N94- 36497
1994031990

Fusing Modeling Techniques to Support Domain Analysis for Reuse Opportunities Identification

5/3-61

Susan Main Hall
Eileen McGuire
SofTech, Inc.
1600 N. Beauregard St.
Alexandria, Virginia 22311
(703)824-4561 FAX: (703)931-6530
email: shall@softech.com

/2695
P- 18

AB

Functional modeling techniques or object-oriented graphical representations, which are more useful to someone trying to understand the general design or high level requirements of a system?

For a recent domain analysis effort, the answer was a fusion of popular modeling techniques of both types. By using both functional and object-oriented techniques, the analysts involved were able to lean on their experience in function oriented software development, while taking advantage of the descriptive power available in object oriented models. In addition, a base of familiar modeling methods permitted the group of mostly new domain analysts to learn the details of the domain analysis process while producing a quality product.

This paper describes the background of this project and then provides a high level definition of domain analysis. The majority of this paper focuses on the modeling method developed and utilized during this analysis effort.

<u>Project Background</u>

The analysis work described in this paper was performed in support of the Software Development Center - Washington, Army Reuse Center (ARC). Using functional descriptions and design documentation of four Army software systems under development and the Department of Defense Technical Reference Model, the application support layer services, such as database services, network communications, and the human machine interface, were studied. In addition, technical references were used to support the development of the description for the User-Machine Interface (UMI). The primary goal of the effort was to develop a complete, understandable model of a generic application support layer system. When completed, this model was utilized to identify potential reuse opportunities between the existing software and future system development efforts. The majority of the work performed by the ARC and its supporting staff focuses on increasing software reuse in the government sector.

Domain Analysis

Domain Analysis is the process of identifying the commonalities in a class of similar systems [Priento-Diaz 90]. Domain analysis could be considered as requirements analysis performed on more than one system. The activities performed during domain analysis include collecting, organizing, analyzing and concisely capturing information from systems which perform similar tasks. System specifications, requirements documents, functional descriptions, design documents, and even users manuals can provide the information needed for domain analysis. The key to successful domain analysis is to have complete descriptions for at least three systems in the software family being studied. At least three systems are needed in domain analysis in order to obtain a non-system-specific view of the domain.

There are two ways to view a family of systems or domain: vertically or horizontally. A vertical domain encompasses systems which perform the same system application. For example, in Figure 1, Embedded Weapons Systems, Management Information

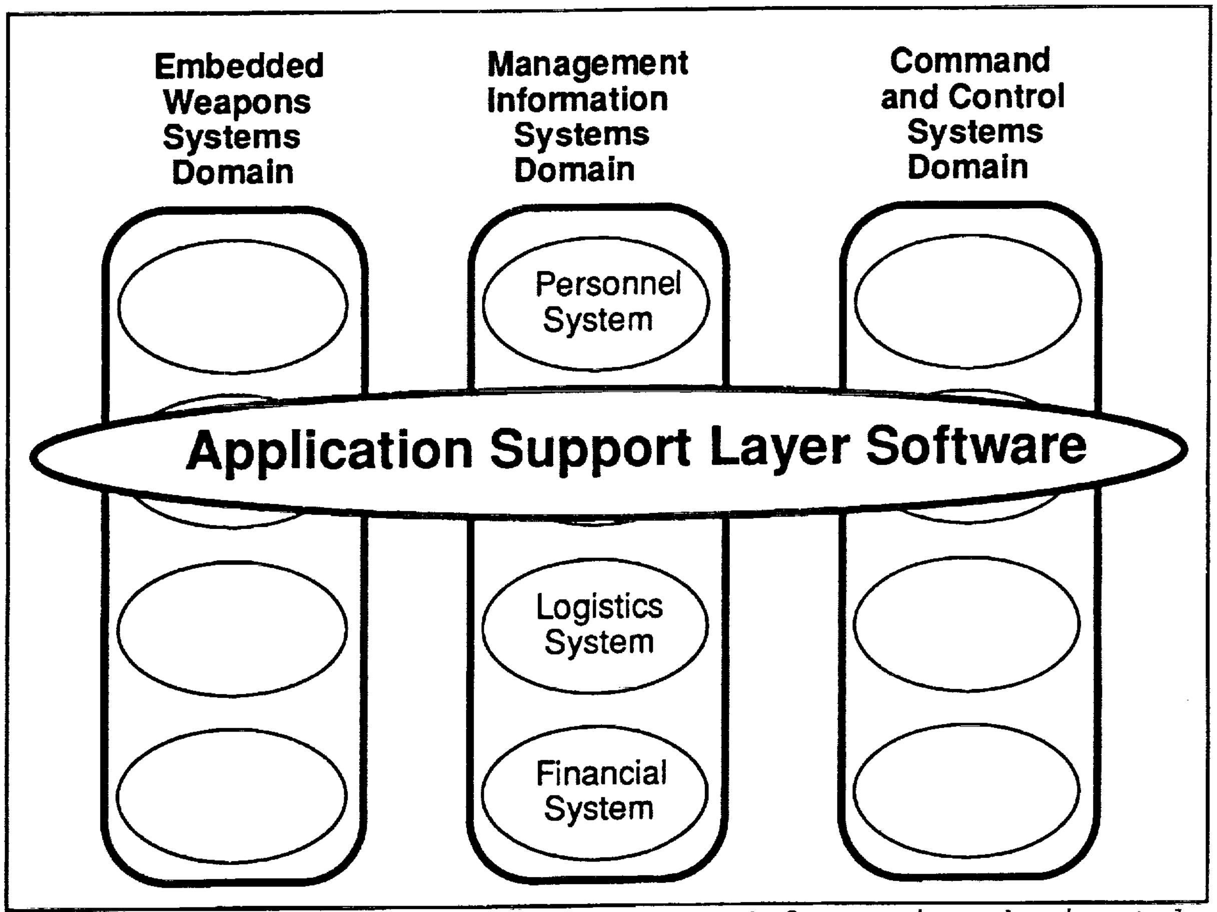

Embedded Weapons Systems Domain
Management Information Systems Domain
Command and Control Systems Domain
Personnel System
Application Support Layer Software
Logistics System
Financial System

Figure 1 *The Application Support Layer Software is a horizontal sub-domain of many software domains.*

2

Systems and Command and Control Systems represent three high-level vertical domains. A horizontal domain is an area of activity or knowledge that spans the vertical application oriented domains. The Application Support Layer Software domain is used as the example domain in this paper to describe the modeling technique/procedure that has been performed by this analysis team.

Unlike other domain analysis methods, the analysis procedure this SofTech team employs incorporates the concept of domain-oriented high demand categories and a knowledge base of domain and system information to support the identification of areas predisposed to reuse. The identification of reuse opportunities is a priority in our domain analysis work. The analysis procedure utilized is also unique because of the effective method of combining several modeling techniques that was developed (see the section below for a detailed description of this fused modeling technique). Note, although a thorough study of the application support layer services was performed, the pictorial description captured was limited to include only a high level illustration of the domain.

Combining Multiple Modeling Techniques

Domain modeling is the process of capturing, in graphical form, the conclusions resulting from analysis of a functional family of systems. Specifically, the operations, data and data attributes need to be recorded in a clear, concise format. In general, since object-oriented system descriptions make reuse opportunities easier to locate [Weesale 92], an object-oriented model was targeted to be the final product of the analysis effort. Unfortunately, one of the greatest challenges our team faced was striving to bridge the gap between systems that are still being functionally designed and the advantages that object oriented technologies offer. Therefore, we also came to the conclusion that a fusion of modeling methods was necessary due to the relative immaturity of the available techniques [Weesale 92].

When comparing and contrasting the understandability of modeling techniques, we have found that one modeling technique could not do the entire job well. Since the development of an understandable generic Application Support Layer (ASL) model is critical to its future use we decided to combine several very different modeling techniques. Our new modeling process includes utilizing functionally oriented models, moving into a functional hierarchical grouping model, and then transitioning into a set of object oriented models. Specifically, we used data flow diagrams [DeMarco 78], state transition diagrams, flow charts, hierarchical diagrams, and object models [Coad/Yourdon 91, Rumbaugh 91].

Studying the ASL software began by creating sketches of and reviewing pre-existing data flow diagrams, state transition diagrams, and flow charts from documentation available on the completed Army systems. These functionally oriented diagrams were beneficial to our understanding the systems because of the analysis team's experience in developing functionally oriented software. In addition, capture of the functionality of an ASL in these types of diagrams was performed quickly, since the example systems being studied had been developed using functionally-oriented methods. Both the analysts' experience and the system development techniques supported the easy understanding of the processes performed by typical ASL software.

High-level data flow diagrams provided the basis for the majority of the analysis work on the processes of the ASL software. For example, Figure 2 is a data flow diagram (DFD) of the primary functions performed by the ASL software, according to the functional descriptions of one of the systems studied. Also, shown in the same DFD are the general data flows between the functions. This type of diagram provided an understanding of the basic activities performed by an actual ASL code module.

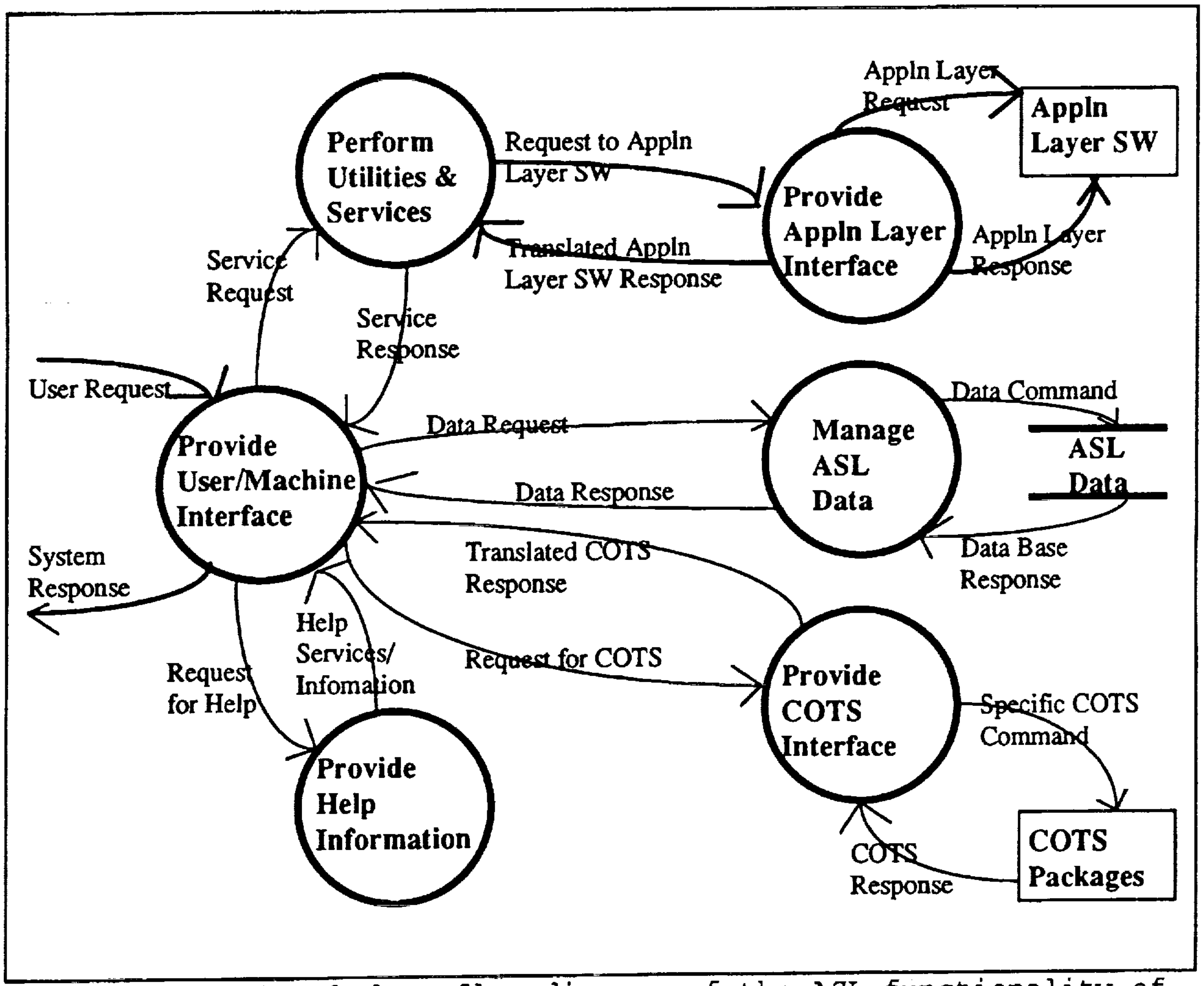


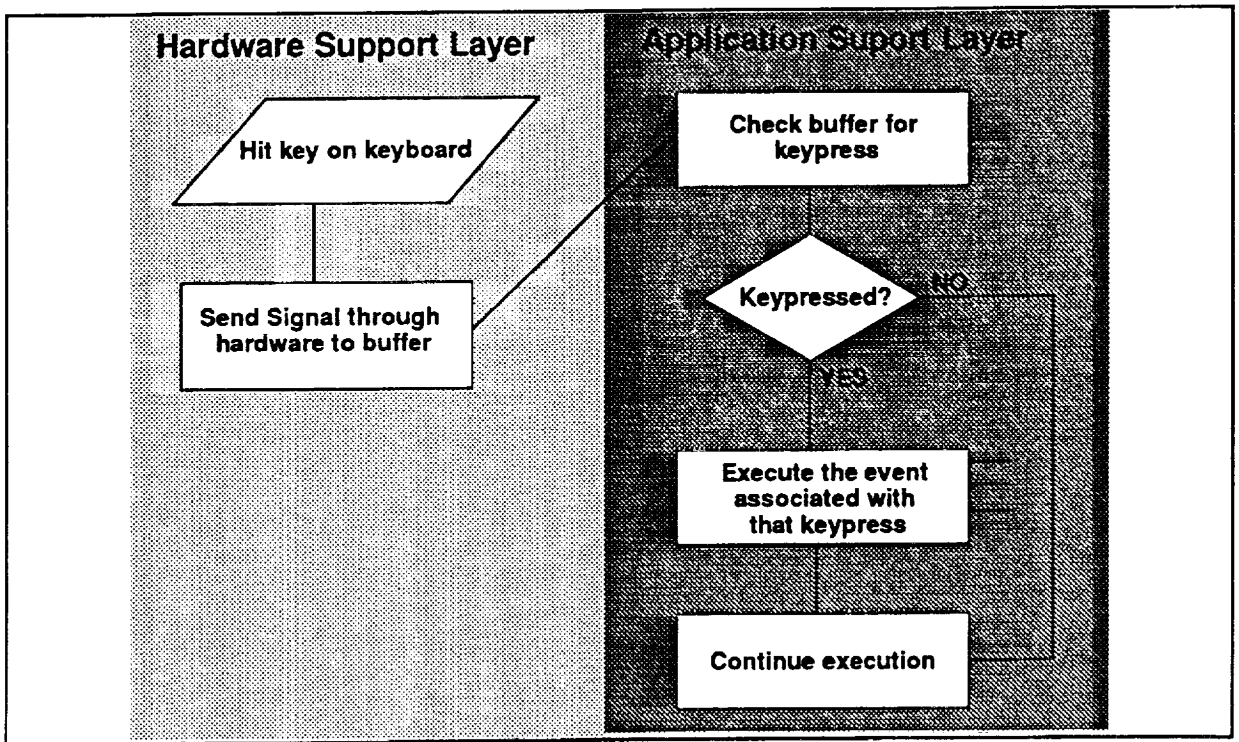
Figure 2 High-level data flow diagram of the ASL functionality of one of the systems studied.

4

Further breakdown of the processes helped to define the specific functions performed, the role of these functions, and the existence of hardware and system dependencies in the ASL software. For instance, the process in Figure 2 called Perform ASL Utilities and Services includes sub-functions such as: Manage Errors, Perform Execution Management, Manage Report Requests, Perform Platform Services, and Handle Interprocess Communication. Note, the last two sub-functions in the previous list are examples of hardware and software dependent activities. Though complete DFDs were not created for each of the processes described by each of the ASL systems studied, select functions were analyzed in greater detail to clarify the data and specific operations involved.

As with many analysis efforts, the most familiar functions proved to be the most difficult to accurately model. State transition diagrams and flow charts were used occasionally to focus the analysis team on actual processing activities and data manipulation details, instead of letting the team rely on sweeping assumptions. In some cases, functions were reviewed at a level of detail much finer than would be captured in the final object-oriented model in order to avoid missing important functionality.

Figure 3 shows an example flow chart of part of the analysis team's discussion on how the ASL software provides the interface between the system user and the



Hardware Support Layer
Application Suport Layer
Hit key on keyboard
Send Signal through hardware to buffer
Check buffer for keypress
Keypressed?
NO
YES
Execute the event associated with that keypress
Continue execution

Figure 3 A partial flow chart representing the level of detail discussed for some of the components of the Application Support Layer domain.

machine. Though details on keyboard use do not define software, they did provide some insight as to the specific software objects involved such as text, lines and shapes. The group reviewed the physical activities (i.e. pushing a function key) to pinpoint the associated software (i.e. the ASL commands that perform the specific data manipulation). By exploring the operations performed by the ASL software, the data objects in this hidden layer of software were identified which assisted in providing a more complete picture for the final object-oriented models.

Moving from a functional model to a object model can result in losing important information. Therefore, in an effort to minimize the impact, a third technique called a functional hierarchical grouping model was applied. This home-grown technique is the fusion between a functional model and an object model. The technique consists of putting the identified functions of the system in a hierarchical model and then grouping the lowest level functions together based on the objects being manipulated. For example, in Figure 4 below, all functions involving the human interaction with the computer system were grouped together to form the basis of an object oriented user machine model.

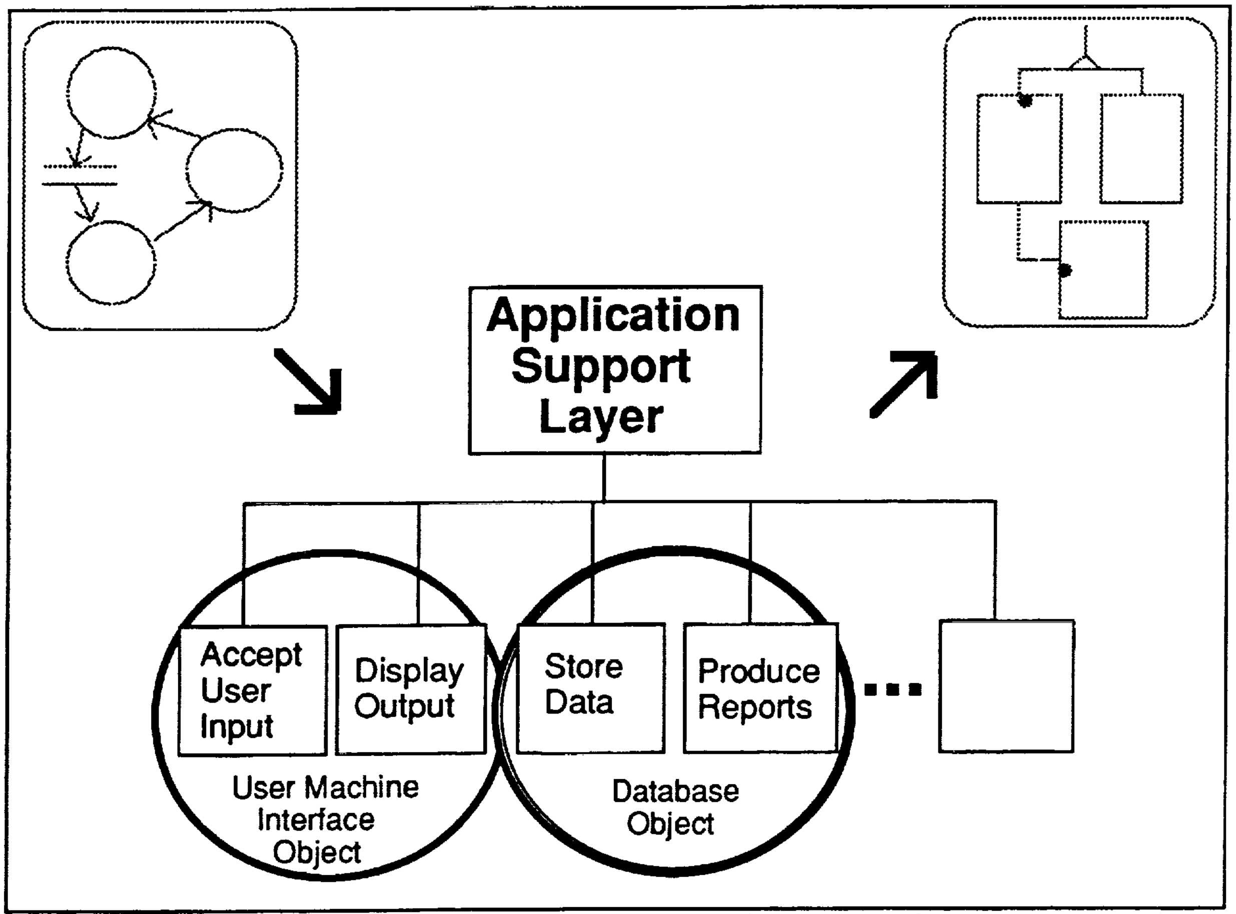


FIGURE 4 *Starting with functional modeling techniques and moving toward object oriented techniques, a fusion of methods occurred.*

6

The first objective in producing the hierarchical model was to list all of the functions potentially performed by an ASL. The word *potential* is used because the interfaces to the ASL also, needed to be defined. Therefore, in this case, too much high-level information is actually helpful. As the top-level processes were broken down into less complex sub-processes, the specific functionality of an ASL became apparent. Activities performed by the application layer or the hardware support layer were removed from the hierarchical model. For instance, one of the analysis team's first hierarchical models included all of the components shown in Figure 5 below, but through a series of iterations several of the components were determined not to be a required part by the typical ASL. Some of the components were hardware support layer activities, like the network functions and some of the components were found to be embedded in other components. The Help functions are an example of this; that is, most of the time, software modules contain their own help files, since Help is so application dependent. In addition, some components were raised in importance based on further analysis. For instance, the Kernel Support sub-function Platform Abstraction in the hierarchical model shown in Figure 5 became a primary area of focus in the final object-oriented model.

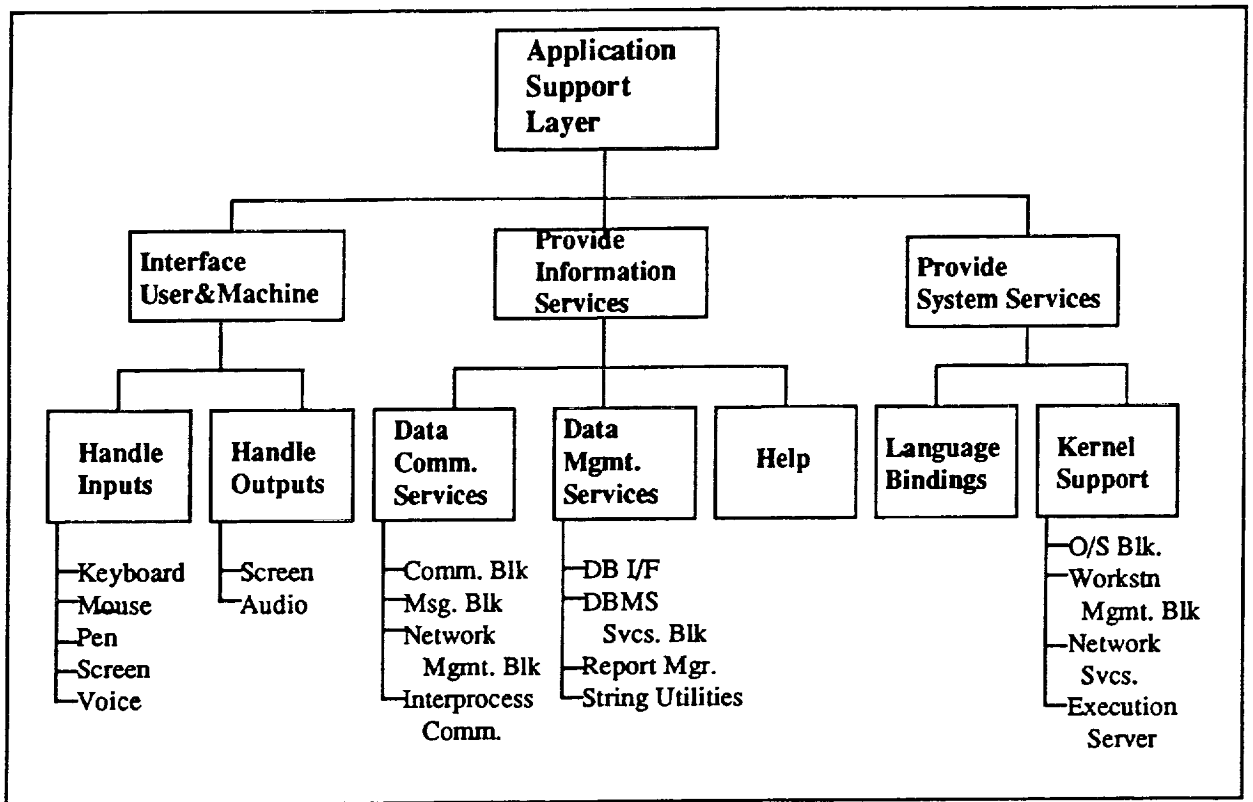


Figure 5 A sample of a draft hierarchical model for the Application Support Layer domain.

One facet of understanding a system that the functional and hierarchical diagrams did not illustrate very well was the commonalities across the different ASL subsystems. This aspect of the system was depicted more accurately by using object-oriented models. Object-oriented models pull all occurrences of the same data-type together, grouping all attributes and operations. Details focus on the data instead of on the functions. This permits code to be written with emphasis on the data being generic or abstract. This data abstraction increases the reusability of the software components - requirements architectures, design models, and code.

Three high-level object-oriented models created during this quick domain analysis were the focus of reuse opportunities identification. These models were the Data Base Management Systems model, the Platform Services model, and the User-Machine Interface model. A simplified version of the Platform Services Object model appears in Figure 6. Note, all data attribute and operation information has been removed in this version of the figure to improve the readability of the model.

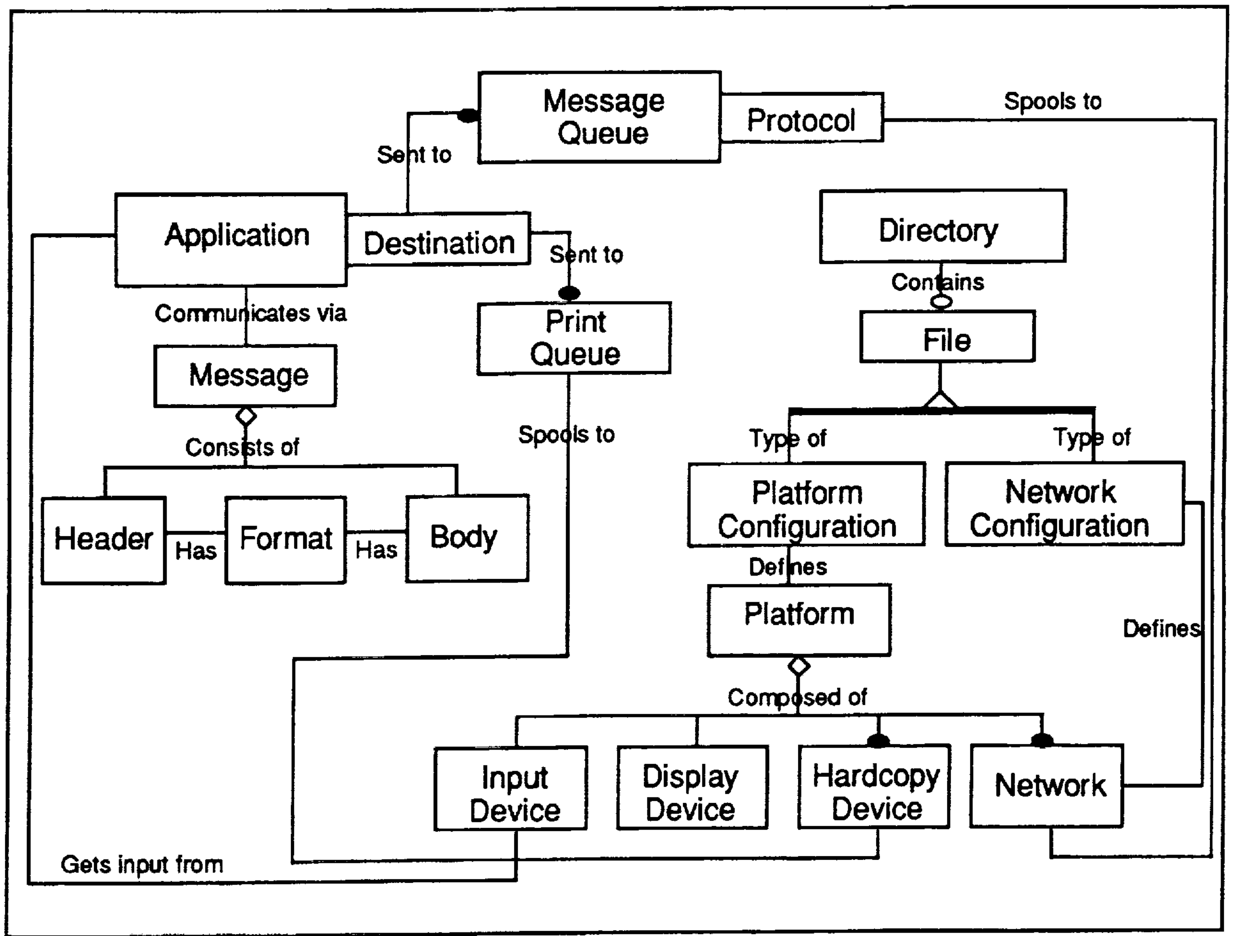


Figure 6 This is a simplified version of the Platform Services Object Model which was used to identify the software's basic functionality.

8

Unlike traditional domain analysis efforts, the primary objective in developing these domain models was not to explicitly define all the details of each of the primitive functions in the domain. Instead, this effort tried to provide an overview of the data relationships and basic interactions. By determining the general data manipulations of a typical ASL, the categories of components which are critical to the functionality of this domain were pinpointed. For example, as shown in Figure 6, specifics of the network were not needed, but understanding the relationship of the network configuration with the rest of the platform configuration proved very useful. The interaction of the ASL and the hardware support layer provided the distinction between potentially reusable software components and those hardware dependent components which require code, for instance, to be system unique.

Full lists of the data attributes and operations were developed for each object in each model. This permitted each object to be treated as a black box; that is, no further breakdown into sub-objects was necessary to expose software functionality. One case of this occurred with the object Platform. One of the operations associated with the object Platform is Enable/Disable Security. This single operation highlighted the importance current software development efforts place on security functions. Security functions are embedded throughout many software products, at multiple software layers. Though the Platform Services Object model does not provide further details of security functions, the high demand that software developers have placed on this category of software was not lost by this analysis team. Security functions were considered as prime reusable component candidates.

Besides providing a visual representation of the domain to assist in reuse opportunity identification, this process of integrating multiple modeling techniques offers an additional benefit. Though the faceted domain analysis approach described by Prieto-Diaz could have been performed to identify reuse opportunities, no product would have been available for future reusable component development. Typically, domain analysis is considered to be divided into two types:

o	Consumer-oriented	associated with reuse opportunities identification, and
o	Producer-oriented	associated with the creation of reusable components [Moore-Bailin].

However, the object-oriented models and their supporting documentation produced by the procedure described in this paper can be used as a basis for reusable requirements models. All of the high-level information on the domain is available in these models and many of the domain component details can gleaned from the analysis process documentation. Therefore, the final object-oriented models produced by this process not only meet current needs, but also some of those for future reuse planning.

Model and Reuse Opportunities Identification

The purpose of performing reuse opportunity identification is to facilitate reuse within one or among several system development efforts. During reuse opportunity identification, systems are evaluated and selected as candidates for reusing software components in their development life-cycle (client systems) and/or for developing and providing reusable software components to support the software development life-cycle of other systems (donor systems). Each potential client and donor system's schedule, language and functionality are studied. This information, along with data on the organization's policies, reuse knowledge and experiences, reuse training, and any other information that might facilitate or limit reuse is researched.

A system's schedule together with the high demand categories (HDCs) of components included in a system are the most crucial pieces of information needed when trying to coordinate reuse between compatible systems. HDCs are classifications of software components that are defined as being a necessity or requirement of all the systems that are in a particular domain. The HDCs are chosen by domain engineers using the generic architectures and domain models resulting from domain analysis. HDCs may be either functional or object-oriented in nature. For this reason, the study of the system's functionality, as well as, the data or object-oriented aspects of the each system involved in the reuse opportunities analysis is important.

This need of both functional and object-oriented views is where fusing modeling techniques proved to be very beneficial. For example, the HDCs that evolved from the domain analysis effort on the application support layer included process network messages, manage data dictionary, user machine interface, and database management system.

Once the analysis team had established the high demand categories from the domain models, we had a basis from which to identify reuse opportunities. We then took the potential client and donor system's schedules and identified which systems would be the clients and which systems would be the donors. This schedule coordination is critical to performing successful reuse opportunity analysis. The goal in this type of analysis is to begin identifying the client-donor relationships as early as possible in the client system's software development life-cycle (i.e. before requirements analysis, if possible). This permits the software reuse to be planned into the client systems' development schedule and thus, the largest cost benefits can be realized.

For systems which have similar software development schedules, if a client-donor relationship is established early enough the systems can perform requirement analysis or design development as a team. Then, one system could be chosen to write the reusable code and donate it to the other. Or the systems could split the code development effort and swap the highly reusable pieces before system integration testing.

After finding compatible systems according to schedule restrictions, the analysis team took the products produced from the conceptual phase and/or the requirements from the client system (depending on where the system was in the life cycle) and matched them to the requirements and design of the donor system. The HDCs and the generic models also guided this matching process by helping the analysts determine what was reusable and what was application specific. Since the generic domain models produced represent what is common (or reusable) among all systems in the ASL domain the analysts using the models were able to quickly identify potential opportunities for opportunistic and systematic reuse.

Summary

The initial use of this fusion of modeling techniques resulted in the development of a complete, understandable high-level object-oriented ASL domain model. Since that time, the technique has been applied successfully to the analysis efforts of other vertical domains including the personnel and budget domains. In most of these efforts, this fused modeling technique was employed to permit a very fast high-level domain analysis for the purpose of reuse opportunities identification. Since traditional domain analysis can take several person years per domain, this quick process (measured in terms of person months, not years) proved to be substantially cost effective.

However, our experience indicates that using multiple types of modeling techniques closely linked together should enhance traditional domain and system analysis efforts in general. Multiple views of a software modules functionality permits easier identification of reuse opportunities, quickly locates inconsistencies in system design, and encourages the development of more complete, reliable software products.

Ms. Susan Main Hall is a Systems Consultant, Management, for SofTech, Incorporated. She directs a technical group which supports the Army Reuse Center through domain analysis, reuse requirements analysis, reuse opportunities identification, library donor component selection, and quality assurance of reusable software components. Additionally, Ms. Hall has over eight years experience in supporting DoD Ada technical development efforts. She has participated in independent verification and validation, modeling, and development. Ms Hall holds a Bachelors of Science degree in Computer Science and a Masters of Science degree in Computer Science with Software Engineering concentration from George Mason University.

Ms. Eileen M. McGuire is an Associate Software Engineer for SofTech, Incorporated. She preforms domain analysis, reuse requirements analysis, reuse opportunities identification, and library donor component selection. Ms. McGuire holds a Bachelors of Science degree in Management Science (Computer Based Decision Support Systems Option) from Virginia Polytechnic Institute and State University.

References

Blaha, Michael, "Models of Models," September 1991

Caldiera, G. and V. R. Basili, "Identifying and Qualifying Reusable Software Components," IEEE Computer, Vol. 24, No. 2, Feb 1991, pp. 61-70

Coad, and Yourdon, **Object-Oriented Analysis**
Englewook Cliffs, NJ: Yourdon Press/Prentice Hall, 1991

Coleman, Derek, Fiona Hayes and Stephen Bear, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," IEEE Transactions on Software Engineering, Vol. 18, No. 1, January 1992

Domain Analysis Guidelines, Draft, SofTech, Inc., May 1992

DeMarco, T., **Structured Analysis and System Specification.** Englewook Cliffs, NJ: Yourdon Press/Prentice Hall, 1978

Fiscal Year 1994 Reuse Opportunities Report, Final, SofTech, Inc., July, 30, 1993

Gomaa, H., L. Kerschberg, C. Bosch, V. Sugumaran and I. Tavakoli, "A Prototype Software Engineering Environment for Domain Modeling and Reuse," 1991

Iscoe, Neil, "Reuse - A Knowledge Based Approach," NASA Software Engineering

Workshop Proceedings, December 1992

Jacobson, Ivar and Frederik Lindstrom, "Re-engineering of Old Systems to an Object-Oriented Architecture," OOPSLA'91

Lubars, Mitchell D., "Domain Analysis and Domain Engineering in IDeA," IEEE 1991

McGarry, Frank, "Lessons Learned", NASA Software Engineering Workshop Proceedings, December 1992

Moore, John M. and Sidney C. Bailin, "Domain Analysis: Framework for Reuse Technical Report", Computer Technology Associates, Rockville, MD, 193

Patel, Sukesh, William Chu, Rich Baxter, Brian Sayrs and Steve Sherman, "A Top-Down Software Reuse Support Environment," 1992

Prieto-Diaz, Ruben, "Domain Analysis: An Introduction," Software Engineering Notes, Vol. 15, No. 2, April 1990

Prieto-Diaz, Ruben, "Domain Analysis for Reusability," Proceedings of COMPSAC '87, pp. 23-29

Rumbaugh, James, Michael Blaha, Wiliam Premerlani, Frederick Eddy and William Lorensen, **Object-Oriented Modeling and Design**, Prentice-Hall, Inc., 1991

Shumate, Ken, "BATCES Solution #1" an Object-Oriented Design from Functional Requirements Analysis," ACM Ada Letters, Nov/Dec 1993, Vol. XIII, Number 6, pp. 133-161

Tracz, Will, "Domain Analysis Working Group Report - First International Workshop on Software Reusability," Software Engineering Notes, Vol. 17 No. 3, July 1992

Wessale, Bill, "Large Project Experience with Object Oriented Methods and Reuse," NASA Software Engineering Workshop Proceedings, December 1992

Fusing Modeling Techniques to Support Domain Analysis for Reuse Opportunities Identification

by
Susan Main Hall
Eileen McGuire

SOFTECH, INC.

Project Background

(Army Reuse Center, Software Development Center-Washington)

Needed quick methodology to:

- Perform Domain Analysis
  - Application Support Layer Services Domain
  - Four Army systems currently under development
  - Systems and analysts were functionally oriented

- Focus on Identification of Reuse Opportunities
  - Object-oriented models make this process easier

SOFTECH, INC.

Problem Statement

Which provide a clearer understanding of high -level system requirements:

Functional Models

or

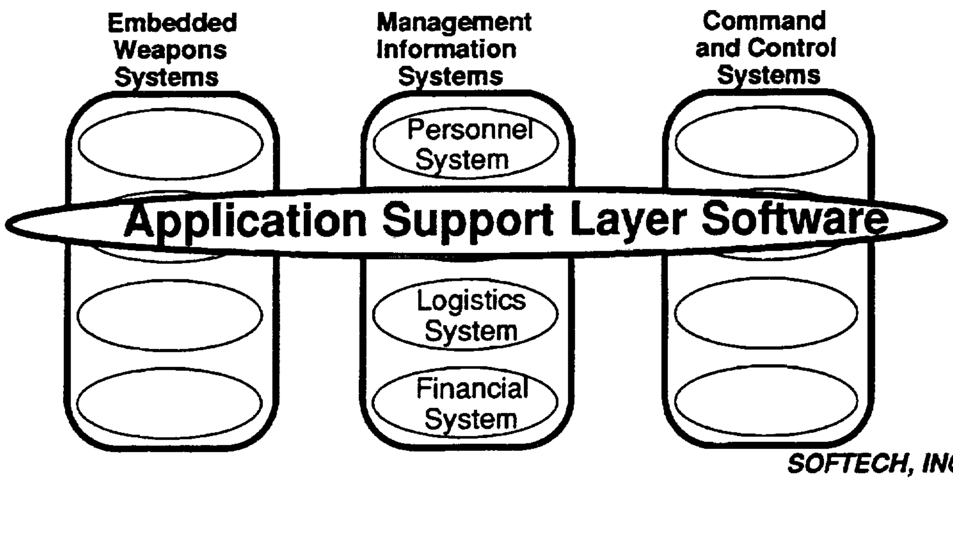
Object-Oriented Graphical Representations

SOLUTION: Use BOTH!

SOFTECH, INC.

Domain Analysis

Process of identifying commonalities in a class of similar systems


Embedded Weapons Systems
Management Information Systems
Command and Control Systems
Personnel System
Application Support Layer Software
Logistics System
Financial System

SOFTECH, INC.

Fusing Modeling Techniques

Since one modeling technique could not capture the domain analysis completely, we:

- Started with FUNCTIONAL models
- Moved to a functional HIERARCHICAL GROUPING model
- Transitioned into a set of OBJECT-ORIENTED models

SOFTECH, INC.

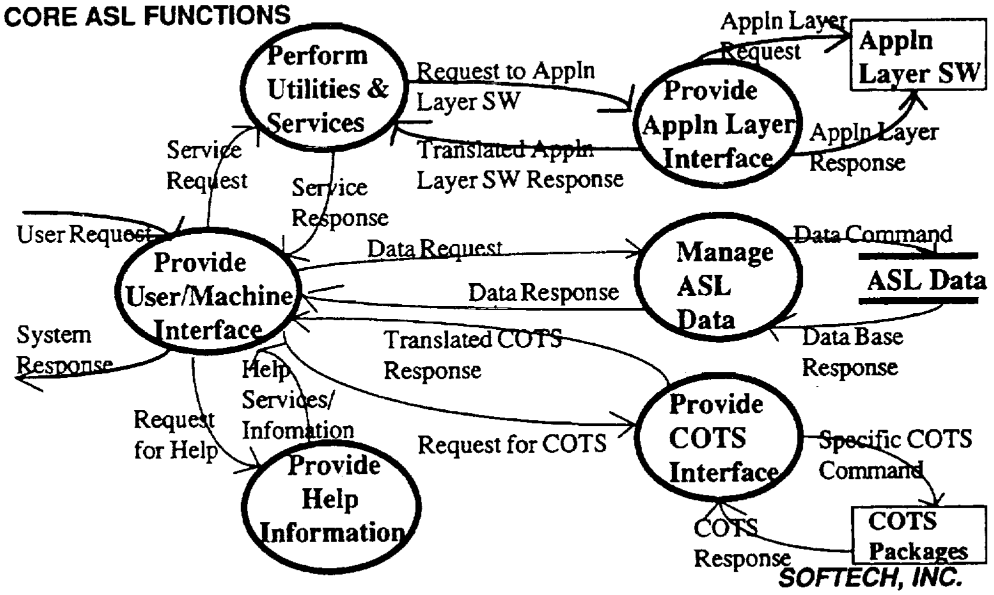
Functional Models

- Began by reviewing existing and creating new:
  - data flow diagrams
  - state transition diagrams
  - flow charts
- Captured basic activities performed by the actual Application Support Layer code module being studied
- Overlaid each system on top of one another to highlight commonalities and differences

SOFTECH, INC.

Functional Models (continued)

CORE ASL FUNCTIONS



SOFTECH, INC.

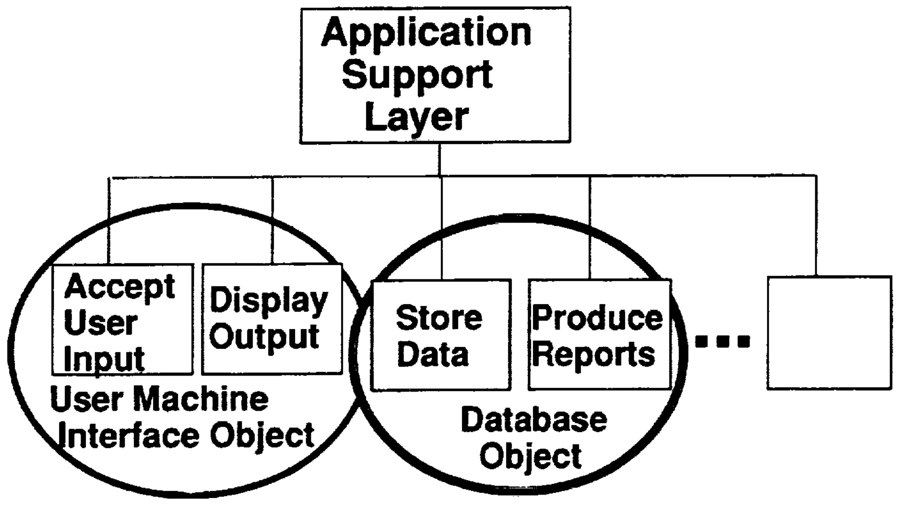
Heirarchical Models

- Moving from a functional to an object model can cause important information to be lost.
- So a home-grown technique was applied.
- This heirarchical technique consists of:
  - listing the identified functions in a heirarchical tree
  - grouping the lowest level functions together based on the objects manipulated
  - dividing functions into those IN the domain and those INTERFACING WITH the domain.

SOFTECH, INC.

Heirarchical Models (continued)

SAMPLE OF HEIRARCHICAL GROUPING TECHNIQUE



SOFTECH, INC.

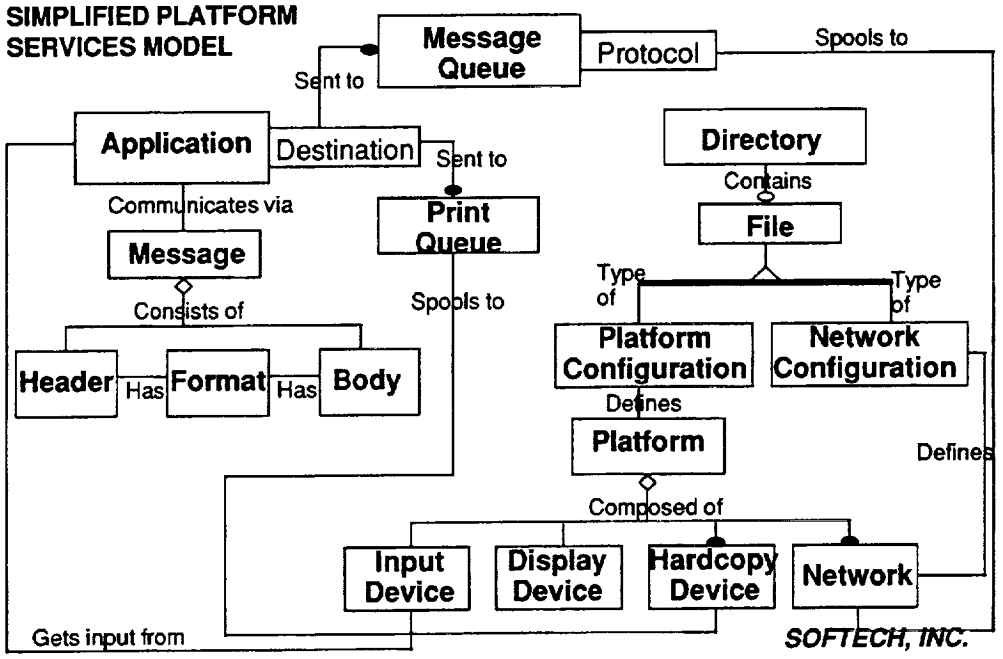
Object-Oriented Models

- Used to illustrate commonalities across Application Support Layer sub-systems
- Grouped occurrences of same data-type together
- Captured data attributes
- Assigned functions to data
- Provided level of data abstraction to increase reusability

SOFTECH, INC.

Object Oriented Models (continued)

SIMPLIFIED PLATFORM SERVICES MODEL



SOFTECH, INC.

Reuse Opportunities Identification

Purpose: to facilitate reuse within one or among several softare development efforts.

- During reuse opportunities identification, systems are evaluated and selected as candidates to:
  - reuse software components in their software development life-cycle (clients)

  AND/OR
  - provide reusable software components to support the software development life-cycle of other systems

SOFTECH, INC.

Models Assisted ROI

- Application Specific/System Unique components were stripped away, using the functional and heirarchical models.
- High Demand Categories were established, using the domain models
  - Functional
  - Data-Oriented.
- Reusable software components were identified, factoring in development schedules
  - Requirements Architectures,
  - Design Models,
  - Code Modules.

SOFTECH, INC.

Summary

- Multiple views illustrate a domain more clearly than a single modeling approach
- This fusion of modeling techniques approach:
  - identified more substantial reuse opportunity candidates,
  - completed more quickly than traditional domain analysis, and
  - provided a basis for future developement of a reusable domain model.

SOFT