REPRINTED FROM OBJECT magazine

Developing a medical diagnostic system with OOA/RD

The question for objects is no longer "Why?" but "How?"

Jim Ladd

TO REALIZE THE full benefits of objects, the object paradigm must extend beyond the programming phase of the software development process. While the spotlight of the software community has been directed at object-oriented programming (OOP), the more important issue of object orientation as a development philosophy remained on the collective periphery. As object technology gains popularity, this issue is forced to become the focal point of the community. The question for objects is no longer Why? but How?

BACKGROUND

Different people develop software differently, yet a common thread is woven through all software development activities. From the solo hobbyist coding to the multiperson teams building the next generation securities trading system, all will experience an understanding of the problem, a selection of a solution, and a creation of that solution.

This process is also known as analysis, design, and implementation:

- **Analysis is the study of a problem.** The analysis of a problem concentrates on what must be done and not how it is to be accomplished. The result of the analysis efforts is a set of descriptions of the problem.
- **Design is the selection of a particular solution for a given problem.** A solution specifies exactly how the problem is to be solved. The results of the design work are specifications and policies that guide the construction of the solution.
- **Implementation is the creation of the chosen solution for the problem.** During the implementation phase, the source code is produced according to the design specifications and guidelines.

Although all software constructors (individual and organizational) perform analysis, design, and implementation, far fewer use formal methods, techniques, and tools to do so. The question should not be whether such things are advantageous. The question is how to forge the methods, techniques, and tools into an effective and efficient software development paradigm.

Jim Ladd is Software Manager at Abbott Laboratories, Irving TX.

YET ANOTHER BLOOD ANALYZER—YABA

One of our more recent projects to follow the Shlaer-Mellor method is one I will refer to as the YABA (yet another blood analyzer). The YABA is a medical diagnostic instrument for determining the characteristics of certain components of white blood cells. The goal of the YABA is to automate the steps of the testing protocol.

The testing protocol is a controlled series of operations performed on the cells. Chemicals are added to the blood sample and the mixture is incubated. Depending on the type of test, additional mixing and incubation cycles may be performed. When the chemical reactions are completed, the sample is exposed to a detection system that gathers raw data. Finally, this information is processed and reduced to a set of final results.

The automation of the testing procedure is accomplished through the intelligent coordination of electro-mechanical devices. The major components of the YABA include a fluidics system for liquid chemical management and distribution, an incubator, an image processing module for assay detection, and two 3-axis robotic platforms for transporting the samples within the instrument. Small electric motors and solenoids actuate the mechanisms of the components.

The requirements for the YABA extend beyond the real-time control of the instrument. The YABA provides the user with test protocol configuration, sample and patient data management, results analysis, report generation, and diagnostics capabilities.

Analysis

The first step of the analysis was to partition the problem into domains. The latest version of the YABA domain chart is shown in Figure 1. At the top of the chart is the application domain that was the most specific to the project. After the initial analysis was completed, the application domain was divided into two subsystems. The instrument subsystem was concerned with the processing of the sample and contained objects of the instrument such as the Input Load Station, the Pipettor, and the Reader Robot. The other application subsystem contained the objects that were of more interest to the user. Objects within this subsystem included the Patient, Results Report, and Test List. A few objects, such as Sample and Test, were shared by the two subsystems.

The application domain required the services of other domains. These domains consisted of the subject matter that was less specific than the application domain. The service domains of the YABA include the following:

- **Scheduler**: This domain reserves the instrument's components according to the resource and timing requirements of the test protocols. It calculates and maintains the timelines for the different resources (pipettor, optics system, etc.) within the instrument. Objects for this domain include the Protocol, the Activity, and the Resource objects.

- **Command sequence:** Once a Protocol reserves a Resource object, that Resource must perform an Activity at the scheduled time. The Activity object is a sequence of commands. This domain parses the Activity into the underlying commands and monitors their execution. The Aspirate Command, Level Sense Command, and Get Tray Command are a few of the objects of this domain.
- **Motor controller:** The Motor Controller domain accepts, queues, and executes motor-specific instructions for the custom-built motor controller. This domain includes the Move Command and the Stop Command objects.
- **Image processing:** The Image Processing domain performs the operations needed to acquire, manipulate, and analyze images. Object-oriented analysis was not used for this domain because the algorithms were "discovered" through theoretical and experimental work and modeled with standard flow charts.
- **Process I/O:** This domain interfaces with the outside world. It is concerned with interface entities such as hardware interrupts and I/O ports.
- **User interface:** This domain deals with the human interface screens and the corresponding user interaction. The Patient Demographics Screen and the Test Ordering Screen are two objects within this domain.
- **GUI:** The GUI domain provides the window management and display functions. The GUI is a commercial package and provides the functions common to products of this type.

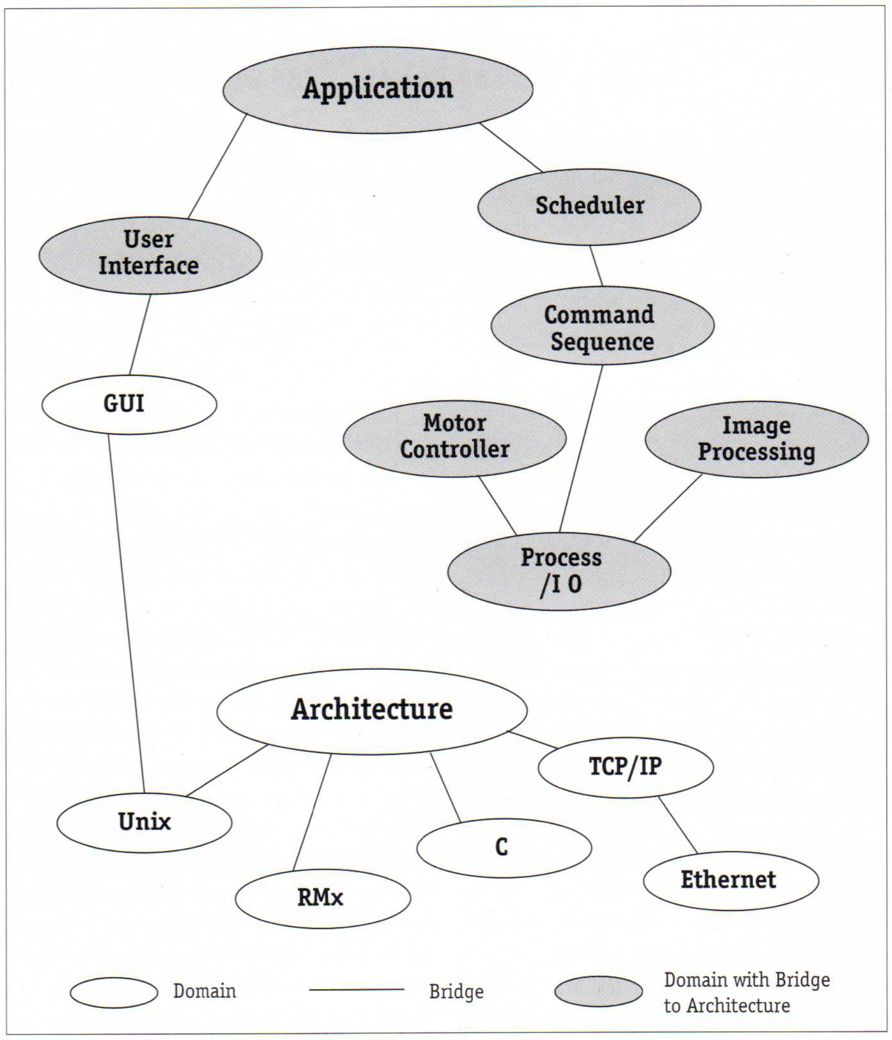


Figure 1. YABA domain chart.

The architecture domain is concerned with the organization of data and control of the software. It was specified during the design phase.

The domains at the lowest level of the chart involve the implementation entities. These domains include the operating systems and programming language. The network interface was a commercially available package and did not require analysis. These domains were also selected during the design phase.

A benefit of domain analysis is the potential for reuse of work. The lower domains have the greatest potential. Obviously, the operating system and programming language domains are extremely reusable. The service domains also provide opportunities for reuse. For example, the objects of the motor controller domain are "unaware" of the YABA system; they operate in a contained world. The analysis models could be easily applied to other projects requiring the service of a motor controller. The application domain offers the least reusability due to its project specificity. However, objects within this domain would provide an excellent starting point for the application domain of a similar project.

After the domains were identified, the analysis within the domains began. A small group (>5) of analysts (in our case, engineers with their analysis hats on) would meet to define a domain's mission statement and brainstorm for obvious objects. At the conclusion of the initial meeting, a single analyst was assigned responsibility for the domain. Over time, if the subject matter of a domain was too much for the solo analyst, the domain was partitioned into subsystems. The boundary of the individual was reduced from the domain to the subsystem level. Review meetings were used to update the other analysts on the progress of the models.

The actual act of analysis can be imposing even at a subsystem level. Several techniques were adopted by the software group to simplify the task. One technique was to model the normal behavior of the objects in the initial analysis. The required failure analysis and error handling were modeled after the normal behavior of the objects became stable.

A major doctrine of the Shlaer-Mellor method is that the analysis models should be exact and complete descriptions of the problem. By embracing this philosophy, the analysis efforts on the YABA drove the discovery process toward a greater understanding of the problem. If the information was not available to complete a model, it was sought. In the worst of cases, assumptions were made and thoroughly documented. The analysts were forced to address the unresolved issues and come to some convergence.

Design

One of the first design decisions was the separation of the real-time control from the human interface functions. These two areas have very distinct objectives, requirements, and constraints. Because the operating systems, utilities, and other programs for one area do not adequately address the needs of the other, an architecture with two complete computing environments was chosen. An off-the-shelf personal computer was selected for the human interface and configured with the standard accessories. UNIX was chosen for the operating system on the human interface computer.

If the present rate of improvement in the CASE arena can be maintained, the future of software engineering automation should be very exciting and rewarding.

A passive-backplane industry-standard architecture (ISA) computer was selected to execute the control software. The control computer contained a motor controller board, image processing board, and input/output board. The control computer would run Intel's iRMX real-time operating system. An Ethernet connection with the TCP/IP protocols provided the communications channel between the two computers.

The C programming language was chosen over C++ because, at the time of the decision, our group had (1) no experience with C++, (2) little confidence in the C++ compilers for the embedded processors, (3) a strong desire for homogeneity across the processor boundaries. Additional work was necessary to integrate C into the Shlaer-Mellor method, but the alternatives were less attractive.

A set of rules was generated that specified the policies and guidelines for the software construction. Included in the rules were the object distribution among the CPUs, data and control organization, and system initialization procedures. The design specification also included templates for the source and header files. These files consisted of formats for the function prototypes, state transition tables, and initialization function. The templates would be completed during the implementation phase.

Another task of the design phase was the construction of the software architecture. The architecture is the mechanism in which objects communicate and events are processed. For the YABA project, an asynchronous state-oriented architecture was designated for each microprocessor in the system. This architecture is basically an engine that manages a state transition table for every instance of every object in the processor space. The engine is very adaptable to different computing environments. It was modified to work with the UNIX and iRMX operating systems. The architecture was also adapted to the motor controller microprocessor, which had no operating system.

Implementation

In the implementation phase, programmers transformed the analysis models into source code according to the design rules. Deviations from the design rules were not allowed. If a question or issue surfaced, it was addressed by modifying the design rules or clarifying the analysis models.

The code generation was very straightforward and mechanical. The source and header templates were completed for each object using information from the analysis models. Each process of the process model became a function in the C programming language. The task of code generation was extremely scalable. The time required for coding was dependent on the number of analysis models and the amount of programming resources available.

On previous projects, there has been much debate on the type and amount of comments to be place in the source code. This argument was finally settled on the YABA project. No comments were to be in the source code. The valuable information is in the analysis models and design rules. The source code is now relegated to the same level as assembly and machine code.

Debugging

The software produced with this process encourages debugging strategies that differ from those used with traditional software. If a problem arose during program execution, first the source code was compared with the analysis models. If a discrepancy was found, the error was fixed and debugging continued. If no deviations were discovered, the analysis models were inspected for errors. Typically, the events of the object were manually traced and the corresponding transitions of the objects observed.

For very evasive bugs, manual event tracing through the models proved difficult and alternative approaches were examined. One approach was to use the executable code as a debugging platform even though the system might be incomplete. With object-based software, attention can be directed to the behavior of a single object or the objects within a domain with minimal effort. Only the events that cause the objects to transition must be provided for this activity; it does not require involvement of the entire system.

CASE Tools

The fundamental concepts of the Shlaer-Mellor method encourage a high level of automation. Unfortunately, potential is one issue and realization is another. When the YABA project was in development, few CASE tools supported OOA/RD. The products that did were somewhat overly optimistic in their claims. Still, while the early tools were limited in features and were at times difficult to use, the software group continuously embraced CASE technologies because (1) while imperfect, the available CASE support added value to the process, and (2) all advancements provided by the tools would improve the analysis and design efforts of the group.

Recently, after a number of years of inactivity, the CASE vendors are now taking notice of the potential for automating the OOA/RD processes. The field is becoming populated with tools offering automation in different parts of the process. Simula-

SHLAER-MELLOR METHOD

Although it is best known for its approach to object-oriented analysis, the Shlaer-Mellor method defines a complete paradigm for the analysis, design, and implementation of software systems. The method combines Object-Oriented Analysis (OOA) and Recursive Design (RD) techniques in a comprehensive approach for software development. OOA describes the notation, models, and procedures necessary to discover and model objects in the problem space. RD provides the transition policies and guidelines from the analysis to implementation. The Shlaer-Mellor method also includes the process for applying OOA and RD to a system.

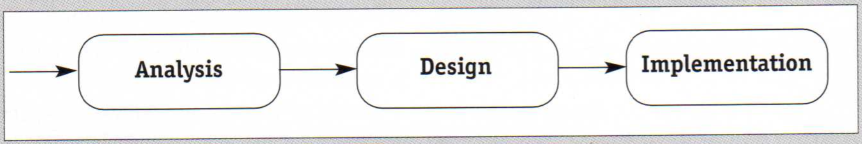
The OOA method uses three primary models to describe the problem space:

- Information models
- State models
- Process models

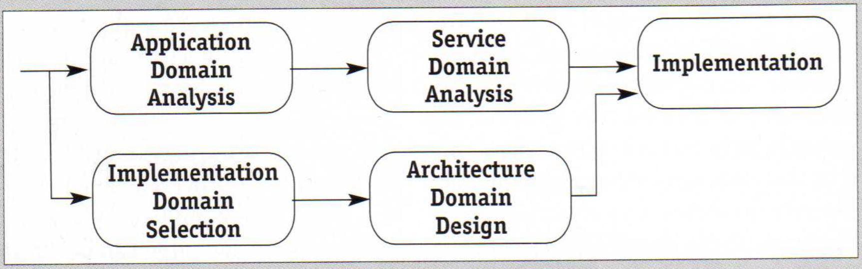
Integrated together in a comprehensive method, the models produced represent a complete and consistent description of the problem. The models are so complete that they are, in fact, executable and offer the opportunity to verify the correctness of the analysis.

The cornerstone of recursive design is the design-by-translation concept. The analysis models are translated directly into code by applying design rules. The design phase focuses on the selection and creation of the rules and not on another set of models. The design rules guide the construction of the solution. During the implementation phase, the rules are applied to the analysis models to generate the source code.

The Shlaer-Mellor development process relies heavily on the concept of domains and bridges. Domains consist of specific subject matter. In the object world, domains are composed of closely related objects. Bridges are the connections between the domains. Recursive design defines four types of domains. The application domain consist of the material that concerns the user. Service domains provide generic functions. The architecture domain dictates the management of data and control of the system. The implementation domains are the programming languages and operating systems.



Traditional process.



Application Domain Analysis
Service Domain Analysis
Implementation
Implementation Domain Selection
Architecture Domain Design

Shlaer-Mellor process.

tion and animation of the state models and analysis-to-code translation are just two of the critical technologies offered by the latest versions of CASE tools. The products are also improving in other, more standard ways including enhanced report generation, better static checking of the models, more intuitive human interfaces, and open databases. If the present rate of improvement in the CASE arena can be maintained, the future of software engineering automation should be very exciting and rewarding.

RECOMMENDATIONS

The paradigm shift to object-oriented technology can be a confusing and frustrating passage. When coupled with the shift to the Shlaer-Mellor method, the task can be overwhelming at times. To accelerate the learning process for OOA/RD, the following recommendations are offered:

1. Acquire formal training whenever possible. Learn the fundamentals of the technology from the masters.
2. Target a microproject (< 6 objects) and take it through the analysis, design, and implementation phases.
3. Document everything. The trail will be valuable in post-project analysis and studies.
4. Avoid the temptation to "just code it." Follow the process by completing the analysis models and design rules before starting the implementation.
5. Don't get discouraged. Rigorous analysis is difficult and requires effort and time to master.

SUMMARY

The results of our efforts with the Shlaer-Mellor analysis and design method were better documentation, improved understanding of the problem, increased communication among the team members, freedom from implementation technologies, and an increased potential of reusable work products. All team members were impressed by the OOA/RD methods and wanted to apply the paradigm to other projects.

The Shlaer-Mellor Method, however, is not for everyone. Those comfortable with traditional development practices might have difficulty adapting to the new paradigm. The shift from a code- or design-based process to an analysis-centric one can be as challenging as the shift to object technologies. To realize the potential benefits, the policies and guidelines of the process must be followed faithfully. For those who wish to migrate from software crafting to software manufacturing, the Shlaer-Mellor development paradigm is an excellent alternative.

For further information, please contact Project Technology at 1-510-845-1484.

PROJECT TECHNOLOGY

Shlaer-Mellor Method