errors undetected by the requirements analyst. However, he is unfamiliar with the "crow's foot" notation used in Figure 1. Instead, he uses ModEx to generate fluent English descriptions of the model, which uses the domain terms from the model. Figure 2 shows an example of a description generated by ModEx for the university model. The administrator can browse through the model using the hypertext interface at his own pace. Suppose that the university administrator notices that the model allows a section to belong to zero courses, which is in fact not the case at his university. He points out the error to the requirements analyst, who fixes the model.

Suppose now that the administrator finds the example texts useful but insufficient. He decides to change the content of the output texts so that they include a more detailed example paragraph instead of the current shorter example paragraph. To do this, he can go to the Text Plan Configuration window for the text he has been looking at, shown in Figure 3. He replaces the Examples-Short paragraph with the Examples-Long paragraph from the list of pre-built constituents (shown in the lower right corner of Figure 3.). He can now return to browsing the model, using texts with extended example sections.

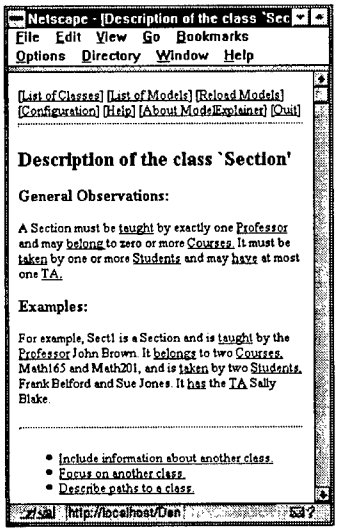

Figure 2. Description Used for Validation

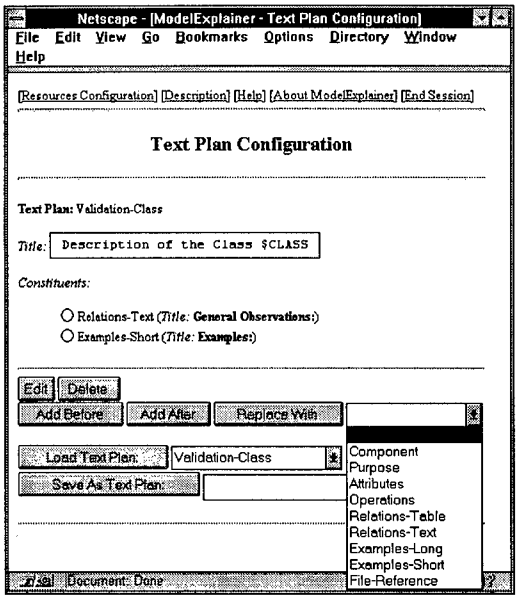

Figure 3. Text Plan for Validation

Once the model has been validated by the university administrator, the requirements analyst needs to document it, including annotations about the purpose and rationale of classes, attributes, and operations, so that other requirements analysts can understand it during evolution later in the life cycle, and so that the designer can properly refine it into a design model of the information system. To document it, she configures an output text type whose content and structure is compatible with her company's standard for OO documentation. The corresponding text plan is illustrated in Figure 4, and an example of a description obtained from this text plan is shown in Figure 5. (This description follows a format close to the Andersen Consulting standard for the documentation.) This description is composed of different types of information: text generated automatically (section Relationships), text entered manually by the user because the information required is not retrievable from the object model (section Purpose), and tables composed both of information generated automatically and information entered manually by the user (sections Attributes and Operations). The analyst then saves the text plan under a new name in a library of text plans and it can be used subsequently for documentation purposes. Note that while the generated documentation is in hypertext format and can be browsed interactively (as in the I-DOC system of (Johnson and Erdem, 1995)), it can of course also be printed for traditional paper-based documentation and/or exported to desktop publishing environments.

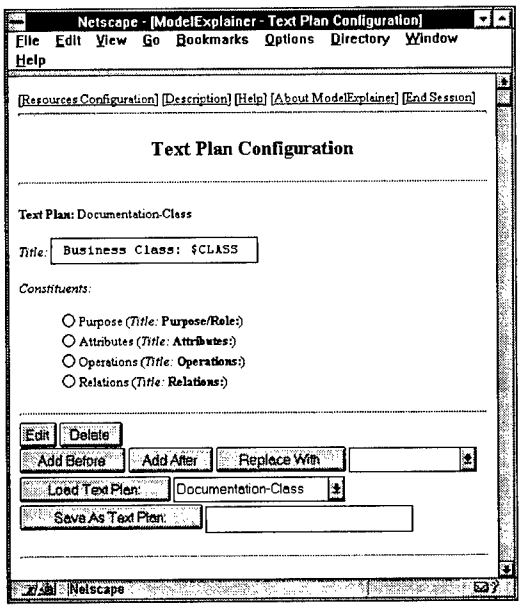

Figure 4. Text Plan for Documentation

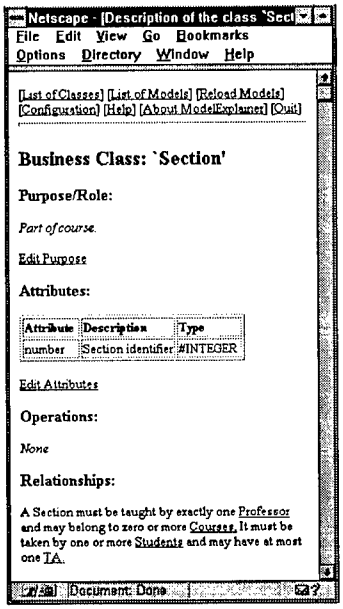

Figure 5. Description Used for Documentation

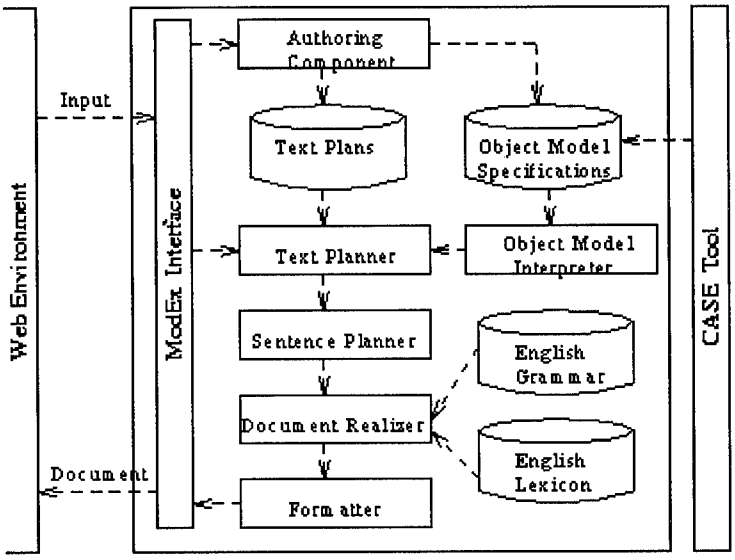

Figure 6. ModEx Server Architecture

2.4 How ModEx Works

As mentioned above, ModEx has been developed as a WWW application; this gives the system a platform-independent hypertext interface. The ModEx Server is composed of seven major components organized in the architecture illustrated in Figure 6. The ModEx server receives control and description requests via a standard Web CGI (Common Gateway Interface) interface, and returns HTML-formatted documents which can be displayed by any standard Web browser. These HTML documents are always generated dynamically in response to a request, there are no `canned' pre-existing HTML documents. In the current implementation, four main types of request are considered:

- Edit a text plan. This generates an HTML document such as the documents shown in Figures 3 and 4, which allow to load/edit/save a text plan.
- Load an object model. This generates a document displaying the list of classes found in the object model.
- Generate a description. This returns a document such as the documents shown in Figures 3 and 5.
- Save human-authored text. This refreshes the current document to reflect the text newly edited by the user.

ModEx is implemented in C++ on both UNIX and PC platforms. It has been integrated with Ptech, a commercial CASE tool in the TRP tool suite and has been fielded at a Software Engineering Laboratory at Raytheon, Inc.

2.5 ModEx for PC PACK

ModEx has been extended to a new CASE environment, PC PACK from Epistemics, Ltd. This new integration extends ModEx's functionalities by integrating several different types of descriptions based on the extended information found in the different PC PACK tools: entity-relationship (E-R) models, also known as object-oriented data models; process models; concept ladders (IS-A hierarchies) and processes ladders (PART-OF hierarchies). Here is a list of description types identified after consultation with personnel from Andersen Consulting and from Epistemics:

- Descriptions for E-R models. This type of description, already supported by the previous versions of ModEx, is based on the information manipulated in the PCPACK E-R Tool. It consists of a description of an entity with its relations to the other entities, including an example paragraph.

- Descriptions for concept definitions. This type of description, based mainly on information from the (Concept) Laddering Tool, defines an entity by comparing it to closely related entities, namely the supertype concept and sister concepts.

- Descriptions for process models. This type of description, based on information from the Control Editor, describes a process with its inputs and outputs (data-stores, exit values) as well as its relationships to the neighbor processes.

- Description for process decomposition. This type of description, based on information from the (Process) Laddering Tool, consists of a description of the process's constituents and the process's container (parent).

In addition, there is an interface that gives access to all types of descriptions that mention a given element from the PCPACK knowledge base.

2.6 Lessons Learned

Our first lesson learned with ModEx concerned the need for a customizable naming convention for labels used in a model. ModEx uses no domain knowledge, since it was designed to be independent of any particular domain. Nevertheless, ModEx must be able to correctly interpret the categories (nominal, verbal, prepositional, etc.) of the labels, in order to generate fluent text. For this reason, we decided to require ModEx users to label their models using a naming convention based on recommendations made by leading authors on object-oriented modeling. From the feedback we received about ModEx, we learned that users were willing to follow ModEx naming conventions. This willingness derives in part from the fact that model designers agree that a systematic naming convention can help a reader to understand a model. Of course, ModEx's naming convention remains idiosyncratic in the sense that it may not match conventions in prior use (if any). This can be a problem, for example, for a user who already has a library of models that do not conform to the ModEx naming convention. We concluded therefore that the ModEx naming convention is useful but too restrictive: it should be made customizable to consider the model designer's own conventions. We did not have the opportunity to extend ModEx in this sense.

A second lesson learned concerned the need for customizable text plans. A text plan is a text generation resource used to determine both the content and the organization of a text that is to be generated. The first versions of ModEx included a library of text plans, but this was built-in in the system. Feedback from the application team users at Raytheon

indicated that there were slight differences in the preferences regarding the content of the text or its organization. For example, for the table of operations generated by ModEx, the preferences regarding its content and the naming of the entries were slightly different for two of the users who gave us feedback. It thus appeared that there was a need for customizable text. We have addressed this need in the final version of ModEx developed under this contract.

3. LIDA: LInguistic assistant for Domain Analysis

LIDA (LInguistic assistant for Domain Analysis) is an add-on to PC PACK (Epistemics, Ltd.) which pre-processes texts so that they can be handled more efficiently in PC PACK's Protocol Editor by the human analyst. In our initial effort, financed by the TRP, we packaged existing technology into an innovative and robust tool. LIDA performs two tasks: it "normalizes" highlighted passages and it searches for the most frequent words, displaying them in the order of frequency. Here, "normalization" refers to a combination of two operations. First, in a purely formatting operation, leading and trailing white spaces and punctuation marks are stripped. Then, in a linguistic operation, the words in the highlighted passage are replaced by their base form (for example, *seen* is replaced by *see* or *employee's* by *employee*). This is important since typically a session in the Protocol Editor will lead to a large number of synonymous concepts that differ only typographically and/or in inflection. The analyst spends much time in identifying such synonyms.

The initial version of LIDA operates directly on the protocol file and therefore requires only minimal integration with PC PACK. LIDA does not itself display the text.

We will further explain the functionality by giving a more detailed account below. In the following, "base form" refers to the grammatical base form (infinitive verb, singular noun, and so on), lower case, without leading or trailing blanks or punctuation marks.

The user may start by loading the text into the protocol editor and doing some editing. When done with initial editing, the user must save the protocol. Then, the user invokes the LIDA interface, which gives him the following options:

- SPECIFY FILENAME. The user enters the name of the file that LIDA will process.
- NORMALIZE. LIDA processes the file and adds the base form of each word in an invisible field. This is done automatically prior to any other LIDA processing.
- SPECIFY TARGET WORD. The user enters a target word.
- HIGHLIGHT TARGET WORD. The user first chooses a color. LIDA then normalizes the target word and searches the file for occurrences. LIDA augments the file in such a manner that the occurrences of words whose base form is the desired form will be shown in the chosen color. If the word is already highlighted, the color is overridden.
- In addition, LIDA can also highlight the newly marked words in a special manner (a red special symbol preceding the occurrence), so that the user can easily find them through the protocol editor. This special marking is removed prior to the next LIDA operation. There is also a REMOVE NEW TAG option that removes all LIDA-inserted occurrences of the special symbol.

- SHOW MOST FREQUENT. LIDA counts the number of occurrences of base forms and display a list of words ranked by frequency (most frequent first, excluding certain pure function words, such as "the" or "does"). The user can then click on one of the words and invoke "Search" (see above).

- DETERMINE SENTENCE BOUNDARIES. LIDA reformats the file in such a way that each sentence starts a new line. (Optionally, the sentences can also be numbered.)

After the user has used LIDA through the LIDA interface, he goes back to the Protocol Editor and loads the modified file.

4. CogentHelp: A Tool for Authoring Dynamically Generated On-line Help

4.1 Introduction

CogentHelp is a prototype tool for authoring dynamically generated on-line help for applications with graphical user interfaces (GUIs). Unlike most help authoring tools, CogentHelp embodies the "evolution-friendly" properties of tools in the literate programming tradition (Knuth, 1992; Ramsey, 1994) — e.g., the `javadoc` utility for generating API documentation from comments embedded in Java source code (Friendly, 1995; cf. also Johnson and Erdem, 1995; Priestly et al., 1996; Korgen, 1996). CogentHelp is also unusual in that it is (to date) one of the few tools to bring natural language generation (NLG) techniques to bear on the problem of authoring dynamically generated documents (cf. also Knott et al., 1996; Hirst and DiMarco, 1995; Binsted, 1995).

4.2 Design Goals

We set out to achieve three main goals in designing CogentHelp, each of which has various aspects. The first of these goals is end user–oriented, whereas the latter two are developer-oriented.

The first goal is to promote quality in the resulting help systems, which includes promoting

- **Consistency** — the grouping of material into help pages, the use of formatting devices such as headings, bullets, and graphics, and the general writing style should be consistent throughout the help system;

- **Navigability** — the use of grouping and formatting should make it easier to find information about a particular GUI component in the help system;

- **Completeness** — all GUI components should be documented;

- **Relevance** — information should be limited to that which is likely to be of current relevance, given the current GUI state;

- **Conciseness** — redundancy should be avoided;

- **Coherence** — information about GUI components should be presented in a logical and contextually appropriate fashion.

The second goal is to facilitate evolution, which includes facilitating

- **Fidelity** — the help author should be assisted in producing complete and up-to-date descriptions of GUI components;

- **Reuse** — wherever possible, the help author should not have to write the same text twice.

The final goal is to lower barriers to adopting the technology. This has principally meant providing an authoring interface that makes the benefits of the system available at a reasonable cost in terms of the understanding and effort required of the help author. As will be explained in the System Overview section below, the switch to the Java AWT should also make CogentHelp accessible to a much larger developer community.

4.3 Automated Documentation

The main idea of CogentHelp is to have developers or technical writers author the reference-oriented part of an application's help system in small pieces, indexed to the GUI components themselves, instead of in separate documents (or in one monolithic document). CogentHelp then dynamically assembles these pieces into a set of well-structured help pages for the end user to browse.

The principal advantage of this approach is that it makes it possible to keep the reference-oriented part of an on-line help system up-to-date automatically, since both the application GUI and the documentation can be evolved in sync. This benefit of generating both code and documentation from a single source has long been recognized, both in the NLG community (cf. Moore, 1995; and references therein) and in the SE community, where it is recognized under the banner of literate programming tradition (Knuth, 1992). Other important benefits stem from supporting the separation of the content of the document to be generated (best left to the author) from the structure of the document (well suited for rule-based processing).

To better understand where CogentHelp fits in, it is instructive to compare it with the closest reference points in the NLG and SE (Software Engineering) communities. On the NLG side, there is the Drafter system (Paris et al., 1995; Paris and Vander Linden, 1996), which generates drafts of instructions in both English and French from a single formal representation of the user's task. Drafter is considerably more ambitious in aiming to automate the production of multilingual, task-oriented help; at the same time, however, Drafter is more limited in that it is not evolution-oriented, aiming only to generate satisfactory initial drafts (whence its name).

On the SE side, the nearest reference points are a bit more distant, as CogentHelp has more in common with developer-oriented tools such as the `javadoc` API (Application Programming Interface) documentation generator distributed with Sun Microsystems' Java Developers Kit (Friendly, 1995) than with currently available help authoring tools.

4.4 System Overview

CogentHelp supports the authoring of various human-written text fragments (or "help snippets") indexed to an application's GUI widgets. The widgets themselves also provide

some useful help-related information in the form of types, labels, locations and part-whole relations for GUI widgets. Using this information, CogentHelp dynamically generates HTML help pages for end users. These pages are designed to support efficient navigation, through the use of intelligent, functionally structured layout as well as through an expandable/collapsible table of contents and a "thumbnail sketch" applet for visual navigation.

In evolving the design of CogentHelp, we have benefited greatly from the feedback provided by our consortium partners at Raytheon. In particular, in working with WCMS team members acting as trial help authors, we came to appreciate the importance of providing some means of visual navigation, as well as the need to support authoring via a custom authoring interface (rather than relying on resource files or existing resource editors).

In order to support the WCMS team as a user group, we initially redesigned CogentHelp's predecessor (which worked with AC's FOUNDATION GUI builder and WinHelp) to work with the Neuron Data cross-platform GUI builder and HTML. CogentHelp itself became a multi-user server written in Java, making it highly cross-platform. To mediate access to the dynamically generated texts, displayed in Netscape, we used the Common Gateway Interface (CGI) built into the NCSA (National Center for Supercomputing Applications) web server; to retrieve run-time information, we made use of Expersoft's PowerBroker ORB.

Towards the end of the project, for reasons having nothing to do with CogentHelp, the WCMS team decided to abandon Neuron Data and switch to using HTML forms. Since we did not consider HTML forms a suitable target for CogentHelp, this change meant the WCMS team would have to become a feedback group, rather than an actual user group. (With HTML forms, since space is at less of a premium, and links are simple to make, we believe there is less need for a sophisticated help-authoring framework.)

While this turn of events was initially troubling, it did have the positive effect of enabling us to focus our efforts on porting CogentHelp to the Java Abstract Windowing Toolkit (AWT) sooner than had originally been planned. We view this switch as positive for several reasons. First, unlike the Neuron Data tool, which is expensive, the Java AWT is distributed as freeware, and various GUI-building environments are available at low cost. Second, the dynamic nature of the AWT fits well with the CogentHelp approach, as a more dynamic interface increases the utility of dynamically generated help. Finally, the AWT enables a simpler architecture, since with the AWT it becomes possible to turn CogentHelp into a lightweight add-in component that runs in its own thread.

4.5 NLG Techniques

Although CogentHelp is by no means a typical NLG system, it does employ certain natural language generation techniques in order to support the design goals described above. These techniques fall into two categories, those pertaining to knowledge representation and those pertaining to text planning.

4.5.1 Knowledge Representation

In developing CogentHelp, we have taken a minimalist approach to knowledge representation following a methodology for building text generators developed over several years at CoGenTex. As will be explained below, this approach has led us to (i) make use of a large-grained "phrasal" lexicon and (ii) devise and implement a widget-clustering

algorithm for recovering functional groupings, as part of an Intermediate Knowledge Representation System (IKRS).

4.5.1.1 Phrasal Lexicon

As Milosavljevic et al. (1996) argue, to optimize coverage and cost it makes sense to choose an underlying representation which

- makes precisely those distinctions that are relevant for the intended range of generated texts; and
- is no more abstract than is required for the inference processes which need to be performed over the representation.

They go on to argue that besides eliminating a great deal of unnecessary `generation from first principles,' this approach complements their use of a phrasal lexicon (Kukich, 1983) at the linguistic level.

Applying essentially the same approach to the design of CogentHelp, we first determined that for the intended range of generated texts it suffices to associate with each widget to be documented a small number of atomic propositions and properties, identifiable by type. Next we determined that since no inference is required, these propositions and properties can be conflated with their linguistic realizations — i.e., the indexed, human-authored help snippets CogentHelp takes as input. While we did not originally think of CogentHelp's collection of input help snippets as a phrasal lexicon a la Milosavljevic et al., in retrospect it becomes evident that this collection can be viewed as tantamount to one. (Of course, since these snippets vary in size from phrases to paragraphs, the term "phrasal" is not entirely accurate.)

The types of snippets in current use include a one-sentence short description of the relevant GUI component; a paragraph-sized elaboration on the short description; various phrase-sized messages concerning the conditions under which it is visible or enabled, if appropriate; and a list of references to other topics. In the case of the phrase-sized messages, each of these fields is accompanied by a syntactic frame which prompts the author to provide consistent syntax — for example, entries in the WHEN_ENABLED field should fit the frame

This widget is enabled when _____.

4.5.1.2 IKRS

To enhance the modularity and robustness of a practical text generator, Korelsky et al. (1993) argue for the use of an Intermediate Knowledge Representation System (IKRS, pronounced "Icarus"). One purpose of an IKRS, they suggest, is to provide a component in which to locate domain-level inferencing not provided by the application program. Note that while this type of inferencing is motivated by text planning needs, it is still about the domain rather than about natural language communication, and thus does not belong in the text planner itself.

In developing CogentHelp, we encountered a need for just this sort of inferencing in order to support sensible layout. The problem we faced was how to logically arrange descriptions of widgets within a help page (or set of help pages) describing a window, which is the basic unit of organization in CogentHelp. As will be explained below, grouping

widgets by type was considered inadequate, because doing so would obscure functional relationships between widgets of different types. A naive spatial sorting was likewise considered inadequate, as this would inevitably separate elements of a functional group appearing in a certain area of the window.

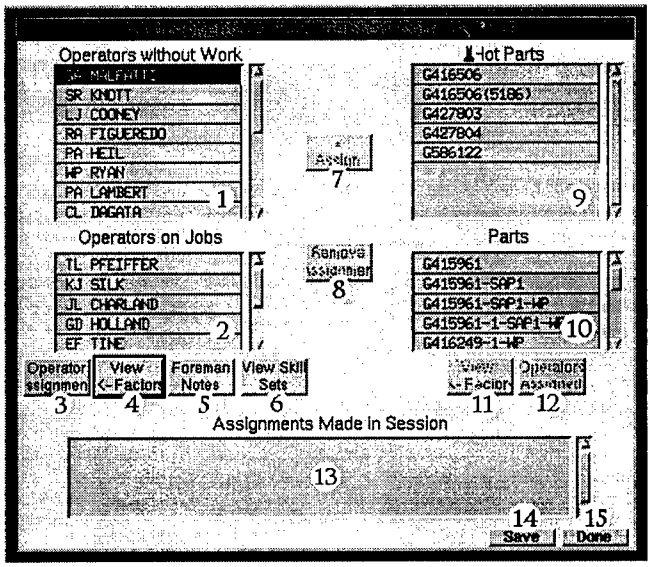

Figure 7: Sample Application Window

To illustrate the problem, consider the sample application window shown in Figure 7, which is from one of the WCMS prototypes. This window, whose purpose is to allow a manufacturing shop floor foreman to assign operators to parts, is organized as follows: on the left there are two list boxes for operators (1, 2), with buttons beneath them to access information about these operators (3, 4, 5, 6); on the right there are two list boxes for parts (9, 10), with buttons beneath them to access information about these parts (11, 12); in the middle there are two buttons for making and removing assignments (7, 8); towards the bottom there is a list box showing the assignments made so far (of which there are none — 13); and at the bottom there are standard buttons such as Save and Done (14, 15 — the Help button would go here). Given this organization, consider first arranging descriptions by type, and alphabetizing: besides cutting across the implicit functional groupings, arranging descriptions in this way would end up putting the two View K-Factors buttons (4, 11) in sequence, without any indication of which was which! Now consider a simple top-down, left-to-right spatial sort: again, this would inevitably yield a rather incoherent ordering, such as the Operators without Work list box (1), the Hot Parts list box (9), the Assign button (7), the Operators on Jobs list box (2), the Parts list box (10), the Remove Assignment button (8), etc.

Our approach to this problem was to develop, as part of our IKRS, a clustering algorithm for recovering these functional groups, using spatial cues as heuristics. The reason this approach might be expected to work is that in a well-designed GUI, functionally related widgets are usually clustered together spatially in order to make the end user's life a bit easier. With any clustering approach, there is always the tricky matter of determining a suitable distance measure: after trying out a variety of features, what we found to work surprisingly well was a simple weighted combination of proximity, alignment and type identity. In particular, in a test suite of around 15 windows provided to us by our trial user group, we obtained reasonable results (no egregiously bad groupings) on all of them

without undue sensitivity to the exact weights. In the case of the window shown in Figure 7, the clustering procedure performed exactly as desired, yielding precisely the groupings used in the description of this window given above — i.e.: (((1, 2), (3, 4, 5, 6)), ((9, 10), (11, 12)), (7, 8), 13, (14, 15)).

4.5.2 Text Planning

At the core of CogentHelp is the text planner. The text planner builds up HTML trees starting from an initial goal, using information provided by the IKRS, following the directives coded in CogentHelp's text planning rules. These HTML trees are then linearized into an ASCII stream by a separate formatter, so that they can be displayed in a web browser.

The text planner is constructed using Exemplars for Java, a lightweight framework for building object-oriented text planners in Java, which has been developed in parallel with CogentHelp. In this framework, text planning rules — the exemplars, so-called because they are meant to capture an exemplary way of achieving a communicative goal in a given communicative context — are objects which cooperate to efficiently produce the desired texts. While space precludes a detailed description, it is worth noting that Exemplars for Java supports abstraction, specialization and content-based revisions, all of which have proved useful in the present effort.

In developing the exemplars used in CogentHelp, we have made use of three NLG techniques: structuring texts by traversing domain relations, automatically grouping related information, and using revisions to simplify the handling of constraint interactions. We will illustrate the first two techniques below; for reasons of space, we will omit further discussion of the third.

4.5.2.1 Capitalizing on Domain Structure

When text structure follows domain structure, one can generate text by selectively following appropriate links in the input (Paris, 1988; Sibun, 1992). In the case at hand, we have chosen to use the group and cluster structure combined with a top-down, left-to-right spatial sort: while such a spatial sort alone is insufficient, as we saw above, a spatial sort which respects functional groups turns out to work well.

Returning to the example of Figure 7, traversing the clusters in this way yields a listing which (naturally!) mirrors the order and groupings of the one-sentence description of the window's organization we have provided — that is, following a general description of the window, there are descriptions of the two operators list boxes (1, 2), followed by descriptions of the four buttons providing additional information on operators (3, 4, 5, 6), followed next by the part list boxes (9, 10) and the buttons associated with them (11, 12), and so on. This is (partially) illustrated in Figure 8, which shows a sample help page generated by CogentHelp; note that the list of widgets in the dynamic TOC (Table of Contents) on the left side of the page is arranged according to this traversal. In the particular topic shown, the user has reached the second button (View K-Factors) of the group of four buttons beneath the Operators list boxes, as can be seen from the highlighting in the thumbnail sketch applet.

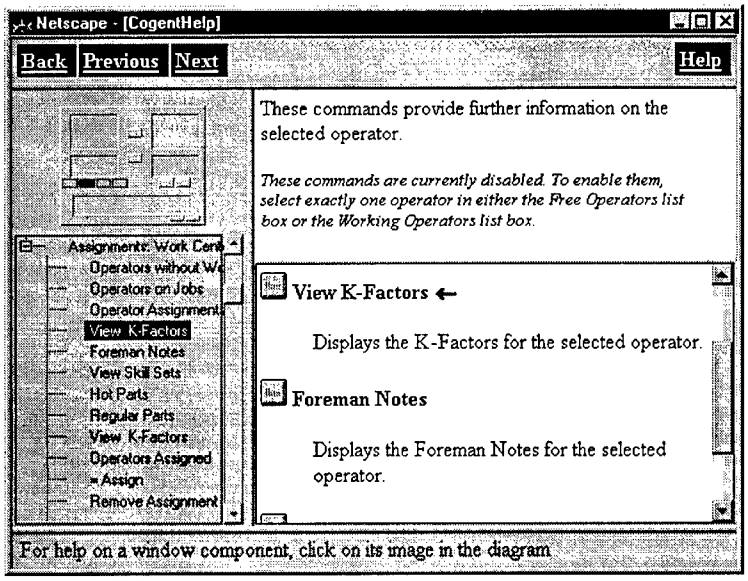

Figure 8: Sample Help Page

4.5.2.2 Grouping

Grouping related information and presenting shared parts just once is a well-known NLG technique for achieving conciseness and coherence (Reiter and Mellish, 1993). In a reference-oriented document such as an on-line help system, similar or identical descriptions will often be appropriate for elements that have similar or identical functions. To indicate these similarities, as well as to avoid redundancy, it makes sense to group these descriptions when possible. Note that these group descriptions are made possible by the use of a phrasal lexicon, which has been designed to allow the author's messages to make sense in a variety of contexts.

To illustrate, let us have another look at Figure 8. In the upper right frame of the page, note that there is the following description of how to enable all of the four buttons below the operators list boxes, rather than a repetition of the same message four times in close proximity:

> *These commands are currently disabled. To enable them, select exactly one operator in either the Free Operators list box or the Working Operators list box.*

This portion of the help text is produced by prepending "These commands are currently disabled. To enable them," to the shared message. Note that the resulting message is sensitive to whether the four buttons are currently disabled (a run-time fact).

4.6 Authoring

In informal usability tests with our WCMS feedback group, we determined that our first try at an authoring interface, which relied on the resource editor of the Neuron Data GUI builder, was unsatisfactory. In particular, it (i) required excessive clicking for the author to navigate from snippet to snippet, and (ii) failed to provide sufficient context, making it unnecessarily difficult for the author to adhere to the CogentHelp authoring guidelines.

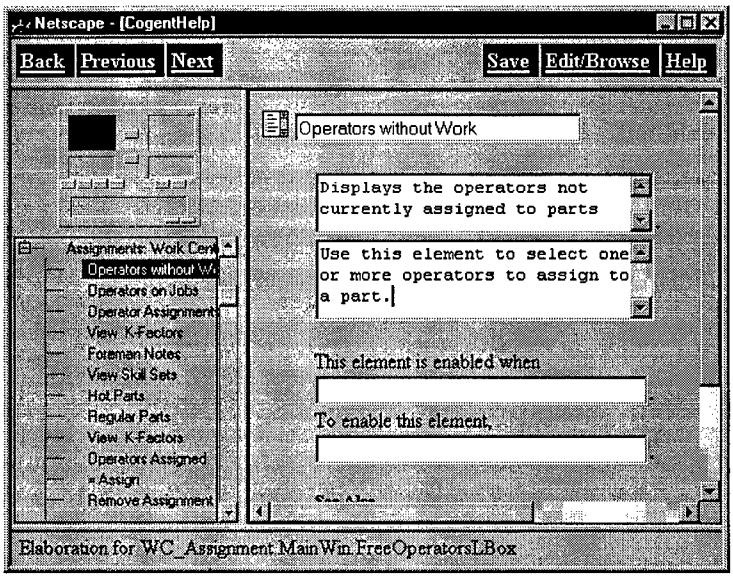

Figure 9: CogentHelp in Authoring Mode

In response to this feedback, we developed the authoring interface shown in Figure 9. This interface uses CogentHelp's existing architecture (together with the HTTP forms protocol) to allow the user to edit the text snippets for each widget in substantially the same context they would inhabit in generated help topics. This design provides maximal realism for the author, especially since one can switch between editing and browsing mode at the click of a button to preview the generated help.

4.7 Lessons Learned

In working with our feedback group at Raytheon, we discovered that it was essential to provide some means of visual navigation, as well as a custom authoring interface. This feedback directly led us to develop the thumbnail sketch applet and our current, browser-based authoring interface, as described in the preceding sections.

Two other lessons learned pertain to the architecture of CogentHelp as ported for the Java AWT. First, to simplify access to run-time state information, we have investigated turning CogentHelp into a lightweight add-in component that runs in its own thread. A problem we faced here was how to communicate with the browser without involving a web server. Fortunately, we discovered that a simple HTTP server could be written in a small amount of Java code that would be sufficient for our purposes. (Note that this much holds for Java applications; we have yet to investigate what changes would be required to support Java applets.)

The second lesson concerns the dynamic nature of Java GUIs, and in particular the absence of a separate resource library for GUI widgets. Since we relied on the names in the Neuron Data resource library in order to identify widgets, we faced the problem of how to reliably identify the widgets in a Java GUI, which when created programmatically can be completely anonymous. (Note that developers of existing WinHelp-based authoring tools will face the same problem in trying to port them to Java.) To resolve this problem, we decided to create our own registry, which allows widgets to be registered and named dynamically via API calls.

At a more general level, while we have gathered substantial feedback on CogentHelp functionality during the TRP effort, it should be stated that because of the many changes CogentHelp has undergone to this point, we will need to perform a post-TRP phase of evaluation to more fully validate the concept. In particular, we expect to face the issue of how to integrate CogentHelp with the various approaches to HTML-based help promoted by Netscape, Sun and Microsoft. Since none of these initiatives appear likely to address the dynamic and evolution-oriented aspects of CogentHelp, we are optimistic that the CogentHelp concept will find its niche and prove valuable to the rapidly growing Java developer community.

5. CogentTIPS

5.1 Introduction

As mentioned in Section 1, CogentTIPS is CoGenTex's facility for Teamware Integrated Performance Support (TIPS). For an overview of the context for CogentTIPS, as well as the notion of Teamware IPS, see Section 1.

5.2 Personalized Task Summaries

CogentTIPS is intended to help a project member understand how a project fits together, as well as its current status. To support project members in this way, CogentTIPS generates personalized task summary pages on demand for display in a web browser. These pages present salient information from the Project Manager tool, and provide hyperlinks to methodological guidance for the task and related assets. As will be explained further below, the methodology links are obtained indirectly from the process specification.

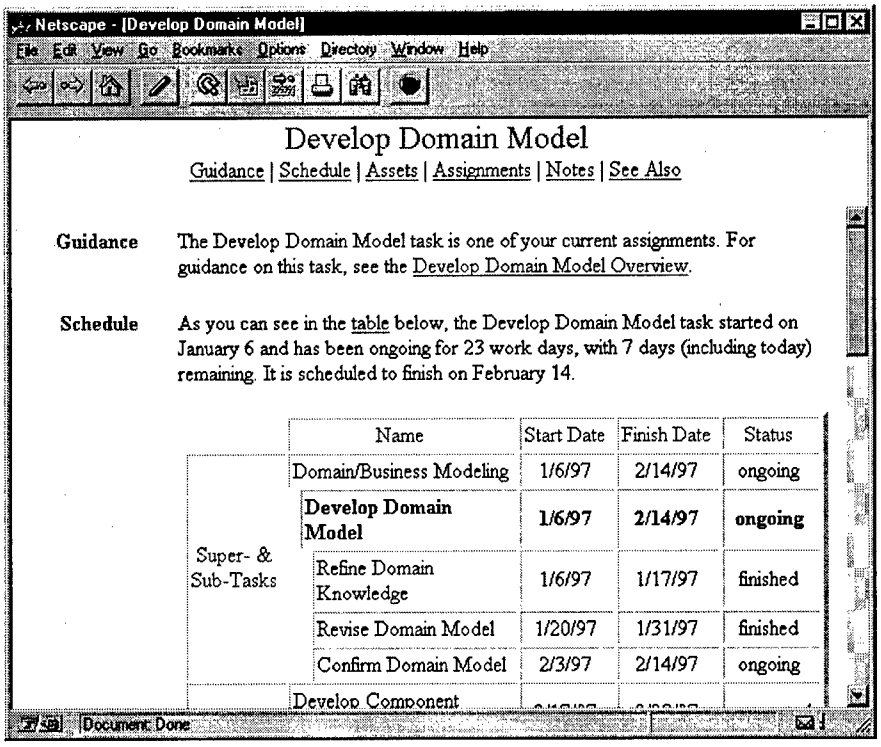

Figure 10: Sample CogentTIPS Page

A sample CogentTIPS task summary page is shown in Figure 10. At the top there is a banner frame. This non-scrolling banner identifies the topic task, and provides navigation support to facilitate access to the various categories of information available. These categories are guidance, schedule, assets, assignments, notes and further related information.

The information presented in the task summary pages is both personalized and contextualized to the current date. To illustrate, let us first consider the Guidance section of the page shown in Figure 10. Here the personalized message "The Develop Domain Model task is one of your current assignments" is provided to draw the reader in, as well as to help confirm that s/he is viewing the desired page. (Of course, if this page is being generated for a project member not currently assigned to this task, this message will not appear.) Following this message, a link is given to the appropriate methodology page for this task.

Next let us consider the Schedule section of this sample page, which also serves to illustrate how natural language text can be used to complement tabular information. Generally speaking, fluent text can complement tables by highlighting certain bits of derived information that a table alone would not make salient. Text can also be used to present information that is difficult to convey in tabular form, e.g. temporal and causal relations. At the same time, tables make information easy to find, are very concise, and add balance to the presentation.

The Schedule section is organized around a table that shows the dates and completion status for the topic task and related tasks. The related tasks are categorized into super- & sub-tasks and other related tasks. The subtask relationship is indicated via indenting, and the topic task is indicated via boldface type. All related tasks contain links to their summary pages.

Rather than jump directly in with a table, this section warms the reader up a bit with a short introductory paragraph highlighting pertinent and derived info for the topic task. Some items of note:

- Tense is used to present the information in a contextualized fashion: the past tense with "start" tells the reader that the start date is in the past — which is difficult to show in a table — and the present perfect with "be ongoing" indicates present relevance.

- The duration and remaining duration of the task in work days is derived information, and not all that obvious. Adding this info to the table, however, would make it too big, as this information is less important for the other tasks listed.

- The start and finish dates are treated asymmetrically in the text; the finish date could still change, whence the "is scheduled to finish" locution.

Following the table, there are three paragraphs of elaboration. The first paragraph establishes the context for this task in terms of its super-task. The second paragraph is used to present salient information for the sub-tasks. The third paragraph describes the temporal relationships between the topic task and the other related tasks (the presence of such relationships is what determines their inclusion in the first place). Note that these relationships are impossible to show in a table in a natural way (though diagrams such as Gantt charts would complement the text). The possible relationships are the four predecessor/successor ones (finish-to-start, start-to-start, etc.), optionally with lead/lag time (e.g. halfway through), and optionally constrained by specific dates (e.g. no later than).

Continuing on with the sections not shown, the Assets section describes how the topic task is linked to other tasks via producer/consumer asset relationships. It also provides links to asset descriptions in the methodology.

The Assignments section is organized around a table showing the project members assigned to the topic task plus their roles, effort levels and work complete percentages. For each project member, a link is provided to his or her project member page. Commentary precedes the table in the form a single paragraph. The paragraph begins with the effort level (avoiding percentages) for the reader, if applicable. It then mentions how many (other) project members are assigned to the task, and their names (unless there are more than three). Finally, it summarizes the work completion status, emphasizing the reader's status.

The Notes section shows any notes that have been attached to the topic task by the project manager.

The See Also section provides follow-up links, including one to the reader's project member page, one to the project overview page, and one to a page describing CogentTIPS.

5.3 System Overview

CogentTIPS draws its input from three sources: the Project Manager tool, the process specification in PSL used to set up the project, and the methodology mapping. The first two of these sources are described in Chapter 1. The last source is created and maintained using the CogentTIPS mapping tool.

The CogentTIPS mapping tool provides a means of creating, storing and maintaining a mapping between the activities, assets and roles specified in the input PSL and the URLs (Uniform Resource Locator) of the HTML pages that provide guidance on these entities. It is intended to be used as follows. Once the desired process has been specified in PSL, and the relevant methodological text converted to HTML, an analyst or methodologist uses the mapping tool to create the mapping for the input process. Later, at the start of a project using this methodology, the project manager tailors the PSL specification of the process to the needs of his or her project, and then invokes the mapping tool to update the mapping for any newly added activities, assets or roles. Since the same tailored PSL specification is used to set up the project, with appropriate links from entities in the project plan back to entities in the process specification, the methodology mapping can be used to provide appropriate hyperlinks to methodological guidance for project-level entities.

The CogentTIPS generator is an HTTP server written in Java. Requests to generate a summary page for a particular task are uniformly encoded as URLs, whether they arise from the TRP Desktop or as follow-up links in a generated task summary page. The texts themselves are generated using the Exemplars for Java text planning framework, which was originally developed for use in CogentHelp and has been extended to make use of the Java API to CoGenTex's RealPro syntactic realizer (which itself has been substantially improved to support its use in ModelExplainer). Note that in writing the exemplars (text planning rules) for CogentTIPS, the presence of RealPro is especially valuable insofar as it simplifies the handling of the syntax and morphology of tense. (For example, RealPro converts features for present perfect on "be" to yield "has been", also taking into account number agreement with the subject.)

5.4 Lessons Learned

Because the IPS infrastructure was slow to come together, CogentTIPS could not be implemented in time to receive substantial feedback during the TRP time frame. Nevertheless, this effort did provide a welcome opportunity to integrate the Exemplars for Java framework used in CogentHelp with the RealPro syntactic realizer used in ModelExplainer, as well as to benefit from the lessons learned during the CogentHelp and ModEx efforts.

6. Conclusions

Participation in the TRP ROAD program has been of great benefit to CoGenTex. As a small company with an emerging new technology, the TRP program provided us with the opportunity to validate this technology in collaboration with two well-established and respected large companies, Andersen Consulting and Raytheon, for use in both commercial and defense-oriented systems. An additional benefit was that we were exposed to two different styles of managing the software development process and its products. This exposure, together with valuable feedback from both consortium partners, allowed us to mature our technology and better situate it in the software development process.

Technically, our initial ideas, which we brought with us from our SBIR program, were both confirmed and modified by exposure to our TRP partners. For **ModelExplainer**, our initial claim was that the system would be a useful tool for developing object-oriented data models. While overall this claim was confirmed, we learned that natural language paraphrases are useful mainly in *validation*. Validation occurs early on the systems engineering process, when the level of description is still less formal. As a result, we have integrated ModelExplainer with PCPACK, which is a requirements acquisition tool, and for which the specific strengths of natural language generation will be most relevant. We also learned that at later stages of the process, the principal role for ModelExplainer is that of *documentation*, which requires a different type of document from validation. As a result, ModelExplainer addresses both validation and documentation needs, offering a customizable document format. For **CogentHelp**, we began with the idea of automatically generating help from GUI definitions and GUI scripts, in order to aid in keeping on-line help up-to-date with a rapidly evolving application. Along the way we learned that while there is a real need for this type of support tool, GUI scripts cannot be relied upon for useful information (a higher-level behavior specification would be necessary). This realization prompted us to focus on authoring, i.e. to focus on supporting the authoring of small pieces of help text (using a consistent style) indexed to GUI widgets. Raytheon feedback was crucial in helping us define this authoring functionality. Finally, we have introduced two new systems, which were motivated directly from our exposure to TRP: **CogentTIPS,** for Teamware Integrated Performance Support (TIPS) in the form of dynamically generated personalized task summary pages, and **LIDA,** for linguistic help at the very early stages of requirements engineering. All of these tools have in common that their role is clearly defined with respect to the systems engineering process: no CASE tool is useful if the user cannot use it as part of his or her overall methodology.

At the program level, our TRP program consortium partners greatly facilitated our efforts to make our technology known to vendors of process modeling, domain modeling and software engineering tools. We are pursuing these commercialization options beyond the TRP. In this regard, we would like to express our support for allowing small businesses to

use SBIR awards for cost sharing purposes, and suggest that the government adopt this policy more broadly. In the absence of this policy, participation in the TRP program would have been impossible, and thus we would have missed out on an excellent opportunity to evolve our technology and establish vital commercialization paths.

Bibliography

Benner, K. (1996). Addressing complexity, coordination, and automation in software development with the KBSA/ADM. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference (KBSE-96),* Syracuse, NY.

Binsted, Kim. (1995). Personalised patient education linked to the medical record. In *AI in Patient Education Workshop*, Glasgow, Scotland, August.

Cattell, R. G. G., editor (1994). *The Object Database Standard: ODMG-93.* Morgan Kaufman Publishers, San Mateo, CA.

Davis, A. M. (1993). *Software Requirements: Objects, Functions, and States.* Prentice-Hall, Inc., Upper Saddle River, NJ, revision edition.

Friendly, Lisa. (1995). The design of distributed hyperlinked programming documentation. In *International Workshop on Hypermedia Design.*

Gulla, J. (1993). *Explanation Generation in Information Systems Engineering.* PhD thesis, Norwegian Institute of Technology.

Hirst, Graeme and Chrysanne DiMarco. (1995). HealthDoc: Customizing patient information and health education by medical condition and personal characteristics. In AI in Patient Education Workshop, Glasgow, Scotland, August.

Johnson, W. L. and Erdem, A. (1995). Interactive explanation of software systems. In *Proceedings of the Tenth Knowledge-Based Software Engineering Conference (KBSE-95),* pages 155–164, Boston, Mass.

Johnson, W. L., Feather, M. S., and Harris, D. R. (1992). Representation and presentation of requirements knowledge. *IEEE Transactions on Software Engineering*, pages 853–869.

Kim, Y.-G. (1990). *Effects of Conceptual Data Modeling Formalisms on User Validation and Analyst Modeling of Information Requirements.* PhD thesis, University of Minnesota.

Knott, Alistair, Chris Mellish, Jon Oberlander, and Michael O'Donnell. (1996). Sources of flexibility in dynamic hypertext generation. In *Proceedings of the Eighth International*

Natural Language Generation Workshop (INLG-96), pages 151–160, Herstmonceux Castle, Sussex, UK.

Knuth, D. E., editor. (1992). *Literate Programming.* CSLI.

Korelsky, Tanya, Daryl McCullough, and Owen Rambow. (1993). Knowledge requirements for the automatic generation of project management reports. In *Proceedings of the Eighth Knowledge-Based Software Engineering Conference (KBSE-93)*, pages 2–9, Chicago, Illinois.

Korgen, Susan. (1996). Object-oriented, single-source, on-line documents that update themselves. In *Proceedings of The 14th Annual International Conference on Computer Documentation (SIGDOC-96)*, pages 229–238.

Lavoie, Benoit, Owen Rambow, and Ehud Reiter. (1996). The ModelExplainer. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG-96)*, Herstmonceux Castle, Sussex, UK.

Martin, J. and Odell, J. (1992). *Object-Oriented Analysis and Design.* Prentice Hall, Englewood Cliffs, NJ.

Milosavljevic, Maria, Adrian Tulloch, and Robert Dale. (1996). Text generation in a dynamic hypertext environment. In *Proceedings of the 19th Australasian Computer Science Conference*, pages 229–238, Melbourne, Australia.

Moore, Johanna, editor. (1995). *Participating in Explanatory Dialogues.* MIT Press.

Murtagh, Fionn. (1985). *Multidimensional clustering algorithms.* Physica-Verlag, Vienna.

Paris, C., K. Vander Linden, M. Fischer, A. Hartley, L. Pemberton, R. Power, and D. Scott. (1995). A support tool for writing multilingual instructions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada.

Paris, Cecile. (1988). Tailoring object descriptions to the user's level of expertise. *Computational Linguistics*, 11(3): 64–78.

Paris, Cecile and Keith Vander Linden. (1996). Drafter: An interactive support tool for writing. *IEEE Computer, Special Issue on Interactive Natural Language Processing*, July.

Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6): 33–42.

Priestley, Michael, Luc Chamberland, and Julian Jones. (1996). Rethinking the reference manual: Using database technology on the www to provide complete, high-volume reference information without overwhelming your readers. In *Proceedings of the 14th*

Annual International Conference on Computer Documentation (SIGDOC-96), pages 23–28.

Rambow, Owen and Tanya Korelsky. (1992). Applied text generation. In *Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

Ramsey, Norman. (1994). Literate programming simplified. *IEEE Software*, 11(5):97–105, September.

Reiter, Ehud and Chris Mellish. (1993). Optimizing the costs and benefits of natural language generation. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, volume 2, pages 1164–1169.

Scott, D. and de Souza, C. (1989). *Conciliatory planning for extended descriptive texts.* Technical Report 2822, Philips Research Laboratory, Redhill, UK.

Sibun, Penelope. (1992). Generating text without trees. *Computational Intelligence*, 8(1):102–22.

Swartout, B. (1982). GIST English generator. In *Proceedings of the National Conference on Artificial Intelligence.* AAAI.

Wanner, Leo and Eduard Hovy. (1996). The HealthDoc sentence planner. In *Proceedings of the Eighth International Natural Language Generation Workshop (INLG-96)*, pages 1–10, Herstmonceux Castle, Sussex, UK.

☆U.S. GOVERNMENT PRINTING OFFICE: 1998-610-130-61170

DISTRIBUTION LIST

addresses	number of copies
DOUGLAS WHITE RL/C3C8 525 BROOKS RD ROME NY 13441-4505	20
DR. COLIN T. SCOTT ANDERSEN CONSULTING 3773 WILLOW ROAD NORTHBROOK IL 60062	5
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
RELIABILITY ANALYSIS CENTER 201 MILL ST. ROME NY 13440-8200	1
ROME LABORATORY/C3AB 525 BROOKS RD ROME NY 13441-4505	1
ATTN: GWEN NGUYEN GIDEP P.O. BOX 8000 CORONA CA 91718-8000	1

AFIT ACADEMIC LIBRARY/LDEE
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765 — 1

ATTN: R.L. DENISON
WRIGHT LABORATORY/MLPO, BLDG. 651
3005 P STREET, STE 6
WRIGHT-PATTERSON AFB OH 45433-7707 — 1

ATTN: GILBERT G. KUPERMAN
AL/CFHI, BLDG. 248
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022 — 1

ATTN: TECHNICAL DOCUMENTS CENTER
OL AL HSC/HRG
2698 G STREET
WRIGHT-PATTERSON AFB OH 45433-7604 — 1

AIR UNIVERSITY LIBRARY (AUL/LSAD)
600 CHENNAULT CIRCLE
MAXWELL AFB AL 36112-6424 — 1

US ARMY SSDC
P.O. BOX 1500
ATTN: CSSD-IM-PA
HUNTSVILLE AL 35807-3801 — 1

COMMANDING OFFICER
NCCOSC RDT&E DIVISION
ATTN: TECHNICAL LIBRARY,CODE D0274
53560 HULL STREET
SAN DIEGO CA 92152-5001 — 1

NAVAL AIR WARFARE CENTER
WEAPONS DIVISION
CODE 4BL000D
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100 — 1

SPACE & NAVAL WARFARE SYSTEMS CMD
ATTN: PMW163-1 (R. SKIANO)RM 1044A
53560 HULL ST.
SAN DIEGO, CA 92152-5002 — 2

SPACE & NAVAL WARFARE SYSTEMS COMMAND, EXECUTIVE DIRECTOR (PD13A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200 — 1

COMMANDER, SPACE & NAVAL WARFARE SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200 — 1

CDR, US ARMY MISSILE COMMAND
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSMI-RD-CS-R, DOCS
REDSTONE ARSENAL AL 35898-5241 — 2

ADVISORY GROUP ON ELECTRON DEVICES
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202 — 1

REPORT COLLECTION, CIC-14
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545 — 1

AEDC LIBRARY
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211 — 1

COMMANDER
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000 — 1

US DEPT OF TRANSPORTATION LIBRARY
FB10A, M-457, RM 930
800 INDEPENDENCE AVE, SW
WASH DC 22591 — 1

AWS TECHNICAL LIBRARY
859 BUCHANAN STREET, RM. 427
SCOTT AFB IL 62225-5118 — 1

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

SOFTWARE ENGINEERING INSTITUTE 1
CARNEGIE MELLON UNIVERSITY
4500 FIFTH AVENUE
PITTSBURGH PA 15213

NSA/CSS 1
K1
FT MEADE MD 20755-6000

ATTN: OM CHAUHAN 1
DCMC WICHITA
271 WEST THIRD STREET NORTH
SUITE 6000
WICHITA KS 67202-1212

PHILLIPS LABORATORY 1
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 2
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

DR. ROBERT PARKER 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR. GARY KOOB 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR. ROBERT LUCAS
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

DR. DAVID GUNNING
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

JOHN SALASIN
DARPA
3701 N. FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

HOWARD SHROBE
DARPA
3701 N. FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

HELEN GILL
DARPA
3701 N. FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

ROBERT LADDAGA
DARPA
3701 N. FAIRFAX DRIVE
ARLINGTON, VA 22203-1714 1

MARK SHULTZ
TRI-STATE RESEARCH
P.O. BOX 2194
NATICK MA 01760 1

DR. TANYA KORELSKY
COGENTEX, INC.
840 HANSHAW ROAD, SUITE 5
ITHACA, NY 14850-1589 1

BARRY HANTMAN
RAYTHEON
MISSLE SYSTEMS DIVISION
50 APPLE HILL DRIVE
TEWKSBURY, MA 01876-0901 1

DR. JOSEPH HINTZ 1
RAYTHEON
MISSLE SYSTEMS DIVISION
50 APPLE HILL DRIVE
TEWKSBURY, MA 01876-0901

TOM GREENE 1
EXPERSOFT
5825 OBERLIN DRIVE
SAN DIEGO, CA 92121