

## **IBM TXSeries and Trifolium Technology**

Bill Ruchte  
Trifolium, Inc.

### **TXSeries and Trifolium Technology**

This paper covers the basic structure of the TXSeries version 5 product from IBM and how the Trifolium Vivo Service Engine is applied for use with this product. The intent is to provide a roadmap for the use of Trifolium technology in conjunction with the TXSeries infrastructure. Given the range of capabilities in each of these product suites, they could work together in many possible ways. Not all of these possibilities are mentioned here. Instead, the focus is on the strategies that Trifolium believes will provide the greatest advantage for most organizations. The following recommended applications of the underlying service engine are combined, packaged and distributed as the Trifolium TXExpress product.

### ***The Structure of TXSeries***

The TXSeries product from IBM provides an infrastructure for the development, deployment and management of distributed applications. The product has evolved as a combination of previously separate technologies, and the divisions between the original products are still largely present in the combined offering. This means that there are a number of fundamental choices to be made by an organization deploying TXSeries that will yield considerably different infrastructures and development challenges.

The main unifying concepts that tie the TXSeries constituent pieces together are: distribution of processing components and distributed resource coordination. Distribution of processing components means that the developer creates units of functionality that are requested as needed by some user or other process. The location, implementation and internal structure of these processing components are largely hidden from the requestor. This allows the infrastructure to manage the processing components without affecting the requestors. For example, multiple processes might be used to handle the volume of processing without the requestor being aware of more than one instance of the process. Distributed resource coordination means that the infrastructure supports mechanisms for various data sources to be impacted by the processing of one or more processing components, with the assurance that any changes will be consistent.

Fundamentally, TXSeries is composed of two different execution environments for distributed application logic: the Encina environment and the CICS environment. The Encina environment, in turn, has several options in how it can be used.

### ***TXSERIES-ENCINA***

The TXSeries-Encina execution environment supports the implementation of distributed transactional applications using a client/server model. The client interactions with the servers are handled using a remote procedure call (RPC) mechanism. This means that client processes refer to logic in the server as if it was a function call within its own program space. The actual execution of the logic is done on the server with any input parameters from the client sent to the server and any results on the server sent back to the client. This transport between the client and server takes place within the infrastructure with little or no impact on how the client code has to be written.

TXSeries-Encina servers are developed as stand-alone programs that make references to special Encina library functions. These library functions allow the server program to communicate with the Encina infrastructure and also to be controlled by Encina. The server program includes special generated code that provides the starting point for the functions that will be called by clients. The actual implementation of these functions is the work that the developer will focus on.

Encina handles the mechanics of locating servers on behalf of clients and binding clients to servers. It handles the transport of data between clients and servers. It handles the invocation of the developer's logic in response to the receipt of a function call at the server.

To support the ability to coordinate changes to multiple data sources, Encina implements an industry standard API known as the XA standard. The Encina libraries provide support for writing code that uses XA transaction capability. Third party vendors of data storage products implement the same standard, assuring that they can be part of Encina distributed transactions.

A complete Encina execution environment will consist of a number of processes that manage the infrastructure (communications, security, monitoring, etc), and some arbitrary number of server processes developed for specific purposes. These developed processes rely on being linked with Encina libraries in order to interoperate with the Encina environment.

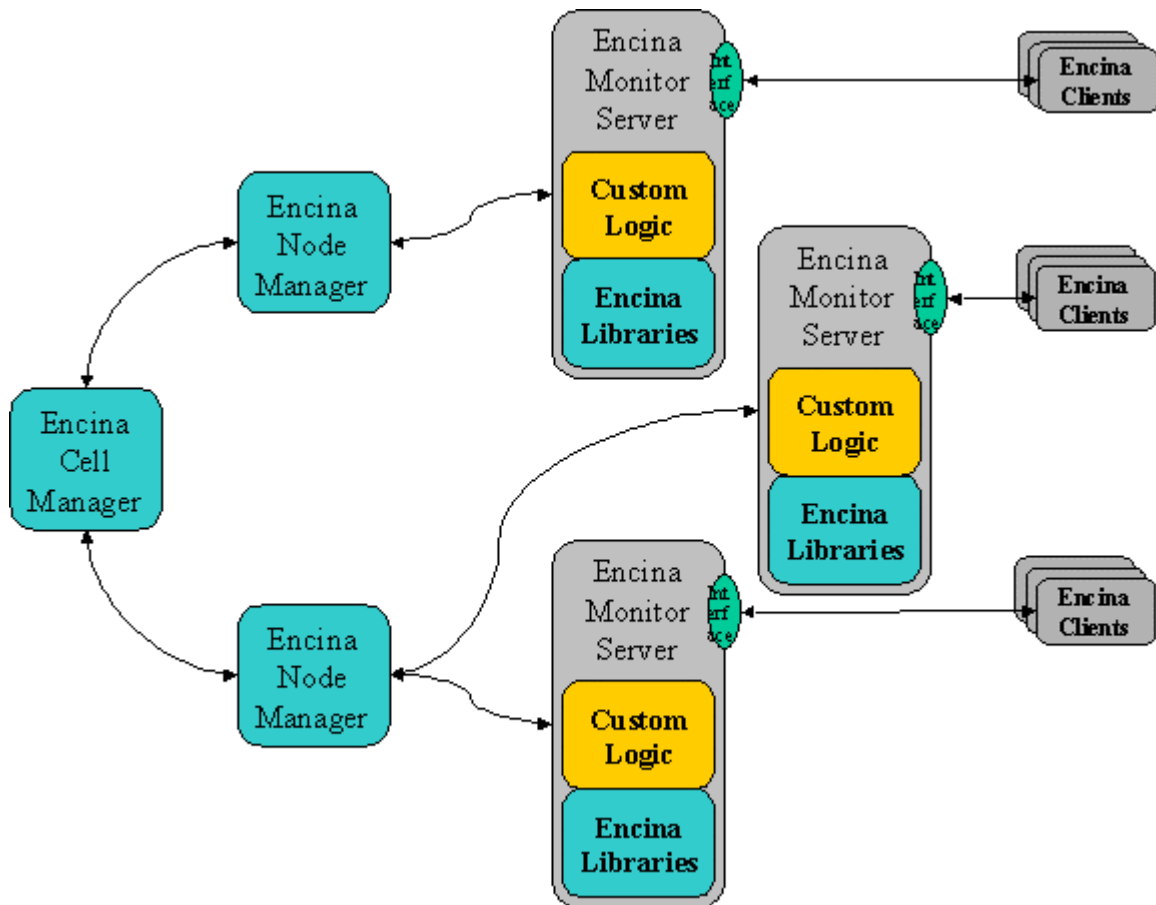


Figure 1 - Structure of a TXSeries-Encina Execution Environment

### TXSERIES-CICS

The TXSeries-CICS execution environment is essentially a complete processing environment supporting terminals that display processing results, engines that execute processing components, data storage mechanisms and other infrastructure.

The model used for distributed processing in CICS is more difficult to characterize. From the standpoint of the normal user, it is a distributed presentation model, where only a display terminal is needed to invoke the logic components. There are a number of other models that can be implemented, including distributing logic components across multiple machines or platforms, and peer-to-peer applications.

The CICS execution environment provides location independence for the processing components that are implemented. These components are mapped from request codes provided by the user, and executed by the environment to produce the result the user sees. Behind the scenes, there may be transfers of control to processing components that reside in other CICS regions.

Unlike Encina, where a new server is constructed to add function implementations to the environment, in CICS there is a pre-existing execution engine for developer-written functionality. The code written by the developer makes references to CICS functions that are provided by underlying libraries, but the program is compiled into a shared object format instead of into an executable. The CICS engine is capable of loading, linking, and running these developer-written shared object files as necessary to support the requests being made by clients. CICS provides built-in support for access to record-oriented data storage and queues. Access to additional resources can be written as part of the developer code that implements some functionality. Distributed coordination of changes to data resources is done through the use of the XA standard, just as with Encina.

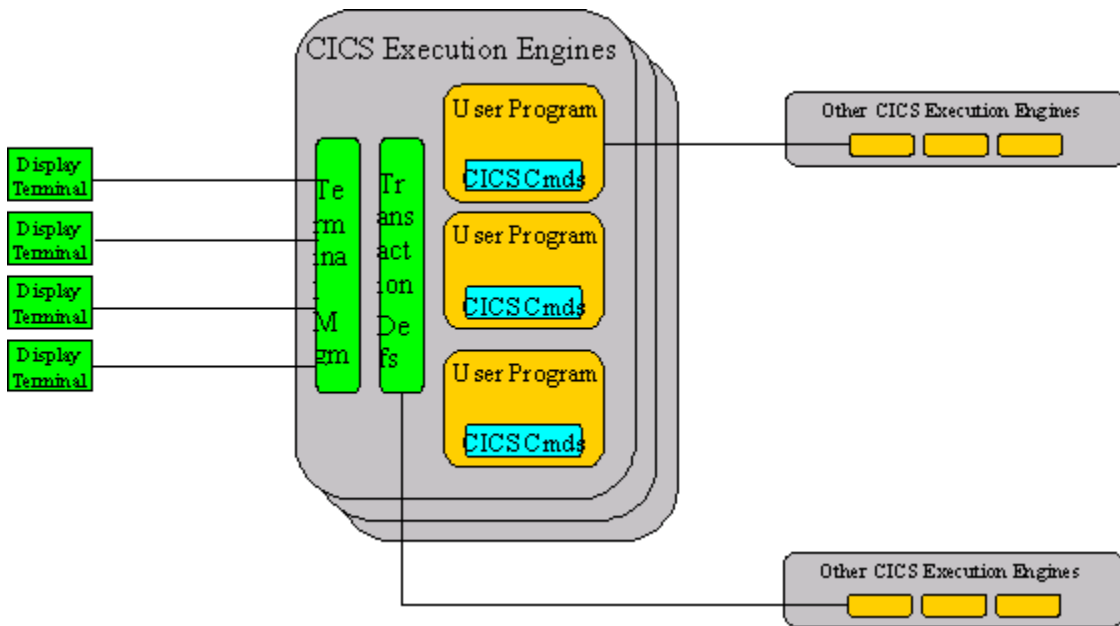


Figure 2 - Structure of a TXSeries-CICS Execution Environment

### The Structure of Trifolium Vivo Service Engine

The Trifolium technology discussed here is the Vivo service engine. Vivo is a complete application development environment based on Trifolium's development infrastructure, namely the Trifolium Enterprise Developer's Kit. The Trifolium EDK is not specifically discussed here. The EDK has been used to implement Vivo, but is not required for use of Vivo in development applications.

### TRIFOLIUM VIVO

The Trifolium Vivo product is an example of an advanced EDK processing engine with specific message handling and processing logic components built into it. The Vivo service engine includes an XML-based configuration system that allows a developer to map incoming requests to defined operations, and then configure these operations using Vivo components.

These components are of two basic types: flow-of-control components and processing components. Full sets of both types of components are built in to the Vivo Service Engine. It is also possible to extend the Vivo service engine with custom components.

The XML configuration system allows an operation configuration to be as simple as mapping to a single built-in component, or as complex as having nested loop and branching structures with multiple types of processing components. There is a sophisticated fault handling mechanism built into Vivo as well. The result is that a Vivo service engine can be deployed and configured to perform a wide range of processing needs with no code, no compiling and no special development tools. It is this notion of "programming" by configuring the sequencing and behavior of built-in software components that is unique to Vivo.

The Vivo service engine is available for use in various contexts. The basic division is between the server version of Vivo and the embedded version. The server version is an executable binary that is capable of running in various middleware environments and accepting client connections directly. The embedded Vivo service engine is a dynamically loadable library with entry points, where a containing program can pass requests to be processed by the service engine. The mechanisms for defining the processing logic are the same for either version of Vivo. A Vivo component program defines an operation available in a Vivo service engine. This component program is a set of configured Vivo components defining both a flow-of-control between components and the detailed behavior of each component. The Vivo flow-of-control components are configurable software modules that perform actions like looping, branching, switching, etc. The Vivo processing components are configurable software modules that perform message manipulation, data access, data mapping, host interactions, etc. Custom Vivo components can be built, either by Trifolium or by a user, to add functionality as required. Custom components are dynamically loaded and used by the Vivo service engine. The Vivo component programs are specified as an XML document where each of the possible components has a specific XML definition for specifying its options. There is fully annotated XML Schema support for all of the built-in components, so standard XML editors can aid development of the XML configurations for Vivo.

The following diagram represents the structure of a Vivo service engine.

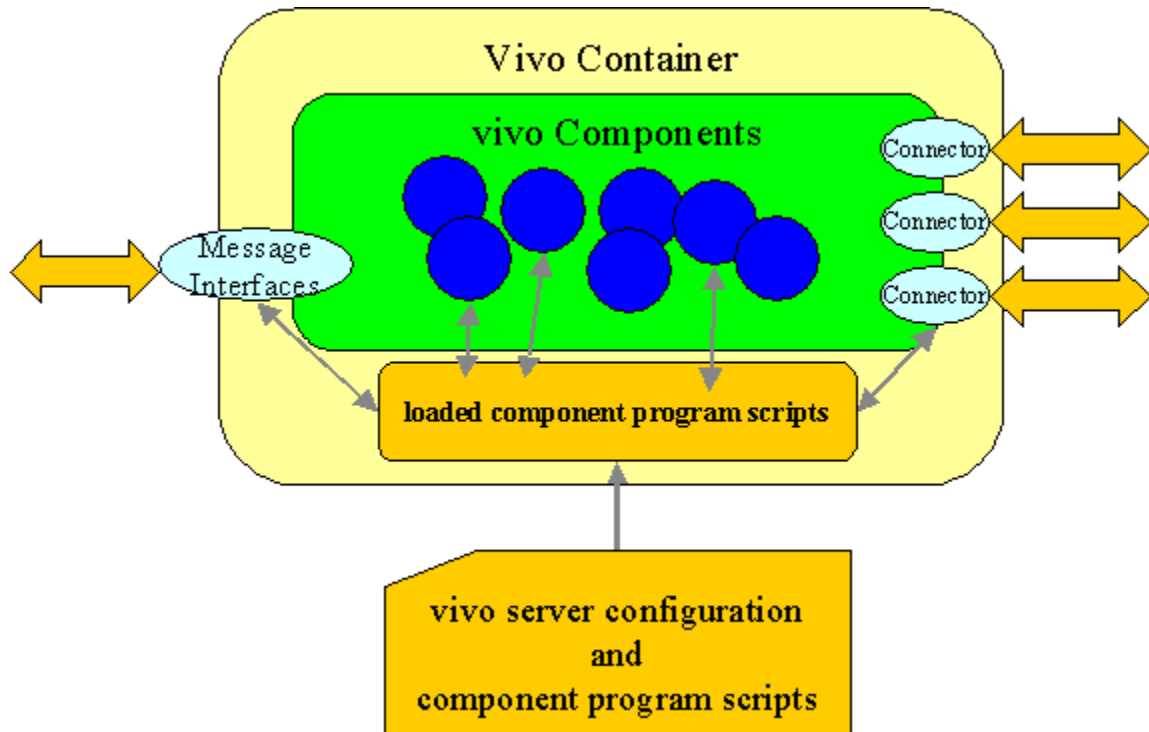


Figure 3 - Structure of a Trifolium Vivo Service Engine

#### Trifolium Vivo with TXSeries-Encina

Given that the heritage of Trifolium technologies is linked to the Encina product, it is not surprising that the use of Vivo with Encina is a natural fit.

#### HOW TRIFOLIUM VIVO SERVERS WORK WITH ENCINA

Trifolium servers interact with Encina by making the appropriate calls to the Encina server management library functions. These functions control the initialization of the server, the registration of recoverable resources, listening for requests and server termination. A special server management component performs all of these functions. In addition, there are XA-compatible versions of the database access components that can be used as recoverable resources in an Encina environment. There are special components for dealing with Encina's Peer-to-Peer Communications (PPC) facility with full distributed-transaction support. Another component supports the use of MQSeries as an XA resource.

By using the components specialized for Encina (where this specialization is necessary) and the XA versions of the communication and data access components, along with the rest of the standard Vivo components, it is straightforward to configure servers that act as Encina Monitor Application Servers within an Encina environment. These servers can have any interface that the developer chooses to design; however there is a default Encina interface that receives messages as strings and returns results as strings. These message strings can be in the various formats supported by the Vivo messaging infrastructure, including XML, fixed-format buffers, name-value streams or "raw" strings.

Vivo allows the use of user-developed interfaces through a dynamic loading process. A developer can create an interface using the standard Encina TIDL mechanisms, generate (manually or automatically) code to transform functions and parameters into Vivo messages, and make the interface and transform code available to Vivo as a loadable library. The functionality required to implement the functions in the interface is then developed using the Vivo component configuration mechanisms.

#### DEVELOPING ENCINA SERVERS WITH TRIFOLIUM VIVO SERVICE ENGINE

When using Vivo, the development of functionality is largely done using the Vivo configuration mechanisms (referred to as "component programming" in Vivo documentation). However, there can be custom development done if there is a need, since Vivo has the capability to dynamically load and use custom extensions. The basic process for using Vivo to create an Encina server is as follows:

1. Determine whether to use the default Encina message-oriented interface or to create a custom interface. If you choose to use the default interface, skip to step 6.
2. Assuming that you chose to create a custom interface, define the interface in the normal way using TIDL. Compile the interface definition in the normal way (tidl, idl) to create "stub" functions for the server.
3. Create code that maps the functions and their parameters into a supported message structure (e.g. XML

- format) and passes the message to the Vivo engine. This can be done manually, or using the Trifolium utility (genVivoEncinaMsgs).
4. Compile and link the server stubs and mapping code into a dynamically loadable library (.dll or .so file).
  5. Add an entry to the Vivo container configuration that points to your new library for dynamic loading.
  6. Develop Vivo operation configurations for each of the functions/messages the server will support. Add these configuration files to the import list for the Vivo engine, and to the routing list for the Vivo container.
  7. Define a new Encina server using the Encina management tools, specifying the standard Vivo binary as the executable, and defining the appropriate environment variables to allow Vivo to find the configuration files you created.
  8. Start the server and test the operations.

Using Vivo with the default message-oriented interface is the quickest way to get up and running with an Encina server. Given the support for rapidly generating support of custom interfaces, Vivo can be an extremely powerful way to create very sophisticated distributed transaction applications extremely quickly.

The diagram below shows an Encina execution environment populated with servers implemented using Vivo service engines.

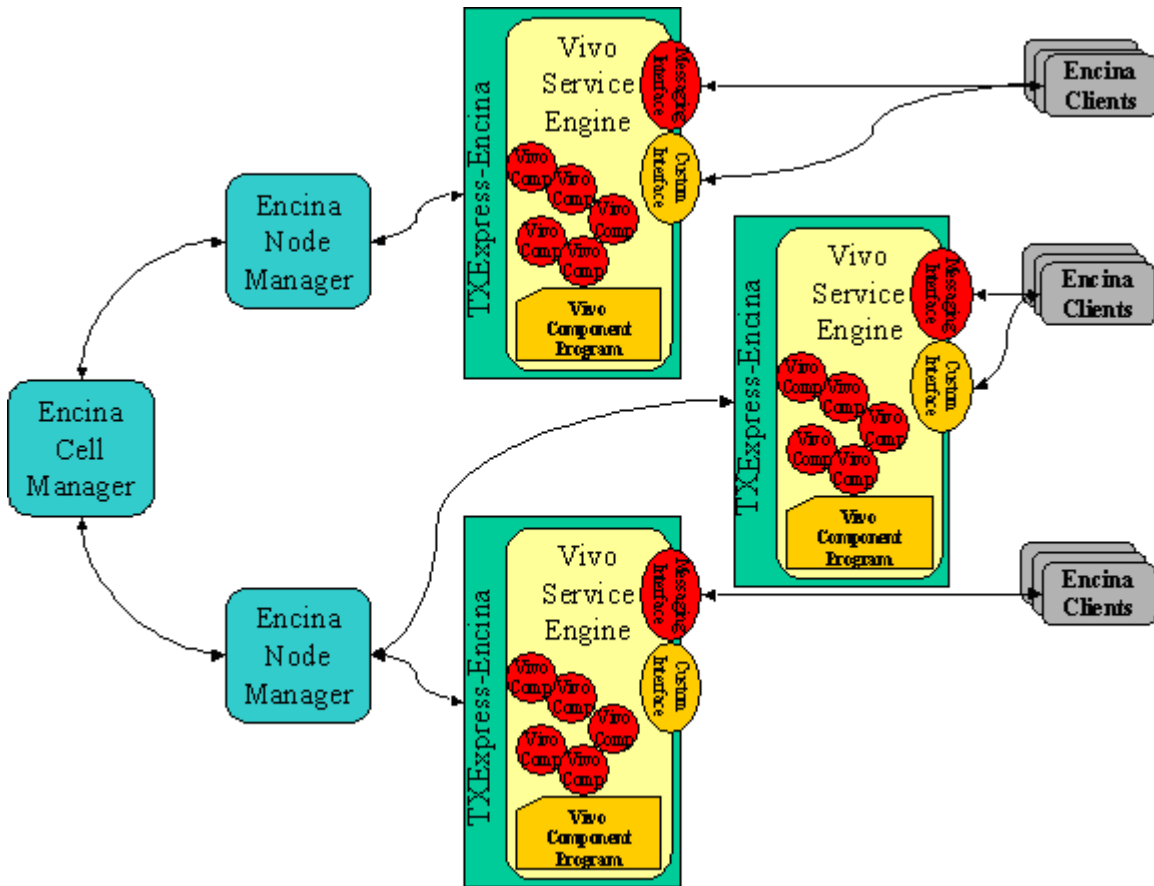


Figure 4 - TXSeries-Encina Execution Environment With Vivo Service Engines

#### Trifolium Vivo with TXSeries-CICS

The nature of how TXSeries-CICS executes user-developed code is different than the TXSeries-Encina model, which means there are some differences in how Vivo is used to develop this functionality. Despite these differences, the fundamental value of using Vivo is the same. The development of custom functionality to execute in the TXSeries-CICS environment is much more rapid, reliable, and predictable.

A CICS program can be intended for use directly from terminal input and produce terminal output, or it might be intended to be the target of a LINK or XCTL from another program or chain of programs. The use of the program will have a large impact on what is included, particularly if the program is doing terminal input and output, in which case the CICS BMS facilities would be used within the program. If the program is simply a processing stage without input or output duties, it may have very little CICS-specific content.

In the TXSeries-CICS environment, we will typically use the embedded version of Vivo. This is convenient since

CICS does dynamic loading of the programs that are defined as implementing incoming transaction requests. The embedded Vivo service engine can be the program that is loaded and executed as the implementation of a CICS transaction. It is also possible to use Vivo in the server mode as an "outboard" server that is connected within a CICS program. This model of usage would take advantage of one of the standard communication techniques between the CICS program and the Vivo server. One interesting option is to use the Encina TPM interface component in the Vivo server to allow it to act as an acceptor of APPC conversations from CICS programs.

The discussion in the rest of this section concerns the embedded service engine approach to using Vivo with TXSeries-CICS.

**USING VIVO AS A CICS PROGRAM**

TXSeries-CICS expects to be able to dynamically load and execute a program when needed to implement a requested transaction. Although Vivo is typically shipped as a stand-alone server, a version of Vivo has been produced which is compiled and linked appropriately for use as a CICS program. This does not change the functionality of Vivo as a service engine at all. It simply allows the Vivo service engine to get requests handed to it from CICS, or another CICS program, instead of receiving its requests directly.

From within any new or existing CICS program, the Vivo-CICS program can be invoked. The data transferred from the invoking program becomes the message that is processed through the Vivo component program. Typically the EDK fixed-format buffer message support will be used to process the passed data, however any of the Vivo message formats are available for use. Although it would be possible to directly use the Vivo service engine to handle terminal-based requests from users, it requires extra steps in the development process. To implement direct terminal support, Vivo must do the terminal screen mapping. Although it is possible, and sometimes acceptable, to do terminal screen mapping without additional support, it is more common to use the facilities of BMS to handle the details. In order to use BMS within Vivo, a Vivo extension component would have to be developed to implement the specific screens involved. In general, it is much easier to use Vivo as a called program and have the calling program handle the terminal input and output.

The diagram below shows the standard use of the Vivo service engine within TXSeries-CICS.

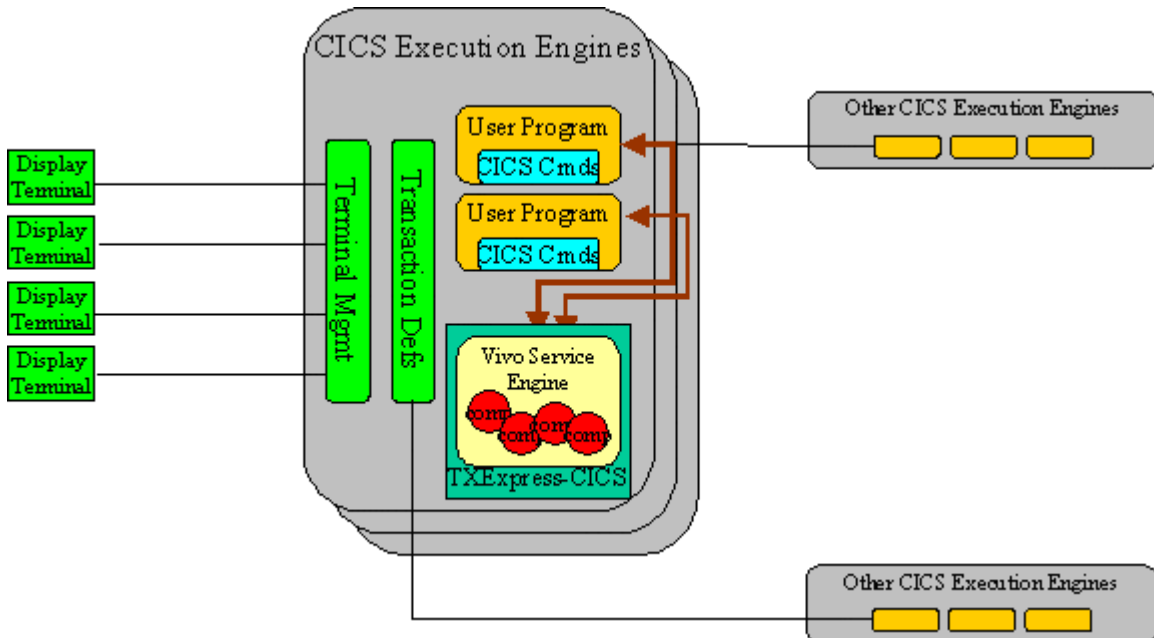


Figure 5 - TXSeries-CICS Functionality Implemented Using Vivo Service Engine

**Conclusion**

With IBM's TXSeries product, it is possible to develop extremely high-volume, fault-tolerant, and high-performance distributed applications. Building these types of applications requires a high degree of skill in application design and construction and long development cycles. Using the Trifolium Vivo technology, an organization can build applications in the TXSeries environment much more reliably, predictably and rapidly. Since the proper use of the TXSeries components is built into the Trifolium product, along with best practices for large distributed application development, using the Trifolium tools for development will yield the volume, resilience and performance you need.