Microsoft® Visio® 2000

Iterative Development Using the UML and Microsoft® Visio® 2000 Enterprise Edition

White Paper

Frank Sternberg, Ph.D.,
Integrated Software Specialists, Inc.

Table of Contents

Published: April 2000

Introduction

Information technology departments that develop high-quality software on a timely basis share common characteristics. One of these is the use of diagrams and models to visually enhance communication among team members, and between the development team and the user community. Communication of business and system requirements and design is more effective when diagrams and pictures augment written text. The Unified Modeling Language (UML) provides a common language for creating visual models of software requirements and architecture.

Most IT organizations are already familiar with Microsoft® Visio® 2000 software and its broad range of uses within an organization. With the release of Microsoft Visio 2000 Enterprise Edition, it has become much more practical and affordable to model an application before building or buying software.

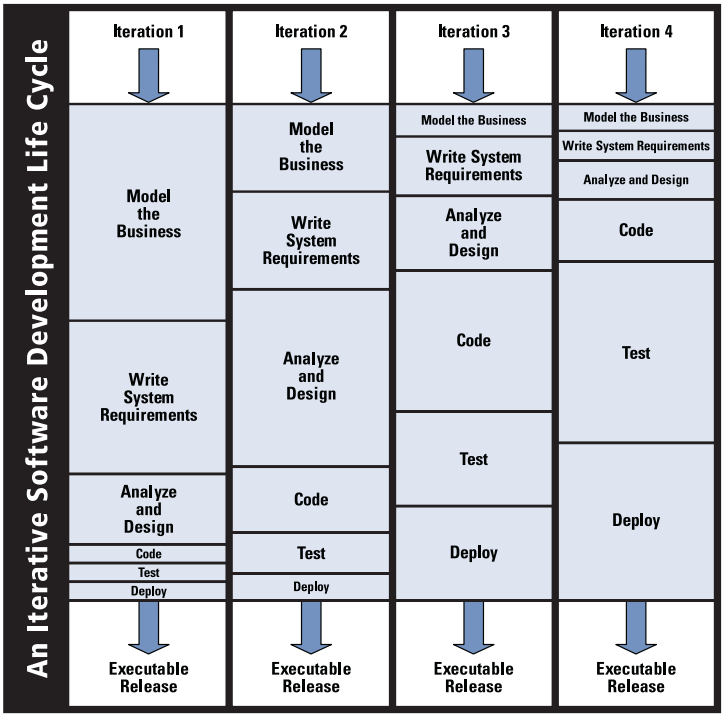
In this paper, we'll describe some of the ways to use Enterprise Edition to help you deliver higher quality and more maintainable software. We'll briefly discuss the value of iterative development, visual modeling, and the UML. Then, we'll walk you through a case study that explains the UML diagrams typically drawn during a software development project and how they are beneficial.

An Iterative, Incremental Development Life Cycle

An iterative process stands in contrast to a waterfall process. In a waterfall process, the entire system under development moves forward through the process' activities *as a whole*, from modeling the business and gathering requirements through analysis, design, coding, testing, and deployment. Users don't see anything running until the product is completed.

With an iterative and incremental process, the product is developed and delivered in stages, with each stage adding functionality. As you can see in the diagram below, the output of one iteration becomes the input to the next. The word 'iterative' means that the process' activities are repeated during each stage of the cycle. The word 'incremental' refers to the functionality added at each iteration.

Communicating through Diagrams



An Iterative Software Development Life Cycle. *Increasingly, developers are moving toward an iterative and incremental process. See the sidebar on page 3 for more information about the benefits of this trend.*

A picture is worth a thousand words. Microsoft Visio 2000 Enterprise Edition makes it easy for business and technical users to communicate more effectively using models. A *model* is a simplified visual representation of your business or the software needed to support your business. A model focuses on the most relevant entities and interactions while ignoring unnecessary details. For example, a *map* is a model of a piece of the earth's surface. A road map focuses on details needed to drive a car, while a contour map may focus on details needed for cross-country skiing.

Like a map, a *visual model* of a business or software facilitates communication. Users and developers can rely on the visual model with its series of diagrams and related text to effectively communicate how software should function. Since most businesses have processes that are used to their competitive

Benefits of an Iterative and Incremental Software Development Process

Deliverables meet business needs and user expectations. *Because functionality is added incrementally, users understanding of the system is more tangible, thereby enabling them to provide higher quality feedback for each subsequent iteration.*

Changing conditions during development can be accommodated in response to new or different requirements. *Because assessment and planning are routinely done between iterations, the process is designed to accommodate change.*

Higher quality software is produced. *The software is tested at the completion of each iteration, rather than all at once at the end. With each iteration, there s the option to improve or modify the internal architecture built in the previous iteration.*

Development projects are more likely to succeed. *High-risk features can be scheduled for early iterations and resolved quickly.*

advantage, these models also represent corporate assets. These models become the foundation upon which future business architecture and related software development projects should be constructed.

Because models drawn and stored in Enterprise Edition can be used for communications throughout the organization, they become even more valuable. The visual model (or any piece of it) can be easily published on a Web site for interested parties to browse or can be inserted into a Microsoft Word document. In this way, Enterprise Edition allows workers to efficiently discuss evolving business requirements and architectural decisions by referring to the same model. This speeds up the process of disseminating information and gaining consensus, thereby making the iterative software development process run more smoothly.

Benefits of communicating visually using models

Here are some specific ways that visual modeling facilitates communication.

By *visualizing* the system to be built:

- Developers and users can better agree on what should be designed and built.
- The system s architecture is made explicit and unambiguous.

By *specifying* the structure and behavior of the system:

- Programmers can write code that easily tracks changing business conditions.
- QA experts and testers can obtain their testing specifications from the model.
- A programmer can use tools to generate code from the model.

By *communicating* the intent of the system to various groups:

- A project manager can use the model to coordinate and synchronize the team and to bring new developers up-to-speed quickly.
- Users are kept involved through continually evaluating and contributing at each iterative step.
- Technical writers can build the documentation and online Help from the model.

By *documenting* the system and the decisions made:

- Programmers can allow the finished system to evolve during future maintenance efforts without violating the integrity of the system.
- Experts can achieve widespread systemic re-use (politics and culture willing).

Unified Modeling Language (UML) and Visio 2000 Enterprise Edition

Over the past few years, the industry has standardized on the UML as the visual language for modeling software systems. The UML is independent of tools and methodologies. The Object Management Group, a consortium of industry organizations, currently manages the UML.

The UML is used to create models in the form of diagrams. Enterprise Edition includes all of the diagram types in the UML, and thus supports the complete life cycle of a development project.

Although the UMLs strength is its ability to convey information *visually*, a UML model also stores critical *textual* information behind the elements of the diagrams.

Using Enterprise Edition, technical developers can print and customize reports that lay out the state of the model.

Enterprise Edition also allows you to import from, and export to, Microsoft Repository a fairly universal format for converting and sharing UML models.

Microsoft Visio 2000 Enterprise Edition also supports many other OO notations besides the UML such as the Booch and OMT notations. These are both precursors to the UML and are still widely used.

Scenario Showing Visio 2000 Enterprise Edition UML Diagrams

Let s explore the use of the UML through a fictitious scenario Lucy s Lemonade business. This scenario will depict how to approach a business problem iteratively using the UML and Microsoft Visio 2000 Enterprise Edition.

Luy s Lemonade is an emerging small business whose owner, Lucy, has a vision for her company. She has some stores and an established brand name. Much unnecessary work is being done manually, and Lucy knows that she needs to automate. She hopes to be selling products over the Web before long.

To increase the quality of management information, Lucy decides first to automate her time-tracking system. Employees will enter their daily work into a daily work log that captures, for each unit of work, the time spent and the task performed. The employee s manager must approve the daily work log. This captured information will be organized and fed into management reports and into payroll processing.

In our iterative process, we will keep repeating certain activities whose results we need to communicate. Some of the critical activities occupy the first column of Table 1. As the table shows, we ll describe the time-tracking scenario by showing and interpreting the UML diagrams drawn with Enterprise Edition during each activity.

Table 1. UML Diagrams used during major software development activities

Some Major Software Development Activities	Principal UML Diagrams Used	Shown in Diagram
Identify and document the business' processes (as is and to be)	Activity diagram	1
Define external requirements in terms of functionality delivered to kinds of users	Use case diagram	2
Capture the primary abstractions of the business (or other problem domain) and the relationships between these abstractions	Class diagram	3
Describe how objects interact to realize the functionality specified in the use case diagrams	Interaction diagrams • Collaboration diagram • Sequence diagram	 4a 4b
Carve up the system into smaller subsystems	Package diagram	5
Deploy components to computers	Deployment diagram	6

Communication is one of the great challenges in a project like Lucy s. How long the project takes, its probability of success, and the quality of the final product depend to a great degree on how well everyone involved communicates. While reading this scenario, please note how well the UML diagrams communicate the business requirements and the structure and behavior of the system.

Neither this list of activities, nor the list of UML diagrams in the table, is exhaustive.

Each of the following sections corresponds to one of the activities in Table 1. Each section contains:

An overview of the activity

Reference to the corresponding UML diagram

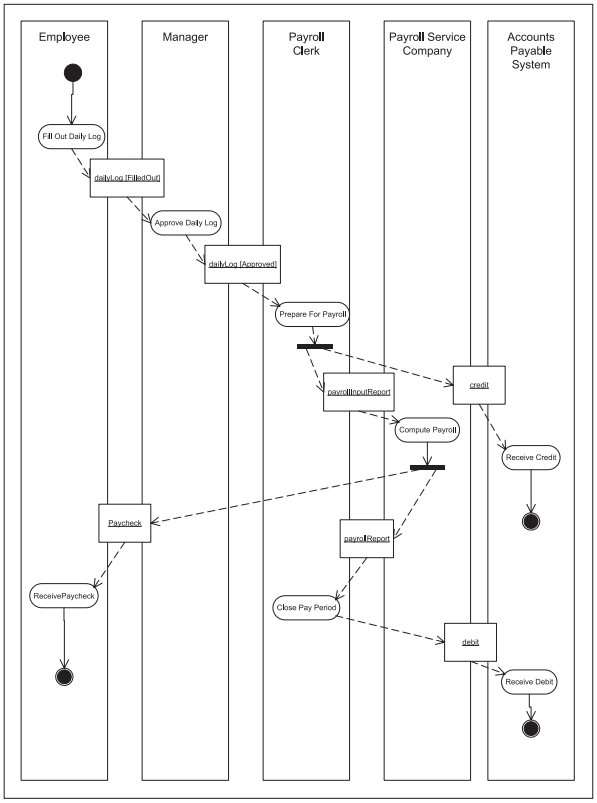
A basic interpretation of the diagram

Benefits obtained by performing this activity and drawing/capturing the corresponding diagram

A final section describes a prototype screen drawn with Visio 2000 Enterprise Edition for the first use case (a screen displaying the user interaction for the first functionality delivered).

1. Discover and document business processes

Diagram 1: Activity diagram



Overview: From a strategic business perspective, graphical modeling provides an excellent way to represent how a business or business process is functioning *today*. More importantly, it will ultimately show how a business *should be* functioning, and how its processes *should be* improved. Such initiatives typically drive Information Technology (IT) projects.

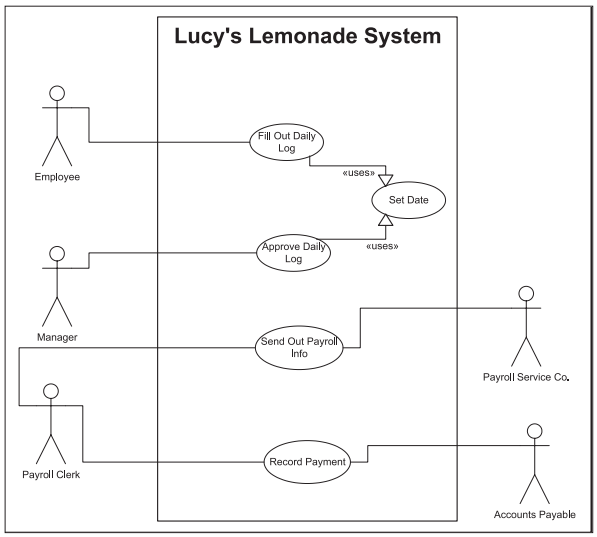
Knowledge of critical business processes usually resides only in the heads of key workers. This step captures those processes in a model that visually communicates the important steps.

Interpretation of Diagram 1, Activity Diagram: After filling out the daily work log, the employee sends it to her manager, who approves it. The manager sends it to the payroll clerk, who uses the incoming daily work logs to prepare for payroll. The payroll clerk enters the appropriate credits into the Accounts Payable system, and sends the payroll input report to the payroll service company, which computes the payroll. The payroll service company sends checks to the employees and sends the payroll report to the payroll clerk, who then debits the Accounts Payable system.

Benefits: It behooves a business or an application development group to *expose* and *understand* the relevant business processes as the first step in a development effort. Capturing the business processes in a UML visual process model using Enterprise Edition can help expose inefficiencies, which can suggest ways to improve the processes.

2. Define external functional requirements

Diagram 2: Use Case diagram



Overview: The UML approach to requirements gathering goes beyond simply listing a set of requirements. Rather, it lists the different kinds of users of the business or software, and then lists the kinds of functionality that the system will deliver to each kind of user. At the highest level, a use case diagram describes this step.

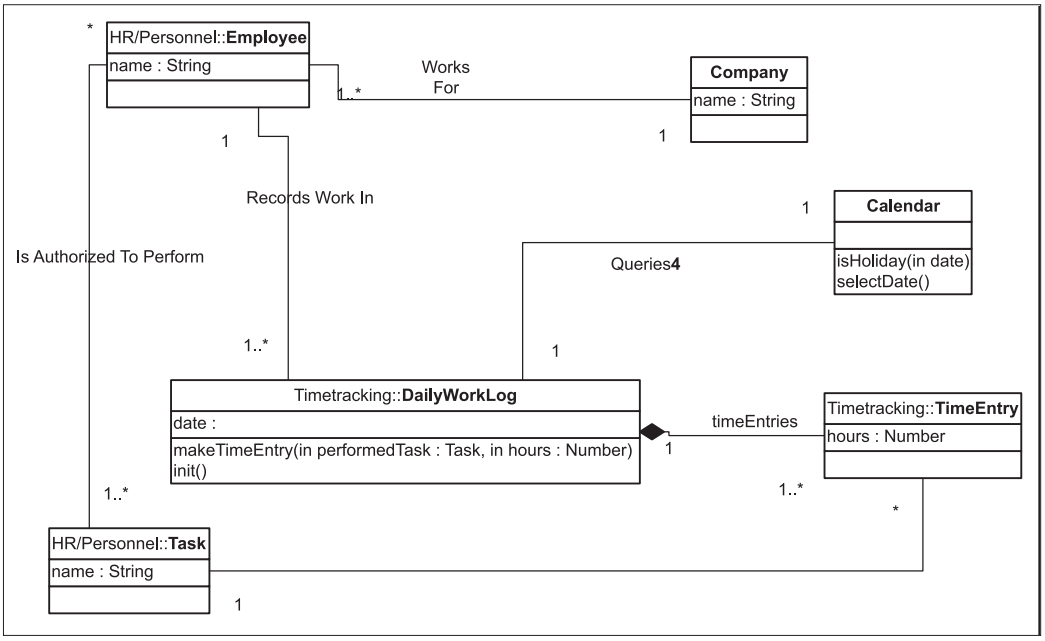
(See also **Figure 1**, which is the prototype screen for this use case.)

Interpretation of Diagram 2, Use Case diagram: Employees, managers, and the payroll clerk use the system. Employees use the system to fill out the daily log. Managers use the system to approve daily logs. The payroll clerk uses the system to send payroll information to the payroll service company and to record payments in the accounts payable system. Two of the high-level use cases make use of the lower-level reusable use case *Set Date*.

Benefits: Use cases have proven to be an excellent means of communicating with users about requirements. Non-technical users understand and relate to use cases because they see themselves in the use case diagrams (as actors) exercising their slices of functionality. The increased involvement of users in defining requirements with Enterprise Edition increases their level of support for the product and the process.

3. Capture the primary abstractions of the problem domain

Diagram 3: Class Diagram of Domain Model



Overview: Listen to people who work in the same business domain (say, telecommunications or order processing) talk to each other, and you will hear them using certain words specific to their domain (e.g., switcher or line item). You need to use these words to specify what the system will accomplish. For an accurate specification, you need precise definitions of these words that everyone can agree on. Besides providing a verbal definition of each word, object methodologies take another huge step forward by embedding these words (these primary abstractions) as classes, attributes, associations, and roles in a class diagram. The *class diagram* precisely depicts the relationship of the words to each other.

Interpretation of Diagram 3, Class Diagram of Domain Model: Every employee works for one employer a company. A company has one or more employees. Every employee is authorized to perform one or more tasks. Employees record work in a daily work log, which contains one or more time entries. A time entry consists of the hours worked and the task performed.

Benefits: The importance of performing this activity derives from the Golden Rule of Object Orientation. The domain classes that we used to describe the problem domain become software classes that form the basic building blocks of the software. Basing the software architecture on business entities enables the finished software system to gracefully evolve and adapt to future changing business requirements.

If you already own Visio Professional 5.0, you can immediately start practicing OO modeling with that tool, because it enables you to draw class diagrams. Just by sharing classes visually and consistently within a development project, team members can rest assured that they are all working with the same set of basic building blocks.

From class diagrams, Visio 2000 Enterprise Edition will generate class definitions in Java, C++, or Microsoft Visual Basic®. You can also use the UML solution in Enterprise Edition to reverse engineer projects in order to document system architecture.

4. Describe how objects and roles interact to realize the functionality

Overview: So far, we have:

Given an external description of the system's functionality.

Represented the principal abstractions in the domain by software classes.

Each software class is like a "template" for a potentially large number of *objects.* When the time-tracking application runs, the various *classes* spawn different kinds of *objects*: employees, tasks, daily logs, a calendar, and so on. Each object, say, Penelope Powe's daily log for 6-Sept-1999, carries around its internal data. But each object is also *intelligent* because it performs services for other objects. For example, Penelope's 6-10-1999 daily log object, when asked, enters time entries into itself. (See Figure 3.)

In an object-oriented system, interacting objects do all work. So we now determine how the business objects will interact with each other to realize the functionality described in the use case. Objects interact by sending messages to each other. For each use case, we use an *interaction diagram* to describe:

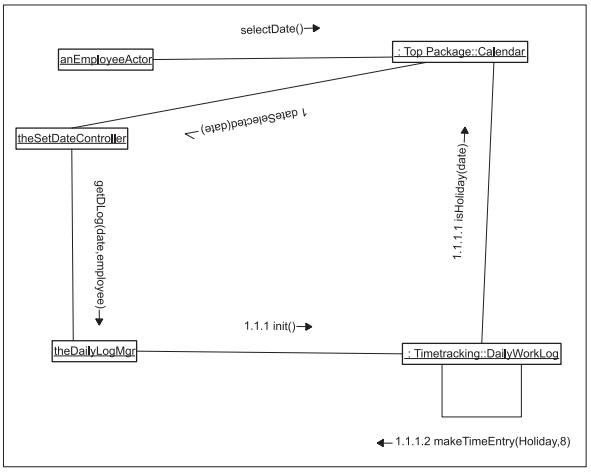
The objects involved in the interaction

and

The sequence of inter-object messages that realize the use case.

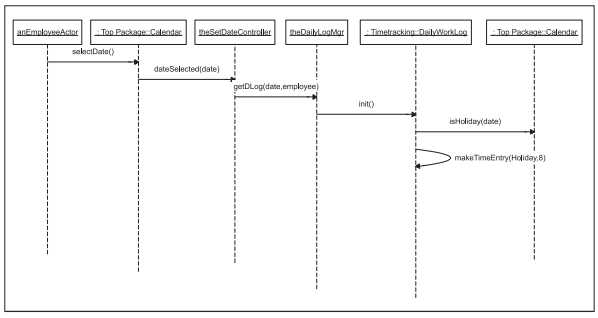
There are two kinds of interaction diagrams: *collaboration diagrams* and *sequence diagrams.*

Diagram 4a: Collaboration Diagram for use case Set Date



Interpretation of Diagram 4a, Collaboration Diagram for use case Set Date and Diagram 4b, Sequence Diagram for use case Set Date: Diagrams 4a and 4b contain exactly the same information: When the employee filling out the daily work log sets a date, here is the behavior that makes it happen. The employee actor (Penelope) selects a date from the calendar. The calendar notifies the controller that a new date was selected. The controller asks the daily log manager to get (create, if needed) the daily log for that date. The returned daily log asks the calendar if the date is a holiday. If so, then the daily log enters eight hours of the holiday task into itself.

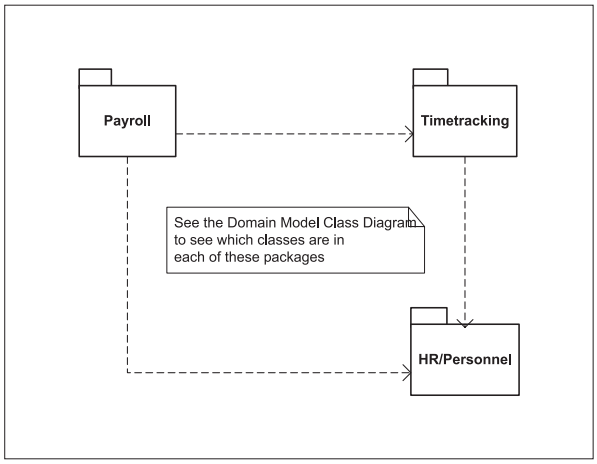
Diagram 4b: Sequence Diagram for use case Set Date



Benefits: Good carpenters use the maxim "Measure twice; cut once." The diagrams tell a programmer how to code the use case. The algorithm can be tested from the diagrams, before any code is written. Errors caught during the modeling and design activities with Visio 2000 Enterprise Edition are easier, cheaper, and faster to correct than those that are allowed to creep into the code.

5. Divide and conquer: carve up the system into subsystems

Diagram 5: Package diagram



Overview: An oft-used and powerful problem-solving technique is to break up a hard problem into simpler problems. In UML modeling, we can decompose our system into lower-level subsystems and show the dependencies among these subsystems.

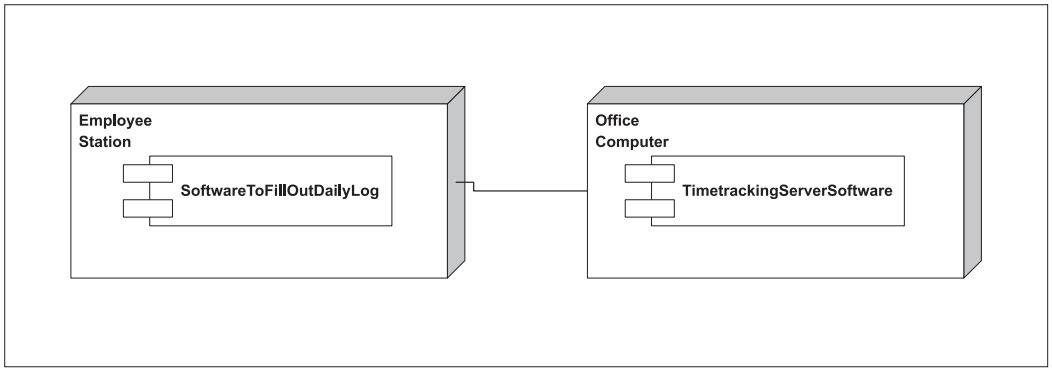
Interpretation of Diagram 5, Package diagram: The top-level system breaks down into three subsystems: *Timetracking*, *HR_Personnel*, and *Payroll*. *Payroll* depends on (uses the services of) both *Timetracking* and *HR_Personnel*. *Timetracking* depends on *HR_Personnel*. The labeling in Diagram 4 shows that the *Timetracking* package contains classes *DailyWorkLog* and *TimeEntry*, while the *HR_Personnel* package contains *Employee* and *Task*.

Benefits: Once subsystems have been identified and documented with package diagrams, it is common practice to parcel them out to small teams to model and develop. Each team can focus on its subsystem while largely ignoring the rest of the system. This also allows specialized developer expertise to be applied to subsystems where the expertise is needed.

Historically, many systems have resisted up-scaling because dependencies were not identified and controlled during development; small changes in one part of the system forced cascading changes and recompiles in far-reaching parts of the system. To enable graceful evolution of a system, it is imperative that subsystem dependencies be identified and minimized early in the process. Package diagrams drawn with Enterprise Edition do that.

6. Deploy components to computers

Diagram 6: Deployment diagram



Overview: All the prior activities model *logical* aspects of the system; however, this activity models *physical* aspects executables, libraries, files, tables, documents all referred to as *components* by UML. Basically, the diagrams identify the system s physical components and the computers on which these components should reside.

Interpretation of Diagram 6, Deployment diagram:
Lucy s time-tracking system uses two kinds of computers: *OfficeComputer* and *EmployeeStation*. The executable component *SoftwareToFillOutDailyLog* runs on *EmployeeStation,* while executable component *TimetrackingServerSoftware* runs on *OfficeComputer*. The two computers talk to each other.

Benefits: These diagrams drawn with Enterprise Edition are useful for identifying runtime elements for potential re-use. They help you plan and schedule deployments.

7. Draw a prototype screen

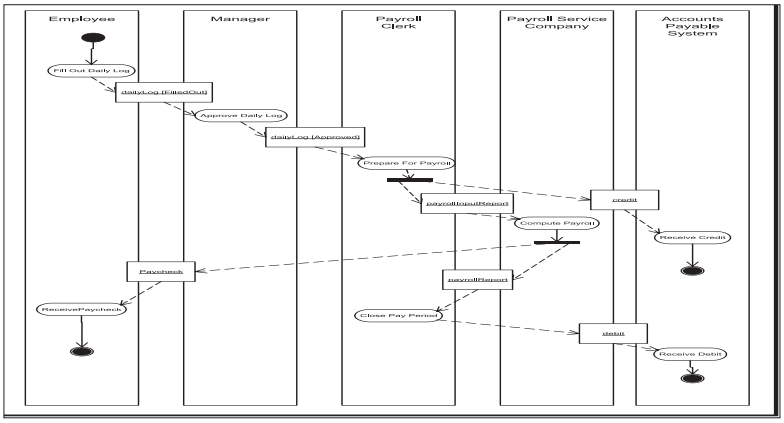


Figure 1. Visio 2000-drawn Prototype Screen for Use Case Fill Out Daily Log

Overview: When designing the system, a good initial rule-of-thumb is to design one window for each use case.

Interpretation of Figure 1, Visio 2000-drawn prototype screen for use case *Fill Out Daily Log*: The screen shows Penelope Power filling out her daily log for October 6, 1999. She has entered seven hours of the *Sell Lemonade* task and is just choosing *Order Supplies* from a list of authorized tasks for the remaining one hour.

Benefits: For most users, the visual screen brings the use case down to earth. Users can look at the prototype screen and imagine what it would feel like to run the use case. This enables you to get quality suggestions from them regarding the functionality of the evolving application, as well as its look and feel.

Recommendations for Getting Started with Visual UML Modeling

If you are new to object modeling and to the UML, then a staged introduction might work well for you. In your initial exploration or project, start with a whiteboard to diagram and capture your first model. This allows you to separate learning how to model from learning how to use a tool. When your model becomes so large that you really can t keep track of things anymore using just whiteboards or scraps of paper (it won t take long), then switch to an easy-to-use UML visual modeling tool like Microsoft Visio 2000 Enterprise Edition.

You don t need to use all the diagrams at once. Start with the most critical diagrams. We recommend the following:

1. Use case diagrams to capture and gain agreement on system requirements.
2. Class diagrams to describe the structure of the business and to define the building blocks of the software.
3. Sequence (or collaboration) diagrams to describe how objects get the work done.

If you are just getting started, you should probably get some training relative to the UML and object modeling.

Microsoft Visio 2000 Enterprise Edition is "methodology neutral." Thus, you can acquire your early OO modeling and development experience without committing to an OO methodology. After your initial experience, you will be in a much better position to understand the issues involved in choosing a methodology that is right for your team.

Enterprise Edition checks your model for you as you build it, identifying and pointing out UML errors. This is an invaluable teaching aid, which also helps you create well-formed models.

Modeling supports *rapid application development*, with the model getting extended and refined with each iteration. This is not to be confused with "*rabid* application development," where visual modeling is neglected in a misguided effort to save time. Model-free development leads, at best, to working applications that are difficult and expensive to maintain, extend, or scale up to meet changing business requirements.

Working in Visio 2000 Enterprise Edition

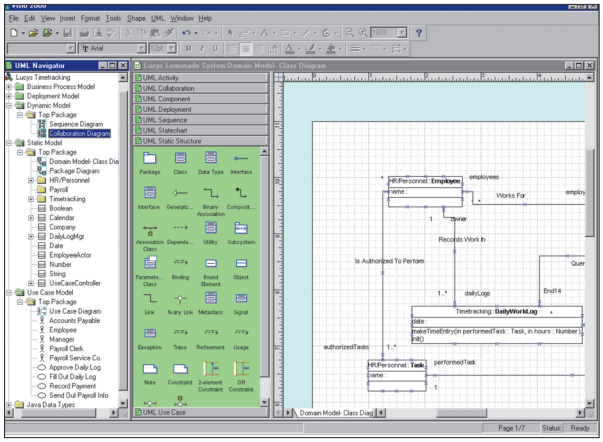


Figure 2. How Enterprise Edition was used to construct Diagram 4

This section contains a few figures that illustrate how Microsoft Visio 2000 Enterprise Edition supports you in defining and capturing a UML visual model.

In the **drawing page** (on the right half of the screen) you see the diagram built using UML model elements displayed in the stencil window (in the center of the screen). The **stencil window** is displaying the UML elements needed for static structure diagrams. Other UML elements appear in the stencil window when one of the other seven UML stencil title bars is clicked. Thus, the tool allows modelers to draw the full suite of the eight generic kinds of UML diagrams.

The **UML Navigator** (on the left) displays the models, packages, diagrams, and most model elements being built by the modeler in the drawing page. As new UML model elements are defined in the drawing page, they appear automatically in the UML Navigator. Modeling elements (e.g., classes) can be dragged into the drawing page from either the UML Navigator or the stencil window. The drawing is intelligent; as a class icon is dragged to better position it, any attached lines and labels follow the class.

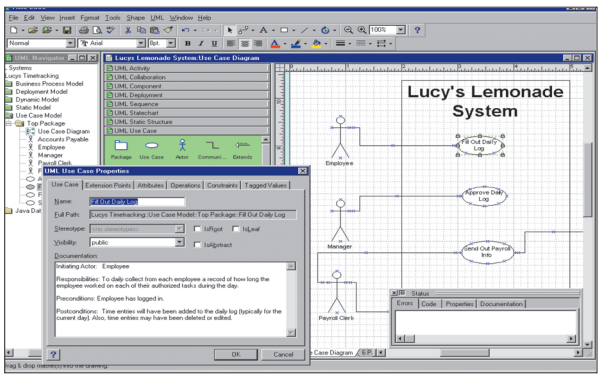


Figure 3. Documenting the use case Fill Out Daily Log

This screen illustrates how use case documentation was captured behind the use case diagram, in the UML **Use Case Properties** dialog box. Semantic information can be captured behind virtually every model element.

Conclusion

These examples have illustrated the many benefits of UML modeling with a visual communication tool like Microsoft Visio 2000 Enterprise Edition.

Aligning a software solution to the business requirement is the key to success for every project. When needs and requirements are visually documented with models, communication between sponsors, business users, customers, and developers is optimized. Visual models facilitate interactive and iterative architecture development, making development projects more likely to succeed.

Visio 2000 Enterprise Edition is ideal for creating the diagrams and models needed to visually communicate how software must function. Using this iterative and incremental development approach throughout the project life cycle maximizes the business value and impact of the software delivered. It also provides a foundation on which a business can create new solutions that take advantage of market opportunities and better use technology as a competitive advantage.

Further Resources

For the Visio Developer Network, visit www.microsoft.com/office/visio and then click Explore Visio on the Web and choose Visio Developer Network.

For other modeling methodologies, visit www.orm.net.

For product information, visit www.microsoft.com/office/visio.

To find out more about how to get started, contact Integrated Software Specialists, Inc. (ISS), or the Visio division of Microsoft Corporation. ISS offers a popular one-day seminar where students learn the prerequisite concepts required to apply object technology effectively. Additionally, ISS offers a four-day boot camp to help people learn how to apply object thinking to analyze, model, and design basic business applications. You can contact ISS at 847.706.6797 or info@issintl.com, or visit www.issintl.com.

 Part No. 098-88443