

APPENDIX D

Technology Tools and Techniques

Appendix D - Technology Tools and Techniques

Table of Contents

D.1 Technology Tools	D-1
D.2 Component Delivery	D-9
D.3 Introduction to UML	D-29
D.4 Legacy System Components	D-33

EXECUTIVE SUMMARY

Successful delivery of the software supporting the Modernization Blueprint is critically dependent upon the technologies, tools, and methods used for system building, analysis and design, code repositories, class libraries, testing, and component-based development (CBD). These tools and technologies are described in this Appendix.

Section D.1 is organized by classification of tool and technique. First, the classification is defined and briefly described. Then, tools and technologies for each classification are presented.

SFA identifies CBD as a tool critical to the success of the Blueprint. Since designing and creating components is a relatively new discipline, industry practice is still being defined and refined. We recommend that ED/SLCDM be supplemented in the area of component design and recommend the Catalysis methodology for this purpose. Catalysis is the dominant methodology for component design, is publicly available, and is supported by several vendors.

In this scenario:

- ED/SLCDM provides the overarching SFA development life cycle linked to management and reporting functions. Business and domain modeling is performed under the ED/SLCDM.
- Component modeling and specification is conducted using Catalysis.
- The ED/SLCDM is used for component implementation as well as taking component specifications through outsourcing, reuse, re-engineering, or development into operation.

Section D.2 describes in detail the life cycle, tools, technologies, and supporting methodologies for CBD.

Section D.3 presents an introduction to the Unified Modeling Language (UML).

Section D.4 discusses legacy system components and CBD

D.1 Technology Tools

As software becomes larger and more complex, the size of the team needed to complete the development process increases. On large-scale projects, this team often will consist of application designers, programmers, graphical interface designers, product testers, technical writers, build engineers, product support staff, and management. On smaller projects, the same team member often handles several of these tasks. When the development team grows past the single programmer, the advantages offered by development tools that facilitate teamwork and collaboration quickly become apparent. With larger teams, such tools are a necessity.

The following subsections will discuss the process support and management tools for software development projects:

- Subsection D.1.1 Analysis, Modeling, and Design
- Subsection D.1.2 Software Configuration Management
- Subsection D.1.3 System Building
- Subsection D.1.4 Automated Software Testing and Quality Assurance
- Subsection D.1.5 Software Project Management
- Subsection D.1.6 Reusable Class Libraries
- Subsection D.1.7 Code Repositories

D.1.1 Analysis, Modeling, and Design

Analysis, modeling, and design tools assist in generating requirements for applications, including data definitions, business rules, and programming specifications. These tools typically use formalized methods and approaches derived from research in computer science and computer-aided software engineering (CASE). They are used in the analysis and design phases to create models with different levels of detail and depth. Modeling and design data are defined centrally and stored in a shared repository. A good model can be understood and critiqued by application experts who are not programmers.

When considering analysis tools, the words “requirements management” (RM) should come to mind. A software requirement can be defined as “a condition or capability to which the system being built must conform.” RM is the process used to ensure that a system conforms to its expectation. Since there are several types of requirements at different levels of detail, which vary in importance and difficulty, it is important to keep a history of each requirement and its associated details. Two key features of a RM tool are the ability for viewing the requirements and their attributes as well as the ability to show traceability (to define a relationship between requirements). Leading RM tools are Rational RequisitePro, Quality Systems & Software DOORS, IcCONCEPT-RTM, SLATE REquire, and Technology Builders, Inc. Caliber-RM.

There are typically two classes of modeling and design tools: CASE products and lightweight modeling tools. The primary distinction between older CASE products and newer or updated modeling and design tools is the latter has a less-rigid structure, greater process flexibility, and the ability to integrate with other development tools. The most basic feature of the modeling and design tools is support for diagrams and drawings, typically using customizable icons for notations and symbols.

Lightweight modeling tools for database applications have become more prominent with the increase in client/server development. Many of these tools now directly support object-oriented systems with models and diagrams useful in designing and documenting the object classes and inheritance to be used in a software project. Advanced features considered desirable in these lightweight tools are the ability to export the requirements to a skeleton of the source code (by use of code generators) and to import or update the data model from changes made in the actual source code (reverse engineering). Popular lightweight database modeling tools used for client/server applications include products such as Logic Works' Erwin and Sybase's Powersoft PowerDesigner (formerly S-Designer). At the high end are tools capable of modeling mainframe and client/server applications, such as Software Through Pictures from Aonix.

D.1.2 Software Configuration Management

Software configuration management (SCM) tools are used by application development teams to provide software revision control, source code versioning, and release management capabilities. Other more advanced features include process management and the ability to track requests for changes and bugs discovered during testing. Development team members are required to check out models, specifications, source code modules, documentation, and other files from a central repository before making changes and then check these files in after changes have been made.

The SCM tools available from various vendors usually provide most of the following capabilities: version management, support for distributed development teams, integrated change request management, customizable compile and build environments, and basic processes for code versioning and promotion. These vendor tools differ the most in their support for these features: large-scale distributed teams, Web access for reporting or version management, special support for Year 2000 teams, degree of process orientation, and ability to integrate with other development tools.

SCM tools preferred by mainframe application developers include Computer Associates' Endeavor, Intersolv's PVCS and PCMS, and Platinum Technology's CCC/Harvest. UNIX programmers gravitate toward the RCS/ CVS tools available with UNIX as well as enhanced products, such as IBM's TeamConnection, Mortice Kern Systems' Source Integrity, Perforce's Perforce, Rational Software's Pure Atria ClearCase, and Starbase's StarTeam. ClearCase and TeamConnection go a step further than other tools and support archiving the complete build environment, including the development tools (the language compiler and library files, for example) used in the build process.

Some SCM tools also focus on process control in addition to the traditional version control functions. These products offer features for organizing and controlling the processes and procedures as they are used in software development and deployment. Table D-1 summarizes these processes and procedures used in software development and deployment. Vendors with the most process control focus are Continuous Software Continuous/CM and Platinum Technology CCC/Harvest. Products from IBM, Rational Software, SQL Software, and TRUE Software also include a strong infusion of process control.

Capability	Description
Version Identification	Versions and releases can be assigned identifiers automatically when they are added to the system. Some tools support the assignment of attribute values for identification.
Change or Version Control	Versions of components must be checked out explicitly to make changes. The team member making the change is recorded automatically. When the changes are checked in, a new version is created and identified with a tag. The old version is never destroyed or overwritten.
Storage Management	Version management tools provide features to reduce the storage space required by the different versions. Most tools try to minimize storage by describing each version in terms of its delta or difference from a baseline version.
Build Environment	To replicate a build, versions of development tools and ancillary files (compiler "include" files, for example) used in the process also must be managed and archived under change control.
History Recording	All changes made to a module or system are recorded and listed.

Table D-1: Software Configuration Management Capabilities

Microsoft offers Visual SourceSafe, a low-end version-management tool that integrates directly with Visual C++, Visual Basic, and Visual J++ integrated development environments (IDEs). Visual SourceSafe is bundled with Microsoft's Visual Studio Enterprise Edition. As a result, Microsoft implicitly has endorsed the use of source code management and is exposing a large number of small developers to these concepts. Microsoft also publishes a set of developer APIs that allows other vendors to integrate SCM tools into the Microsoft IDEs. This allows a migration path for users who outgrow the basic Visual SourceSafe capabilities and require a more robust and full-featured SCM product. Inprise now bundles a version of Intersolv's PVCS with its Delphi, JBuilder Enterprise, and C++ Builder Enterprise suites.

D.1.3 System Building

In modern program development, it is common to use tools that automate the build process of an application and ensure the latest approved source code files are being compiled. Most of these build tools are based on the "make" utility originally developed on UNIX. The purpose of these tools is to identify and perform the minimum amount of work needed to build a new version of the application. For example, there is no need to recompile a module that has not changed since the last compilation.

There are three steps in the system build process:

- Step 1: Generate or create the dependency structure;
- Step 2: Compile source code; and
- Step 3: Link object-code modules.

System building tools require a list of components or modules needed for a specific build, a structure that documents the dependency between various components, and information about where these files are stored. Some build tools can create the dependency structure automatically from a list of files. Many modern compilers support an option to generate this file dependency list while parsing the source code. Other tools, such as "makedepend" that is bundled with the X Window System on UNIX, can identify these dependencies. Without these tools, this

dependency structure must be created manually. Opus Make with MKMF from Opus Software is a third party "make a make file" tool that can generate these dependencies automatically. It runs on a variety of platforms and has numerous features that are missing from standard "make" packages.

The most recent advance available in some of these tools is a parallel and distributed "make" feature. Parallel "make" takes advantage of multiprocessor machines, spreading the build process across multiple processors in the same machine by executing parallel compiles of different source files. Distributed "make" spreads the work across multiple computers in a distributed environment. The goal of a parallel or distributed make feature is to shorten the time needed to rebuild the application. SunSoft's Workshop integrated environments are bundled with a parallel "make" facility. The freely available GNU "make" utility often used in UNIX environments also supports parallel "make."

The last step in building an application is the linking phase. In this step, the numerous object-code modules that constitute an application are brought together and linked into one whole. The link step can be slow and resource-intensive, especially on large projects. The most advanced linkers implement incremental linking, in which only the modified modules are relinked, rather than relinking all parts of the executable program from scratch every time a single module is changed. This process results in significantly shorter link steps. On UNIX systems, incremental linking is included in GNU C++, HP Softbench, and SunSoft's Workshop products. Rational Software's PureLink provides the same function in a stand-alone linker product for UNIX platforms. Inprise's C++ Builder, Microsoft's Visual C++/Visual Studio, and Watcom's C++ support this capability on the PC.

D.1.4 D.1.4 Automated Software Testing and Quality Assurance

Automated software testing and quality assurance tools represent a wide range of processes and technologies used to ensure that software does not contain "bugs."

The tools in this category have been divided into the following subsections:

- Subsection D.1.4.1 Static Analyzers
- Subsection D.1.4.2 Software Metrics
- Subsection D.1.4.3 Execution Profilers
- Subsection D.1.4.4 Defect Testing, GUI Testing, and Load/Performance Testing
- Subsection D.1.4.5 Run-time Error Detection
- Subsection D.1.4.6 Code Coverage

Table D-2 summarizes the classification of these tools.

Approach	Description
Static Analysis	Analyzes the source code to locate potential defects in programming.
Software Metrics	Analyzes the source code or documentation and generates metrics based on complexity.
Execution Profilers	Determines how much time is spent during execution in various functions and components to locate the heavily used sections that could be limiting performance.
Defect Testing	Automates finding areas where the program does not conform to its specifications, usually based on a test manager application using specially created application test suites.
GUI Testing	Test the GUI component of an application by simulating user keystrokes and mouse movements.
Load/Performance Testing	Stresses the application by running it with heavy simulated workloads to determine its load/performance characteristics.
Run-time Error Detection	Monitors and locates certain classes of run-time errors, particularly memory leaks.
Code Coverage	Determines how much source code in an application is covered by automated tests and defined test suites.

Table D-2: Software Testing and Quality Assurance Tools

D.1.4.1 Static Analyzers

Static analysis tools, such as the "lint" utility used with C files, are designed to check the source code for potential programming errors. These tools are called static because they analyze the source code rather than the running program itself. Modern static analysis includes sophisticated checks on control flow, data use, function interfaces and parameters, information flow, and paths of execution. Static analyzers for C source code are Compuware CodeReview, Gimpel Software PC-Lint Gimpel Software FlexeLint, and IPT's lint-PLUS.

Several static analyzers for checking Fortran source code are available, including Cobalt Blue's FOR_STUDY, IPT's FORTRAN-lint and FORTRAN90-lint, Leiden University's ForCheck, Polyhedron Software's plusFORT GXCHK tool, and Quibus's ForWarn. The FEI product from IPT is unusual in this regard; it represents an example of the integration of static analysis (FORTRAN-lint) with other powerful development and reverse-engineering tools that make up Cayenne Software's Ensemble package. (In August 1998, Cayenne merged with Sterling Software.)

D.1.4.2 Software Metrics

Software metrics is another form of static analysis. The tool parses the source code and assigns values based on complexity and other measurements to lines of code, functions, and components to indicate where the more complex, less-understandable, and statistically more error-prone programming occurs. Similar metrics for complexity and readability can be applied to documentation files. Based on these metrics, it is possible to focus testing in the areas that have been identified as more error-prone. Tools that determine software metrics include McCabe and Associates' Visual Quality Toolset, which is available on UNIX and Windows and can analyze code in Ada, C, C++, COBOL, Fortran, Pascal, PL/1, Visual Basic, and several other languages.

D.1.4.3 Execution Profilers

Execution profiling tools, bundled with many integrated development environments, monitor a specially linked version of the running program and report how much processor time is used in executing each line of program code. This information helps developers pin-point areas that should be targeted for optimization and performance improvement efforts. UNIX systems include the "gprof" command-line utility for this task. The newer profiling tools incorporate an easy-to-use graphical front end for controlling profiling and reviewing the results. Compuware's TrueTime products on Windows 95/98/NT, Intel's VTune on Windows 95/98/NT, and Rational Software's Pure Visual Quantify available on UNIX and Windows NT represent enhanced profilers with graphical interfaces. There is also increasing interest in profilers that support Java development such as Compuware's TrueTime for Java, Intel's VTune, Intermetrics' JWatch, and KL Group's JProbe.

D.1.4.4 Defect Testing, GUI Testing, and Load/Performance Testing

Software defect testing usually is handled by tools that run the application through predefined test suites. Automated defect testing can be applied to unit testing of individual components, module testing of a collection of dependent components, and system testing of the entire application. Tools are also available that automate testing for defects in the GUI and the interaction between the client and server.

A new category of load-and-stress testing can also be applied to client/server and Web-based applications to ensure reliability, robustness, and performance. Major vendors of GUI and distributed client/server testing tools are Mercury Interactive, Rational Software (having acquired SQA), and Segue Software. Tool vendors that offer automated testing of mainframe applications include Computer Associates, Compuware, Cyrano (formerly IMM and Performance Software), and IBM. Vendors offering load testing tools include Cyrano and Rational Software.

D.1.4.5 Run-time Error Detection

One of the most popular categories of products now used for quality assurance of C and C++ programs is run-time error detection. These products check for memory management problems (memory and resource leaks and other heap errors) as well as parameter or range errors that occur when calling operating system or library functions. These tools interface with the application and the operating system or libraries to track memory management calls and other system function calls to catch parameter errors dynamically at run time. Run-time error detection tools for C/C++ include Compuware's (formerly NuMega Technologies') BoundsChecker (MS-DOS, Windows 3.x/95/98/NT), Parasoft's Insure++ (UNIX and Windows 95/98/NT), and Rational Software's Purify (UNIX and Windows 95/98/NT). Compuware also has introduced SmartCheck, a run-time debugging tool for Visual Basic that automatically detects and diagnoses errors and translates vague error messages into more detailed problem descriptions. Other products that have run-time error checking built in are the C/C++ development environments from CenterLine Software and SunSoft's Workshop tools.

D.1.4.6 Code Coverage

A code coverage tool measures what portion of the source code for an application is actually executed during automated testing. The goal is to test as high a fraction of the code base as possible. Run-time error detection and code coverage are often included as background processes

while automated defect testing is running. Visual tools in this category include Parasoft's TCA (bundled with Insure++) and Rational Software's Pure Coverage available on UNIX and Windows 95/98/NT.

D.1.5 Software Project Management

Project management involves planning and scheduling software development projects. Another important component of project management is tracking changes in user requirements over the lifetime of a software project. Several packages can be used to manage staff and resources as well as produce time lines, critical-path diagrams, and project evaluation and review technique (PERT) charts. Tools also are available to estimate the costs associated with a software project. Currently, a large number of these products are targeted specifically for estimating costs associated with Year 2000 mitigation projects. Two software project management tools are Microsoft Project and Software Productivity Research KnowledgePLAN.

D.1.6 Reusable Class Libraries

For years, language compilers have included run-time libraries to support functions and operations that are required by the language standards but are not directly part of the programming language. For example, Fortran compilers come with a large library of mathematical and other functions that are defined by the Fortran 90 standard but must be implemented outside the language compiler. One enduring strength of C has been the definition of the standard C library, which includes a large collection of portable functions for I/O, character and string handling, mathematics, and other purposes. The C++ language standard defines a similarly rich standard C++ library, the cornerstone of which is the standard template library (STL) of generic container classes.

One key benefit of an OO language (Smalltalk, for example) is its class library, a repository of reusable code that has been refined and cultivated over time. Libraries of reusable classes and components for most of the popular OO languages are available on the Internet.

Several companies now sell mature and tested class libraries designed for a particular set of tasks. This option allows development teams to buy reusable class libraries as an alternative to building the classes themselves. Rogue Wave Software sells a large collection of popular C++ class libraries that are available across platforms (MS-DOS, Windows, Macintosh, OS/2, and most versions of UNIX) and can be purchased in both source code and object form. These class libraries include basic tools, the STL, mathematics, and database access, networking, and financial functions. Rogue Wave Software's Tools.h++ class library is also bundled with many of the major C++ compilers and development environments. Many C++ class libraries have been rewritten as Java class libraries and are now becoming available. Rational Software also offers C++ and Java class libraries targeted for specific applications. Many vendors now sell software components designed to work with Visual Basic, C++, Inprise's Delphi, and Java.

ILOG, one of the largest software components vendors, began shipping its JViews product in late 1997. JViews is a pure Java library of sophisticated graphical objects, behaviors, and data structures. KL Group also sells a set of highly regarded Java components. KL Group began shipping JProbe, its Java profiler, in March 1998. JProbe is an interesting product for KL Group because it takes the company beyond the components business and into the tools arena.

One interesting class library for networking is the Adaptive Communication Environment (ACE) toolkit originally developed by Doug Schmidt at Washington University. The ACE toolkit is a

large, rich framework of high-level C++ classes for writing portable network applications on UNIX and Windows 95/98/NT. It uses design patterns, a recent advance in software development technology, in the design of its C++ classes. A version of the ACE toolkit implemented in Java also is available. The ACE toolkit is provided as freely available source code and has been ported to most UNIX systems and Windows 95/98/NT.

D.1.7 Code Repositories

Code repositories offer hope for greater code reuse. The primary problem a development organization faces is the work involved in creating a useful repository. For code repositories to work effectively, software development teams need to be in agreement on processes, classification procedures, and other criteria. Several vendors now offer software tools that facilitate setting up and maintaining code repositories, but these are only as good as the procedures actually employed by software developers. Vendors offering software for creating and maintaining code repositories include Intersolv, Platinum, Rational Software, and Transtar. Some high-end development tools (Sterling Software's COOL:Gen, for example) are adding code repository capabilities in their latest releases. This market still is in its infancy.

D.2 Component Delivery

Developing software applications from reusable components is termed *component-based development* (CBD). The technology might be more aptly called, “application *delivery* through the assembly of software components”. Components are a means for delivering enterprise business applications. The subsections listed below describe the tools, technologies, and methodologies for application development using components, as applied to the requirements of SFA:

- Subsection D.2.1 Introduction
- Subsection D.2.2 Strategic Directions for Component-Based Development
- Subsection D.2.3 CBD Tools
- Subsection D.2.4 Methodologies for Component Design
- Subsection D.2.5 Recommendations to SFA
- Subsection D.2.6 Tool Support for Catalysis

D.2.1 Introduction

The ability to create and deliver large software applications on a predictable schedule in a rapidly changing environment has not matched the growth of hardware and network capacities. Large package applications such as PeopleSoft and SAP can be an alternative to custom code. However, packages require long and expensive implementation and integration projects and it can be expensive and difficult to customize packages to a user’s business needs. OO design and distributed objects are proving to be successful in large software projects. The discipline of OO design, however, takes time to grow in a development team. CBD promises to address these problems. First, component-based applications are designed to be flexible, with the ability to adapt to the changing requirements of business and technology. Second, components can increase the speed with which applications are developed and delivered. Third, component design and assembly does not require object-oriented techniques. Finally, components provide a means of integrating legacy applications and current technologies.

The basic design principles of components are simple and can be implemented in a variety of technologies. Central to the notion of components is one basic idea: designing an application using components is designing for reuse, not for obsolescence. A component:

- separates what is performed from how it is performed;
- sets up specific boundaries between functions;
- has no unexpected side effects or interactions with other components;
- provides services around which applications can be constructed;
- requires design architecture; and
- demands good documentation.

Components are built to deliver one or more *services* within an application. A design focused on services separates the capability of an application from its technical implementation. A software component’s services are accessed through an *interface*, which must be consistent (i.e., work the same way every time) and published (i.e., tell other components how to use it). Constructing an

application begins by identifying the services required, then gathering the components that offer those services. Connecting the components builds the application.

The *service* provided by a component isolates that component from the consumer of its services. Microsoft defines a service as “a set of functionality that supports activities and/or yields information. A service is accessed through a consistent, published interface. A service represents some computing capability. A description of this capability can be used to represent a contract between the provider of the capability and the potential consumers. Using the description, an arm’s-length deal can be struck that allows the consumer to access the capability.” [Microsoft, 1996].

The creators of the Catalysis method for component design (D’Souza and Wills, 1998) present the following definition of component:

A component is “an independently deliverable unit of software that encapsulates its design and implementation, and offers interfaces to the outside, by which it may be composed with other components to form a larger whole.”

CBD is then a means of creating software applications from prefabricated, pre-tested, and reusable pieces of software. An application assembled from components would be flexible and could be built and rebuilt quickly. In designing and assembling an application, the designer wants to draw from a catalog of components, each plug-compatible with the other, each with a clear and obvious function. Achieving this kind of interoperation and flexibility requires a high degree of standardization. Achieving standardization requires a disciplined approach to the design, development and deployment of components

D.2.2 Strategic Directions for Component-Based Development

No single CBD direction dominates the market today. Industry analysts single out four major strategic directions for CBD:

1. Enterprise JavaBeans (EJB) and CORBA, the Sun Microsystems and Object Management Group (OMG) component model.
2. COM/DCOM, the Microsoft component model.
3. Component models promoted by the major package software vendors such as SAP, PeopleSoft, and Oracle.
4. Components created by development tools, promoted by tool vendors such as Forte, Sterling and Platinum.

Before discussing the strategic directions for CBD, it is essential to establish a distinction between objects and components. An object might be a component but a component is not necessarily an object.

- Components provide coarse-grained functionality that can be understood by business users. Objects tend to be much smaller pieces of functionality, of a more technical nature.
- Components are usually defined through interface standards such as COM/DCOM or CORBA whereas objects typically rely on low level programming languages.
- The internal workings of components are concealed (the technical term is *encapsulated*). On the other hand, objects are not completely encapsulated because of their property of *inheritance*. An object class can be defined as a subclass of another (parent) class, and it will *inherit* the attributes and methods of this parent class. This allows reuse of the

existing class structures when defining new objects, but it compromises encapsulation for components because some knowledge of the existing classes internal workings are required.

Table D-3 presents a list of other properties contrasting components and objects.

Components	Objects
Business-oriented	Technology-oriented
Coarse-grained	Fine-grained
Standards-based	Language-based
Multiple interfaces	Single interface
Provide services	Provide operations
Fully encapsulated	Use inheritance
Understood by everyone	Understood by developers
Open standards	Proprietary standards
source: Select Software	

Table D-3: Components and Objects

As component technologies and their supporting tools continue to evolve, the distinction between CBD and the straight OO approach to design, construction, and implementation will become more apparent.

D.2.2.1 EJB and CORBA

CORBA was developed by OMG, an industry consortium. Work on CORBA began in 1989, long before anyone was using the term “component”. OMG created CORBA to enable objects to communicate with each other in a variety of different languages across a number of different platforms.

EJB grew out of the Sun JavaBeans initiative. Enterprise Java Beans are Java-language components for building applications. They differ from pure objects in that EJBs have an “external interface”, called the deployment descriptor, which allows a tool or other EJB to read what the component is designed to do and to connect it up properly to other EJBs. EJBs are designed to operate on a server, using a set of interfaces that allow them to delegate the work of managing services such as security, transactions, and persistence to an EJB container. Containers provide management and control services, isolate the EJBs from different hardware and operating systems, and ensure consistent services are available to the EJBs. By providing a consistent set of service interfaces, EJB containers isolate the EJBs from the details of different underlying middleware such as transaction monitors or object request brokers. When EJB containers are married together via CORBA, their components interoperate in a distributed, multi-tier, enterprise-wide environment.

January 1999 saw the release of Sun’s Jini. Built on Java technology, Jini organizes Java components into a flexible, distributed system. Under the Jini infrastructure, components can be added and removed at will. Jini will benefit CBD by promoting the development of entire systems based upon components and providing a new mass market for Java components. Jini will further the design of services-based architectures which are essential to component design. On the other hand, because Jini is so new, the challenges of real-world implementation are not well understood. Early adopters take on considerable risk.

IBM's SanFrancisco project is designed to help application developers build distributed, object-oriented business applications. It is a collection of business objects (such as accounts payable, accounts receivable, and general ledger) written in the Java language conforming to the EJB component model. By offering a framework for business objects on a large number of IBM platforms, SanFrancisco should attract software vendors to build components for the large installed base. IBM plans to use EJB to unify its wide range of computer platforms to a single standard.

For applications deployed at the enterprise level, EJB/CORBA holds a major share of the marketplace, supporting a growing third-party component market. Compared to COM/DCOM however, the EJB/CORBA market is small and immature. The market is complicated by two factors. First, the standards process to which OMG adheres slows the adoption of standards for the OMG business object component architecture. (Work has begun on the design of components for vertical industries, but products are not ready for release.) Second, Sun continues to evolve the EJB specification, presenting a challenge to developers who prefer a stable design base. Nevertheless, EJB and CORBA are attractive in environments requiring platform-independent, reusable, distributed components.

D.2.2.2 COM and DCOM

Microsoft has been promoting COM, its component object model, as its alternative to CORBA for distributed software component architecture. DCOM refers to *distributed COM*, which is the means for communicating among COM objects over a network. DCOM reflects the evolution of the Microsoft component model that began with the linking of application components on a single desktop and evolved to distributed components. Growing from the extensive set of GUI controls built with Microsoft's ActiveX technology, the COM/DCOM component market is the largest and most developed. Recently, COM/DCOM component types have broadened beyond GUI controls to mathematical and statistical calculation, credit card verification, and remote database access. This includes infrastructure and dozens of business application services.

Like EJB, DCOM components operate in a sort of container: Windows NT. As a component platform, Microsoft NT supports DCOM components interconnected by a rich set of distributed services collectively called the Microsoft Distributed Component Architecture (MDCA). COM and DCOM components are becoming the standard means of building NT applications. They create a new market for component design, development, and delivery on that platform. COM and DCOM dominate the desktop and workgroup environments, especially in organizations committed to NT.

D.2.2.3 Software Package Components

Packages such as PeopleSoft, Oracle Financials, and Baan are some of the largest software systems in operation. The burden of keeping these packages integrated, maintained, and updated led the package vendors to devise component strategies. The package component technologies provide customers with faster and cheaper ways to deploy and update package applications. Package components permit incremental upgrades and integration of parts from other vendors. However, working with package components has uncovered some important lessons. Organizations soon wrestle with the complex task of managing package component configurations and upgrades.

The major ERP vendors are starting to explore repository technology as a means to better manage their component configurations. Through the DCOM Component Connector, SAP uses MS Repository to store information about business API (BAPI) objects and make them accessible to

Independent Software Vendor (ISV) tools. More information on the MS Repository may be found in subsection D.2.3.3, Component Repository and Configuration Management. Other packages, such as those from Siebel and Lawson, are built around the Microsoft DCOM component model and Microsoft repository.

D.2.2.4 Development Tool Components

Model-based development tools may be used to create components and to define their interfaces. Most design tools use Universal Modeling Language (UML) to store and retrieve component definitions. Development tools create a component by modeling it, creating the code, and then building the component for distribution. Using development tools in this way ensures that the following key principles of component-based development are honored:

- Component specification is separate from design and implementation
- Design focus is on the interface
- Behavior of the component is formally documented

Examples of development tools that support components include products from Platinum Technology, Forte Software, and Sterling Software. Platinum Technology offers Paradigm Plus for CBD, based on the Catalysis interface-based design methodology. Forte creates applications from the start as a collection of cooperating components. These components can run locally or remotely, connected by several standard protocols. Components created outside Forte, for example, EJB or COM/DCOM components, can be integrated directly with Forte components.

Sterling Software presents two different approaches to building components. The first approach, embodied in the COOL:Spex tool, uses Catalysis. The second approach, employing COOL:Gen, is based upon Sterling's CBD96 component standard. CBD96 describes the means of creating, specifying, and using components in COOL:Gen. CBD96 includes a documented set of conventions, naming standards, and best practices.

At the moment, the "commodity" software development tools cannot accept components constructed by the model-based development tools. For example, Visual Basic cannot exchange component descriptions with COOL:Gen. The COOL:Gen component marketplace is thus limited to components created by the COOL:Gen tool. Over time, as more components are delivered using EJB and COM, the vendor-specific components will be caught in a shrinking market. In response to this shrinking market, tool vendors will begin to sell their components through the same channels that EJB/CORBA and COM/DCOM components are sold, and will gradually transform their products away from proprietary models to adopt the commodity component standard.

In light of the four strategic directions just described, a life cycle and tools to support CBD are the discussed in the subsections that follow.

D.2.3 CBD Tools

Designing and developing components requires the support of both methodologies and tools. Component methodologies are discussed in subsection D.2.4. This subsection describes tools supporting CBD within the context of a life cycle that consists of three major activities: (1) building components, (2) finding components, and (3) using components. The tools supporting CBD are described in the following subsections:

- Subsection D.2.3.1 Component Modeling

- Subsection D.2.3.2 Component Construction
- Subsection D.2.3.3 Component Repository and Configuration Management
- Subsection D.2.3.4 Component Assembly
- Subsection D.2.3.5 Component Integration

Figure D-1 graphically depicts the categories of component tools.

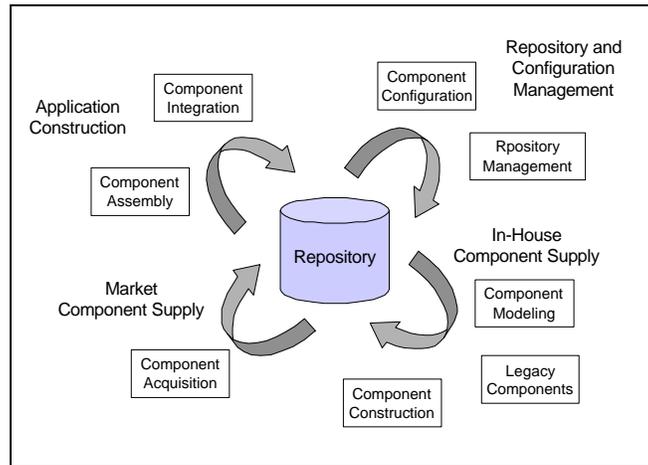


Figure D-1: Component Tool Categories

In the subsections that follow, the tools for CBD are described in further detail.

D.2.3.1 Component Modeling

A model is a technical language and notation that helps developers specify, visualize, and document software systems. UML is transforming the modeling tool market. UML is both a language and a notation. It allows designers to share modeling information among many tools. A common modeling language such as UML offers these benefits: interoperable tools, a common language to describe software systems, and a means to plug in standard groups of components. UML also facilitates communication between the developer and the client, facilitates reuse, provides a single means for describing components as well as indexing their catalogs, and enables the exchange of model data among tools. UML, by its design, can be stored to and retrieved from a relational database.

While UML is gaining increased popularity, it suffers from two problems. First, business process modeling and database design are extensions to the basic UML capability and have not been fully integrated into its other capabilities. Secondly, components are not well supported by the basic releases of UML. (UML components in the 1.0 release of UML are not really components at all - they are packaged objects.) Generally, modeling tools should detect components and interfaces early in the business process modeling stage. The D'Souza Catalysis method, implemented in tools from Sterling and Platinum, is currently the leading means for identifying components early in the development life cycle. More information on Catalysis may be found in subsection D.2.4, Methodologies for Component Design . Table D-4, below, presents tools and technologies for modeling components and their interfaces.

Tool Name	Description	Vendor
-----------	-------------	--------

Tool Name	Description	Vendor
COOL:Spex	COOL:Spex is an implementation of the Catalysis component modeling methodology, complete with the required UML extensions. Once component interface models have been identified and specified in COOL:Spex, models are handed off to downstream tools for generating the components.	Sterling Software www.cool.sterling.com/cbd
Delphi	Component modeling support for the Delphi application development and deployment environment. Support for Delphi components only. Not UML-based.	Inprise www.inprise.com
Forte	Supports component modeling within the Forte application environment. Forte applications can include EJB/CORBA and COM/DCOM components, but the Forte tools do not create these components. The component models used by Forte may be exported in the UML format.	Forte Software www.forte.com
Paradigm Plus	Tool for component modeling within a full development life cycle. Supports UML extensions for Catalysis methodology.	Platinum Technology www.platinum.com/dreamteam
Rational Rose	Tool for component modeling within the context of a full development life cycle. Components identified through the Rational Unified Methodology are specified in extensions to UML.	Rational Software www.rational.com
SELECT Enterprise	An implementation of the SELECT Perspective Method for component design supported by the SELECT Component Manager. More information on SELECT is found in Table D-6 Component Repository and Configuration Management.	SELECT Software www.selectst.com/component
SoftModeler / Business	Java tool for the modeling, design, and deployment of EJB and JavaBeans components. Supports UML notations for components (including Use Cases) and component distribution (local and remote methods and properties both in Class / Component diagrams and Sequence Diagrams).	Softera, Ltd. www.softera.com/products.htm
UNIFACE	Tool for component modeling for the UNIFACE application environment. No EJB/CORBA or COM/DCOM support.	COMPUWARE www.compuware.com

Table D-4: Tools for Component Modeling

D.2.3.2 Component Construction

Component construction tools support the process of creating the components themselves. This contrasts with component assembly tools that create an application by snapping together a series of existing components.

A complete component consists of the following elements:

- Model-based specifications.
- Native specifications (specifications in a specific programming language).
- Interfaces for the different component models it supports.
- A means of testing the component.
- A digitally signed certificate guaranteeing that the component conforms to the specification.
- Documentation for the interface, the services of the component, and the process by which the component was created.

Component construction tools are in their early stages of development. The programmer still does most of the work creating the component parts and gathering them together. Sterling's COOL:Gen can construct components using the CBD96 standard. Forte software can construct components for use in their own environment. Component construction tools from Sun Microsystems, IBM, and Sterling should be offered in early 1999. Table D-5 describes tools for constructing components.

Tool Name	Description	Vendor
[Internal Sterling Product Name]	Tool for model-based EJB construction. Not yet released as of the date of this publication.	Sterling Software www.sterling.com
AION	Integrated development environment for rule-based applications. AION components may include interfaces to CORBA, DCOM, IBM Encina, and BEA Tuxedo.	AION division of Platinum Technology www.platinum.com/products/appdev/aion_ps.htm
COOL:Gen	Tool for model-based development, closely aligned with TI/IEF and information engineering. Used as a tool to develop for the CBD96 component standard.	Sterling Software www.sterling.com
EJB Development Kit	Container generation tools, EJB server.	IONA Technologies (acquired from EJBHome in February 1999) www.ejbhome.com
JavaWorkshop and JDK	EJB construction tool.	Sun Microsystems www.sun.com

Table D-5: Tools for Component Construction

Tool Name	Description	Vendor
JBuilder	Visual Java development tools with capability of creating EJBs.	Inprise Corporation (formerly Borland) www.borland.com/jbuilder/
jBusiness	Encapsulates enterprise data and processes into Java-based Enterprise Business Objects (Novera-specific components) compatible with CORBA and EJB. Integrates with Sun NetDynamics application server announced in February 1999. The fate of the existing Novera application server is unstated. Special attention to the management of distributed applications. Designed as an application packaging, deployment, and management environment, not a Java development environment.	Novera Software, Inc. www.novera.com
jDeveloper	Java and EJB development environment.	Oracle Corporation www.oracle.com
OrbixStudio Graphical Server Builder	Components for CORBA-based middleware may be identified and constructed using the Graphical Server Builder within OrbixStudio. No UML, EJB, or COM/DCOM support.	IONA Technologies www.iona.com
PowerJ	Java development system including support for JavaBeans and (future) EJB.	Sybase Corporation www.sybase.com
StructureBuilder	A visual, model-based component and application development tool for EJB/CORBA and JavaBeans. Does not specify support for a component development methodology.	Tendril Software http://www.tendril.com
Visual Café Symantec	Development environment for Java and EJB.	Symantec www.symantec.com
Visual J++	Visual development environment for Java. Includes the many Microsoft extensions to Java that impact code portability.	Microsoft Corporation www.microsoft.com
VisualAge for Java	Java development and EJB construction tool designed specifically for large scale team development. It is built around a shared code repository.	IBM Corporation www.ibm.com

Table D-5: Tools for Component Construction, continued

Note: A growing market for EJB means the list of supporting tools changes often. A current list may be found at java.sun.com/products/ejb/tools1.html.

D.2.3.3 Component Repository and Configuration Management

Repositories are shared databases designed to facilitate component sharing and tool interoperation across the development life cycle. “Best-of-breed” tools can be selected for each situation when models can be shared among all the tools. Repositories also promote team-based development,

resource management, and dependency tracking. A well run, well-institutionalized repository is essential for component development, software reuse, and legacy integration.

A repository provides the necessary infrastructure for effective component management. Repository-based development allows the sharing of application files among development teams. Advances in middleware technology take this concept even further by promising the sharing of information (components) over the World Wide Web (WWW). Repository-based component management offers developers at multiple sites the ability to cooperatively manage their shared software assets and to thereby realize the full benefits of CBD. Key features of the repository that support component-based development include:

- **Synchronized Component Views** - the component specifications, models, source code and executable must be kept synchronized in the repository.
- **Dependencies and Compatibility** - groups of components used together may include dependencies that must be tracked and communicated to component developers and application assemblers.
- **Version Control** - versions of a component must be managed and tracked through the repository.
- **Backward Compatibility** - as a component is enhanced, its name must be tracked. The name must be changed if it is no longer compatible with earlier versions.
- **Publish, Catalog, and Notify** - publishing components allows easy search and retrieval for both the component specification and the various implementations. Notification linked to the publishing process enables developers and component assemblers to become aware of components in their area of interest.
- **Extended Search** - ideally, searches for components should extend across local boundaries to external repositories and custom developers.
- **Ownership and Change Control** - an owner who is responsible for the maintenance and support of the component must be associated with a component .
- **Component Usage Metrics** - the repository must track component usage for the purposes of licensing, version control, problem notification and “recall”, and enhancement notification.

Among industry repositories, the Microsoft Repository is a key driver in the application development market. The MS Repository offers support for UML and database schema models. It provides a point of integration for tools including application modeling, application testing, debugging, diagramming, performance testing, defect tracking. In the CBD marketplace, MS Repository is becoming the de facto standard for online component catalogs and specifications.

The MS Repository is based on an open database connection (ODBC) database, rather than an OO database management system (ODBMS). Some competitors in the pure object-oriented (OO) repository market are turning to ODBMSs as an implementation solution (e.g., IBM TeamConnection using ObjectStore and Unisys UREP using Versant). However, other vendors such as Oracle have had success building repositories on top of relational engines, so there is no clear advantage to an OO backend. Other competitors including Platinum Technology's

Repository/Open Enterprise Edition, Viasoft/R&O's Rochade, and Softlab's Enabler are not using ODBMSs, but instead rely on either relational databases or proprietary file systems.

Table D-6 provides a list of repository technologies.

Tool Name	Description	Vendor
MS Repository	Code and model repository, native to Microsoft development tools and NT; ported to MVS and Unix by Platinum Technology.	Microsoft www.microsoft.com
PLATINUM Repository	PLATINUM development and metadata repository.	Platinum Technology www.genevasoft.com/products/dataw/reoeetk1.htm
Intersolv	Code repository with maintenance tools.	Intersolv
Rational Repository	Code repository with maintenance tools.	Rational Software
TeamConnection	IBM's development repository uses the ODBMS backend ObjectStore.	IBM Corporation www.ibm.com
Universal Repository (UREP)	The Unisys software repository. Backend is the Versent ODBMS.	Unisys Corporation www.marketplace.unisys.com/urep/capguide/prodove.html
SELECT Component Manager	Component Manager enables developers to store, catalog, search and retrieve components. It includes a facility for registering interest in catalogs of components and for event notification. It links multiple repositories over an intranet or the Internet as well as to the MS Repository, Unisys UREP repository, and the Softlab Enabler.	Select Software www.selectst.com/component/Default.ASP

Table D-6 Component Repository and Configuration Management

D.2.3.4 Component Assembly

Creating applications from components is referred to as *component assembly*. Component assembly combines the building of the new application's workflow with creating the user interface, connecting to legacy applications, and attaching to services provided through middleware and other enterprise service providers.

There are several ways to accomplish component assembly:

- Certain small projects may begin with component assembly. If components can be found by designers, or in some cases by business users themselves, that implement clearly understandable business-oriented interfaces, then assembly may be performed automatically by tools that “wire together” the components to produce an application.
- Complex applications may apply “glue logic” to underlying frameworks and components. Skilled developers would be required to operate advanced CASE tools, OO development systems, or other component assembly tools that permit linking components using scripting languages. For example, MS Frontpage, a layout and component assembly tool for Web applications, uses Visual Basic to define glue logic around ActiveX controls and components.

- Workflow management applications may be assembled by providing a workflow manager with the details of the business process it is required to support. Within the workflow manager, human and computer activities are fully qualified and linked to the underlying software components that provide application support.

Tool support is rudimentary. Most component assembly occurs in development tools, which can limit the choice of components to those created by a particular tool. Many of the component development tools listed in Table D-5: Tools for Component Construction, **continued**

Tool Name	Description	Vendor
CBToolkit	Component-based application development environment. Generates the code necessary for components to be managed in the Component Broker environment.	IBM http://www.software.ibm.com/ad/cb/cbfactj2.html
UNIFACE Assembly Workbench	Assemble and manage components of the following types: UNIFACE, Visual Basic, Java, 3GL, CORBA and COM/DCOM. Displays detailed component interaction diagrams	COMPUWARE www.compuware.com
Visual Studio	Visual environment for component development. Integrated with the Visual Component Manager for finding, tracking, and cataloging components.	Microsoft Corporation msdn.microsoft.com/vstudio/downloads/solutions.asp
Visual Concepts	Visual Concepts is a repository-based tool for assembling ActiveX, CORBA, and Java components visually. EJB support is planned for the future.	SuperNova www.supernova.com
Conductor	Visual application construction tool for Forte components. Includes EJB/CORBA.	Forte www.forte.com

Table D-7: Tools for Component Assembly

D.2.3.5 Component Integration

Component integration software solves the problem of connecting components of different component models across different platforms. Several types of software are used for connecting components. These include ERP packages such as SAP and PeopleSoft, transaction processing monitors (TPMs) such as BEA Tuxedo, and object request brokers (ORBs) such as Iona's Orbix. Combining messaging, transaction monitors, and object request brokers in one package, object transaction monitors (OTMs) include BEA's WebLogic Enterprise, IBM's Component Broker, and Microsoft's Transaction Server (MTS).

Complex environments with several standards in place may require more than one or two integration tools to connect all the needed systems. Supporting several of these tools is a technical challenge. Vendors are gradually gravitating to a small number of standards, but this consolidation will take time.

Table D-8 presents tools for component integration.

Tool Name	Description	Vendor
BEA M3	CORBA ORB with CORBA Object Transaction Service, compatible with Tuxedo; Java based; EJB interface.	BEA Systems www.beasys.com
BEA ObjectBroker	Implements Object Bridge to enable bi-directional communication between DCOM and CORBA. Refer to the entry for Object Bridge in this table for more information.	BEA Systems www.beasys.com
BEA Tuxedo	TP monitor includes an EJB container; includes connectors to other component models	BEA Systems www.beasys.com
BEA WebLogic Enterprise	Was WebLogic; started life as a deployment platform; WebLogic Enterprise is a merge of the BEA WebLogic Web application server with BEA M3™ object transaction manager (OTM). The BEA WebLogic Enterprise provides advanced Java development services, Web integration, support of CORBA and Enterprise JavaBeans (EJB).	BEA Systems www.beasys.com
CBConnector, CBToolkit	Component Broker Connector and the supporting Component Broker Toolkit; middleware and application development technology. EJB support	IBM www.software.ibm.com/ad/cb/cbfactj2.html
WebEnterprise	Integration adapters to packages, host systems, COM/DCOM, CORBA/IIOP, JavaBeans, EJB, BEA Tuxedo, IBM Encina and MQSeries. Includes Forte Application Server technologies.	Forte Software Inc. www.forte.com
Inprise	Application integration engine from Borland. It was first developed as a deployment platform.	Inprise www.borland.com
Jacada Connects	Jacada Connects to Domino and HTML.	CST Inc. www.cst.com
Jaguar CTI	TP monitor serving as EJB container; connectors to other component models	Sybase Incorporated www.sybase.com
MTS	Microsoft Transaction Server	Microsoft Corporation www.microsoft.com
NetDynamics	Java application server with interfaces to CORBA, SAP, PeopleSoft, RDBMS, and other enterprise data stores. NetDynamics began as a Java and EJB deployment platform, though its system integration capabilities are widely used today.	Sun Microsystems www.netdynamics.com (Sun purchased NetDynamics in 1998)

Table D-8: Tools for Component Integration

Tool Name	Description	Vendor
Netron Frameworks	A framework for distributed systems. One may produce components from legacy COBOL or use Netron to derive business rules from code. Components are specific to Netron. Netron connectivity framework and custom application framework are proprietary.	Netron, Inc. www.netron.com
Object Bridge	Mechanism for bi-directional communication between DCOM and CORBA. Other Visual Edge products link business components with SAP.	Visual Edge www.vedge.com
Orbix 3	CORBA middleware	IONA Technologies www.iona.com
OrbixOTM 3	Enterprise middleware suite	IONA Technologies www.iona.com
Persistence PowerTier	PowerTier Live Object Server for EJB is an application server using the same technology as Persistence PowerTier C++. Native connectivity to databases.	Persistence Software, Inc. www.persistence.com
UIE	Universal Integration Engine, an EAI solution. EJB support in the future	SuperNova www.supernova.com
UNIFACE	UML-based repository that links UNIFACE to leading object modeling and CASE tools such as Rational Rose.	COMPUWARE www.compuware.com
UNIFACE Universal Request Broker	Integrates UNIFACE and non-UNIFACE components in distributed environments. Supports COM/DCOM, CORBA, and JavaBeans. Includes a Web application server; TP services through a link to Tuxedo and Encina.	COMPUWARE www.compuware.com/products/uniface/station/reading/read_arc.htm

Table D-8: Tools for Component Integration, continued

D.2.4 Methodologies for Component Design

Component design is the process of identifying reusable chunks of functionality within a large software system. The three central questions of component design are:

1. What functions within a system should be grouped together into a component?
2. What are potentially useful interfaces for the component?
3. How well can the interface and its functionality be used in future systems?

Designing components requires attention to three views of the system: the what, who, and how. These are described as follows:

- **What** - describes the behavior of system elements participating in some joint activity. To describe any single element (or component), its context must be understood. This level of description can represent the *design* for some higher level. At the high level, this is domain modeling.
- **Who** - determines and assigns responsibilities among the participants and specifies the interactions underway during each joint activity. This description leads to defining the services provided by each responsible entity. These services are composed into collaborations that implement the joint activities (now called *transactions*).
- **How** - refines these collaborations and specifies the implementation of services for each component. A description of these implementations can be refined further with *what*, *who*, and *how*. Successive refinement of *who* and *how* together is the process of development.

These three system views together describe a Use Case. A Use Case describes the joint behavior of a set of objects or components. It may be refined both in terms of the granularity of interactions of the objects, as well as in terms of the actual objects providing the services required. The rigorous specification of Use Cases, as formalized in this what/who/how model, leads to the well-specified interfaces required by components. Component modeling is performed within a larger context of component-based software development. Component modeling is the first step in the component life cycle. Figure D-2 below, presents the interconnections among the stages of component modeling and construction.

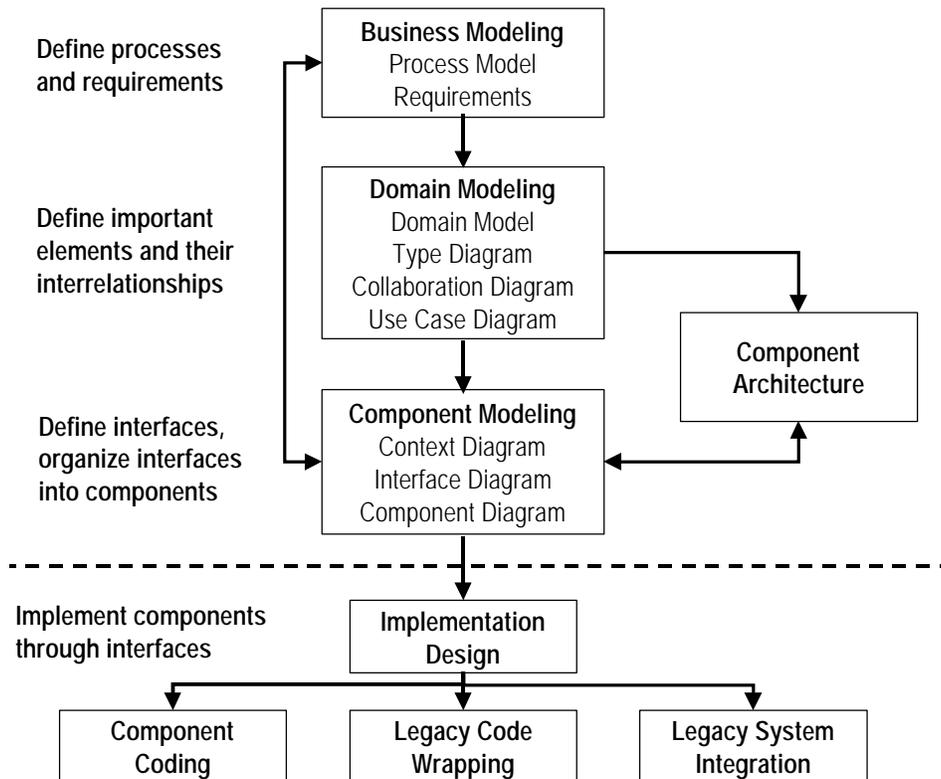


Figure D-2: Component Modeling and Construction

Each element of Figure D-2 is described in the subsections that follow.

D.2.4.1 Business Process Modeling and User Requirements

This is not necessarily the start of every project, but when required, business processes and user requirements are modeled and recorded. This may include as-is processes modeling and documentation of the supporting applications, as well as any number of phased, to-be models of re-engineered target processes and applications.

D.2.4.2 Domain Modeling

Domain modeling begins the process of determining reusable parcels of services. An OO approach is used to understand requirements and translate these requirements into interface descriptions. Tasks within Domain Modeling include Use Case Modeling, Collaboration Modeling, and Type Modeling.

When objects model the domain, and their behavior is observed and analyzed during key events and actions for the domain, interface specifications can be derived and refined as the basis for component specifications. As the interface specification model is constructed, sufficient detail is recorded about the behavior of the operations to allow matching against catalogs of already defined interfaces (i.e., components). When a close match is found, the interface that has been previously defined may be reused as part of the specification of the interface under construction.

As the interface specification is passed to teams who will implement the interface, they check for enough detail for unambiguous construction. In effect, the interface is the contract between the implementation team and future users of the component. These activities are enhanced by two kinds of reusable specifications: (1) domain model patterns called *frameworks* containing generic collaborations and types, and (2) already-published interfaces used in specifying new ones. When an already-published interface is reused, the designers can look for potential component reuse.

D.2.4.3 Component Modeling

During component modeling, the overall architecture of the application is defined. Interfaces will first be allocated to components. Interface catalogs will be consulted to find components that support them. The protocols of detailed collaborations will be translated into interactions between the components comprising the applications. Extensions to specification models of interfaces for existing black and white box components may be made here. Architectural design patterns delivered frameworks containing generic component collaborations will be used to guide component architecture design.

D.2.4.4 Implementation Design

Once the overall component architecture is decided, implementation projects may begin to deliver the required components. Three main categories describe these projects:

- **New build projects**, in which new components are built using software development tools. If an OO programming system is used, classes will be designed to do the job, perhaps assisted by OO design tools, and various class libraries. Design patterns delivered as framework models and code can speed up the process. If an OO implementation approach is not viable, other mechanisms will be used to accomplish the

implementation to fit the interfaces specification (for example, a procedural Information Engineering-based approach).

- **Acquisition projects** to outsource, purchase, or lease components that meet interface specifications that cannot, or should not be built with in-house resources. This may involve browsing or searching in component catalogs. Any wrappers or extensions to acquired packages and components will be designed and implemented at this time.
- **Legacy harvesting projects**, in which use is made of existing application resources to satisfy the needs of new applications. This may be part of a transition strategy, in which older application and packages are analyzed and then discarded, wrapped, or re-engineered to produce components that support interfaces required by the emerging, newer applications.

What has been described above are the ideas behind the Catalysis method for component design. Catalysis is the leading methodology for the design and development of components. Catalysis differs from most OO methods in the way it defines components. This is important since, as observed earlier, a component is not necessarily an object. Clear definition of services is essential to good component design. Catalysis focuses on the definition of each component's services, not just their interactions as object methods typically do. The result is clear, specific service definitions. The Catalysis process, however, is not a full life cycle methodology. The method begins at the creation of domain models and ends just short of implementation. Therefore it meshes well with full life cycle models (e.g., Method/1), leaving analysis, implementation, and testing to the full life cycle methodology. This discussion leads directly to specific recommendations to integrate Catalysis into the CBD and life cycle management strategy of SFA.

D.2.5 Recommendations to SFA

Designing and creating components is a relatively new discipline and industry practice is still being defined and refined. CBD methodologies are generally an assembly of techniques closely monitored by technical experts in competing companies. To provide SFA with defensible recommendations for CBD, the following requirements were defined:

- The CBD approach and method should not be proprietary, although specific implementations may be proprietary.
- Supporting technical tools for the approach and method should be publicly available, preferably from a number of competing sources.
- Notations and symbols used should be publicly available.
- The CBD approach and methods should harmonize with current and planned ED guidance, recommendations, and standards.

Facing SFA is the problem of joining a CBD method with the ED/SLCDM. We recommend that the ED/SLCDM be supplemented in the area of component design and recommend Catalysis for this purpose. Catalysis is the dominant methodology for component design. Other approaches are less complete or tie component development too closely to object-oriented design.

In this scenario,

- ED/SLCDM provides the overarching SFA development life cycle, linked to management and reporting functions, and business and domain modeling is performed under the ED/SLCDM;
- component modeling and specification are conducted using Catalysis; and
- the ED/SLCDM is used for component implementation, taking component specifications through outsourcing, reuse, re-engineering, or development, into operation.

D.2.6 Tool Support for Catalysis

This subsection addresses tool support for Catalysis. Catalysis, described in the earlier subsection on component-based development, is a methodology for creating components. While it is not tied to OO methods, it does expand on OO design concepts and uses UML with extensions as its notation. In its initial form, Catalysis was a “paper and pencil” methodology without the support of a software package. Its author, Desmond D'Souza, created a software implementation of Catalysis when he worked for ICON Corporation. The package was acquired in 1998 by Sterling Software and was renamed COOL:Spex.

Catalysis is based on the notion of *interface-based design*. An interface is a contract between the supplier of the component services and the users. Like a contract, an interface specification must:

- be clear and explicit;
- provide a definition of terms in a common vocabulary;
- describe the roles and responsibilities of each part of the software involved; and
- explain the details of the functionality.

The component’s interface captures everything that its potential clients can rely on. Expressing the interface clearly and completely is essential to the component’s design, and is the reason that Catalysis focuses so heavily upon interfaces.

While Catalysis uses UML diagrams, most OO tools are not complete enough to support Catalysis without the use of workarounds. The usual mechanism for extending a UML tool is through UML *stereotypes*. Stereotypes would allow Catalysis modeling elements to be classified as a particular type by these general-purpose UML modeling tools. However, because of the following drawbacks with the use of stereotypes, general-purpose UML tools are not suitable for Catalysis:

1. The meaning of the stereotyped modeling elements cannot be defined because there is no underlying metadata.
2. Additional modeling semantics cannot be understood. For example, Specification Types cannot have operations but Classes can. Ensuring correctness of the model would require additional code solely to support the stereotyped element within the tool.
3. Because of the way stereotypes are stored, typographical errors result in more than one concept defined.
4. Code generators in UML tools would not recognize the Catalysis component specification.

For full use of Catalysis, the following UML diagrams are required: Component Architecture, Collaboration Diagram, Use Case Diagram, Interface Diagram, and Type Diagram. Each of these diagrams is discussed below.

- A Collaboration Diagram captures and models the dynamic behavior of a domain, including actions between elements of the domain.
- A Interface Diagram specifies the detailed underlying behavior of an interface including pre- and post-conditions.
- A Type Diagram is visual modeling support for recording the elements of interest within a domain. Components may be visually identified in their environment. Through the Type Diagrammer, terms common across an interface can be identified and defined as a contract.
- A Component Specification by Interface matrix documents which component specification offers which interfaces.
- A Collaboration by Collaboration matrix documents which collaboration is a refinement of another collaboration.

Tools supporting part or all of Catalysis are described below.

COOL:Spex

COOL:Spex is a product of Sterling Software, acquired in 1998 from ICON Corporation. It extends the Catalysis methodology with elements of the Sterling Advisor methodology, covering business modeling to code generation.

The model-based approach embodied in COOL:Spex has the advantage that the specification of components can be very precisely defined. Also, the semantic relationship between the specification's operations and type model is tightly integrated. Furthermore,

- Models of interfaces provide more easily understood semantics of behavior.
- Interfaces with models are easier to inspect to identify if they might be useful behavior bundles.
- Components whose interfaces have model information are self-describing.
- The impact on a model using someone else's interface is easier to gauge if that interface has a (semantic) model.

Component specifications are accessed through a repository, which, in the case of COOL:Spex, is the MS Repository.

Rational Software

Rational's Unified Process is a framework that covers the complete software life cycle. The solution, based upon components, relies heavily on UML. Rational Rose 98 includes the Behavioral Elements package, a set of extensions to UML. This set of UML extensions supports the Catalysis methodology for component modeling.

Rational adds these diagrams through their Unified Process:

- Use Case Diagrams
- Class Diagrams
- Component Diagrams
- Deployment Diagrams
- Sequence Diagrams
- Collaboration Diagrams

- Statechart Diagrams
- Activity Diagrams

The web site for Rational Software is www.rational.com/products/rup

Platinum Technology

The Platinum Technology Inc. PARADIGM Plus is a process framework and tool that, through UML extensions, supports component development and can be used with the Catalysis methodology. Desmond D'Souza, the inventor of the Catalysis method, now works for Platinum Technology, Inc.

The web site for Platinum Technology is www.platinum.com.

Select Software Tools

The SELECT Perspective Method is a general component design method supported by the SELECT Component Manager. The SELECT Perspective incorporates a collection of industry best-practice modeling techniques that are applied and adapted using process templates within an architectural framework across a wide range of developments in a component-based setting. SELECT is in the process of merging Catalysis concepts into their own component methodology. The SELECT web site is www.selectst.com/component.

D.3 Introduction to UML

The Unified Modeling Language (UML) is a common modeling language for building software systems. This subsection introduces modeling and UML, and why UML is important to Blueprint delivery within SFA:

- Subsection D.4.1 Modeling and UML
- Subsection D.4.2 UML and SFA Software Development
- Subsection D.4.3 UML Documents
- Subsection D.4.4 Terminology

D.3.1 Modeling and UML

Modeling languages and supporting tools specify, construct, visualize, and document the design and construction of a software-intensive system. Models are basically essential to successful software development in large organizations. Specifically, models function to fulfill the following.

- preserve requirements;
- facilitate communication;
- help manage complexity;
- capture essential parts of a system;
- specify business processes;
- define software architecture; and
- promote reuse.

The need for modeling software systems has been so great that dozens of modeling languages have been invented. Fortunately it is not necessary to learn them all. Over the last three years these modeling languages have converged in the development of UML. UML has captured virtually all of the important work in modeling, with special emphasis on OO and CBD methods. UML includes the methods and notations of the three important thought leaders in OO design: Grady Booch, Ivar Jacobson, and John Rumbaugh. UML includes elements for the Catalysis method for component design. The result of the merge of these modeling languages and methods is a single, widely applicable modeling language for users of these and other methods.

The UML meets the following four design goals:

- UML enables the modeling of systems (not just software) using object-oriented concepts
- UML establishes an explicit link between system concepts and requirements with the implementation of these concepts and artifacts in executable code
- UML addresses the need to manage problems of scale in complex, mission-critical systems
- UML defines a modeling language usable by both people and machines

In the next subsection, the link between the UML and software development is explored.

D.3.2 UML and SFA Software Development

Within the disciplines of software development, UML provides the following benefits to SFA projects:

- Applicability to different development life cycle methodologies, including the ED/SDLCM
- Support for SFA software architecture directions, including CBD
- Design for shared repositories to support software development in a distributed environment

UML focuses on a standard modeling language, not a standard process. Although the UML must be applied in the context of a process, different development projects require different processes. UML provides consistency over the different processes by providing a common modeling language (a common semantics) and a common notation (a picture of the semantics). UML fits well with development processes that are architecture-centric, iterative, and incremental. ED/SDLCM deliverables include diagrams that may be modeled with UML-based tools.

The UML specifies the notation and semantics for eight core diagrams employed in building software system models. These models are included as a reference point for readers who may be familiar with OO methods, and in order to link them with ED/SDLCM deliverables.

- **Requirement Diagrams** - Use Case and Class.
- **Behavior Diagrams** – State Chart and Activity.
- **Interaction Diagrams** – Sequence and Activity.
- **Implementation Diagrams** – Component and Deployment.

These diagrams provide multiple perspectives of the system under analysis or development. The underlying model integrates these perspectives so that an internally consistent system can be analyzed and built. These diagrams, along with supporting documentation, are the primary artifacts that a modeler sees, although the UML and supporting tools will provide for a number of derivative views.

UML provides the following capabilities to support the distributed software development environment planned for SFA:

- Modeling support for component technology, distributed computing, and frameworks
- A model interchange among a variety of software and design tools
- An interface to repositories for the sharing and storage of models and specifications

The latest release of UML includes the following new concepts:

- Extensibility mechanisms
- Patterns/collaborations and activity diagrams for business process modeling
- Concurrency and distribution for modeling ActiveX/DCOM, EJB and CORBA
- Refinement to handle relationships between levels of abstraction
- Interfaces and components
- Constraint language

Component-based development methods, including Catalysis, the Rational Unified Process, and the SELECT component development framework, exploit the new UML support for concurrency modeling, patterns, and interfaces.

D.3.3 UML Documents

As the sponsoring organization for UML, the Object Management Group (OMG) performs two major supporting functions. First, the OMG manages the standards development process by which the UML evolves. Second, OMB maintains the core UML documentation presented in the list that follows.

- **UML Semantics** - provides the abstract syntax, model rules and constraints, and semantics.
- **UML Notation Guide** - presents the UML notation with examples.
- **UML Extensions** - documents the extensions to UML for business modeling and for Rational Software's development methodology.
- **UML Standard Elements** - describes the UML elements and their interrelationships.
- **UML Glossary** - defines words related to UML and to system modeling in general.

These documents are accessible from the OMG web site, www.omg.org.

D.3.4 Terminology

Definitions for words and phrases commonly associated with objects and modeling are included in Table D-9 as a reference.

Term	Definition
Class	A programming language construct for defining how the state and behavior of a set of objects is implemented. A class may implement multiple types, and vice-versa.
Collaboration	A set of transactions which have some common purpose, with a common level of abstraction or detail, and involving objects playing different roles; often corresponds to a temporal relation between a set of finer-grained transactions which meets the specification of some higher-level transaction.
Component	An object, possibly complex, which has definite responsibilities assigned to it, and which will have an implementation to support one or more interfaces. Not all components will be implemented as instances of classes.
Design	A recursive process of refinement and decomposition of transactions: A distinct level of description that addresses how the required behaviors will be provided by some pattern of lower-level collaborations and finer-grained components.
Design Pattern	A proven design technique, presented with a discussion of its applicability and trade-offs, which suggests a transformation from a specification to the next level of design, or from one design to another.
Design Type	A type introduced to circumscribe some portion of the (next level of) implementation, with an interface of service transactions. Design types take part in transactions. Members of this type will be identifiable components in the implementation.
Model Type	A type introduced as part of a specification model, purely to support a specification. Model types do not directly take part in transactions, but they can have queries, invariants, etc. Also called specification type.

Table D-9: Modeling Terms and Definitions

Term	Definition
Object	An individual with identity and behavior; a member of some types; an identifiable component with an interface; an instance of a class.
Query	A hypothetical read-only function modeling the state of an object and used only in pre/post conditions; often depicted as a diagrammatic link or a typed attribute.
Role	A place for a participant in a collaboration; a view of an object from the perspective of some other object which collaborates with it. The mapping from design object to role is many-to-many at any point in time, and is dynamic.
Snapshot/ Instance diagram	A diagram of an instantiation of a type model at some instant in time, showing the interesting aspects of the objects, and the results of their specification queries (depicted as links and attributes).
Specification	A description of guaranteed behavior of some object, together with the conditions under which that behavior is guaranteed; often described with pairs of pre-conditions and post-conditions in terms of a specification model, sometimes with an associated temporal constraint.
Specification Model	A set of queries - called specification queries - which supports some specification; often depicted as types, attributes, and associations in a set of diagrams.
Specification Type	see Model Type
Transaction - joint and service	A unit of interaction or information exchange between participant objects playing some roles, with a specified effect on those objects. We support both joint transactions (multiple participants, described symmetrically with no distinguished receiver) and service transactions (attached to a distinguished receiver which is assigned responsibility for that transaction). Transactions can be refined in several ways. Our transactions do not require the atomicity and serializability properties required of traditional database transactions.
Type	A specification of externally visible behavior of objects - members of that type are all objects that conform to its behavioral specification. A type makes no statement about implementation.
Type Model	see Specification Model
Use case	A transaction which accomplishes a meaningful objective to an external user of a system; often a joint transaction, refined to describe the roles that the different participants play in the collaboration.

Table D-9: Modeling Terms and Definitions, continued

Source: Catalysis

D.4 Legacy System Components

In order to meet rapidly changing business needs, organizations will create new software components and assemble them into new applications. Just as they did when applications were evolving to client/server, managers are forced to address the question of how to best move their application portfolios forward to the new environment. Making the change must be gradual and carefully managed. Not every legacy system will make the transition. As desirable as it might be to convert a legacy application to components, there will be no justification without a business driver. Furthermore, the scale of legacy portfolios means that organizations simply can not begin again from scratch.

Fortunately, CBD provides a technical means to integrate legacy applications into new systems. Components make no assumptions about the underlying technology; components are about how the application's interfaces are designed, not about how the internals of the components are implemented.

To provide guidance for the choices for transitioning legacy systems to components, this section presents an overview of the major considerations:

- Subsection D.4.1 Objectives for Legacy System Components
- Subsection D.4.2 Design Choices
- Subsection D.4.3 Conclusion

D.4.1 Objectives for Legacy System Components

A legacy system can be integrated into component-based systems without necessarily making a wholesale conversion of the system to components. Five possible objectives drive the construction of components from legacy applications:

- **Reuse.** Through reuse, parts of existing applications are transformed into components and reused in new applications. A key design choice is whether the specific code is to be copied and reused in multiple systems, or whether the application logic is to be converted to a service and called from systems as required.
- **Integration.** Through integration, legacy applications are connected to new component-based systems. To accomplish the integration, an application programming interface is installed in the legacy code and made accessible to the external component application.
- **Replacement.** Parts of the legacy application are replaced with new or upgraded code, designed as components.
- **Enhancement.** New requirements force new functionality into the legacy system.
- **Encapsulation.** By inserting an interface, the service provided by a legacy application (the “what”) is separated out from the way the service is implemented (the “how”).

Clearly, converting an entire legacy application to components would meet all of the above objectives, but it is usually not necessary to accomplish them all. It is more likely that only a few objectives are required at any given time, perhaps because resources are short, or the package is scheduled for replacement shortly.

D.4.2 Design Choices

Six different scenarios capture the technical means by which legacy systems are adapted for component technology. The scenarios are:

- **Migration** - the application is migrated to a new platform or technology. Unless the new technology is based upon components, this step contributes little to the component strategy.
- **Wrapping** - legacy code is reused by wrapping it in a component interface. Making use of a tool and this wrapped componentized code, the services can be included in new component applications.
- **Restructuring** - legacy source code is restructured into component interfaces and separate components.
- **Enhancement** - in the process of creating components, new functionality is added to the legacy code and presented through the component interface to other applications.
- **Legacy Interfacing** - an application making use of the legacy application calls it via one of its legacy interfaces.
- **Component Interfacing** - an application integrates directly with services presented by the new component interface.

Among these scenarios, wrapping, legacy restructuring, and component interfacing offer the best choices for code and service reuse. Integration is best accomplished through wrapping, restructuring, and either legacy or component interfacing.

D.4.3 Conclusion

Deciding whether a legacy system is candidate for component-based reuse begins with an assessment of how closely the system meets to-be business and technical requirements. A system becomes a candidate for legacy component harvesting when it has been through the Component Design Methodology of subsection D.2.4. With the components identified through that process, developers can choose the objectives and scenarios for implementing the legacy components.