

Prolifics

Technical Briefing

© 1998 by JYACC, Inc., 116 John Street, New York, New York 10038, USA

All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express prior written consent of the publisher.

For more information about Prolifics products, contact Prolifics at 1-800-458-3313 or via email at sales@prolifics.com.

Prolifics, JYACC, and JAM are trademarks of JYACC, Inc.

TUXEDO AND BEA ARE REGISTERED TRADEMARKS OF BEA SYSTEMS, INC. OTHER BRANDS AND PRODUCT NAMES APPEARING IN THIS DOCUMENT MAY BE TRADEMARKS OR REGISTERED TRADEMARKS OF THEIR RESPECTIVE COMPANIES.

CONTENTS

CONTENTS	ii
AN INTRODUCTION TO ENTERPRISE COMPUTING	1
What is the Prolifics Family?	1
What is Multi-tier computing?	1
JETNET	2
BEA TUXEDO	2
What is JetNet and the JetNet Manager?	2
What is Prolifics Client?	2
What is Prolifics Web Server?	2
What is Prolifics Application Server?	3
TRADITIONAL TRANSACTION PROCESSING	3
CLIENT/SERVER AND MULTI-TIER COMPUTING	5
The client/server approach?	5
Reasons to choose client/server?	6
Three models for client/server	6
The Remote Data Access Model - Simple Client/Server	6
The Database Server Model	8
Three-tier/Multi-tier: The Application Server Model	10
Why choose JetNet in Prolifics <i>Standard Edition</i> for distributed TP?	11
Why choose Tuxedo for a 3-tier solution using distributed TP?	12
DEVELOPMENT TOOLS FOR MULTI-TIER COMPUTING	13
3GL solutions	13
First generation solutions using client-only tools	13
Second generation solutions using complete client/server tools	14
PROLIFICS ARCHITECTURE	14
Prolifics in the multi-tier environment	14

Development components of a Prolifics application	15
Data management using the Visual Object Repository	15
Middleware Management using Prolifics' JetNet Manager	16
Procedural and Deployment Management using Prolifics' Interface File	16
PROLIFICS FEATURES	16
Speedy and Flexible Development	16
Full Web and GUI Development	17
Manageable Development	17
JetNet Feature Support	18
TUXEDO Feature Support	19
Deployment	19
Database Support	19
BENEFITS OF USING PROLIFICS	20
Rapid Development	20
Improved application quality	20
A better production system	20
DEVELOPMENT EXAMPLES	22
PROLIFICS SUPPORTED PLATFORMS	26

AN INTRODUCTION TO ENTERPRISE COMPUTING

What is the Prolifics Family?

The Prolifics product family is designed to help you meet the challenges of building and maintaining enterprise-class, mission critical applications. Prolifics is a complete solution for the rapid development of transactional applications that are your core business systems.

Prolifics applications can be deployed throughout the enterprise, on “super Thin” clients of Web Browsers, as well as GUI clients such as Windows, Macintosh, and Motif.

The product family consists of:

Prolifics Development Environment: The development environment is utilized for developing your transactional applications. Once installed, the developer has an almost unlimited amount of resources available to assist in building application objects in the Visual Object Repository. The Visual Object Repository enables breakthrough developer productivity by providing a central point for development of all your client and server components (utilizing a GUI environment) for an industrial strength, enterprise-class, mission-critical application.

Prolifics Web Server: The Prolifics Web Server is a stand-alone server environment, which, combined with your HTTPD, provides transactional application functionality to the Internet. The Web Server provides support for dynamic generation of HTML, and allows you to add your own VBScript, JavaScript, and ActiveX components to your pages from within the Prolifics development environment.

Prolifics Client: The Prolifics Client resides on your user’s workstations. It is used to deploy your application in a GUI environment on most popular workstations, including Windows (3.1, 95 and NT), Macintosh, Motif and UNIX character mode.

Prolifics Application Server: The Prolifics Application Server provides the deployment components required for your Prolifics application.

What is Multi-tier computing?

Multi-tier computing, also referred to as enhanced client/server, 3-tier, or “distributed applications,” is a flexible environment for distributing the three key components of an application:

- Data presentation and formatting
- Application logic, or business rules
- Data storage

A more detailed discussion of the 3-tier architecture is included below.

Multi-tier computing requires some form of *messaging middleware* that manages the communications between the client and the server. Prolifics is offered with two different middleware options, Prolifics *Standard Edition* that includes Prolifics *JetNet* middleware, and Prolifics *Tuxedo Edition* that is used with BEA TUXEDO (purchased directly from BEA).

JETNET

JetNet is the integrated middleware in Prolifics *Standard Edition*. A subset of BEA TUXEDO (see below), JetNet provides the connectivity between the various components of a multi-tier application built with Prolifics *Standard Edition*.

BEA TUXEDO

BEA TUXEDO is a controlled, manageable environment for building and running multi-tier applications. It lets you partition your application into logical components which can be distributed to run across several machines. It helps you see what is happening at runtime and tune for proper performance. TUXEDO also provides security mechanisms, failover and recovery, and distributed data control.

Because TUXEDO has these features, it is often used to develop distributed Transaction Processing (TP) applications. The characteristics of distributed TP are described below.

Prolifics *Tuxedo Edition* is the development tool of choice when the advanced features of BEA TUXEDO are required for the application.

What is JetNet and the JetNet Manager?

As described above JetNet is a subset of TUXEDO. It delivers many of the core functions of TUXEDO, such as asynchronous service requests, runtime load balancing and request prioritization, server replication, and peer-to-peer messaging. These features are necessary for building a high performance three-tier architecture. JetNet allows you to partition your application into components that can be distributed over several machines. However, JetNet does not provide the security mechanisms, support for domains, and some other high end features of TUXEDO.

JetMan, the JetNet Manager, is a graphical tool for managing JetNet. It simplifies middleware management so thoroughly that users of Prolifics Standard Edition need know nothing about BEA TUXEDO to configure and run Prolifics Standard Edition applications.

What is Prolifics Client?

Prolifics Client adds front-end capabilities to the client facilities of a distributed TP system. It allows you to construct screens, maintain visual consistency through the Visual Object Repository, access local databases, and attach application logic to events. Part of that application logic can consist of requests for service through the distributed TP system, requests that will be fulfilled by a remote server. To the developer, requesting the service looks much like a making a normal subroutine call, except that the code for the "subroutine" executes on a remote machine.

What is Prolifics Web Server?

Prolifics Web Server lets you deploy your transactional application over the Internet using the same screens and servers developed for your GUI application. Prolifics Web Server takes existing Prolifics GUI screens, dynamically converts them to HTML and serves them to user's browsers over the Internet. Prolifics Web Server can also serve Prolifics screens and

screen components to existing HTML, and can be enhanced with HTML, ActiveX components, JavaScript, and VBScript using Prolifics' ProActive Web extensions.

What is Prolifics Application Server?

Prolifics Application Server lets you construct a TP server visually—not requiring the use of C or C++ to code the individual services. (Although you can use C/C++ where appropriate.) Visual Server Development hides the complexity of server construction from the developer, making building servers much easier. Its built-in database interface capability lets you specify database interactions using Prolifics' screen editor instead of writing SQL. If you do need to write SQL you can write vendor-independent SQL so your application can be easily ported to another database, or vendor-specific SQL to take advantage of powerful internal facilities provided by your database vendor.

TRADITIONAL TRANSACTION PROCESSING

Many organizations rely heavily upon Online Transaction Processing (OLTP) systems technologies for their mission-critical applications. Among the industries that have traditionally made significant use of transaction processing are transportation and financial services. But OLTP is increasingly used in other areas, such as retail point of sale, manufacturing control, telecommunications, government, healthcare, insurance, and image processing.

Applications running in OLTP environments are characterized by:

- Large numbers of simultaneous users
- High throughput, volume, and performance
- Continuous, non-stop, real-time processing
- The need to provide highly secure access to data and detailed control over its availability
- The capability to maintain a consistent data view through any failures

The services necessary to create these environments are provided by software systems called *OLTP Systems* or *TP Monitors*. Originally TP monitors, such as IBM's CICS, were supplied only by mainframe vendors whose proprietary operating systems furnished many of the underlying security and data management capabilities required for OLTP. As a result, OLTP applications reflected the hierarchical, terminal-to-host architecture typical of mainframe applications.

Figure 1 and Figure 2 illustrate this traditional OLTP model.

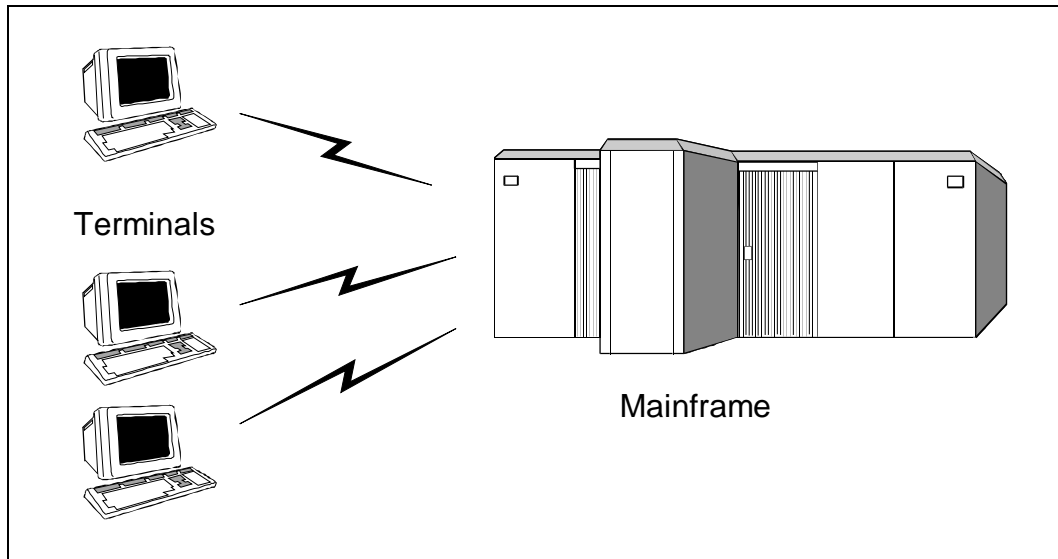


Figure 1: Mainframe OLTP Physical Model

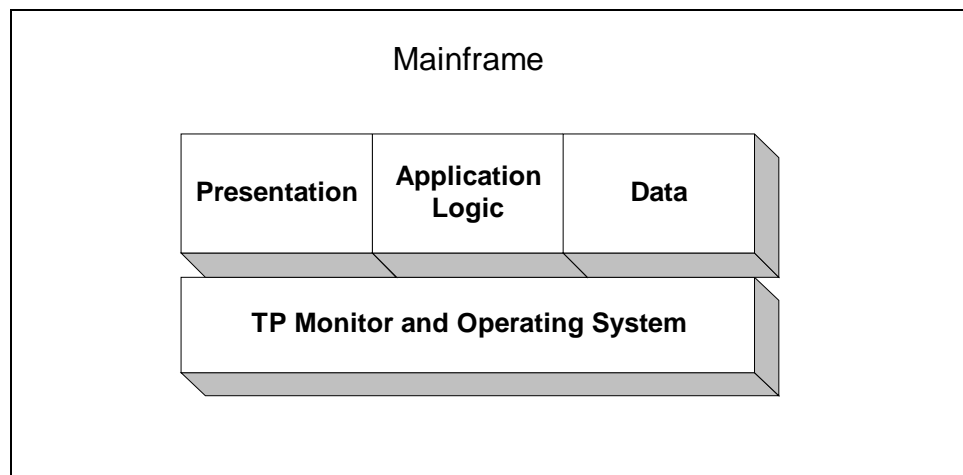


Figure 2: Mainframe OLTP Logical Model

CLIENT/SERVER AND MULTI-TIER COMPUTING

The client/server approach?

The client/server approach to computing recognizes that there are three distinct components to a typical application:

- Presentation and input services
- Application logic
- Permanent data store

Using the traditional model, these three components would all reside on the same machine. A semi-intelligent terminal might perform some limited presentation and input processing.

The client/server approach consists of distributing these three components across multiple machines. There are machines that are responsible for direct interaction with the user, drawing windows and data, collecting user input, and probably controlling the application flow. These are called client machines. Other machines that are responsible for management of resources (such as printers, disk drives, or databases) are called servers. Client machines run applications which request services from the server machines.

The physical architecture of client/server is illustrated below.

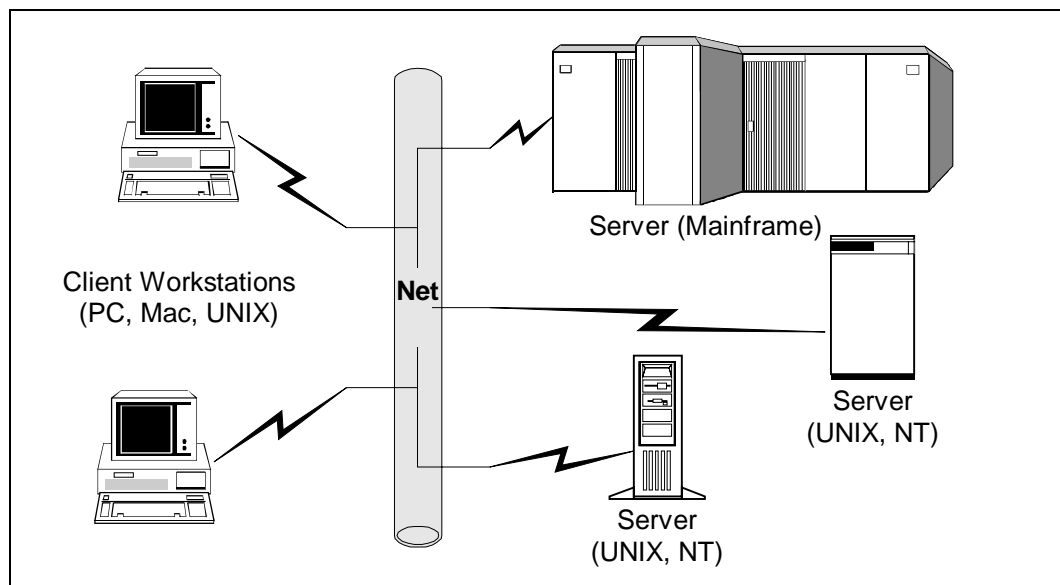


Figure 3: Application Server Physical Architecture

Reasons to choose client/server?

In general, the choice of client/server is driven by these advantages over single-computer solutions:

- **Flexibility** - The application logic can be spread across existing computer resources, regardless of location.
- **Performance** - Distributing the computing load can increase performance, particularly with respect to tasks related to the client. And performance can often be added incrementally by adding server machines.
- **Improved interfaces** - Client machines are typically workstations with superior (but computation-intensive) graphical presentation capabilities.
- **Better application integration** - Many common applications (such as spreadsheet programs and word processors) are oriented toward workstations. Using a client/server approach lets you integrate with these other workstation-based applications.
- **Lower cost** - In many cases the cost of a client/server solution is lower than that of a single-machine solution, because of the superior price/performance ratios of smaller computers.

Three models for client/server

There are several different styles, or models, of implementation for client/server. The differences between them relate to where the three main components of an application are distributed.

The Remote Data Access Model - Simple Client/Server

The Remote Data Access model splits the components between two places. One machine has the presentation and input services and the application logic, and another machine has the permanent data store (a database). This is a “data-shipping” model, sometimes referred to as 2-tier computing, and is the most commonly implemented of the client/server approaches. Figure 4 illustrates this model.

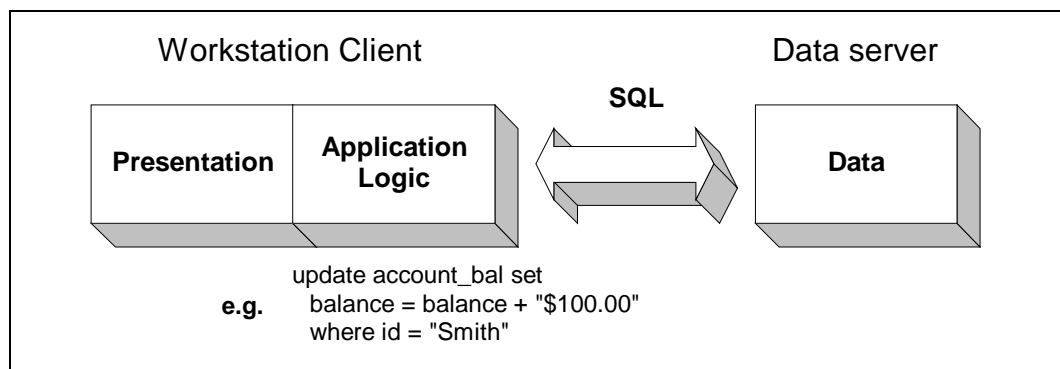


Figure 4: Remote Data Access Model

Advantages to Remote Data Access or Simple Client/Server

The advantages to this model relate to its simplicity: most of the application resides on a single machine (the client), yet that client can be a PC or workstation, with advanced graphics and input capabilities.

- The architecture is fairly simple.
- Development solutions, such as tools, are abundant and easy to use.
- Deployment is easy.

For these reasons, simple applications are best served with a Remote Data Access approach to client/server.

Disadvantages or limitations to Remote Data Access

There are several architectural and practical limitations to Remote Data Access. These include:

- **Performance** - Performance is often a problem with applications using Remote Data Access techniques. Each client consumes a substantial amount of server-machine resources, since it must have a dedicated connection to a database server machine. Often many database queries, each requiring many network packet exchanges, are necessary to perform one logical data exchange.
- **Security** - Since the application logic is wholly stored on the user desktop, and these desktops are not secure, the application is usually vulnerable. Data flowing over the network is not encrypted. Each user has a database login, and direct modification of application data using SQL is difficult to prevent. The application logic itself may become altered inappropriately.
- **Web Security** - For web-deployed applications security is a special problem. The remote data access model requires that the database API be accessible on your web server, which is outside your firewall. This essentially makes your data available to any hacker.
- **Development and deployment management** - Change management of these 2-tier applications is difficult. Since application logic is stored directly on the desktop, it must be reliably updated whenever the application changes. And the application logic is directly tied to the data model, so it has to change whenever the data model changes. As simple a modification as splitting the database table into 2 separate tables (perhaps for performance reasons) requires the application to be updated on all desktops.
- **Reliability** - Since each desktop client requires a direct network connection to the data server, Remote Data Access or 2-tier applications are vulnerable to network or machine failures. Handling failover to a backup machine or network path is left to the application developer.
- **Online management** - Online management of the application is very difficult. The networked solutions provided by the database vendor do not allow for bringing server machines up and down within the application in response to performance or maintenance needs. There is no way to migrate users to backup machines since they have inflexible, direct database connections.
- **Network Intensive** - The requirement to ship large amounts of data to the client (where the business rules to process it reside) can clog the network.

- **Limited communication mechanisms** - Only simple request/response communications mechanisms are supported. Asynchronous requests are limited or simply not supported. There is no provision for achieving parallelism by issuing multiple requests simultaneously. There is also no way for users to communicate with each other or for external events to interrupt the desktop application.

Consider a stock trading application where prices continuously change and certain users need to be notified of particular price changes. Using the Remote Data Access model requires an expensive and untimely polling mechanism to note changes, since there is no mechanism to send events or messages to particular clients.

The Database Server Model

The database server model circumvents some of the disadvantages of the Remote Data Access model by allowing logic to reside on the database server. This new logic is in the form of DBMS stored procedures. Figure 5 illustrates this model.

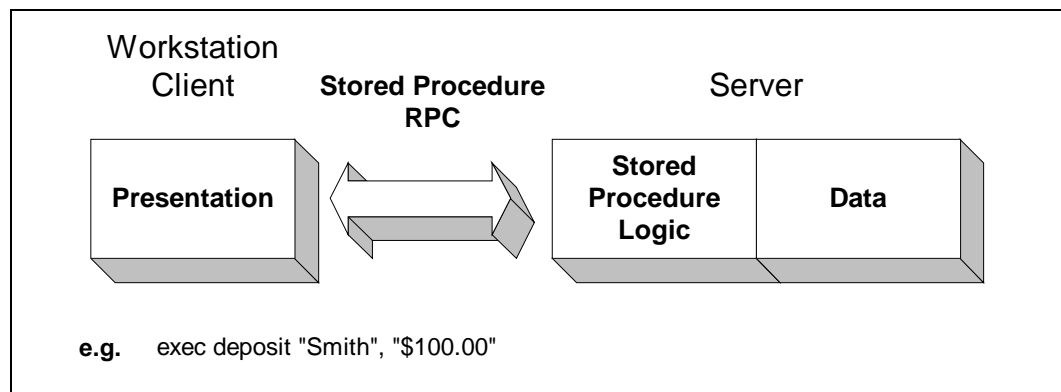


Figure 5: Database Server Model

Advantages to the Database Server Model

There are several advantages to using database stored procedures over direct table access:

- **Management** - Stored procedures can eliminate the dependencies between the data storage mechanism and the client application logic, providing a kind of data abstraction. This, plus the fact that some of the application logic can be stored in the database server, can make managing changes easier.
- **Performance** - Database stored procedures sometimes provide performance enhancements.
- **Security** - Stored procedures can improve security by moving logic away from the desktop and by allowing access to be determined by operation rather than broad-based data items.

Disadvantages to the Database Server Model

Problems still exist with this approach, arising from these areas:

- **Stored Procedure limitations** – Most database stored procedure implementations are so limited that complete abstraction does not exist. For example, when retrieving data using a stored procedure, the client may have to “know” how many tables are accessed to provide the data.
- **Limited languages** - Stored procedure languages have limited flexibility, and standard languages such as C/C++ or COBOL are no longer an option. Without access to 3GL languages you cannot extend the environment itself. For example, you cannot easily access 3rd-party mathematical libraries, external communication links, or non-DBMS data sources.
- **Database vendor lock-in** - Stored procedure languages are proprietary to the database vendor.
- **Performance** - In some cases stored procedures hurt application performance. They suffer from the fact that for optimal efficiency the procedures themselves must run on the machine where the data is located. In most applications the database server machine is the performance bottleneck, and moving extra processing there just adds to the problem.
- **Security** - Data still flows over vulnerable networks with no encryption. Authentication is provided by the database and cannot be altered.
- **Web Security** - The database API is still accessible on your web server.
- **Reliability** - The same problems exist as with Remote Data Access.
- **Online management** - The same problems exist as with Remote Data Access.
- **Limited communication mechanisms** - There are no new communications mechanisms provided by using stored procedures.
- **Increased complexity** - A new language for implementing stored procedures must be learned, and at least two languages must be maintained. Code must be developed on at least three different classes of machines.
- **Development tool support** - Tools do not support stored procedures as well as direct table access.

Three-tier/Multi-tier: The Application Server Model

The application server model, like the database server model, allows for distribution of application logic to either the client or the server or both. But it adds a new software layer that eliminates unreliable point-to-point connections in favor of an architecture that more closely resembles a software bus.

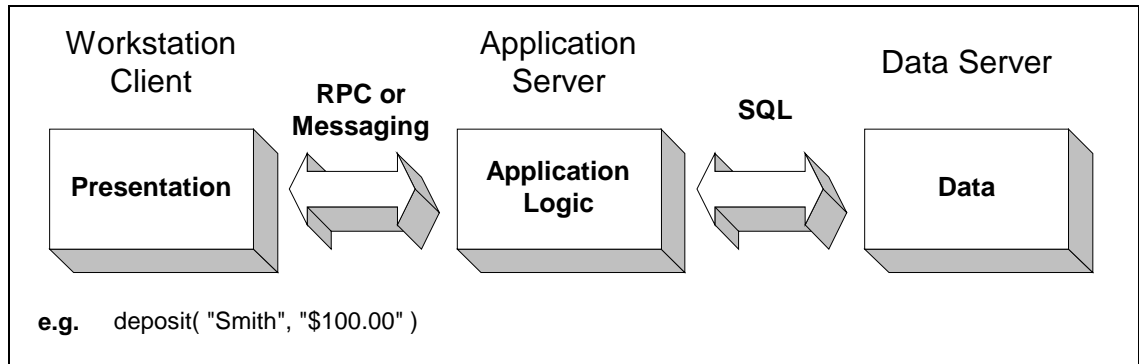


Figure 6: Three-tier Application Server Model

The physical architecture is similar to that of the other client/server models, except that it more easily allows distribution of the server component across multiple machines.

The logical architecture of this model is significantly different from that associated with the Database Server model. There is no point-to-point network connection; communication between machines occurs on an as-needed basis. A separate database server process is not needed for each client. A single server process can handle many clients, which often results in better performance.

BEA TUXEDO is an application development and deployment environment that uses this architecture to provide a distributed, transactional, high-performance, controlled runtime system. In this capacity, it can be used as a *distributed TP system*.

Advantages to the Application Server Model

The advantages to the 3-tier Application Server Model are:

- **Better languages** - Application logic can be implemented in expressive 3GL and 4GL languages, the same ones which are also used on the client.
- **Better application modularity** - Dependencies between clients and server data structures can be eliminated. This improves modularity, and thus decreases development time and improves quality and maintainability. Changes to your data model can be transparent to your client applications.
- **Performance** - There is no fixed per-client overhead on your server machine. Network bandwidth requirements can be significantly less.
- **No database vendor lock-in** - Server languages are database vendor independent. Client application logic is independent of the database entirely.

- **Development and deployment management** - You can build abstract interfaces between your data and your desktop clients.
- **Security** - Security mechanisms can be established for access to all types of processing, not just database access. You can link in data encryption where needed.
- **Web Security** - For web-deployed applications the Web Server is still outside the firewall, but the application servers that contain business logic are behind the firewall and thus within your security perimeter. Your database API is on the application servers, not the web server.

Disadvantages to the Application Server Model

Disadvantages to this model include:

- **Increased complexity** - You must manage an extra application component — the middleware itself. With Prolifics *Standard Edition*, however, management is simplified significantly by the JetNet Manager, *JetMan*.
- **Development tool support** - There are very few tools which support the Application Server Model at a high level. Prolifics solves this problem.
- **More Difficult to Test** – Because a single transaction is processed on multiple platforms debugging becomes more complicated. Prolifics includes tools that aid multi-platform debugging.

Why choose JetNet in Prolifics *Standard Edition* for distributed TP?

JetNet is a high-performance environment and is often the best choice as a basis for many business applications. Where all of the features of TUXEDO are not required, JetNet provides a cost-effective yet powerful middleware environment for your 3-tier or multi-tier application.

- **Performance** - Often, using a distributed TP system such as TUXEDO is the only way to achieve performance goals. Since TUXEDO supports the multiplexing of many client processes into fewer, context-free server processes, per-client overhead can be greatly reduced. Asynchronous requests and event brokerage can improve throughput and eliminate costly polling. *Load balancing* and *request prioritization* help with managing longer operations.
- **High availability** - Machine-to-machine failover, replication of data and services, and duplication of management processes create a highly available environment.
- **Online management** - Online management facilities are at the heart of JetNet. You can bring machines up or down within an application in response to changing usage patterns or maintenance needs. You can migrate processing from machine to machine. You can set up logging and tracing facilities.
- **Flexibility** - JetNet lets you move processing from machine to machine without affecting your running application.
- **Scalability** - JetNet supports *matrix scalability* — the ability to add heterogeneous resources without changing the application architecture.

- **Multiple communications mechanisms** - More communication mechanisms are available, such as synchronous or asynchronous request processing and direct messaging.

Why choose Tuxedo for a 3-tier solution using distributed TP?

TUXEDO is a feature-rich, high-performance environment and is often the best choice as a basis for mission-critical business applications. These are some of the additional features to consider in selecting TUXEDO as your middleware:

- **Improved request management** - BEA TUXEDO provides service request management features such as *request prioritization* and *data-dependent routing*. Data dependent routing is particularly useful because it lets you change the flow of requests depending on the data contained within the request. This is performed as a function of application configuration and is independent of application logic. For example, it makes it easier to split data sets (such as DBMS tables) into segments on separate machines.
- **Extended communications mechanisms** - Additional communication mechanisms are available, such as reliable queuing, request forwarding, and advanced event brokering..
- **Better security** - TUXEDO supports a variety of highly secure mechanisms for user authentication, access authorization, and data encryption. With a networked DBMS solution, verification of user identity is typically unreliable and prone to compromise. Often access to data cannot be controlled in a finely-grained fashion, and sensitive data is transmitted without encryption. TUXEDO's open architecture lets you solve these problems.
- **XA-compliant transactions** - TUXEDO supports XA standard calls.
- **Mainframe connectivity** - TUXEDO provides LU6.2 connectivity to CICS using the BEA Connect extension.
- **Cost** - TUXEDO usually provides distribution services less expensively than a DBMS networked option. A leading industry analyst estimates total system cost savings of greater than 30%, depending on system scale.

DEVELOPMENT TOOLS FOR MULTI-TIER COMPUTING

Despite the advantages of the multi-tier application server model, particularly for large applications, it has not been implemented nearly as frequently as the other two models. There are several reasons for this:

- Full-featured distributed environments are generally a recent development. (TUXEDO has been around for 10 years, but was originally developed for internal use at Bell Labs.)
- Companies are more cautious with larger applications.
- With the increased capabilities of multi-tier computing comes increased complexity.
- There are few tools available to make multi-tier application development easier.

As companies are becoming more ambitious with their application development, they are more and more often turning to 3-tier computing as a solution. They are faced with three main alternatives.

3GL solutions

Most existing multi-tier applications involve the writing of significant amounts of 3GL code to implement both client and server components. Typically, this is C/C++ or COBOL code. The limitations here are that every developer must be a proficient C or COBOL programmer, and must be familiar with the API offered by the multi-tier middleware vendor. Development in this type of environment is usually slow, difficult to manage, or simply unfeasible.

First generation solutions using client-only tools

The 1st generation solution to this problem is to attempt a marriage of existing client-oriented development tools with distributed TP environments such as TUXEDO. These client-only tools are typically database front-end tools that run under the Microsoft Windows environment. They allow part of the application to be developed without 3GL coding; however, they suffer these drawbacks:

- They do not help with developing application servers.
- They usually do not allow full exploitation of the TUXEDO's feature set.
- They do not have an efficient multiple developer environment, with a controlled repository and object inheritance.
- There is no built-in way of insuring data or procedural consistency among databases, application servers, and clients.
- They are oriented towards tight integration with a database, but since there is no direct database connectivity their methodology is inappropriate.
- They are oriented to smaller development efforts, and so lack general-purpose flexibility and large application control.

Second generation solutions using complete client/server tools

The 2nd generation solution is a complete one. It provides the functionality necessary to develop a complete application, including both client and server components, without resorting to C/C++ or COBOL coding. Equally important, it is aware of the capabilities of a distributed environment such as TUXEDO and allows for their exploitation. A distributed environment adds complexity that is often difficult to justify if the full feature set is not available.

PROLIFICS ARCHITECTURE

Prolifics in the multi-tier environment

Since Prolifics helps you build a complete distributed multi-tier application, it provides both client and application server components. Application logic can be stored either on the multi-tier client, on the application server, or both. You can even use database stored procedures in conjunction with a Prolifics application.

The figure below demonstrates the architecture of Prolifics

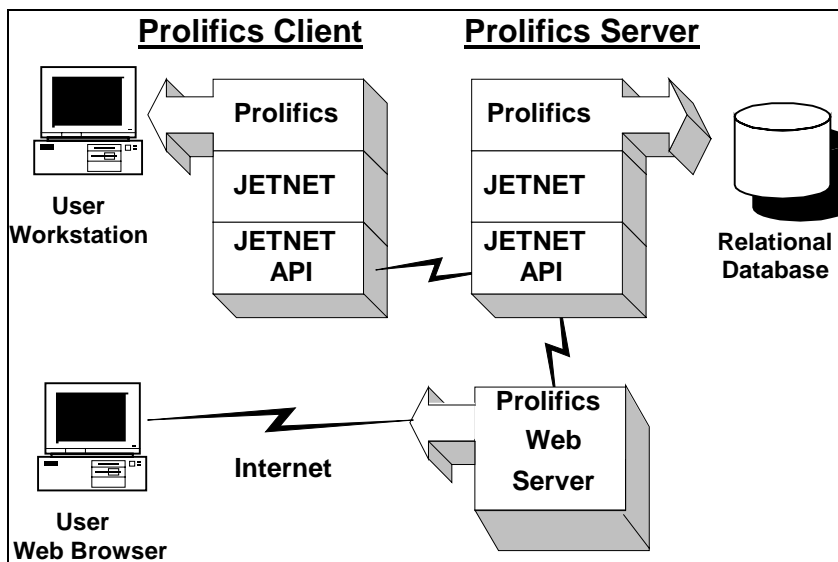


Figure 7: Prolifics Architecture

Both client and server components of Prolifics interact through the JetNet (or TUXEDO) API. Prolifics Client and Prolifics Application Server pieces are both full-fledged application clients and servers, capable of taking advantage of all the features of the environment, including the ability to request service from non-Prolifics servers and servicing requests from non-Prolifics clients. Prolifics is an open solution. The Prolifics Web Server acts as a “client” on behalf of the user with a web browser.

Development components of a Prolifics application

Within the environment provided by using Prolifics there are several key components.

- **The Visual Object Repository** is the heart of the Prolifics development environment. It centralizes control of data elements, ensuring consistency among client screen objects, server data objects, and the database. It inherits metadata from the database that describes tables, views, columns, and relations. The metadata is augmented with business rules and look-and-feel constraints so developers work with a common source of development components.
- **Client Screens** interact with the user. They perform most validation, consistency checking, and data formatting internally through non-procedural properties.
- **Client scripts** perform local procedural business logic.
- **The Prolifics Web Server** is the interface between the user's browsers and the Prolifics Application Servers. It is a "server" to the client web screens that are displayed on their browsers and a "client" to the Prolifics Application Servers.
- **Service containers** reside on the Prolifics Application Servers and contain the "business logic" of the application. They perform validation, consistency checking, and potentially data formatting. They map user data to database tables and columns to allow for automatic SQL generation. Service containers are created using a screen metaphor with the Prolifics screen editor and screen wizard. Service Containers are distributed deployment components and may reside on any server or on more than one server in a multiple server environment.
- **Service scripts** perform service procedural business logic and can control database interaction.

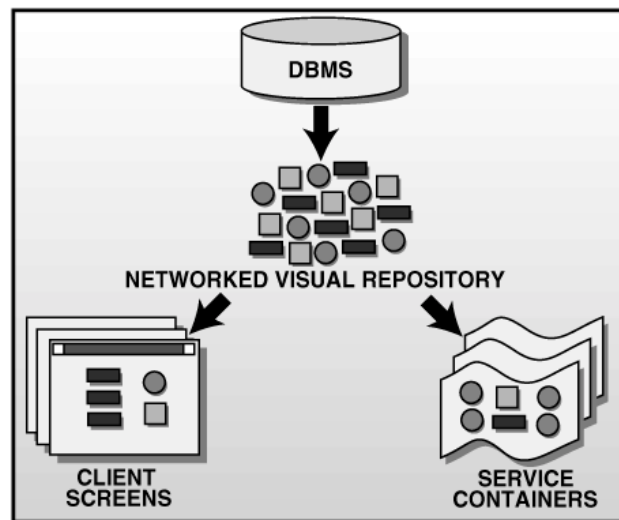


Figure 8: Prolifics *Development Components*

Data management using the Visual Object Repository

Using Prolifics' ability to import information from many popular relational databases, you can establish the basis of a team repository—Prolifics' Visual Object Repository. Other non-

database objects can be stored there as well. The Repository can then be used to build both client screens and service containers using Prolifics' Screen Wizard and drag-and-drop techniques. Whether these objects are on the client or server, they are kept synchronized, since they both can establish inheritance from a single Repository, on an attribute-by-attribute basis.

Middleware Management using Prolifics' JetNet Manager

Most multi-tier middleware software requires special training to manage the complex setup, configuration, and tuning procedures. Prolifics Standard Edition includes the JetNet Manager, an intuitive graphical utility for configuring JetNet. The JetNet manager allows you to configure servers, move services between servers and host machines, specify the number of servers that support each service, specify initialization and termination procedures, specify database connections, and generally manage the entire middleware layer.

And, if your requirements change and you find it necessary to move to full TUXEDO, your JetNet application is upwards compatible to Prolifics *Tuxedo Edition*.

Procedural and Deployment Management using Prolifics' Interface File

The Interface File is an application component which is edited with a graphical editor. It describes the service interfaces between Prolifics clients and servers, or between Prolifics and non-Prolifics clients and servers, letting you manage procedural consistency across your distributed application.

The Interface File also contains grouping information. Most commonly, groups within the Interface File are used to control deployment of services in the runtime environment. By simply assigning group membership you can cause services to be advertised to the application or removed from the advertised list. This can be done *dynamically* while the application is running. New services can be built and added to the system without bringing down any part of your application—a *feature unique to Prolifics*.

PROLIFICS FEATURES

Prolifics has all the features critical to development of large applications. Prolifics' flexible approach means that you can work using the methodology you want. Both top-down, Rapid Application Development and bottom-up, structured development approaches are available.

Speedy and Flexible Development

- **Rapid Application Development** - Traditionally, Prolifics has provided superb tools for rapid prototyping and development. Prolifics lets you run an application just as the user would, yet make changes to it on-the-fly, without compilation, linking, or re-setup of the application state. Turnaround time for application changes or bug-fixes is measured in seconds rather than minutes or hours.

Prolifics brings these same capabilities to the *Standard Edition*. You can add new services to the server component or make new calls from the client without any recompilation. Client changes can be made while running the application.

- **Screen Wizard for client *and* server development** - You can use Prolifics' Screen Wizard to build both the client and the server parts of your application. The Screen Wizard walks you through the process of defining a client screen or the data portion of an application service. On the server side, the Wizard builds server data objects which support complex transactions, such as nested master-detail/sub-detail transactions with multi-table updates—*without writing a single line of SQL*.
- **A 4GL-debuggable application server** - Using Prolifics Debuggable Server under UNIX you can interactively debug your services. This eliminates the need to fall back to a complex C/C++ or COBOL debugger environment to track down application problems.
- **Dynamic modification of service offerings** - Service offerings can be modified dynamically at runtime based on changes made to the Interface File.
- **Access from C/C++** - Full access to all the functionality in Prolifics is available from both JPL and C/C++.

Full Web and GUI Development

- **GUI and/or Web Deployment** – Applications developed with Prolifics can be deployed via the World Wide Web, GUI (Windows, Mac, and Motif), or any combination. Screens are created once and will run on any supported presentation platform.
- **WYSIWYG Web Pages** – Web-based applications can be written entirely with Prolifics. The screen you draw with the Prolifics screen editor is dynamically converted to HTML for display on a web browser. The same screen will work on a GUI workstation.
- **JavaScript and VB Script** – Prolifics screens intended for web deployment can contain JavaScript or VBScript event handlers.
- **ProActive Features** – Prolifics applications can be extended to take advantage of features of specific platforms.
 - **ProActive HTML** – Prolifics screens you create with the screen editor can be enhanced by embedding HTML in the screen or embedding the screen in a larger HTML page. You can also include only specific components from a Prolifics screen in the displayed page.
 - **ProActiveX** – Prolifics screens intended for GUI or web deployment can contain embedded ActiveX components. All properties of the ActiveX component are directly accessible to the Prolifics developer.

Manageable Development

- **Data control: the VOR** - Prolifics' Visual Object Repository with multi-level inheritance provides an excellent single point of control over the data elements in a large application. Prolifics' Repository is binary portable across all platforms and supports multi-developer access via automatic check-in/check-out. Plus, you can store your Prolifics Repository under external configuration management, such as SCCS and PVCS, if you need more advanced features such as versioning.

The Visual Object Repository can be used in a multi-tier application to maintain client/server consistency, and consistency with a DBMS.

- **Procedure control: Prolifics' Interface File** - Prolifics includes an Interface File. The Interface File is a central repository of service information that you can edit using Prolifics' graphical editor. This helps maintain consistency between services and their invocation by clients. It also allows grouping of services for easier assignment to server instances and better management of your application.
- **Many types of events** - Prolifics generates events at critical points during execution, events that can be handled by your own JPL or C/C++ routines. These events include:
 - Exceptions (informational, warning, and error events)
 - Receipt of unsolicited messages
 - Initiation and termination of service requests, including asynchronous ones
 - Dynamic advertising and unadvertising of services
 - Modification of the Interface File (through changing services or group memberships)
 - Termination of an application server

Using these events, you can control global application behavior, fine-tuning the way Prolifics behaves. You can also make these changes in a single place, reducing coding errors.

- **A hierarchy of installable event handlers** - For each type of event supported, Prolifics provides a hierarchy of locations where event handlers may be installed, from individual service requests to the application as a whole. Prolifics provides default handlers and commands to allow event chaining between hierarchy levels. This allows you to inherit general functionality yet override details on a request-by-request or transaction-by-transaction basis.

JetNet Feature Support

- **Request options** - Prolifics provides options for service prioritization, requests with and without replies, requests with and without time-outs.
- **Asynchronous service requests** - Prolifics lets you invoke services in both a synchronous and an asynchronous fashion. Multiple outstanding asynchronous requests are possible, and you can wait for them in several ways or let Prolifics handle service completion implicitly. Outstanding requests can also be canceled.
- **Prolifics-specific JAMFLEX buffer type** - Prolifics supports the JAMFLEX type, an easy-to-use structure that is transparently integrated with Prolifics.
- **Broadcast/notify support** (tpbroadcast and tpnotify) - Prolifics provides full support for the broadcast and notify capabilities of TUXEDO. Unsolicited messages from the current server (via a notify) or other clients (via a broadcast) can be handled by your own handler routines written in JPL or C.
- **Authentication** - Prolifics Standard Edition supports authentication via application passwords.
- **Log file support** - There is support for writing to the central TUXEDO log file.

TUXEDO Feature Support

If your needs go beyond JetNet, you can upgrade to Prolifics *Tuxedo Edition* and get the following additional features:

- **Rich TUXEDO feature support** - All the features of JetNet, plus: Prolifics provides runtime functions to help you integrate 3GL code which directly accesses TUXEDO's API.
- **FML, STRING and Prolifics-specific JAMFLEX buffer types** - Prolifics *Tuxedo Edition* supports access to the most important TUXEDO buffer types — FML and CSTRING. Plus it supports the JAMFLEX type, which is application code compatible with FML, yet easier to use since no FML field file is required.
- **Improved transactional control** - Full transactional capabilities of the distributed TP system are supported. Transactions can be demarcated with **BEGIN...END** blocks, allowing Prolifics to automatically generate **ROLLBACK** and **COMMIT** commands.
- **TUXEDO authentication** - Prolifics supports all three of the TUXEDO authentication mechanisms: UNIX system security, an application password, and full client authentication via a TUXEDO service.
- **Reliable Queues** – Prolifics *Tuxedo Edition* supports TUXEDO's queue management.
- **XA Compliance** - The XA standard is supported through TUXEDO.
- **Publish & Subscribe** - Prolifics provides native TUXEDO Publish & Subscribe support.

Deployment

- **Development flexibility vs. Deployment performance** - Prolifics provides different built-in behavior depending on whether you are developing or deploying an application. Built-in handlers respond differently, some allow easier on-the-fly changes, while others minimize system overhead and disk I/O.
- **Incremental movement to 3GL** - At the same time Prolifics provides an on-the-fly development environment, it also lets you take steps to fine-tune the performance of your application. Services can be defined graphically and coded in JPL scripting language, or moved over to 3GL code if performance needs improvement. In general, this will only be the case for services that do heavy computation (e.g., math calculations) rather than database access.

Database Support

- **Full database feature support** - For each database, Prolifics generally supports the broadest feature set, including stored procedures, all types of transactional control, and browse mode.
- **Cross-database portability** - Prolifics lets you import database information into the Repository and then add attributes and properties. From there you build screens using the Prolifics Screen Wizard and drag-and-drop techniques—without writing SQL code. The resulting applications are instantly portable across all supported databases.

If for some reason you do wish to write SQL, Prolifics provides features to help make that code portable as well.

- **Support for XA-compliant databases** - Prolifics has support for the major XA-compliant database products.

BENEFITS OF USING PROLIFICS

Rapid Development

The key benefit of using Prolifics is that you can quickly build reliable applications.

- **Less coding** - You end up writing vastly less code. Most of the logic of your application can be represented visually by building attribute-based objects. Any code you do need to write to represent your business logic is much simpler and more robust than the equivalent C/C++ or COBOL code.
- **Interactive, dynamic development** - You can build your application and immediately test it without going through any compilation or relinking. You don't even have to know TUXEDO administrative commands. The resulting development is fluid, responsive and quick.
- **Debugging** - Prolifics ships with an integrated debugger that lets you correct errors on-the-fly and retest your application.
- **Reduced training needs** - Because of the reduced complexity, your training costs are greatly reduced. You can have business analysts build parts of your application directly, rather than exclusively using C/C++ or COBOL programmers.

Improved application quality

The simplifications provided by Prolifics can make a huge difference in application quality.

- **Centralized control of vital functionality** - The Visual Object Repository and the Interface File, which can be used by both Prolifics Client and Prolifics Web and Application Servers, centralize information about the data objects and procedures in your application. Prolifics' hierarchical event handlers let you centrally control common programmatic behavior. These reduce application inconsistencies.
- **More reliable applications** - Since developers are using Prolifics' proven scripting language, JPL, and the higher level distributed JetNet interface provided by Prolifics, the chance of application crashes due to minor programming errors is greatly reduced. Prolifics detects most programming errors immediately and reports them so they can be fixed immediately—before the user finds them.
- **Better consistency with the database** - The Visual Object Repository, when used by Prolifics Application Server, can increase the consistency of your application with respect to the database.

A better production system

- **Improved application maintainability** - The centralized control provided by Prolifics lets you make global changes quickly and confidently. And less coding makes for quicker application updates.

- **Flexible logic partitioning** - Functionality can be quickly moved from client to server or vice versa.
- **Production-system performance** - Prolifics uses the native mechanisms provided by the distributed TUXEDO, insuring maximum performance. Linked-in support under Windows (rather than dynamic DLL lookups) means superior Windows client performance, especially with larger data sets.
- **Native-style TUXEDO services** - Some tools generate incomprehensible TUXEDO service names, or bypass the native TUXEDO messaging mechanisms. Prolifics Application Servers and services are natural, native-style TUXEDO entities, so you can manage them easily using TUXEDO administrative facilities.

DEVELOPMENT EXAMPLES

To provide an example of the power of Prolifics several samples are provided below of a realistic implementation of a familiar transaction: depositing an amount in a bank account, with a possible application error message generated by the server if something goes wrong. Note that these are very simple examples. Most real-world applications would contain more complex transactions for which the differences illustrated below would be even greater.

Figure 9 and Figure 10 illustrate the service request and implementation written in C. The client portion uses the API offered by one of the distributed TP systems. The client is still integrated with Prolifics for its display and control capabilities. The server portion uses embedded SQL with C.

Figure 11 shows this same request coded in JPL, Prolifics' scripting language, using Prolifics. All errors that could occur are automatically intercepted by Prolifics' default error handler, which displays a message and rolls back the transaction if one exists. This handler can be completely replaced, or individual exceptions can be overridden.

Figure 12 and Figure 13 implement the service using Prolifics with JPL. Figure 12 uses explicit SQL in the service, while Figure 13 uses automatic SQL generation through Prolifics' SQL Transaction Manager. The advantage with using Prolifics' SQL Transaction Manager is that the data is described on a Prolifics screen—created with Prolifics' graphical screen editor—and so can take advantage of direct importation of database information, plus inheritance from Prolifics' Visual Object Repository.

```

#include <stdio.h> /* UNIX */
#include <string.h> /* UNIX */
#include <fml.h> /* TUXEDO */
#include <atmi.h> /* TUXEDO */
#include <userlog.h> /* TUXEDO */
#include <deposit.h> /* FML fields */
#include "smdefs.h" /* Prolifics */

int client_deposit()
{
char *error_message;
char *receive_buffer;
char *send_buffer;
char *account;
char *amount;

/* Begin transaction */

if (tpbegin( 10, 0 ) < 0)
{
print_monitor_error( "tpbegin" );
return( -1 );
}

/* Allocate and load send data buffer */

send_buffer = tmalloc( "FML", NULL, 1024 );
receive_buffer = tmalloc( "FML", NULL, 1024 );
if ((send_buffer == NULL) ||
(receive_buffer == NULL))
{
print_monitor_error( "tpalloc" );
return( -1 );
}

account = sm_n_fptr( "ACCOUNT" );
if (account == NULL)
{
print_error( "Invalid field: ACCOUNT" );
return( -1 );
}
}

```

```

Fadds((FBFR *) send_buffer,
ACCOUNT,
account );

amount = sm_n_strip_amt_ptr( "AMOUNT", NULL );
if (amount == NULL)
{
print_error( "Invalid field: AMOUNT" );
return( -1 );
}
Fadds((FBFR *) send_buffer, AMOUNT, amount );

/* Execute the call */

if (tpcall( "DEPOSIT",
send_buffer,
0,
&receive_buffer,
&receive_length, 0 ) == -1)
{
if (tperrno != TPESVCFAIL)
{
tpabort( 0L );
print_monitor_error( "tpcall" );
return( -1 );
}
tpabort( 0L );
error_message =
Fvals( (FBFR *) receive_buffer,
MESSAGE,
0 );
if (error_message == NULL)
{
print_error(
"Invalid FML field: MESSAGE" );
return( -1 );
}
sm_emsg( error_message );
return( -1 );
}
}

```

Figure 9: 3GL Service Request Example

```

#include <stdio.h>      /* UNIX */
#include <string.h>     /* UNIX */
#include <fml.h>        /* TUXEDO */
#include <atmi.h>       /* TUXEDO */
#include <userlog.h>    /* TUXEDO */
#include <deposit.h>   /* FML fields */

EXEC SQL INCLUDE sqlca;

void DEPOSIT( transb )
TPSVCINFO *transb;
{
char amount_buffer[20];
char *return_buffer;
int status;
EXEC SQL BEGIN DECLARE SECTION;
double amount;
char *account;
EXEC SQL END DECLARE SECTION;

/* Allocate return buffer */

return_buffer = tmalloc( "FML", NULL, 1024 );
if (return_buffer == NULL)
{
userlog( "Memory allocation failure" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}

/* Unload input buffer */

status = Fget( (FBFR *) transb->data,
              AMOUNT,
              0,
              amount_buffer,
              0 );
if (status == -1)
{
userlog( "Invalid FML field: AMOUNT" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}

amount = atof( amount );
account = Fgets( (FBFR *) transb->data,
                ACCOUNT,
                0 );
if (account == NULL)
{
userlog( "Invalid FML field: ACCOUNT" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}
}

```

```

/* Execute embedded SQL */

EXEC SQL UPDATE ACCOUNT
SET AMOUNT = AMOUNT + :amount
WHERE ACCOUNT = :account;

if (sqlca.sqlcode != 0)
{
sqlca.sqlcode = 0;
status = Fadds( (FBFR *) return_buffer,
               MESSAGE,
               "Account update failed." );
if (status == -1)
{
userlog( "Invalid FML field: MESSAGE" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}
treturn( TPFALL,
        0,
        return_buffer,
        0L,
        0 );
}
if (sqlca.sqlerrd[2] == 0)
{
status = Fadds( (FBFR *) return_buffer,
               MESSAGE,
               "Invalid account number." );
if (status == -1)
{
userlog( "Invalid FML field: MESSAGE" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}
treturn( TPFALL, 0,
        return_buffer, 0L, 0 );
}

/* Load return buffer and return */

Fadds( (FBFR *) return_buffer,
      MESSAGE,
      "Update successful." );
if (status == -1)
{
userlog( "Invalid FML field: MESSAGE" );
treturn( TPEXIT, 0, NULL, 0L, 0 );
}
treturn( TPSUCCESS, 0,
        return_buffer, 0L, 0 );
}

```

Figure 10: 3GL Service Implementation Example

```
proc client_deposit
    SERVICE_CALL "DEPOSIT" ({ACCOUNT, AMOUNT}, {MESSAGE})

    if (@jam->tp_svc_outcome != TP_SUCCESS)
        msg emsg MESSAGE
    return
```

Figure 11: Prolifics Service Request

```
proc DEPOSIT
    RECEIVE ARGUMENTS ({ACCOUNT, AMOUNT})

    DBMS SQL UPDATE ACCOUNT SET AMOUNT = AMOUNT + :amount WHERE ACCOUNT = :account

    if (@dmretcode != 0)
        SERVICE_RETURN EXIT ({MESSAGE = "Account update failed."})
    if (@dmrowcount == 0)
        SERVICE_RETURN FAILURE ({MESSAGE = "Invalid account number."})

    SERVICE_RETURN SUCCESS ({MESSAGE = "Update Successful."})
```

Figure 12: Prolifics Service Implementation with Explicit SQL

```
proc DEPOSIT

    RECEIVE ARGUMENTS ({ACCOUNT, AMOUNT})
    call sm_tm_command( "SELECT" )
    if (ACCOUNT == "")
        SERVICE_RETURN FAILURE ({MESSAGE = "Invalid account number."})
    BALANCE = BALANCE + AMOUNT
    call sm_tm_command( "SAVE" )

    SERVICE_RETURN SUCCESS ({MESSAGE = "Update Successful."})
```

Figure 13: Prolifics Service Implementation with Automatic SQL

PROLIFICS SUPPORTED PLATFORMS

Both Prolifics Client and Prolifics Servers are available on the following platforms:

Platform	Operating System
BULL DPX/20 Series	AIX
Data General	DG/UX
DEC Alpha/AXP	OSF/1
HP 9000	HP/UX
IBM RS/6000	AIX
Intel x86	Windows NT UnixWare SCO Unix DG/UX
Sun Sparc	SunOS 4 Solaris

Many other popular UNIX platforms may also be available — check with your local sales representative for details.

In addition to both client and server support, Prolifics Client is available for MS Windows 3.1, Windows 95, Windows NT and Macintosh.

Prolifics**Corporate Headquarters**

116 John Street
20th Floor
New York, NY 10038
Phone: 212-267-7722
Fax: 212-608-6753

Prolifics Chicago

125 South Wacker Drive
Suite 300
Chicago, IL 60606
Phone: 312-214-4910
Fax: 312-214-4956

Prolifics Washington, DC

7799 Leesburg Pike
Suite 900 North
Falls Church, VA 22043
Phone: 703-847-8276
Fax: 703-847-8269

Prolifics Orlando

20 North Orange Avenue
Suite 706
Orlando, FL 32801
Phone: 407-420-9474
Fax: 407-420-9484

Prolifics San Francisco

One Sansome Street
Suite 2000
San Francisco, CA 94104
Phone: 415-951-1070
Fax: 415-951-1080

Prolifics France

7 place de la Défense
92974 Paris La Défense Cedex
Phone: (33) (0) 1-46-92-72-72
Fax: (33) (0) 1-46-92-72-70

Prolifics UK

4 Chiswell Street
London, EC1Y 4UP
Phone: (44) 171-786-9555
Fax: (44) 171-786-9556

International Distributors

Prolifics has distributors in the following other international locations:

Croatia • Finland • Germany • India
Israel • Italy • Mexico • Netherlands
Russia • Saudi Arabia • Slovenia
Spain • Sweden • Switzerland