BEA Tuxedo

Using the BEA Tuxedo Domains Component

BEA Tuxedo Release 8.0
Document Edition 8.0
June 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using the BEA Tuxedo Domains Component

Document Edition	Date	Software Version
8.0	June 2001	BEA Tuxedo Release 8.0

Contents

2. Planning and Configuring ATMI Domains

3. Planning and Configuring CORBA Domains

4. Administering Domains

About This Document

This document explains how to configure and administer BEA Tuxedo® domains, for either an data-dependent BEA Tuxedo ATMI environment or an object-oriented BEA Tuxedo CORBA environment.

This document covers the following topics:

- Chapter 1, “About Domains,” provides an overview of the BEA Tuxedo Domains component.
- Chapter 2, “Planning and Configuring ATMI Domains,” explains how to plan and configure a domain for a BEA Tuxedo ATMI environment.
- Chapter 3, “Planning and Configuring CORBA Domains,” explains how to configure a domain for a BEA Tuxedo CORBA environment.
- Chapter 4, “Administering Domains,” explains how to administer BEA Tuxedo domains.

What You Need to Know

This document is intended mainly for administrators who configure operational parameters that support mission-critical the BEA Tuxedo systems. It assumes a familiarity with the BEA Tuxedo platform.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the BEA Tuxedo documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the BEA Tuxedo documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following documents provide related information about BEA Tuxedo software.

- *Installing the BEA Tuxedo System*—paper copy distributed with the CD
- *BEA Tuxedo Release Notes*—paper copy distributed with the CD
- *Setting Up a BEA Tuxedo Application*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to set up and administer the BEA Tuxedo system.

- *Administering a BEA Tuxedo Application at Run Time*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to administer BEA Tuxedo applications at run time.
- *Scaling, Distributing, and Tuning CORBA Applications*—available through the BEA Tuxedo Online Documentation CD, this guide describes how to tune and scale CORBA applications that run in the BEA Tuxedo CORBA environment.

For more information about configuring and administering BEA Tuxedo ATMI and BEA Tuxedo CORBA environments, refer to the *CORBA Bibliography* at http://edocs.bea.com/.

Contact Us!

Your feedback on the BEA Tuxedo documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA Tuxedo documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 8.0 release.

If you have any questions about this version of BEA Tuxedo, or if you have problems installing and running BEA Tuxedo, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR

Convention	Item
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 About Domains

This topic includes the following sections:

- What Is the BEA Tuxedo Domains Component?
- Building a Multiple-domain Configuration
- Example of an Application Using Domain Gateways
- Messaging Paradigms Supported by Domain Gateways
- Defining Transaction and Blocking Timeouts in Domains
- Specifying How Your Domains Connect
- What Is the Domains Configuration File?
- Converting the Domains Configuration File
- Features of BEA Tuxedo System Domains

What Is the BEA Tuxedo Domains Component?

The BEA Tuxedo application programming framework simplifies the development of open *online transaction processing* (OLTP) distributed applications by hiding the complexity associated with the distribution of application processing. The framework consists of the following:

- An extended client/server model that hides the heterogeneity of different computers and application programs, as well as the location of application programs.
- A centralized administration system that allows application administrators to control all cooperating machines as a single application.

As a business grows, application developers may need to organize different segments of the business by sets of functionality that require administrative autonomy but allow sharing of services and data. It may not be appropriate to structure a group of applications as a single distributed application because of the functionality, geographical location, confidentiality requirements, and potential growth of each. Also, an enterprise may want to expand business by cooperating with other organizations that provide OLTP services under the control of different transaction processing monitors, such as BEA's TOP END, Transarc's Encina, IBM's CICS, Bull's TDS, Bull's TP8, ICL's TPMS, and so forth.

Each set of functionality defines an application that spans one or more computers, and is administered independently from other applications. Such a functionally distinct application is referred to as a domain; in practice, the organization often uses the domain's functionality as part of its name so you find applications with names like the "accounting" domain or the "order entry" domain. A BEA Tuxedo domain is a single computer or network of computers controlled by a single BEA Tuxedo configuration file.

Business Operations Interoperating with Each Other

The BEA Tuxedo System Domains feature provides a framework for interoperability among the domains of a business that continues the BEA Tuxedo enhanced client/server model. Interoperability means more than merely the capability of communicating from one domain to another. By transparently making access to services of a remote domain available to users of the local domain (or accepting local service requests from users of a remote domain), Domains, in effect, break down the walls between the business applications of an organization. Application programmers can use the ATMI interface to access the services provided by remote domains, or to define services that can be executed by a remote domain.

The Domains feature also enables BEA Tuxedo applications to cooperate with dozens of applications running in other administrative domains. The BEA Tuxedo system provides a common framework for controlling very large applications that may include domains running other transaction processing systems.

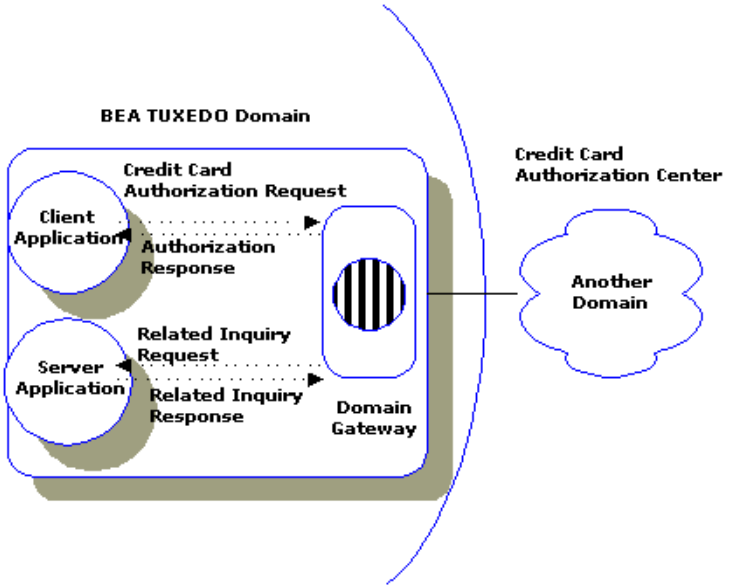
Building a Multiple-domain Configuration

To build a multiple-domain configuration, you need to consider the following tasks:

- Integrate your existing BEA Tuxedo application with other domains
- Ensure interoperability across domains
- Preserve or restrict access to services across domains
- Accept or deny service requests across domains

Domains achieve these tasks through a highly asynchronous, multitasking, multithreaded gateway. A domain gateway (`DGW`) is a BEA Tuxedo-supplied server that handles requests to remote domains and from remote domains. Any request can be processed within a transaction. The following figure illustrates how one BEA Tuxedo domain communicates with another domain via a domain gateway.

Figure 1-1 Two-way Communication Through a Gateway



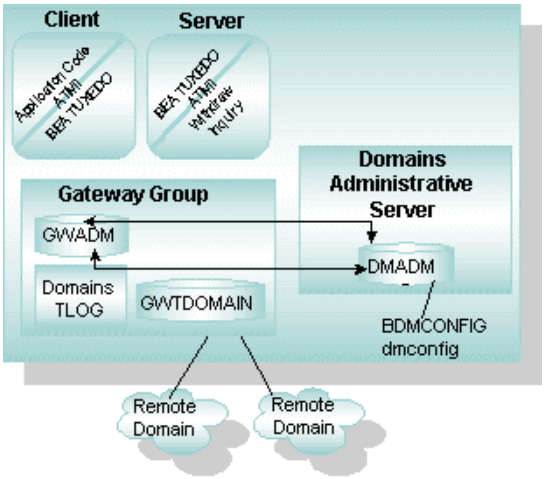
In this illustration, the gateway processes outgoing credit card authorization requests to another domain. The gateway also handles incoming authorization responses.

Domain gateways manage all the communication between domains. The gateway processes include a *gateway administrative server* (`GWADM`) that enables run-time administration of the domain gateway group and a *Domains administrative server (*`DMADM`*)* that enables run-time administration of the BEA Tuxedo application-wide Domains configuration information.

Tools to Set Up and Maintain a Multiple-domain Application

The following illustration shows the tools provided by the BEA Tuxedo system for setting up and maintaining a multiple-domain configuration.

Figure 1-2 Domains Administrative Tools



Domains Administrative Tool	Description
`dmadmin(1)`	A command that allows you to configure, monitor, and tune domain gateway groups dynamically. Use this command to update the `BDMCONFIG` file while an application is running. The command acts as a front-end process that translates administrative commands. These commands send requests to the `DMADMIN` service, a generic administrative service advertised by the `DMADM` server. `DMADMIN` invokes functions that validate, retrieve, or update information in the `BDMCONFIG` file.
`DMCONFIG(5)` `BDMCONFIG`	`DMCONFIG` is the text version of the configuration file for a multiple-domain configuration; `BDMCONFIG` is the binary version.
`dmloadcf(1)` and `dmunloadcf(1)`	`dmloadcf`—reads the `DMCONFIG` file, checks the syntax, and optionally loads a binary `BDMCONFIG` configuration file. `dmunloadcf`—translates the `BDMCONFIG` configuration file from binary to text format.

Domains Administrative Tool	Description (Continued)
DMADM(5)	An administrative server that enables you to manage a Domains configuration at run time. DMADM provides a registration service for gateway groups. This service is requested by GWADM servers as part of their initialization procedure. The registration service downloads the configuration information required by the requesting gateway group. The DMADM server maintains a list of registered gateway groups, and propagates to these groups any changes made to the configuration file (BDMCONFIG).
GWADM(5)	An administrative server that supports run-time administration of a specific gateway group. This server registers with the DMADM server to obtain the configuration information used by the corresponding gateway group. GWADM accepts requests from DMADMIN for run-time statistics or changes in the run-time options of the specified gateway group. Periodically, GWADM sends an "I-am-alive" message to the DMADM server. If no reply is received from DMADM, GWADM registers again. This process ensures the GWADM server always has the current information about the Domains configuration for its group.
GWTDOMAIN(5)	A gateway process that receives and forwards messages from clients and servers in all connected domains (for BEA Tuxedo Domains).

Types of Domain Gateways

The BEA Tuxedo system provides different types of gateways to accommodate various network transport protocols used to communicate with remote domains. Access to remote domains that use the same communication and transaction commitment protocol is provided through a group of gateways that implement the configuration defined for a particular local domain. Following are the different types of domain gateways:

- BEA Tuxedo Domains (TDomains) gateway (that is, the GWTDOMAIN gateway)—provides interoperability between two or more BEA Tuxedo applications through a specially designed TP protocol that flows over network transport protocols such as TCP/IP.

Note: GWTDOMAIN gateways should not be specified as members of an MSSQ set. They should not have a reply queue (REPLYQ=N should be specified). GWTDOMAIN gateways are recommended to be restartable.

- BEA eLink OSI TP gateway—provides interoperability between BEA Tuxedo applications and other transaction processing applications that use the OSI TP standard. OSI TP is a protocol for distributed transaction processing defined by the International Standards Organization (ISO).
- BEA eLink Adapter for Mainframe SNA gateway—provides interoperability between clients and servers in a BEA Tuxedo domain, and clients and servers in an MVS/CICS or MVS/IMS environment in a remote SNA domain. It also supports communication with multiple SNA networks.
- BEA eLink Adapter for Mainframe TCP:
  - For CICS gateway—makes it possible for non-transactional tasks within BEA Tuxedo regions to access services provided by CICS application programs and vice-versa. It enables a BEA Tuxedo domain to communicate, via the TCP/IP network transport protocol, with a CICS environment.
  - For IMS gateway—provides transparent communications between client and server transactions in an IMS system and a BEA Tuxedo domain, a CICS system, or another IMS system.
- BEA TOP END Domain Gateway (TEDG)—provides interoperability between TOP END systems and BEA Tuxedo domains.

Functionality Supported by Domain Gateways

Domain gateways support the following functionality:

- Administration—gateways can be booted or shut down exactly as any other BEA Tuxedo server. Run-time administration is provided through an administrative server, DMADM. Using DMADM, application administrators can make changes to a domains configuration file, and tune the performance of a gateway group. (The DMADM administrative server should be booted before gateway groups.)
- ATMI—gateways can access the programming interface between a domain and the BEA Tuxedo system ensuring access to the following messaging models:

  - Request/Response Model—application programs using the BEA Tuxedo system can request services from applications running in other domains. Also, remote applications can request services from local servers. (No changes are required to the application program to accommodate this interdomain functionality.)
  - Conversational Model—application programs can establish conversations with programs running in other domains. Remote domains can establish conversations with conversational services offered by local servers. (No changes are required to the application program to accommodate this interdomain functionality.)
  - Queuing Model—application programs using the BEA Tuxedo system can store data on queues. Any client or server can store messages or service requests in a queue in a remote domain and all stored requests are sent through the transaction protocol to ensure safe storage. (No changes are required to the application program to accommodate this interdomain functionality.)

- Multidomain Interaction—gateways can communicate with multiple domains.
- Multinetwork Support—gateways can communicate with other domains via a variety of network protocols, such as `TCP/IP`, `IPX/SPX`, and others. However, a gateway is limited by the capabilities of the networking library to which it is linked. In other words, a gateway typically supports a single type of network protocol.
- Transaction Management—application programs can interoperate with other domains within a transaction. The gateway coordinates the commitment or rollback of transactions running across domains.
- Typed Buffer Support—gateways can perform encoding and decoding operations for all the types of buffers defined by the application.

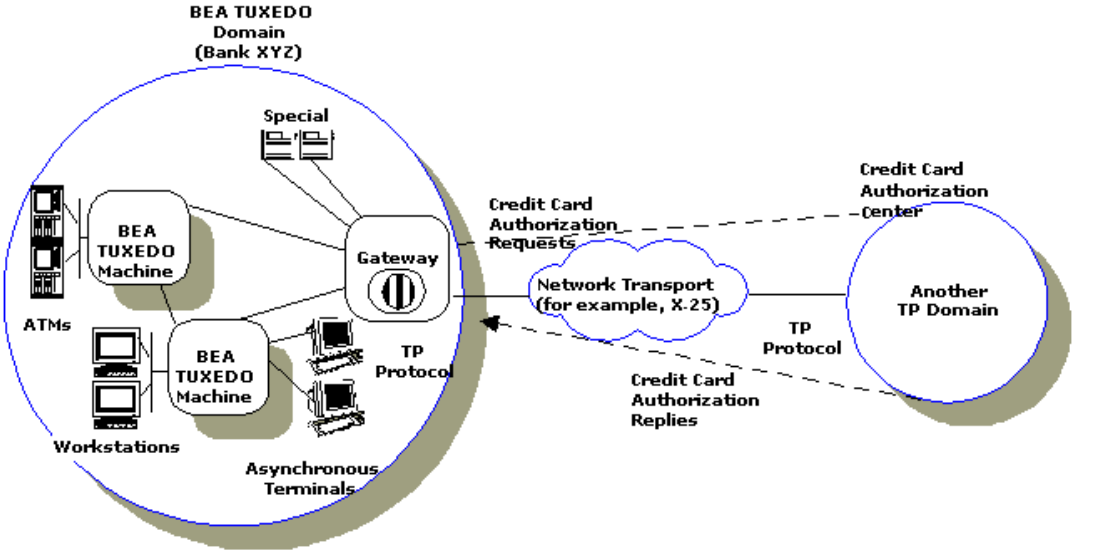
See Also

- “What Is a Multiple-domain Configuration?” on page 3-47 in *Introducing BEA Tuxedo ATMI*
- “Example of an Application Using Domain Gateways” on page 1-9
- “Messaging Paradigms Supported by Domain Gateways” on page 1-11

Example of an Application Using Domain Gateways

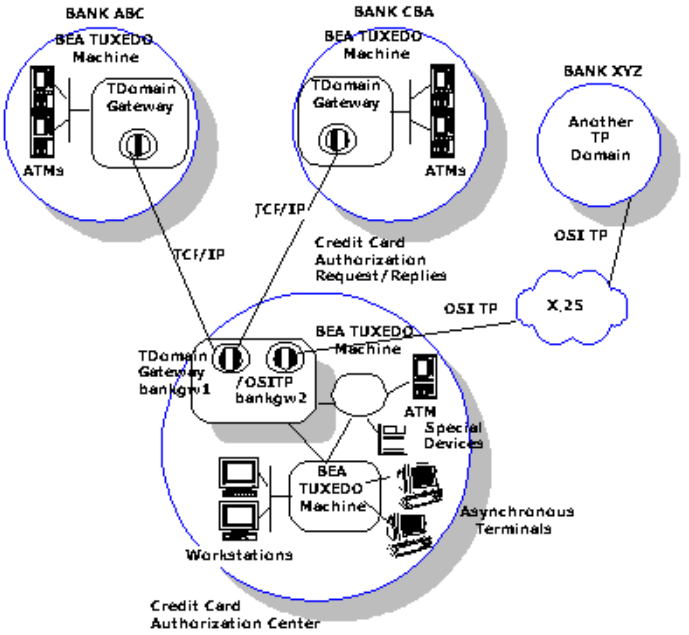
The following figure shows a BEA Tuxedo application that requires services (in this case, credit card authorizations) from a remote domain.

Figure 1-3 High-level View of Two Communicating Domains



The application also accepts service requests (for example, balance inquiries) from remote domains. The gateway process provides bidirectional transaction control, and administrative tools for configuring a local domain to interoperate with other domains. `BDMCONFIG`, the configuration file for a multiple-domain application, identifies exported services, imported services, addressing, and any access control lists to be used. The following figure shows a more detailed view of a sample Domains environment.

Figure 1-4 Example Domains Environment



The example shows a credit card authorization center running under the control of the BEA Tuxedo system. The authorization center has two gateway groups: `bankgw1` (which uses the TCP/IP protocol) and `bankgw2` (which uses the OSI TP protocol). `bankgw1` provides access to two remote BEA Tuxedo domains (Bank ABC and Bank CBA); `bankgw2` provides access to one remote domain (Bank XYZ) using the OSI TP protocol.

In this example, Bank ABC generates service requests to the credit card authorization center. These requests are received by a gateway running within group `bankgw1`. This gateway issues a service request, on behalf of the remote domain, to the credit card authorization service provided by a local server. The server processes the request and sends the reply to the gateway, and the gateway forwards the reply to Bank ABC.

The credit card authorization center may also issue service requests. For example, the authorization center may send balance inquiries to Bank XYZ. Domains makes this possible by providing a gateway that acts like a local server that advertises services available in other domains as if they were local services.

Domains provides the notion of a *local domain* that controls incoming requests and provides a generic addressing framework for the application. Local domains help to provide partial views of an application, that is, a subset of the local services available to a set of remote domains. Each local domain is always represented by a single gateway server group.

Messaging Paradigms Supported by Domain Gateways

The functions of the BEA Tuxedo client/server model are supported by the following messaging paradigms in domain gateways:

- "Request/Response Communication Between Local and Remote Services" on page 1-11
- "Conversational Communication Between Local and Remote Services" on page 1-13
- "Queued Messaging for Data Storage" on page 1-13

Request/Response Communication Between Local and Remote Services

Domain gateways provide support for the request/response model of communication defined by the ATMI interface. A BEA Tuxedo application can request remote services exactly as if they were offered locally.

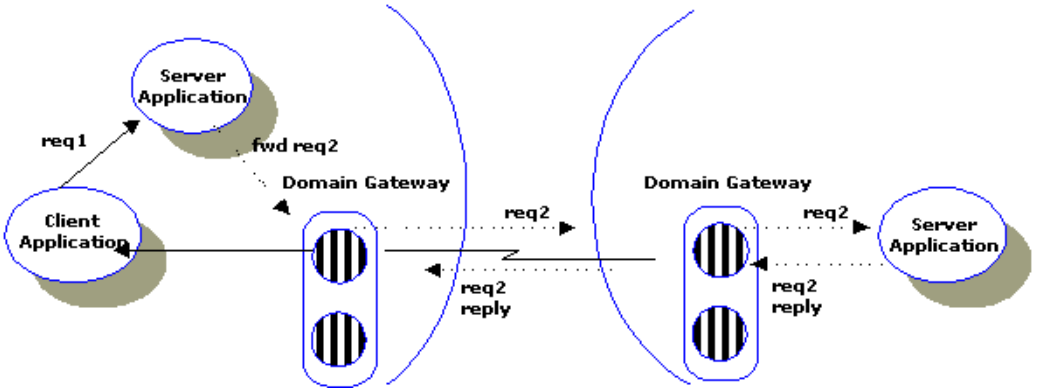
Support for ATMI Functions

The following BEA Tuxedo ATMI functions are logically limited to use within a single application and are not supported across domains:

- `tpinit(3c)`/`tpterm(3c)`—BEA Tuxedo applications do not attach to the environment of a remote domain; they use Domain gateways to access a remote domain. Therefore, an extra `tpinit()`/`tpterm()` sequence is not needed for remote applications.
- `tpadvertise(3c)` and `tpunadvertise(3c)`—cannot be used across domains. Domain gateways do not support dynamic service advertisements across domains.
- `tpnotify(3c)` and `tpbroadcast(3c)`—Domains does not support the unsolicited communication paradigm provided by these primitives.
- Event posting (`tppost(3c)`) and notification of events (`tpsubscribe(3c)`)—Domains does not support these functions across domains.

Support for `tpforward(3c)` is provided to preserve application portability. Forwarded requests are interpreted by domain gateways as simple service requests. This process is shown in the following diagram, which illustrates the simple scenario of a service using `tpforward` to send a request to a remote service.

Figure 1-5 Using tpforward to Send a Request to a Remote Service



Conversational Communication Between Local and Remote Services

The ATMI is a connection-oriented interface that enables clients to establish and maintain conversations with services programmed in the conversational paradigm.

BEA Tuxedo applications use `tpconnect(3c)` to open a conversation with a remote service, `tpsend(3c)` and `tprecv(3c)` to communicate with this service, and `tpdiscon(3c)` to end the conversation. Domain gateways maintain the conversation with the remote service, and support the same semantics for returns (that is, `tpreturn` with `TPSUCCESS` or `TPFAIL`) and disconnects that are defined for BEA Tuxedo conversational services.

Note: The ATMI connection-oriented functions provide half-duplex conversations; `tpforward(3c)` is not allowed within a conversational service.

Application administrators indicate that a remote service is conversational by specifying `CONV=Y` in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.

Queued Messaging for Data Storage

The BEA Tuxedo system enables messages to be queued to persistent storage (disk) or to non-persistent storage (memory) for later processing or retrieval. ATMI provides primitives that allow messages to be added (that is, `tpenqueue`) or read (that is, `tpdequeue`) from queues. Reply messages and error messages can be queued for later return to clients. An administrative command interpreter (that is, `qmadmin`) is provided for creating, listing, and modifying queues. Servers are provided to accept requests to enqueue and dequeue messages (that is, `TMQUEUE` server), to forward messages from the queue for processing (that is, `TMQFORWARD` server), and to manage the transactions that involve queues (that is, `TMS_QM` server).

Domain gateways extend support for queued messaging services across domains.

See Also

- "What Is Request/Reply Communication?" on page 2-14 in *Introducing BEA Tuxedo ATMI*
- "What Is Conversational Communication?" on page 2-9 in *Introducing BEA Tuxedo ATMI*
- "What Is Queue-based Communication?" on page 2-13 in *Introducing BEA Tuxedo ATMI*

Typed Buffers to Package Data

In BEA Tuxedo applications, typed buffers are used to send data between clients and servers. The typed buffer mechanism allows application programmers to transfer data without needing to know which data representation scheme is used by the machines on which the application's clients and servers are running.

A domain gateway can receive and process service requests sent from workstations, BEA Tuxedo machines, and remote domains with different machine representations. A typed buffer switch decodes the data sent with the service request. The administrator must define the typed buffer switch appropriate for the application.

Data-dependent routing depends upon matching specified criteria to fields within data. If data is encoded, however, there is no way to determine the contents of that data in order to route that data accurately. In addition, a domain gateway needs access to the contents for the following reasons:

- The gateway may have to apply data-dependent routing to select the appropriate remote domain for the service requested. (Data-dependent routing criteria for remote domains are defined in the Domains configuration file.)
- Different data formats may be used within different domains, depending on the networking protocols implemented or used in each domain.

Therefore a domain gateway always tries to decode any service request that is received encoded.

OSI terminology provides a useful distinction between abstract syntax (that is, the structure of the data) and transfer syntax (that is, the particular encoding used to transfer the data). Each typed buffer implicitly defines a particular data structure (that is, its abstract syntax) and the encoding rules (or typed buffer operations) required to map the data structure to a particular transfer syntax (for example, XDR).

The BEA Tuxedo system provides a set of predefined buffer types (STRING, CARRAY, FML, FML32, VIEW, VIEW32, X_C_TYPE, X_OCTET, X_COMMON, and XML) and the encoding rules required to map these types to the XDR transfer syntax.

Note: A programmer can supply a custom buffer type by adding an instance to the tm_typesw array in *TUXDIR*/lib/tmtypesw.c (see tuxtypes(5) and typesw(5)), and supplying routines for the new type (see buffer(3c)).

See Also

- "What Are Typed Buffers?" on page 2-24 in *Introducing BEA Tuxedo ATMI*
- "Customizing a Buffer" on page 3-28 in *Programming BEA Tuxedo ATMI Applications Using C*
- tuxtypes(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- typesw(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Defining Transaction and Blocking Timeouts in Domains

The BEA Tuxedo system provides two timeout mechanisms: a *transaction timeout* mechanism and a *blocking timeout* mechanism. The transaction timeout is used to define the duration of a transaction, which may involve several service requests. The

timeout value is defined when the transaction is started (with `tpbegin(3c)`). The blocking timeout is used to define the duration of individual service requests, that is, how long the application is willing to wait for a reply to one service request.

The BEA Tuxedo transaction timeout mechanism is used unchanged in the Domains framework. Use of the same transaction timeout mechanism is necessary because domain gateways implement the TMS functionality and therefore are required to handle the `TMS_TIMEOUT` messages generated by the Bulletin Board Liaison (BBL).

Domain gateways, however, cannot use the BEA Tuxedo *blocking timeout* mechanism. The blocking timeout mechanism uses information stored in the registry slot assigned to each client or server. (Information in the registry slot is used by the local `BBL` to detect requesters that have been blocked for a time greater than `BLOCKTIME`.) Domain gateways, however, are multitasking servers that can process several service requests at a time, which means they cannot use the registry slot mechanism. When a blocking timeout condition arises, the domain gateway sends an error/failure reply message to the requester, and *cleans* any context associated with the service request.

Specifying How Your Domains Connect

You can specify the conditions under which a local domain gateway tries to establish a connection to a remote domain by selecting one of the following connection policies:

- Connect at boot time (`ON_STARTUP`)
- Connect when a client program requests a remote service (`ON_DEMAND`)
- Accept incoming connections but do not initiate a connection automatically (`INCOMING_ONLY`)

Determining the Availability of Remote Services with the Dynamic Status Feature

The gateway process (GWTDOMAIN) advertises those services that are imported from one or more remote domains in the bulletin board. These services typically remain advertised regardless of whether the remote service is reachable.

The capability of the BEA Tuxedo domain gateways known as Dynamic Status reports the status (as determined by the BEA Tuxedo system) of remote services.

When Dynamic Status is in effect, the status of a remote service depends on the status of the network connection between the local and remote gateways. Remote services are considered available whenever a connection to the domain on which they reside is available. When a network connection to a remote domain is not available, services in that domain are considered unavailable. This policy is invoked when the connection policy is ON_STARTUP (that is, when a local domain gateway tries to establish a connection to a remote domain at boot time) or INCOMING_ONLY (that is, when a local domain gateway does not try to establish a connection to remote domains upon starting).

For each service, the gateway keeps track, not only of the remote domains from which the service is imported, but also of which remote domains are available. In this way, the gateway provides intelligent load balancing of requests to remote domains. If all the remote domains from which a service is imported become unreachable, the service is suspended in the bulletin board.

For example, suppose a service called RSVC is imported from two remote domains, as specified by the following entries in the DM_REMOTE_SERVICES section of the configuration file:

```
RSVC RDOM=R1
RSVC RDOM=R2
```

When connections to both R1 and R2 are up, the gateway load balances requests for the RSVC service. If the connection to R1 goes down, the gateway sends all requests for RSVC to R2. If both connections go down, the gateway suspends RSVC in the bulletin board. Subsequent requests for RSVC are either routed to a local service or another gateway, or fail with TPENOENT.

Note: When the connection policy is ON_DEMAND, a connection is attempted only when either a client requests a remote service or an administrative "connect" command is run.

How Your Connection Policy Affects Dynamic Status

Dynamic Status is not available in all Domains configurations; whether it is available depends on which connection policy you establish between your domains. The following table describes how each connection policy affects Dynamic Status.

Table 1-1 Availability of Dynamic Status

Under This Policy...	Dynamic Status Is..
ON_STARTUP	On. Services imported from a remote domain are advertised as long as a connection to that remote domain is active. A connection can be established in any of the following ways: ■ Automatically ■ Manual through the dmadmin command ■ Through an incoming connection
ON_DEMAND	Off. Services imported from remote domains are continually advertised. Ways in which a connection can be established are: ■ Client request ■ Manually through the dmadmin command ■ Through an incoming connection
INCOMING_ONLY	On. Remote services are initially suspended. A domain gateway is available for incoming connections from remote domains, and remote services are advertised when the local domain gateway receives an incoming connection or when a manual connect command is issued. A connection can be established in the following ways: ■ Manually through the dmadmin command ■ Through an incoming connection

What Is the Domains Configuration File?

All domains configuration information is stored in a binary file called BDMCONFIG. You can create and edit a text version of this file, DMCONFIG, with any text editor. You can update the compiled BDMCONFIG file while the system is running by using the dmadmin(1) command when using Domains. In a multi-domain application, a separate BDMCONFIG file must be created for each participating domain.

System access to the BDMCONFIG file is provided through the Domains administrative server, DMADM(5). When a gateway group is booted, the gateway administrative server, GWADM(5), requests from the DMADM server, a copy of the configuration file required by that group. The GWADM and DMADM servers also ensure that run-time changes to the configuration are reflected in the corresponding domain gateway group.

Descriptions of Sections of the DMCONFIG File

The following table provides a description of each section in the DMCONFIG file.

Table 1-2 DMCONFIG File Sections

Section	Purpose
DM_LOCAL_DOMAINS	Describes the environment for a particular domain gateway group. It assigns a logical application name, LDOM, to the subset of local services available to remote domains. You can use multiple entries in this section to define multiple gateway groups within a single BEA Tuxedo application. Each gateway group can provide access to domains of different types.
DM_REMOTE_DOMAINS	Identifies the remote domains available to clients and servers of this Domains application.
DM_LOCAL_SERVICES	Describes the local services provided by a local domain (LDOM) in the DM_LOCAL_DOMAINS section. Specification of services can also be used to restrict access to local services from remote domains; only services specified are available to remote domains.

Section	Purpose (Continued)
DM_REMOTE_SERVICES	Describes the set of services provided by remote domains. It also names the local gateway group (through the LDOM parameter) that provides access to the remote service.
DM_ROUTING	Specifies criteria for data-dependent routing used by gateways to route service requests to specific remote domains.
DM_ACCESS_CONTROL	Specifies an access control list (ACL) that names (via RDOMs) the remote domains permitted to request local services. The ACL parameter in the DM_LOCAL_SERVICES section can be used by setting ACL=*name_of_list* to restrict access to a particular local service to the listed set of remote domains.
DM_*dmtype*	Defines the parameters required for a particular Domains configuration. Currently, the value of *dmtype* can be OSITP, SNAX, TOPEND, or TDOMAIN. This topic focuses only on TDOMAIN. Consult BEA eLink for Mainframe documentation for information about OSITP and SNAX. Consult *Using the BEA Tuxedo TOP END Domain Gateway* for information about TOPEND. Each domain type must be specified in a separate section. In a DM_TDOMAIN section, entries associated with a remote domain can be specified more than once, with different network addresses, to implement the *mirrored* gateway facility. (See DMCONFIG(5) for a description and an example of a mirrored gateway.)

Domains Terminology Improvements

For BEA Tuxedo release 7.1 or later, the Domains MIB uses improved class and attribute terminology to describe the interaction between local and remote domains. The improved terminology applies to the DM_MIB classes, reference page, and error messages, the DMCONFIG file syntax, and various DMCONFIG error messages.

For backwards compatibility, aliases are provided between the DMCONFIG terminology used prior to BEA Tuxedo release 7.1 and the improved Domains MIB terminology. For BEA Tuxedo release 7.1 or later, DMCONFIG accepts both versions of the terminology. For details, see “Domains Terminology Improvements” in the DM_MIB(5) reference page.

See Also

- "Configuring a Domains Environment" on page 2-18
- `DMCONFIG(5)` in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- "Converting the Domains Configuration File" on page 1-21

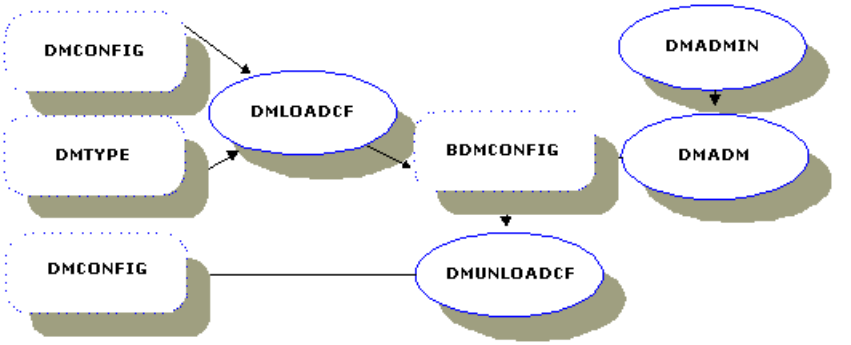
Converting the Domains Configuration File

This section provides instructions for converting a text version of a Domains configuration file (`DMCONFIG`) to a binary version (`BDMCONFIG`), and vice versa.

Converting DMCONFIG to a Binary File

The `dmloadcf(1)` command parses `DMCONFIG` (a text file), and loads the information about the Domains configuration into a binary file called `BDMCONFIG`. The command uses the environment variable `BDMCONFIG` to point to the directory in which the configuration should be stored. The `BDMCONFIG` file can be stored on the same device as the `TUXCONFIG` file (or the binary version of the `UBBCONFIG` file).

Figure 1-6 Relationships Between Configuration Commands and Files



The dmloadcf(1) command, through the -c option, also provides an estimate of the IPC resources needed for each local domain specified in the configuration.

As shown in the preceding figure, the dmloadcf command uses the $TUXDIR/udataobj/DMTYPE file. It checks the DMTYPE file to verify that the domain types specified in the configuration file are valid. Each Domains instantiation has a domain type. The type is used as a tag in the file *TUXDIR*/udataobj/DMTYPE. Each line in this file has the following format:

dmtype:access_module_lib:comm_libs:tm_typesw_lib:gw_typesw_lib

The file has the following entry for TDOMAIN:

```
TDOMAIN:-lgwt:-lnwi -lnws -lnwi::
```

Converting the BDMCONFIG File to a Text File

To unload a binary version of a Domains configuration file (that is, to convert it from binary to text format), run the dmunloadcf(1)command.

Features of BEA Tuxedo System Domains

- Aliasing capability—allows you to define map service names between local and remote applications, allowing for easy integration of applications that use different naming schemes.
- Availability—allows you to specify alternate destinations to handle failure conditions.
- Scalability and modular growth—programmers can structure their applications for modularity, isolation of failures, and independent growth. Interoperation with other transaction processing applications is achieved easily by adding to the Domains configuration the description of services used by a remote application.
- Security—an access control list (ACL) facility is provided to restrict access to local resources from a particular set of remote domains. Domains also provides encryption and password verification.
- Transparency and independence—application programmers need no knowledge of how services are distributed. A service may be available on the same machine as a client, on another machine in the local domain, or on a remote domain. Client application programmers do not need to know the implementation changes made to a service, the location of a service, network addresses, and so on.
- Transaction management and reliability—this Domains feature is integrated with the BEA Tuxedo transaction management capabilities.

CHAPTER

2 Planning and Configuring ATMI Domains

This chapter describes how to plan and configure domains in a BEA Tuxedo ATMI environment. For information about configuring domains in a BEA Tuxedo CORBA environment, refer to Chapter 3, “Configuring CORBA Domains.”

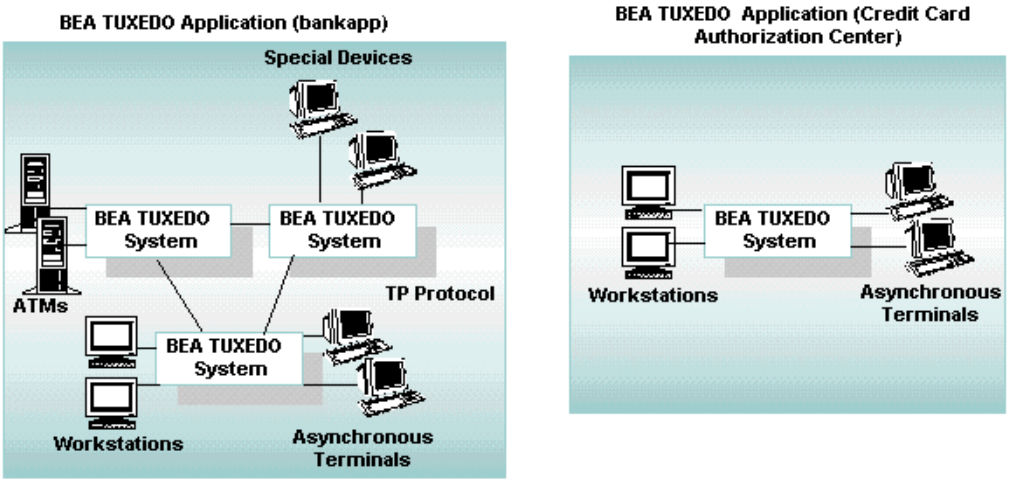
This topic includes the following sections:

- Planning to Build Domains from Multiple BEA Tuxedo Applications
- Sample Domains Application: creditapp
- Configuring a Domains Environment
- How to Compress Data Between Domains
- How to Route Service Requests to Remote Domains
- Setting Up Security in Domains
- Configuring the Connections Between Your Domains
- Configuring Failover and Failback in a Domains Environment

Planning to Build Domains from Multiple BEA Tuxedo Applications

Suppose a bank has developed the two BEA Tuxedo applications shown in the following figure: bankapp and a credit card authorization center.

Figure 2-1 Two BEA Tuxedo Applications



The bankapp application connects ATMs at various bank branches to the central bank office. The Credit Card Authorization Center processes applications for credit cards. Over time, the bank realizes that their customers would be better served if the bankapp application could communicate directly with the credit card authorization application. In this way, they could offer instant credit cards to anyone opening a new account.

bankapp is distributed as a sample application with the BEA Tuxedo software. The credit card authorization application is a hypothetical extension of bankapp.

Take a look at the configuration file (represented in the following sample code) to see how to implement bankapp as a multiple-machine application: *TUXDIR*/apps/bankapp/ubbmp.

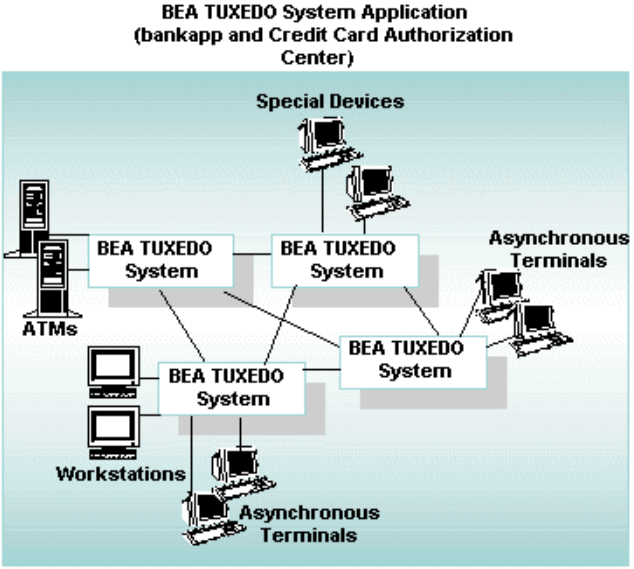
You have the following options:

- "Option 1: Reconfigure the Applications" on page 2-4
- "Option 2: Redefine the Applications as Separate BEA Tuxedo Domains" on page 2-9

Option 1: Reconfigure the Applications

One solution is to combine two BEA Tuxedo applications into one, as shown in the following figure.

Figure 2-2 Combining Two BEA Tuxedo System Applications



In the process of combining the two applications into a single configuration, the following changes are made:

- `OPTION=LAN` is specified and a `NETWORK` section is included.
- Server migration is enabled by specifying `OPTION=MIGRATE`; at the same time a backup master site is defined.
- The gateway server is redefined as three other servers: `TLRA`, `ACCTA`, and `CRDT`.
- Credit Authorization services are added.

Configuration File for Combining the Sample Applications

The following code shows a possible configuration file for the combined applications.

Listing 2-1 Sample Configuration File for the Combined Application

```
*RESOURCES
IPCKEY          76666
UID             0000
GID             000
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MASTER          SITE1,SITE2
SCANUNIT        10
MODEL           MP
LDBAL           Y
OPTIONS         LAN,MIGRATE
MAXGTT          100
MAXBUFTYPE      16
SCANUNIT        10
SANITYSCAN      5
DBBLWAIT        6
BBLQUERY        50
BLOCKTIME       2
#
#
*MACHINES
#
mach1 LMID=SITE1
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/bankapp"
                ENVFILE="/home/mylogin/bankapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/bankapp/tuxconfig"
                ULOGPFX="/home/mylogin/bankapp/ULOG"
                TYPE="type1"
#
mach2 LMID=SITE2
                TUXDIR="/home/mylogin/tuxroot"
```

```
                APPDIR="/home/mylogin/bankapp"
                ENVFILE="/home/mylogin/bankapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/bankapp/tuxconfig"
                ULOGPFX="/home/mylogin/bankapp/ULOG"
                TYPE="type2"
#
mach3 LMID=SITE3
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/bankapp"
                ENVFILE="/home/mylogin/bankapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/bankapp/tuxconfig"
                ULOGPFX="/home/mylogin/bankapp/ULOG"
                TYPE="type2"
#
mach4 LMID=SITE4
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/bankapp"
                ENVFILE="/home/mylogin/bankapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/bankapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/bankapp/tuxconfig"
                ULOGPFX="/home/mylogin/bankapp/ULOG"
                TYPE="type1"
#
*GROUPS
#
DEFAULT:  TMSNAME=TMS_SQL    TMSCOUNT=2
BANKB1    LMID=SITE1         GRPNO=1
OPENINFO="TUXEDO/SQL:/home/mylogin/bankapp/bankdl1:bankdb:readwrite"
BANKB2              LMID=SITE2         GRPNO=2
OPENINFO="TUXEDO/SQL:/home/mylogin/bankapp/bankdl2:bankdb:readwrite"
BANKB3              LMID=SITE3         GRPNO=3
OPENINFO="TUXEDO/SQL:/home/mylogin/bankapp/bankdl3:bankdb:readwrite"
BANKB4    LMID=SITE4         GRPNO=4
OPENINFO="TUXEDO/SQL:/home/mylogin/bankapp/bankdl4:bankdb:readwrite"
#
#
*NETWORK
#
SITE1     NADDR="<network address of SITE1>"
          BRIDGE="<device of provider1>"
          NLSADDR="<network listener address of SITE1>"
SITE2     NADDR="<network address of SITE2>"
          BRIDGE="<device of provider2>"
          NLSADDR="<network listener address of SITE2>"
```

```
SITE3    NADDR="<network address of SITE3>"
         BRIDGE="<device of provider3>"
         NLSADDR="<network listener address of SITE3>"
SITE4    NADDR="<network address of SITE4>"
         BRIDGE="<device of provider4>"
         NLSADDR="<network listener address of SITE4>"
#
*SERVERS
#
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A"
# Servers for the bankapp part
TLR     SRVGRP=BANKB1   SRVID=2
TLR     SRVGRP=BANKB2   SRVID=3         RQADDR=tlr2       CLOPT="-A -- -T 600"
TLR     SRVGRP=BANKB3   SRVID=4
XFER    SRVGRP=BANKB1   SRVID=10
XFER    SRVGRP=BANKB2   SRVID=6
XFER    SRVGRP=BANKB3   SRVID=8
ACCT    SRVGRP=BANKB1   SRVID=11
ACCT    SRVGRP=BANKB2   SRVID=7
ACCT    SRVGRP=BANKB3   SRVID=13
BTADD   SRVGRP=BANKB1   SRVID=12
BTADD   SRVGRP=BANKB2   SRVID=14
BTADD   SRVGRP=BANKB3   SRVID=16
# Servers for the Credit Authorization Part
TLRA    SRVGRP=BANKB4   SRVID=5                            CLOPT="-A -- -T 600"
ACCTA   SRVGRP=BANKB4   SRVID=9
CRDT    SRVGRP=BANKB4   SRVID=15
#
#
*SERVICES
#
DEFAULT:        LOAD=50         AUTOTRAN=N
# Services for the bankapp part
BR_ADD          PRIO=20         ROUTING=BRANCH_ID
TLR_ADD         PRIO=20         ROUTING=BRANCH_ID
WITHDRAWAL      PRIO=50         ROUTING=ACCOUNT_ID
DEPOSIT         PRIO=50         ROUTING=ACCOUNT_ID
TRANSFER        PRIO=50         ROUTING=ACCOUNT_ID
INQUIRY         PRIO=50         ROUTING=ACCOUNT_ID
CLOSE_ACCT      PRIO=40         ROUTING=ACCOUNT_ID
OPEN_ACCT       PRIO=40         ROUTING=BRANCH_ID
# Services for the Credit Authorization part
WITHDRAWALA     PRIO=50
INQUIRYA        PRIO=50
OPENCA          PRIO=40
CLOSECA         PRIO=40
DEPOSITA        PRIO=50
OPEN_ACCT2      PRIO=40
OPENC           PRIO=40
```

```
#
#
*ROUTING
#
ACCOUNT_ID          FIELD=ACCOUNT_ID
                    BUFTYPE="FML"
                    RANGES="10000-39999:BANKB1,
                             40000-69999:BANKB2,
                             70000-109999:BANKB3,
                             *:*"
BRANCH_ID       FIELD=BRANCH_ID
                    BUFTYPE="FML"
                    RANGES="1-3:BANKB1,
                             4-6:BANKB2,
                             7-10:BANKB3,
                             *:*"
#
```

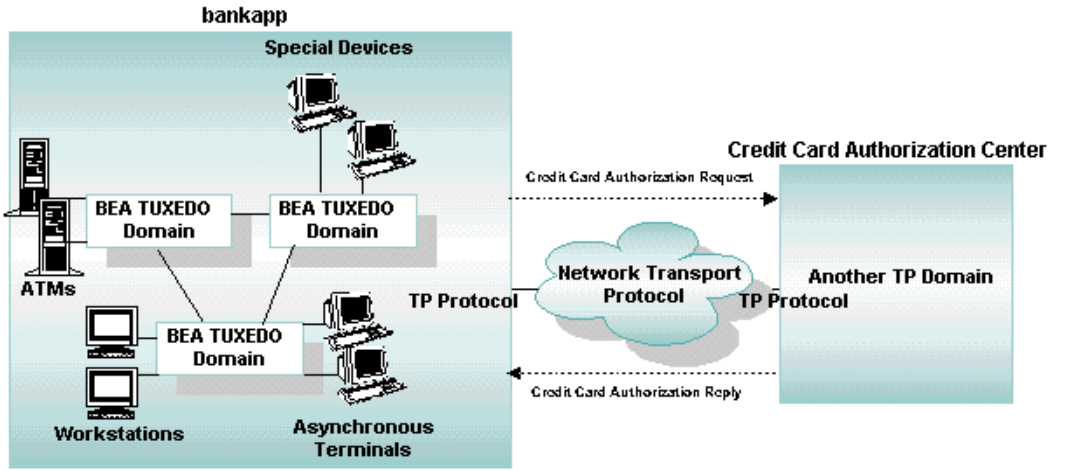
Limitations of Option 1

- Administering a single large application can be more cumbersome than administering two smaller ones; each smaller one has its own administrative interface.
- Booting a networked application can be more costly because of the time required to boot each server and because of the need to propagate bulletin boards across the network. Smaller, separate applications can be booted simultaneously.

Option 2: Redefine the Applications as Separate BEA Tuxedo Domains

The following figure shows the combined application reconfigured as four BEA Tuxedo domains (TDomains). (Three of the domains are in the left-hand box.)

Figure 2-3 Domains Configuration



Modifying the Application Configuration Files

To reconfigure the combined application as TDomains, make the following changes to the `UBBCONFIG` files:

- Change `MODEL` to `SHM`.
- Remove the `NETWORK` section.

  Note: You can use `MP` mode and also write the `NETWORK` section in a multiple domain environment depending on your specific application needs.

- Add domain-specific servers, for example `DMADM`, `GWADM`, and `GWTDOMAIN`, to the `SERVERS` section.

The following code shows a sample converted UBBCONFIG file.

Listing 2-2 Converted UBBCONFIG File

```
*RESOURCES
IPCKEY          76666
UID             7901
GID             601
PERM            0660
MAXACCESSERS    40
MAXSERVERS      35
MAXSERVICES     75
MAXCONV         10
MASTER          SITE1
SCANUNIT        10
MODEL           SHM
LDBAL           Y
MAXGTT          100
MAXBUFTYPE      16
SCANUNIT        10
SANITYSCAN      5
BBLQUERY        50
BLOCKTIME       2
#
*MACHINES
sfexpz          LMID=SITE1
                TUXDIR="/home/mylogin/tuxroot"
                APPDIR="/home/mylogin/creditapp"
                ENVFILE="/home/mylogin/creditapp/ENVFILE"
                TLOGDEVICE="/home/mylogin/creditapp/TLOG"
                TLOGNAME=TLOG
                TUXCONFIG="/home/mylogin/creditapp/tuxconfig"
                ULOGPFX="/home/mylogin/creditapp/ULOG"
                TYPE="type1"
#
#
#
*GROUPS
DEFAULT:        LMID=SITE1
BANKB1          GRPNO=1  TMSNAME=TMS_SQL          TMSCOUNT=2
OPENINFO="TUXEDO/SQL:/home/mylogin/creditapp/crdtdl1:bankdb:readwrite"
BANKB2          GRPNO=2
BANKB3          GRPNO=3
BANKB4          GRPNO=4
DMADMGRP  LMID=mach1 GRPNO=5
#
#
```

```
#
*SERVERS
#
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=Y CLOPT="-A"
GWADM           SRVGRP=BANKB2    SRVID=30
                REPLYQ = N RESTART = Y GRACE = 0
GWTDOMAIN       SRVGRP=BANKB2    SRVID=31
                REPLYQ = N RESTART = Y GRACE = 0
GWADM           SRVGRP=BANKB3    SRVID=24
                REPLYQ = N RESTART = Y GRACE = 0
GWTDOMAIN       SRVGRP=BANKB3    SRVID=25
                REPLYQ = N RESTART = Y GRACE = 0
GWADM           SRVGRP=BANKB4    SRVID=20
                REPLYQ = N RESTART = Y GRACE = 0
GWTDOMAIN       SRVGRP=BANKB4    SRVID=21
                REPLYQ = N RESTART = Y GRACE = 0
DMADM           SRVGRP="DMADMGRP"   SRVID=50
                REPLYQ = N RESTART = Y GRACE = 0
TLRA      SRVGRP=BANKB1  SRVID=2         CLOPT="-A -- -T 100"
BTADD     SRVGRP=BANKB1  SRVID=3
ACCTA     SRVGRP=BANKB1  SRVID=4
CRDT      SRVGRP=BANKB1  SRVID=5
CRDTA     SRVGRP=BANKB1  SRVID=6
*SERVICES
DEFAULT:        LOAD=50
INQUIRYA        PRIO=50
WITHDRAWALA     PRIO=50
OPEN_ACCT2      PRIO=40
OPENC           PRIO=40
OPENCA          PRIO=40
CLOSECA         PRIO=40
BR_ADD          PRIO=20
TLR_ADD         PRIO=20
```

Adding DMCONFIG Files

You also need to create four DMCONFIG files, as shown in the following sample.

Listing 2-3 Sample DMCONFIG File

```
#
#
*DM_LOCAL_DOMAINS
#
#
QDOM1   GWGRP=BANKB2
        TYPE=TDOMAIN
        DOMAINID=QDOM1
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"

QDOM2   GWGRP=BANKB3
        TYPE=TDOMAIN
        DOMAINID=QDOM2
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"
        DMTLOGNAME="DMTLOG_TDOM2"
QDOM3   GWGRP=BANKB4
        TYPE=TDOMAIN
        DOMAINID=QDOM3
        BLOCKTIME=10
        MAXDATALEN=56
        MAXRDOM=89
        DMTLOGDEV="/home/mylogin/creditapp/DMTLOG"
        AUDITLOG="/home/mylogin/creditapp/AUDITLOG"
        DMTLOGNAME="DMTLOG_TDOM3"
#
*DM_REMOTE_DOMAINS
#
#
TDOM1   TYPE=TDOMAIN
        DOMAINID=TDOM1
```

```
TDOM2   TYPE=TDOMAIN
        DOMAINID=TDOM2

TDOM3   TYPE=TDOMAIN
        DOMAINID=TDOM3
#
#
*DM_TDOMAIN
#
QDOM1    NWADDR="0x0002DEEF93026927"
         NWDEVICE="/dev/tcp"
QDOM2    NWADDR="0x0002BEEF93026927"
         NWDEVICE="/dev/tcp"
QDOM3    NWADDR="0x0002CEEF93026927"
         NWDEVICE="/dev/tcp"
TDOM1    NWADDR="0x0002DEEF93026947"
         NWDEVICE="/dev/null"
TDOM2    NWADDR="0x0002BEEF9302691D"
         NWDEVICE="/dev/tcp"
TDOM3    NWADDR="0x0002CEEF9302690E"
         NWDEVICE="/dev/tcp"
#
#
#
*DM_LOCAL_SERVICES
#
#
WITHDRAWALA
INQUIRYA
OPENCA
CLOSECA
```

Sample Domains Application: creditapp

A sample application, `creditapp`, is distributed with the BEA Tuxedo system. `creditapp` is a runnable version of the hypothetical application that was the basis for separating `bankapp` and the credit card application into domains, as discussed earlier in this topic.

The application is located in *`TUXDIR`*`/apps/creditapp` and includes the following files.

Listing 2-4 creditapp Files

```
ACCT.ec         ACCTA.ec        AUDITC.c        BAL.ec          BALANCE.m
BALANCEA.m      BALC.ec         BTADD.ec        CBALANCE.m      CCLOSE.m
CDEPOSIT.m      CLOSE.m         COPEN.m         CRDT.ec         CRDTA.ec
CRMENU.m        CRMENU2.m       CTRANSFER.m     CWITHDRAW.m     DEPOSIT.m
DEPOSITA.m      FILES           HCBALANCE.m     HCCLOSE.m       HCLOSE.m
HCOPEN.m        HCWITHDRAW.m    HELP.m          HOPEN.m         OPEN.m
README          RUNME           RUNME.sh        SETUP.sh        TLR.ec
TLR1.ec         TLR2.ec         TLR3.ec         TLRA.ec         TRANSFER.m
WITHDRAW.m      WITHDRAWA.m     XFER.c          appinit.c       aud.h
aud.v           audit.c         auditcon.c      bank.flds       bank.flds.h
bank.h          cleanup.sh      crbank.sh       crbankdb.sh     crdt_app.mk
crdt_app2.mk    crdt_app3.mk    crdt_app4.mk    crdt_flds.h     crdtvar
crdtvar2        credit.flds     crtlog.sh       crtlog2         crtlog2.sh
domcon1         domcon2         domcon3         domcon4         driver.sh
envfile.sh      gendata.c       gentran.c       hostmk          listnr
populate.sh     run.sh          setenv          ubbdom1         ubbdom2
ubbdom3         ubbdom4         util.c
```

The creditapp README File

The following `README` file is from the `creditapp` directory. The `README` file documents a script that installs and runs `creditapp`. It has been edited to include a few things that were not included in the original script.

Listing 2-5 README File for creditapp

```
SIMPLE BUILD PROCEDURE

The creditapp application is an enhancement of the
bankapp and hostapp applications.

The creditapp application is designed to be a four domain application,
so the software must be built on four machines. The RUNME.sh
script will lead you through the necessary steps.


Step 1: Copy the Software for creditapp.

Make a new directory under your $HOME directory and copy all of
the source files from <TUXDIR>/apps/creditapp into that directory.
```

```
TUXDIR is the root directory under which your BEA TUXEDO System
software is installed. We call the new directory
$HOME /creditapp. The rest of the steps in this procedure are
done in the directory $HOME/creditapp.


Step 2: On each of the remaining three machines:

Make a directory creditapp in a directory that can be used for the application.

We call this directory $HOME/creditapp.

Make a note of the full directory path for $HOME/creditapp and TUXDIR
for each machine. These will be needed by the RUNME.sh script.


Step 3: On the “master site” execute the “RUNME.sh” script.

The shell script “RUNME.sh” is an interactive program designed to
lead you through initialization, booting, shutdown and cleanup
of the four domain creditapp application. The shell is interactive
and requires no command line arguments. All you need in the directory
is the source from the TUXDIR/apps/creditapp directory that you
copied in Step 1.

You will be prompted to enter values for RSH and RCP
environment variables, or accept the defaults.

IT IS VERY IMPORTANT THAT VALUES FOR RSH AND RCP BE ENTERED AS THEY ARE
USED TO REMOTE COPY AND EXECUTE THE NECESSARY SCRIPTS.

The following environment variables are important. The script picks up
the values for TUXDIR and APPDIR from your environment and
prompts you (in OPTION 4) for BLKSIZE:

TUXDIR          Root directory of the BEA TUXEDO System where you have
                  installed the software.

APPDIR           Directory in which the creditapp application resides.
                  crdtvar.dm1 initially is set to allow this to default
                  to the current working directory, which agrees with
                  our intention to use $HOME/creditapp. This is the
                  directory into which you copied the creditapp files in
                  Step 1.

BLKSIZE          Logical blocksize for the database in bytes.
                  Must be an integral multiple of the physical
                 page size of the computer (for example, 512 bytes or 4096 bytes).

When you invoke RUNME.sh you are shown a menu with 10 options (11 counting “quit”).
Here is the list of choices:
```

```
          1) Initialize configuration files and makefiles.
          2) Copy files to remote sites.
          3) Build crdtapp clients and servers.
          4) Create databases.
          5) Generate binary tuxconfig and bdmconfig files.
          6) Create Transaction Log file.
          7) Boot the application.
          8) Populate the database.
          9) Shutdown the application.
          10) Cleanup IPC Resources, database files and log files.
          q) Quit.

To go through the complete process of building and running the sample
application, start with choice No. 1. When the script completes a step,
the menu is displayed for your next choice.


OPTION 1. Initialize configuration files and makefiles.
          This option sets up makefiles, UBBCONFIG and DMCONFIG files that are
          necessary for the application.

          All questions must be answered.

          ENTER the system name: enter uname for machines you are using
                                 beginning with the current machine you are on.


          ENTER TUXDIR for each machine.

          ENTER APPDIR for each machine.


          Continue to answer all queries.


          An example of 4 hexadecimal digits may be (beef, cfff, 6774, aeef).
NOTE: EACH MACHINE MUST HAVE A UNIQUE HEX SEQUENCE.


OPTION 2. Copies the files to the other domains in the configuration.


OPTION 3. Builds clients and servers on all machines.

          NOTE: CAREFULLY CHECK THAT THE BUILDS ARE COMPLETED SUCCESSFULLY ON
                EACH SITE. IF NECESSARY YOU MAY RUN THE BUILD YOURSELF.

                ON THE SPECIFIC SITE ENTER
                nohup make -f CRDT{$MACH}.mk2
```

```
                where ${MACH} is the uname for the machine you are building on.
                For example,

                        nohup make -f CRDTtux1.mk2

OPTION 4. Builds the databases on each site.

          NOTE: ON EACH SITE MAKE SURE THE BLKSIZE VALUE IN files

                crdt${MACH}.dm1 for the primary site

                or crdt${MACH}.dm2 for the remote sites

                where ${MACH} is the uname for the machine you are building on

                ARE CORRECT FOR THAT SPECIFIC MACHINE

OPTION 5. Generates the tuxconfig and bdmconfig files.

All other options are similar to bankapp.

After OPTION 8 : Populate the database

                  Enter q to Quit the menu.

RUNNING CREDITAPP.
__________________

            On each machine a script run.sh exists.

            Execute run.sh.

            run
At the response :

               Is this machine the Credit Card Authorization Center(y/n)?

               If machine is the primary machine answer y .
               If machine is any other answer n.

On the primary machine a different menu will be seen than the other 3 machines.

All Credit accounts exist on primary machine and all machines can access any
account.

            ACCOUNTS 10000000 - 120000000
```

```
Machines 2,3,4 are the enhanced bankapp application.

            ACCOUNTS 10000 - 39999 exist on machine 2
            ACCOUNTS 40000 - 79999 exist on machine 3
            ACCOUNTS 80000 - 109999 exist on machine 4

All processing is done using the /DOMAIN software.

A tail -f of the ULOG###### will show the actual processing of the requests.

On the machine that will process the request enter :

           tail -f ULOG###### where ###### is today's date.
```

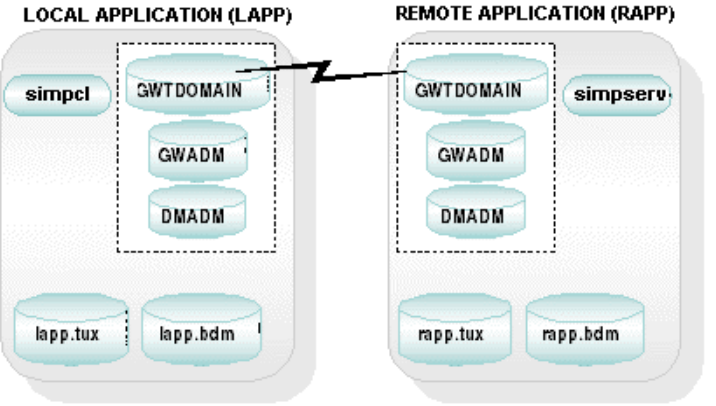
Configuring a Domains Environment

To configure a Domains environment, an administrator needs to specify all the information a BEA Tuxedo domain needs to know about other domains. This information includes services imported from other domains, addressing and security parameters for contacting remote domains, access control lists, services exported to these domains, whether data-dependent routing is used, and parameters for controlling access to exported services. This information is defined in the `UBBCONFIG` configuration file and in the `DMCONFIG` configuration file.

Configuring a Sample Domains Application (simpapp)

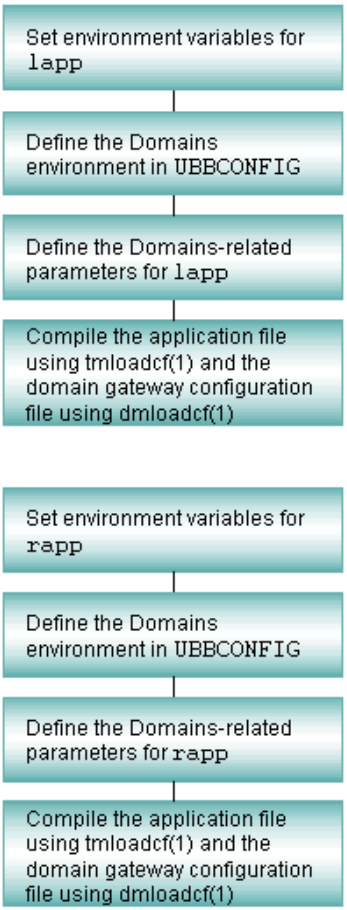
The Domains example illustrated in the following figure consists of two applications: `lapp`, a *local application*, and `rapp`, a *remote application*. Both are based on the `simpapp` example provided with the BEA Tuxedo system. `lapp` is configured to allow its clients to access a service called `TOUPPER`, which is advertised in `rapp`.

Figure 2-4 Local and Remote Applications in simpapp



Configuration Tasks

The following tasks are required to configure the `simpapp` domain consisting of two applications: `lapp` (the local application) and `rapp` (the remote application).



How to Set Environment Variables for lapp

You need to set the following environment variables for the application to be configured successfully:

- TUXDIR—the BEA Tuxedo system root directory (for example, /opt/tuxedo)
- TUXCONFIG—the application configuration file (for example, lapp.tux)
- BDMCONFIG—the domain gateway configuration file (for example, lapp.bdm)
- PATH—must include *TUXDIR*/bin
- LD_LIBRARY_PATH—must include *TUXDIR*/lib (this pathname varies, depending on your operating system)

Example

```
$ TUXDIR=/opt/tuxedo
$ TUXCONFIG=/home/lapp/lapp.tux
$ BDMCONFIG=/home/lapp/lapp.dom
$ PATH=$TUXDIR/bin:$PATH
$ LD_LIBRARY_PATH=$TUXDIR/lib:$LD_LIBRARY_PATH
$ export TUXDIR TUXCONFIG BDMCONFIG PATH LD_LIBRARY_PATH
```

How to Define the Domains Environment for lapp (in the ubbconfig File)

For the sample local application configuration file, `lapp.ubb`, only the required parameters are defined. Default settings are used for the other parameters.

Two server groups are defined:

- The first group contains the Domains administrative server (`DMADM`).
- The second group contains the gateway administrative server (`GWADM`) and the domain gateway (`GWTDOMAIN`).

  Note: For a gateway type other than `GWTDOMAIN`, an executable other than `GWTDOMAIN` must be used. Refer to the BEA eLink for Mainframe documentation and *Using the BEA Tuxedo TOP END Domain Gateway* for additional information.

Server Definitions

- `DMADM`—the Domains administrative server enables run-time modification of the configuration information, required by domain gateway groups, that resides in the binary Domains configuration file. `DMADM` supports a list of registered gateway groups. There must be only one instance of `DMADM` per Domains application.
- `GWADM`—the gateway administrative server enables run-time administration of a particular domain gateway group. This server gets Domains configuration information from the `DMADM` server. It also provides administrative functionality and transaction logging for the gateway group.
- `GWTDOMAIN`—the Domains gateway server enables access to and from remote Domains, allowing interoperability of two or more BEA Tuxedo domains. Information about the local and remote services it needs to export and import is included in the Domains configuration file. The Domains gateway server should always be configured with `REPLYQ=N`.

Example of an Application Configuration File for lapp

Listing 2-6 Example of an Application Configuration File (lapp.ubb)

```
# lapp.ubb
#
*RESOURCES
IPCKEY          111111

MASTER          LAPP
MODEL           SHM

*MACHINES
giselle

                LMID=LAPP
                TUXDIR="/opt/tuxedo"
                APPDIR="/home/lapp"
                TUXCONFIG="/home/lapp/lapp.tux"

*GROUPS

LDMGRP          GRPNO=1 LMID=LAPP
LGWGRP          GRPNO=2 LMID=LAPP

*SERVERS

DMADM           SRVGRP=LDMGRP SRVID=1
GWADM           SRVGRP=LGWGRP SRVID=1
GWTDOMAIN       SRVGRP=LGWGRP SRVID=2 REPLYQ=N

*SERVICES
```

How to Define Domains Parameters for lapp (in the DMCONFIG File)

For the sample local Domain gateway configuration file, `lapp.dom`, only the required parameters are defined. Default settings are used for optional parameters.

The `DM_LOCAL_DOMAIN` section identifies the local domains and their associated gateway groups. This section has one entry, `LAPP`, and specifies the following parameters required for the domain gateway processes in that group:

- `GWGRP` specifies the name of the gateway server group as specified in the application.
- `TYPE` of `TDOMAIN` indicates that the local domain will be communicating with another BEA Tuxedo domain. This parameter indicates the protocol used by the gateways. Other options include SNA, OSI TP, TOP END Domain gateway, TCP for CICS, and TCP for IMS.
- `DOMAINID` identifies the name of the domain gateway and must be unique across all domains.

The `DM_REMOTE_DOMAINS` section identifies the known set of remote domains and their characteristics. This section has one entry (`RAPP`). `TYPE` is used to classify the type of domains. `DOMAINSID` is a unique domain identifier.

The `DM_TDOMAIN` section defines the addressing information required by the BEA Tuxedo Domains component. Following are entries in the section for each local and remote domain specified in this configuration file:

- `NWADDR` specifies either the network address at which connections will be accepted from other BEA Tuxedo domains (local domain entry), or the network address at which connections to other BEA Tuxedo domains will be made (remote domain entry).

The `DM_LOCAL_SERVICES` section provides information about the services that are exported. This section of our sample file has no entries because no services are being exported.

The DM_REMOTE_SERVICES section provides information about the services that are imported. The TOUPPER service is imported so that it can be accessed by clients in the local domain.

Example of a Domain Gateway Configuration File for lapp

Listing 2-7 Example of a Domain Gateway Configuration File (lapp.dom)

```
#
# lapp.dom
#
*DM_LOCAL_DOMAINS

LAPP          GWGRP=LGWGRP
              TYPE=TDOMAIN
              DOMAINID=”111111"

*DM_REMOTE_DOMAINS

RAPP          TYPE=TDOMAIN
              DOMAINID=”222222"

*DM_TDOMAIN

LAPP           NWADDR=”//mach1:5000"

RAPP           NWADDR=”//mach2:5000"

*DM_LOCAL_SERVICES

*DM_REMOTE_SERVICES

TOUPPER
```

How to Compile Application and Domains Gateway Configuration Files for lapp

The local application configuration file (`lapp.ubb`) contains the information necessary to boot the local application. You must compile this file into a binary data file (`lapp.tux`) by running `tmloadcf(1)`.

The local domain gateway configuration file (`lapp.dom`) contains the information used by the domain gateway for one domain for communication with other domains. You must compile this file into a binary data file (`lapp.bdm`) by running `dmloadcf(1)`.

To compile both configuration files, complete the procedure shown in the following sample session.

```
$ cd /home/lapp
$ TUXCONFIG=/home/lapp/lapp.tux; export TUXCONFIG
$ tmloadcf -y lapp.ubb
$ BDMCONFIG=/home/lapp/lapp.dom; export BDMCONFIG
$ dmloadcf -y lapp.dom
```

Once you create both the local and remote domains, you can then boot the application using `tmboot(1)`. The order in which the two domains are booted does not matter. Monitor the applications with `dmadmin(1)`. Once both applications are booted, a client in the local application can call the `TOUPPER` service residing in the remote application.

```
$ tmboot -y
```

How to Set Environment Variables for rapp

You must set the following environment variables for an application to be configured successfully:

- TUXDIR—the BEA Tuxedo system root directory (for example, /opt/tuxedo)
- TUXCONFIG—the full path name of the application configuration file (for example, rapp.tux)
- BDMCONFIG—the full path name of the domain gateway configuration file (for example, rapp.bdm)
- PATH—must include *TUXDIR*/bin
- LD_LIBRARY_PATH—must include *TUXDIR*/lib (this pathname varies, depending on your operating system)

Example

```
$ TUXDIR=/opt/tuxedo
$ TUXCONFIG=/home/rapp/rapp.tux
$ BDMCONFIG=/home/rapp/rapp.dom
$ PATH=$TUXDIR/bin:$PATH
$ LD_LIBRARY_PATH=$TUXDIR/lib:$LD_LIBRARY_PATH
$ export TUXDIR PATH LD_LIBRARY_PATH TUXCONFIG BDMCONFIG
```

How to Define the Domains Environment for rapp (in the UBBCONFIG File)

For the sample remote application configuration file, `rapp.ubb`, only the required parameters are defined. Default settings are used for optional parameters.

The following three server groups are defined:

- The first server group (`SRVGP=RDMGRP`) contains the Domains administrative server (`DMADM`).
- The second server group (`SRVGP=RGWGRP`) contains the gateway administrative server, `GWADM`, and the domain gateway, `GWTDOMAIN`.
- The third server group (`SRVGP=APPGRP`) contains the application server `simpserv`.

The following four servers are defined:

- `DMADM`—Domains administrative server
- `GWADM`—gateway administrative server
- `GWTDOMAIN`—Domains gateway server
- `simpserv`—application server for `simpapp` that advertises the `TOUPPER` service, which converts strings from lowercase to uppercase characters

Example of an Application Configuration File for rapp

Listing 2-8 Example of an Application Configuration File (rapp.ubb)

```
# rapp.ubb
#
*RESOURCES
IPCKEY          222222

MASTER          RAPP

MODEL           SHM

*MACHINES

juliet

                LMID=RAPP
                TUXDIR="/opt/tuxedo"
                APPDIR="/home/rapp"
                 TUXCONFIG="/home/rapp/rapp.tux"

*GROUPS

RDMGRP          GRPNO=1 LMID=RAPP
RGWGRP          GRPNO=2 LMID=RAPP
APPGRP          GRPNO=3 LMID=RAPP

*SERVERS

DMADM           SRVGRP=RDMGRP SRVID=1
GWADM           SRVGRP=RGWGRP SRVID=1
GWTDOMAIN       SRVGRP=RGWGRP SRVID=2 REPLYQ=N
simpserv        SRVGRP=APPGRP SRVID=1

*SERVICES
TOUPPER
```

How to Define Domains Parameters for rapp (in the DMCONFIG File)

For the sample remote Domain gateway configuration file, `rapp.dom`, only the required parameters are defined. Default settings are used for optional parameters.

This configuration file is similar to the local Domains gateway configuration file. The difference is that the two files list different services to be exported and imported.

The `DM_LOCAL_SERVICES` section provides information about the services exported by each local domain. In this example, the `TOUPPER` service is exported and included in the `DM_LOCAL_SERVICES` section. No service is imported so there are no entries in the `DM_REMOTE_SERVICES` section of our sample file.

Example of a Domain Gateway Configuration File for rapp

Listing 2-9 Example of a Domain Gateway Configuration File (rapp.dom)

```
# rapp.dom
#

*DM_LOCAL_DOMAINS

RAPP        GWGRP=RGWGRP
            TYPE=TDOMAIN
            DOMAINID=”222222"

*DM_REMOTE_DOMAINS

LAPP        TYPE=TDOMAIN
            DOMAINID=”111111"

*DM_TDOMAIN

RAPP        NWADDR=”//mach2:5000"

LAPP        NWADDR=”//mach1:5000"
```

```
*DM_LOCAL_SERVICES
TOUPPER
*DM_REMOTE_SERVICES
```

How to Compile Application and Domain Gateway Configuration Files for rapp

The remote application configuration file (`rapp.ubb`) contains the information used by the domain gateway for one domain, for communication with other domains. You must compile this file into a binary data file (`rapp.tux`).

The remote domain gateway configuration file (`rapp.dom`) contains the information used by domain gateways to initialize the context required for communications with other domains. This configuration file is similar to the local domain gateway configuration file. The difference is that the two files list different services to be exported and imported. You must compile this file into a binary data file (`rapp.bdm`).

```
$ cd /home/rapp
$ TUXCONFIG=/home/rapp/rapp.tux; export TUXCONFIG
$ tmloadcf -y rapp.ubb
$ BDMCONFIG=/home/rapp/rapp.dom; export BDMCONFIG
$ dmloadcf -y rapp.dom
```

Once you create both the local and remote domains, you can then boot the application using `tmboot(1)`. The order in which the two domains are booted does not matter. Monitor the applications with `dmadmin(1)`. Once both applications are booted, a client in the local application can call the `TOUPPER` service residing in the remote application.

```
$ tmboot -y
```

See Also

- “What Is the Domains Configuration File?” on page 1-19
- “How to Compress Data Between Domains” on page 2-32
- “How to Route Service Requests to Remote Domains” on page 2-32

- "Converting the Domains Configuration File" on page 1-21
- DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- UBBCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

How to Compress Data Between Domains

Data sent between domains can be compressed for faster performance. To configure compression, set the CMPLIMIT parameter in DMCONFIG.

See Also

- DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- "Compressing Data Over a Network" on page 4-2 in *Administering a BEA Tuxedo Application at Run Time*

How to Route Service Requests to Remote Domains

Data-dependent routing information used by gateways to send service requests to specific remote domains is provided in the DM_ROUTING section of the DMCONFIG file. The FML32, VIEW32, FML, VIEW, X_C_TYPE, and X_COMMON typed buffers are supported.

To create a routing table for a domain, specify the following:

- Buffer type for which the routing entry is valid

- Name of the routing entry and field
- Ranges and associated remote domain names of the routing field.

The following table describes these fields.

Routing Table Field	**Description**

Buffer type	A list of types and subtypes of data buffers for which this routing entry is valid. The types may be included: `FML32`, `VIEW32`, `FML`, `VIEW`, `X_C_TYPE`, and `X_COMMON`. No subtype can be specified for type `FML`; subtypes are required for the other types. The * (or *wildcard*) value is not allowed. Duplicate *`type`*/*`subtype`* pairs cannot be specified for the same routing criteria name; one criteria name can be specified in multiple routing entries as long as the *`type`*/*`subtype`* pairs are unique. If multiple buffer types are specified for a single routing entry, the data types of the routing field for all buffer types must be the same.
	
	Valid values for *`type`* are:
	`[:`*`subtype1`*`[,`*`subtype2`* `. . .]]`
	`[;`*`type2`*`[:`*`subtype3`*`[,`*`subtype4`* `. . . ]]] . . .`
	The maximum total length of 32 *`type`*/*`subtype`* combinations is 256 characters.
	Valid values for *`subtype`* may not include semicolons, colons, commas, or asterisks.
	Example: `FML`

Domain routing criteria	The name (identifier) of the routing entry. A valid value is any string of 1-15 characters, inclusive. Example: `ROUTTAB1`
Routing field name	The name of the routing field. It is assumed that the value of this field is a name identified in an `FML` field table (for `FML` buffers) or an `FML VIEW` table (for `VIEW`, `X_C_TYPE`, or `X_COMMON` buffers). A valid value is an identifier string that is 1-30 characters, inclusive. Example: `FIELD1`

Routing Table Field	Description (Continued)
Ranges	A value comprised of a set of numbers (that must have numeric values) and an alphanumeric string (that must have string values) associated with remote domain names (`RDOM`) for the routing field. The routing field can be of any data type supported in `FML`. String range values for `string`, `carray`, and character field types must meet the following criteria: ■ Placed inside a pair of single quotes and not preceded by a sign. ■ Short and long integer values are a string of digits, optionally preceded by a plus or minus sign. ■ Floating point numbers are of the form accepted by the C compiler or a `tof()` as follows: an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer. When a field value matches a range, the associated `RDOM` value specifies the remote domains to which the request should be routed. An `RDOM` value of * indicates that the request can go to any remote domain known by the gateway group. Valid values for this field are a comma-separated ordered list of *`range`*/*`RDOM`* pairs where a *`range`* is one of two types: (a) a single value (signed numeric value or character string in single quotes); or (b) a range of the form *`lower-upper`* (where *`lower`* and *`upper`* are both signed numeric values or character strings in single quotes). Note that *`lower`* must be less than or equal to *`upper`*. Within a *`range`*/*`RDOM`* pair, the range is separated from the `RDOM` by a colon (:). `MIN` can be used to indicate the minimum value for the data type of the associated `FIELD:` ■ For `strings` and `carrays`, it is the null string ■ For character fields, it is `0` ■ For numeric values, it is the minimum numeric value that can be stored in the field. `MAX` can be used to indicate the maximum value for the data type of the associated `FIELD:` ■ For `strings` and `carrays`, it is an unlimited string of octal-255 characters ■ For a character field, it is a single octal-255 character ■ For numeric values, it is the maximum numeric value that can be stored in the field. Thus, `MIN - -5` is all numbers less than or equal to -5; and `6 - MAX` is the set of all numbers greater than or equal to 6. The metacharacter * (*`wildcard`*) in the range position indicates any values not covered by other ranges previously seen in the entry; one wildcard range is allowed per entry, which should be listed last in the field (ranges following it are ignored). Example: `1-100:REMDOM3`

Setting Up Security in Domains

The BEA Tuxedo ATMI environment provides the following application security mechanisms:

- Authentication—proves the stated identity of users or system processes; safely remembers and transports identity information; and makes identity information available when needed. The default authentication plug-in provides security at three levels: no authentication (`NONE`), application password (`AP_PW`), and user-level authentication (`USER_AUTH`).
- Authorization—controls access to resources based on identity or other information. The default authorization plug-in provides security at two levels: optional access control lists (`ACL`) and mandatory access control lists (`MANDATORY_ACL`).
- Auditing—safely collects, stores, and distributes information about operating requests and their outcomes. Default auditing security is implemented by the BEA Tuxedo EventBroker and userlog (ULOG) features.
- Encryption—security mechanisms to convert data to coded format that is unintelligible to users. The default encryption is RC4 symmetric key encryption.
- Security Plug-in Interface—allows installation of third-party security systems such as custom authentication, authorization, and auditing. The plug-in interface is available to applications running BEA Tuxedo release 7.1 or later software. For information on setting up security in domains using the security plug-in interface, see “Establishing a Link Between Domains” on page 2-24 in *Using Security in ATMI Applications*.

Impact of BEA Tuxedo Application Security on Domains Security

The BEA Tuxedo security mechanisms provided for individual applications and those provided for Domains configurations are relatively independent but compatible:

- The BEA Tuxedo ATMI environment provides the following security mechanisms for Domains configurations:
  - authentication of remote domains
  - access control on exported local services for remote domains
  - encryption mechanisms to protect interdomain communication
- If BEA Tuxedo application security is set to `USER_AUTH` or above, *and* the `ACL_POLICY` and `CREDENTIAL_POLICY` parameters are set to `GLOBAL` in the `DM_REMOTE_DOMAINS` section of the `DMCONFIG`, then user IDs flow across domains with requests. `ACL` checking is performed on the user IDs.
- If BEA Tuxedo application security is set to `USER_AUTH` or above, *and* the `ACL_POLICY` and `CREDENTIAL_POLICY` parameters are set to `LOCAL` in the `DM_REMOTE_DOMAINS` section of the `DMCONFIG`, then user IDs do *not* flow across domains with requests. `ACL` checking is handled via the Domain ID of the requesting domain.
- Even if you assign a security level of `NONE` to your BEA Tuxedo application, you can still set the Domains connection security to `DM_PW`. Note, however, that in order to use an application password in a Domains configuration, you must have already set a value of `APP_PW` (or higher: `USER_AUTH`, `ACL`, or `MANDATORY_ACL`) for the security level in each participating application.

Domains Security Mechanisms

Because distinct domains may exist under different ownership, the native BEA Tuxedo application password scheme may not, of itself, provide sufficient security. Domains, therefore, provides additional security mechanisms:

- Access Control Lists—restricts availability of resources in a local domain to a list of selected remote domains. You configure this security level in the DM_ACCESS_CONTROL section of DMCONFIG.
- Domains Authentication—techniques are required to ensure the proper identity of each remote domain. Domains provides three levels of password security: NONE specifies no authentication; APP_PW is authentication using the application password, which must match on the two domains; and DM_PW, which is authentication using specific passwords per local/remote domain pair. Each of these is selected by setting the SECURITY parameter in the DM_LOCAL_DOMAINS section for the local domain access point involved to the required level (NONE, APP_PW, DM_PW).
- Link-Level Encryption—you can use encryption across domains to ensure data privacy. In this way, a network-based eavesdropper cannot learn the content of BEA Tuxedo messages or application-generated messages flowing from one domain gateway to another. You configure this security mechanism by setting the MINENCRYPTBITS and MAXENCRYPTBITS parameters in the DMCONFIG file.
- Local Domains Access—restricts local services to remote domains. If a service is not exported to remote domains, it is simply unavailable to them. A service is exported by placing an entry in the DM_LOCAL_SERVICES section of the DMCONFIG file for the service.
- User Identity Mapping to Mainframes—provides a mechanism whereby user identities within a domain can be mapped to and from external user identities. This mechanism is currently used by BEA eLink for Mainframe-SNA to map to and from RACF (remote access control facility) user names on IBM LU6.2 mainframes. To use this mechanism, refer to the following dmadmin configuration commands:
  - addumap—add local user mappings to remote user mappings for a local/remote domain pair. Mappings are defined to be inbound, outbound, or both.

- `addusr`—add remote usernames and passwords to the remote user and password tables of a remote domain.
- `delumap`—delete local to remote user mappings for a local/remote domain pair.
- `delusr`—delete remote usernames and passwords from the remote user and password tables of a remote domain.
- `modusr`—change remote passwords in the password tables of a remote domain.

See Also

- “How to Create a Domains Access Control List (ACL)” on page 2-39
- “How to Set Up Domains Authentication” on page 2-42
- “Examples of Coding Security Between Domains” on page 2-44
- `dmadmin(1)`in the *BEA Tuxedo Command Reference*

How to Create a Domains Access Control List (ACL)

To create a domain ACL, you must specify the name of the domain ACL and a list of the remote domains that are part of the list (the Domain Import `VIEW` List) in the `DM_ACCESS_CONTROL` section of the `DMCONFIG` file. The following table describes these two fields.

Domain ACL Field	Description
Domain ACL name	The name of this ACL. A valid name consists of a string of 1-30 characters, inclusive. It must be printable and it may not include a colon, a pound sign, or a newline character. Example: `ACLGRP1`
Remote Domain list	The list of remote domains that are granted access in this access control list. A valid value in this field is a set of one or more comma-separated remote domain names. Examples: `REMDOM1,REMDOM2,REMDOM3`

Using Standard BEA Tuxedo Access Control Lists with Imported Remote Services

A remote service imported from a remote domain is viewed simply as a service within a BEA Tuxedo domain. The standard BEA Tuxedo ACL mechanism then, can be used to restrict access to this service by particular groups of users.

For information on using BEA Tuxedo access control lists, refer to the following entries in the *BEA Tuxedo Command Reference*: `tpacladd(1)`, `tpaclmod(1)`, `tpacldel(1)`, `tpusradd(1)`, `tpusrmod(1)`, `tpusrdel(1)`, `tpgrpadd(1)`, `tpgrpmod(1)`, and `tpgrpdel(1)`.

Setting the ACL Policy for a Remote Domain

As the administrator, you can use the following configuration parameters to set and control the access control list (ACL) policy for remote domains running BEA Tuxedo release 7.1 or later software.

Parameter Name	Description	Setting
ACL_POLICY in DMCONFIG (TA_DMACLPOLICY in DM_MIB)	May appear in the DM_REMOTE_DOMAINS section of the DMCONFIG file for each remote domain access point. Its value for a particular remote domain access point determines whether or not the local domain gateway modifies the identity of service requests received from the remote domain.*	LOCAL or GLOBAL. Default is LOCAL. LOCAL means that the local domain modifies the identity of service requests received from this remote domain to the principal name specified in the LOCAL_PRINCIPAL_NAME parameter for this remote domain. GLOBAL means that the local domain uses any credential it might receive from the remote domain on inbound service requests. If no credential is received from the remote domain then the service request will be forwarded to the service without credentials (which will usually fail). **Note:** This parameter controls whether or not the local domain *accepts* a credential from a remote domain. A parameter related to this one is CREDENTIAL_POLICY, which controls whether or not a local domain *sends* credentials to the remote domain.
LOCAL_PRINCIPAL_NAME in DMCONFIG (TA_DMLOCALPRINCIPALNAME in DM_MIB)	May appear in the DM_REMOTE_DOMAINS section of the DMCONFIG file for each remote domain access point. If the ACL_POLICY parameter is set (or defaulted) to LOCAL for a particular remote domain access point, the local domain gateway modifies the identify of service requests received from the remote domain to the principal name specified in LOCAL_PRINCIPAL_NAME.	1 - 511 characters. If not specified, the principal name defaults to the DOMAINID string for the remote domain access point.

* A remote domain access point is also known as an RDOM (pronounced “are dom”) or simply *remote domain*.

For more information about ACL Policy, refer to Chapter 2, "Administering Security," in *Using Security in ATMI Applications*.

Setting the Credential Policy for a Remote Domain

As the administrator, you can use the following configuration parameters to set and control the credential policy for remote domains running BEA Tuxedo release 8.0 or later software.

Parameter Name	Description	Setting
CREDENTIAL_POLICY in DMCONFIG (TA_DMCREDENTIAL POLICY in DM_MIB)	May appear in the DM_REMOTE_DOMAINS section of the DMCONFIG file for each remote domain access point. Its value for a particular remote domain access point determines whether or not the local domain gateway modifies the identity of service requests received from the remote domain.*	LOCAL or GLOBAL. Default is LOCAL. If the policy is LOCAL then the domain will *not* attach the credentials of the user that originated a request with the invocation to the remote domain. If the policy is GLOBAL then the domain will attach the credentials of the user that originated a request with the invocation to the remote domain. **Note:** This parameter controls whether or not user credentials are *sent* to a remote domain. A parameter related to this one is ACL_POLICY, which controls whether or not incoming credentials are *accepted* by a domain.

* A remote domain access point is also known as an RDOM (pronounced "are dom") or simply *remote domain*.

For more information about Credential Policy, refer to Chapter 2, "Administering Security," in *Using Security in ATMI Applications*.

How to Set Up Domains Authentication

Domain gateways can be made to authenticate incoming connections requested by remote domains and outgoing connections requested by local domains. The authentication mechanism is optional and compatible with the BEA Tuxedo mechanism specified in the TUXCONFIG file.

Application administrators can define when security should be enforced for incoming connections from remote domains. You can specify the level of security used by a particular local domain by setting the SECURITY parameter in the DM_LOCAL_DOMAINS section of the DMCONFIG file. There are three levels of password security:

- No Security (using the NONE option)—incoming connections from remote domains are not authenticated.
- Application Password (using the APP_PW option)— incoming connections from remote domains are authenticated using the application password defined in the TUXCONFIG file. The BEA Tuxedo application password is administered with tmloadcf(1), which prompts for the password when the SECURITY option is enabled in the TUXCONFIG file. The password is automatically propagated with the TUXCONFIG file to the other machines in the configuration. You can update the password dynamically using the tmadmin command.
- Remote Domains Password (using the DM_PW option)—BEA Tuxedo Domains uses this feature to enforce security between two or more BEA Tuxedo domains. Connections between the local and remote domains are authenticated using passwords defined in the DM_PASSWORDS section of the BDMCONFIG file. These passwords are added to the binary configuration file after dmloadcf has been run, using the passwd subcommand of the dmadmin(1) command. Each entry contains the password used by a remote domain to access a particular local domain and the password required by the local domain, in turn, to access the remote domain.

If the SECURITY parameter is not set in TUXCONFIG (that is, if it defaults to NONE or if it is set explicitly to NONE), the Domains configuration can still require the Domain gateways to enforce security at the DM_PW level. If the DM_PW option is selected, then each remote domain must have a password defined in the DM_PASSWORDS section of the BDMCONFIG file. In other words, incoming connections from remote domains without a password are rejected by domain gateways.

T_DM_PASSWORDS MIB Class Definitions

The T_DM_PASSWORDS class represents configuration information for inter-domain authentication through local and remote access points of type TDOMAIN. The T_DM_PASSWORDS class contains the following entries for each remote domain.

- TA_DMLACCESSPOINT—the name of the local domain access point to which the password applies.
- TA_DMRACCESSPOINT—the name of the remote domain access point to which the password applies.
- TA_DMLPWD—the local password used to authenticate connections between the local domain access point (identified by TA_DMLACCESSPOINT) and the remote domain access point (identified by TA_DMRACCESSPOINT).
- TA_DMRPWD—the remote password used to authenticate connections between the remote domain access point (identified by TA_DMRACCESSPOINT) and the local domain access point (identified by TA_DMLACCESSPOINT).

  Note: Passwords are stored securely in encrypted format.

Setting Domains Passwords

BEA Tuxedo Domains passwords (DM_PW) are set using the dmadmin(1) command, as follows:

```
passwd [-r] local_domain_name remote_domain_name
```

This command prompts the administrator for new passwords for the specified local and remote domains.

See Also

- "Examples of Coding Security Between Domains" on page 2-44
- dmadmin(1) in the *BEA Tuxedo Command Reference*

- DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*
- DM_MIB(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Examples of Coding Security Between Domains

The SECURITY parameter in the DM_LOCAL_DOMAINS section of the DMCONFIG file specifies the security type of a local domain. If authentication is required, it is done every time a connection is established between the local domain and a remote domain. If the security types of the two domains are incompatible, or if the passwords do not match, the connection fails.

Example 1: Setting Security to NONE

If SECURITY is set to NONE for a local domain, incoming connection attempts are not authenticated. Even with SECURITY set to NONE, a local domain can still connect to remote domains that have SECURITY set to DM_PW, but before such a connection can be established, you must define the passwords on both sides by running dmadmin(1) or by using DM_MIB(5).

Listing 2-10 Setting Security to NONE for Both Application and Domains

```
DOM1: SECURITY in UBBCONFIG set to NONE
       SECURITY in DMCONFIG set to NONE

DOM2: SECURITY in UBBCONFIG set to NONE
       SECURITY in DMCONFIG set to DM_PW
```

In this example, DOM1 is not enforcing any security but DOM2 is enforcing DM_PW security. On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM1
   DOMAINID=DOM1
   SECURITY=NONE

 *DM_REMOTE_DOMAINS
DOM2 DOMAINID="DOM2"
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM2
   DOMAINID=DOM2
   SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
DOM1 DOMAINID="DOM1"
```

After the required attributes have been set in the TUXCONFIG and BDMCONFIG files, boot the applications on DOM1 and DOM2.

```
On DOM1:
     dmadmin
      passwd DOM1 DOM2
       Enter Local Domain Password:foo1
       Reenter Local Domain Password:foo1
       Enter Remote Domain Password:foo2
       Reenter Remote Domain Password:foo2

On DOM2:
     dmadmin
     passwd DOM2 DOM1
     Enter Local Domain Password:foo2
     Reenter Local Domain Password:foo2
     Enter Remote Domain Password:foo1
      Reenter Remote Domain Password:foo1
```

Once passwords have been created on both domains, a connection can be established and services can be invoked on the remote domain.

Listing 2-11 Setting Application Security to NONE and Domains Security to DM_PW

On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
 DOM1
 DOMAINID=DOM1
  SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
 DOM2 DOMAINID="DOM2"
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 SECURITY=NONE

DMCONFIG
 *DM_LOCAL_DOMAINS
 DOM2
  DOMAINID=DOM2
  SECURITY=DM_PW

 *DM_REMOTE_DOMAINS
 DOM1 DOMAINID="DOM1"
```

After the required attributes have been set in the TUXCONFIG and BDMCONFIG files, boot the applications on DOM1 and DOM2:

```
On DOM1:
     dmadmin
      passwd DOM1 DOM2
      Enter Local Domain Password:foo1
      Reenter Local Domain Password:foo1
      Enter Remote Domain Password:foo2
       Reenter Remote Domain Password:foo2
```

```
On DOM2:
     dmadmin
      passwd DOM2 DOM1
      Enter Local Domain Password:foo2
      Reenter Local Domain Password:foo2
      Enter Remote Domain Password:foo1
      Reenter Remote Domain Password:foo1
```

Once passwords have been created on both domains, a connection can be established and services can be invoked on the remote domain.

Example 2: Setting Security to APP_PW

If the SECURITY parameter in the UBBCONFIG is set to APP_PW or higher, then SECURITY in the DMCONFIG can be set to NONE, APP_PW, or DM_PW. Because you can define multiple views of a domain in one DMCONFIG file (one view per local domain definition), you can assign a different type of security mechanism to each of those views.

Note: If SECURITY is set to APP_PW for a local domain access point in the DMCONFIG, then SECURITY in the UBBCONFIG must be set to APP_PW or higher.

Listing 2-12 Setting Security to APP_PW for Both Application and Domains

```
DOM1: SECURITY in UBBCONFIG set to APP_PW
      SECURITY in DMCONFIG set to APP_PW

DOM2: SECURITY in UBBCONFIG set to APP_PW
      SECURITY in DMCONFIG set to APP_PW
```

In this example, both DOM1 and DOM2 enforce APP_PW security.

On the initiator side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows:

```
UBBCONFIG
 SECURITY=APP_PW

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM1
  DOMAINID=DOM1
  SECURITY=APP_PW

 *DM_REMOTE_DOMAINS
DOM2 DOMAINID="DOM2"
```

On the responder side, the pertinent attributes in UBBCONFIG and DMCONFIG are set as follows.

```
UBBCONFIG
 SECURITY=APP_PW

DMCONFIG
 *DM_LOCAL_DOMAINS
DOM2
  DOMAINID=DOM2
  SECURITY=APP_PW

 *DM_REMOTE_DOMAINS
DOM1 DOMAINID="DOM1"
```

After the TUXCONFIG and BDMCONFIG files have been created, boot the applications on DOM1 and DOM2.

Configuring the Connections Between Your Domains

You can specify *the* conditions under which a local domain gateway tries to establish a connection to a remote domain. To specify these conditions, assign a value to the CONNECTION_POLICY parameter in the Domains configuration file. You can select any of the following connection policies:

- Connect at boot time (ON_STARTUP)
- Connect when a client program requests a remote service (ON_DEMAND)
- Accept incoming connections but do not initiate a connection automatically (INCOMING_ONLY)

For connection policies of ON_STARTUP and INCOMING_ONLY, Dynamic Status is invoked. Dynamic Status is a BEA Tuxedo Domains capability that checks and reports the status of remote services.

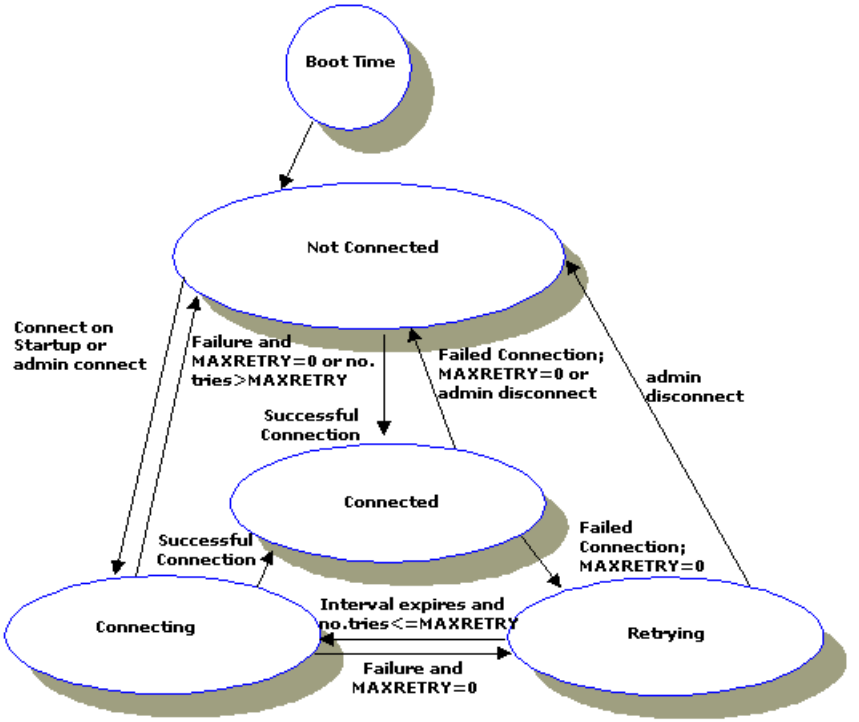
How to Request Connections at Boot Time (ON_STARTUP Policy)

A policy of ON_STARTUP means that a domain gateway attempts to establish a connection with its remote domains when the gateway server is initialized. By default, this connection policy retries failed connections every 60 seconds, but you can specify a different value for this interval (using the RETRY_INTERVAL parameter). This policy invokes Dynamic Status.

```
CONNECTION_POLICY=ON_STARTUP
```

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is ON_STARTUP.

Figure 2-5 Connections Made with an ON_STARTUP Policy



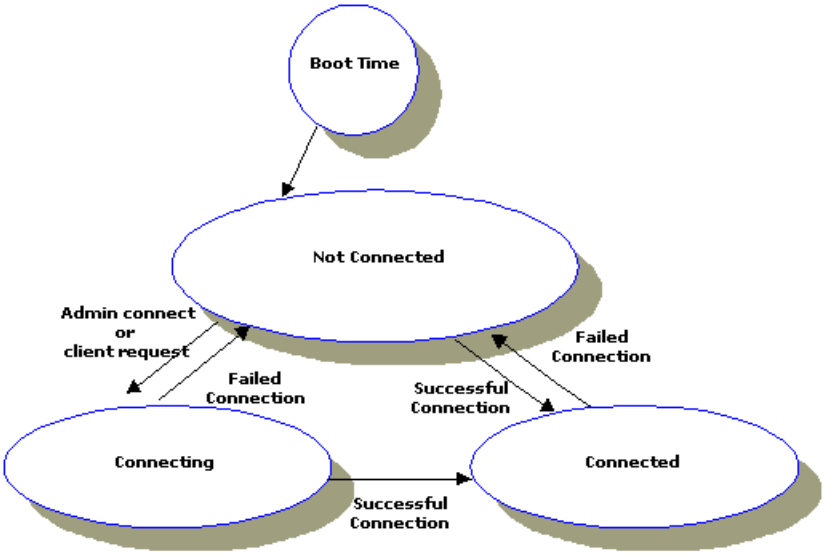
How to Request Connections for Client Demands (ON_DEMAND Policy)

A connection policy of ON_DEMAND means that a connection is attempted only when either a client requests a remote service or an administrative "connect" command is run. The default setting for CONNECTION_POLICY is ON_DEMAND. Connection retry processing is not allowed when the connection policy is ON_DEMAND. This policy does not invoke Dynamic Status.

```
CONNECTION_POLICY=ON_DEMAND
```

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is `ON_DEMAND`.

Figure 2-6 Connections Made with an ON_DEMAND Policy



How to Limit Connections to Incoming Messages Only (INCOMING_ONLY Policy)

A connection policy of `INCOMING_ONLY` means that a domain gateway does not try to establish a connection to remote domains upon starting. Connection retry processing is not allowed when the connection policy is `INCOMING_ONLY`. This policy invokes Dynamic Status.

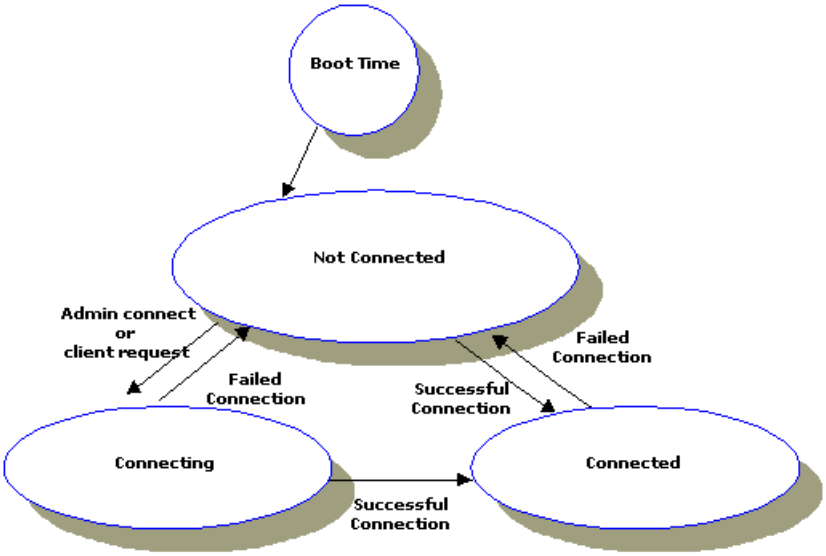
To use this policy, enter the following line in your Domains configuration file:

```
CONNECTION_POLICY=INCOMING_ONLY
```

Note: You can also establish a connection manually using the `dmadmin connect` command.

The following diagram shows how connections are attempted and made by a gateway for which the connection policy is INCOMING_ONLY.

Figure 2-7 Connections Made with an INCOMING_ONLY Policy (accept incoming connections)



How to Configure the Connection Retry Interval for ON_STARTUP Only

When the CONNECTION_POLICY parameter is set to ON_STARTUP, then the connection retry capability is available. The connection retry capability enables a domain gateway to retry, automatically, a failed attempt to connect to a remote domain. As an administrator, you can control the frequency of automatic connection attempts. To do so, specify the length (in seconds) of the interval during which the gateway should wait before trying, again, to establish a connection. You can specify the retry interval by setting the RETRY_INTERVAL parameter in the DM_LOCAL_DOMAINS section of the Domains configuration file as follows:

RETRY_INTERVAL=*number_of_seconds*

Note: You can specify between 0 and 2147483647 seconds.

If the connection policy is ON_STARTUP and you do not specify a value for the RETRY_INTERVAL parameter, a default of 60 is used.

The RETRY_INTERVAL parameter is valid only when the connection policy is ON_STARTUP. For the other connection policies (ON_DEMAND and INCOMING_ONLY), retry processing is disabled.

How to Configure the Maximum Retry Number

You indicate the number of times that a domain gateway tries to establish connections to remote domains before quitting by assigning a value to the MAXRETRY parameter: the minimum value is 0; the default and maximum value is the value of the MAXLONG parameter.

- If you set MAXRETRY=0, automatic connection retry processing is turned off. The server does not attempt to connect to the remote gateways automatically.
- If you set MAXRETRY=*number*, the gateway tries to establish a connection the specified number of times before quitting.

  Note: The RETRY_INTERVAL is rounded up to a multiple of SCANUNIT.
- If you set MAXRETRY=MAXLONG, retry processing is repeated indefinitely or until a connection is established.

The MAXRETRY parameter is valid only when the connection policy is ON_STARTUP. For the other connection policies (ON_DEMAND and INCOMING_ONLY), retry processing is disabled.

Table 2-1 Example Settings of the MAXRETRY and RETRY_INTERVAL Parameters

If You Set...	Then...
CONNECTION_POLICY=ON_STARTUP RETRY_INTERVAL=30 MAXRETRY=3	The gateway makes 3 attempts to establish a connection, at 30 seconds intervals, before quitting.
CONNECTION_POLICY=ON_STARTUP MAXRETRY=0	The gateway attempts to establish a connection at initialization time but does not retry if the first attempt fails.
CONNECTION_POLICY=ON_STARTUP RETRY_INTERVAL=30	The gateway attempts to establish a connection every 30 seconds until a connection is established.

See Also

- "Controlling the Connections Between Domains" on page 2-55
- "Configuring Domains-level Failover and Failback" on page 2-58
- DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Controlling the Connections Between Domains

As the administrator, you can control the number of connections you want to establish between domains. You can also break the connections between local and remote domains.

How to Establish Connections Between Domains

To establish a connection between a local gateway and a remote domain, run the `dmadmin` command with the `connect` (`co`) subcommand, as follows:

```
dmadmin co -d local_domain_name
```

By default, connections are established between the local domain you have specified and all remote domains configured for the local gateway. If you want to establish a connection to only one remote domain, specify that domain on the command line with the `-R` option, as follows:

```
dmadmin co -d local_domain_name -R remote_domain_name
```

If a connection attempt fails and you have configured the domain to try again, repeated attempts to connect (via automatic connection retry processing) are made.

How to Break Connections Between Domains

To break a connection between a local gateway and a remote domain (making sure that the gateway does not try to reestablish the connection through automatic connection retry processing), run the `dmadmin` command with the `disconnect` (`dco`) subcommand, as follows:

```
dmadmin dco -d local_domain_name
```

By default, all remote domains configured for the local gateway are disconnected. If you want to end the connection to only one remote domain, specify that domain on the command line with the `-R` option as follows:

```
dmadmin dco -d local_domain_name -R remote_domain_name
```

Automatic connection retry processing is stopped by this command, regardless of whether there are any active connections when the command is run.

How to Report on Connection Status

Using the `printdomain` command, you can generate a report on connection status and the connections being retried. The `connect` command reports whether a connection attempt has succeeded. The `printdomain` command prints information about the specified local domain, including a list of remote domains, a list of remote domains to which it is connected, an a list of remote domains to which it is trying to establish connections.

The following example shows a `dmadmin` session in which the `printdomain` command is issued (in its abbreviated form, `pd`) for a local domain called `LDOM`.

```
$ dmadmin
dmadmin - Copyright (c) 1996 BEA Systems, Inc.
Portions * Copyright 1986-1997 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
TUXEDO is a registered trademark.

pd -d LDOM
Local domain :LDOM
  Connected domains:
  Domainid:  RDOM1
  Disconnected domains being retried:
  Domainid: RDOM2

 dco -d LDOM -R RDOM1
Operation completed successfully. Use printdomain(pd) to obtain
results.

 dco -d LDOM -R RDOM2
Operation completed successfully. Use printdomain(pd) to obtain
results.
```

```
 co -d LDOM -R RDOM3
Operation completed successfully. Use printdomain(pd) to obtain
results.

pd -d LDOM
Local domain :LDOM
  Connected domains:
  Domainid: RDOM3
```

Configuring Failover and Failback in a Domains Environment

Two types of failover can be performed in a Domains environment: link-level failover and Domains-level failover. This section provides instructions for both:

- "How to Configure Domains to Support Link-level Failover" on page 2-57
- "Configuring Domains-level Failover and Failback" on page 2-58

If you want failover and failback functionality in your domain, you must configure your Domains configuration file to support it.

For details about the Domains configuration file, see the `DMCONFIG(5)` in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

How to Configure Domains to Support Link-level Failover

Link-level failover is a mechanism that ensures that an alternate network link becomes active when a primary link fails. To use link-level failover, the primary and alternate gateways must reside on different remote domains (that is, *gateway mirroring* must be used). Currently, link-level failover does not support multiple alternate links to the same gateway.

To implement link-level failover, specify it in the DM_TDOMAINS section of the Domains configuration file (DMCONFIG) as follows:

```
*DM_TDOMAINS
RDOM1 NWADDR=//addr1:0
RDOM1 NWADDR=//addr2:0
```

The first entry refers to the primary network link for remote domain RDOM1; the second entry refers to the alternate link.

Link-level failback is a manual procedure. When the primary link is restored, the administrator must bring down the alternate link manually. This operation may cause requests that are in progress to fail, and new traffic to be resumed over the primary link.

Note: For more detailed information on gateway mirroring, see DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Configuring Domains-level Failover and Failback

Domains-level failover is a mechanism that transfers requests to alternate remote domains when a failure is detected with a primary remote domain. It also provides failback to the primary remote domain when that domain is restored.

This level of failover/failback depends on Dynamic Status. The domain must be configured with a CONNECTION_POLICY of ON_STARTUP or INCOMING_ONLY to enable Domains-level failover/failback.

Domains-level failover/failback defines a remote domain as available when a network connection to the remote domain exists, and unavailable when a network connection to the remote domain does not exist.

Prerequisite to Using Domains-level Failover and Failback

To use Domains-level failback, you must specify ON_STARTUP or INCOMING_ONLY as the value of the CONNECTION_POLICY parameter.

A connection policy of ON_DEMAND is unsuitable for Domains-level failback as it operates on the assumption that the remote domain is always available. If you do not specify ON_STARTUP or INCOMING_ONLY as your connection policy, your servers cannot fail over to the alternate remote domains that you have specified with the RDOM parameter.

Note: A remote domain is *available* if a network connection to it exists; a remote domain is *unavailable* if a network connection to it does not exist.

How to Configure Domains to Support Failover

To support failover, you must specify a list of the remote domains responsible for executing a particular service in your Domains configuration file. Specifically, you must specify such a list as the value of the RDOM parameter in the DM_REMOTE_SERVICES section. You can also specify alternate domains, as follows:

```
RDOM=identifier_1, identifier_2, identifier_3
```

Example

Suppose the TOUPPER and TOUPPER2 services are available from three remote domains: R1 (the primary remote domain), R2, and R3. Include the following entry in your Domains configuration file:

```
*DM_REMOTE_SERVICES
DEFAULT: RDOM=R1, R2, R3
TOUPPER
TOUPPER2
```

How to Configure Domains to Support Failback

Failback occurs when a network connection to the primary remote domain is reestablished for any of the following reasons:

- Automatic retries (ON_STARTUP only)
- Incoming connections
- Manual dmadmin connect command

Note: For automatic retries, connection retry must be turned on (that is, MA54ETRY>0).

3 Planning and Configuring CORBA Domains

Domains in a BEA Tuxedo CORBA environment are an extension of the core ATMI BEA Tuxedo environment domains. A domain is a construct that is entirely administrative. There are no programming interfaces that refer to domains. Everything concerning domains is done by configuration files; only an administrator is aware of domains.

This topic includes the following sections:

- Configuring Multiple CORBA Domains
- Types of CORBA Domain Configurations
- Examples: Configuring Multiple CORBA Domains

Overview of Multiple CORBA Domains

Since an enterprise can have many different kinds of applications, be geographically dispersed, and be organized into different areas of responsibility, there might be many separate domains. Each domain is a separately administered unit. Perhaps it is organized for geographical considerations (all the machines in a given location). Perhaps it is organized on departmental grounds within an enterprise (accounting, manufacturing, shipping, and so on).

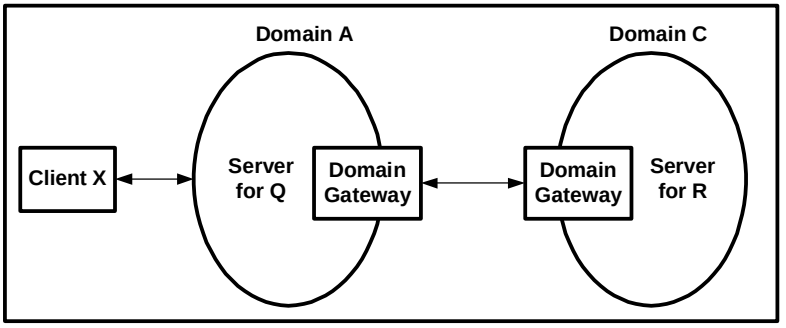
Eventually, an enterprise wants the different applications in those domains to be able to cooperate. It is often impossible to expand a single domain to encompass the enterprise. However, the size of an expanded domain in terms of the number of machines and services would be impractical. Since a single domain must be administered as a whole, the configuration would rapidly become huge and require more effort in administering than in developing and implementing applications.

Therefore, to keep a domain relatively compact for administration, there must be a way to separate applications into multiple domains and still allow applications in one domain to access services in other domains. This capability for interdomain communication is what is generically called "BEA Tuxedo domains."

Interdomain Communication

The following figure shows a simple multiple-domain configuration.

Figure 3-1 Multiple-domain Configuration



The following steps describe single-domain communication between Client X and Domain A:

1. Client X connects to Domain A using the Bootstrap object. The client application uses the Bootstrap object to locate a FactoryFinder and then uses the FactoryFinder to ask for a factory for objects of type Q. (The FactoryFinder call is itself an invocation on Domain A.)
2. When the FactoryFinder returns a factory, the client then invokes that factory in Domain A.

3. The factory returns a reference to an object of type Q, called Q1.

4. The client now invokes on object Q1 in Domain A.

Note: Throughout all of these steps, the client does not know where any of the objects are, or which domains they are in. It might not even know that there is something called a domain. The administrative actions for connecting a client to Domain A are relatively simple for a client, because the client is a simple machine and has very little infrastructure; it stands alone for the most part. Indeed, the connection to a BEA Tuxedo domain is the primary administration for a client. The actual administrative chore is setting the address of the ISL that is in Domain A.

For multiple-domain communication, Q1 needs the services of Object R1, which is in Domain C; therefore, object Q1 must execute operations similar to those described in steps 1 through 4 above, but across domain boundaries. The actual steps are as follows:

1. Object Q1 uses a Bootstrap object to locate a FactoryFinder and then uses the FactoryFinder to ask for a factory for objects of type R.

2. When the FactoryFinder returns a reference to a factory in Domain C, Object Q1 invokes that factory.

3. The factory returns a reference to an object of type R, called R1.

4. Object Q1 invokes on Object R1.

Note: As with Client X, there must be some administration to allow Object Q1 to get at the factories and objects in Domain C. As Figure 3-1 shows, the mechanism for communication between domains is a domain gateway. A domain gateway is a system server in a domain.

A system server is different than a user-written server because it is provided as part of the BEA Tuxedo product; other system servers are the name servers, FactoryFinders, and ISLs. A domain gateway is somewhat similar in concept to an ISL because it is the “contact” point for a domain. It is different from an ISL, however, because a domain gateway connects to another domain gateway, which is itself a contact point for a domain; that is, a domain gateway’s job is to connect to another domain gateway. Thus, the pair of domain gateways cooperate to make sure that invocation on objects that inhabit different domains are routed to the correct domain.

For domain gateways to operate in this manner, they must be configured properly. That configuration is the subject of the following sections.

Functions of Multiple-domain Configuration Elements

The following elements work together to accomplish the configuration of multiple domains:

- BEA Tuxedo configuration file

  The `UBBCONFIG` file names a domain and identifies the group and service entry for a domain gateway server. No attributes of domain gateways are specified in the `UBBCONFIG` file; all such attributes are in the `DMCONFIG` file.

- Domain configuration file

  The domain configuration file (`DMCONFIG`) describes the remote domains that are connected to the local domain. If there is no `DMCONFIG` file, there are no connections.

- FactoryFinder domain configuration file

  One FactoryFinder domain configuration file (`factory_finder.ini`) is required for each domain that is connected to one or more other domains. If a domain is not connected to another domain, there is no need for this file.

  This file specifies which factories can be searched for or found across domain boundaries. You must carefully coordinate the `factory_finder.ini` file with the `DMCONFIG` so that they both have information about the same connected domains and provide the same connectivity.

- Invocation of an object in a remote domain

  The whole point of the BEA Tuxedo domains feature is for an application in one CORBA domain to be able to make an invocation on an object in another CORBA domain, without either the client or server applications being aware that domains are a factor. Configuration information is intended to allow such invocations to cross domain boundaries and to hide the fact of those boundaries from applications.

  Being able to make an invocation on a reference for an object in a remote domain depends on a satisfactory set of three configuration files—the `UBBCONFIG`, `DMCONFIG`, and `factory_finder.ini` files—for each domain and

on the coordination of two of those configuration files—the DMCONFIG and factory_finder.ini files—between domains. As the number of domains grows, the coordination effort grows.

- References to objects in a remote domain

  Any object reference may specify a local domain or a remote domain. A reference to a remote domain typically happens when a FactoryFinder returns a reference to a factory in a remote domain. It also happens when that factory, in turn, creates and returns a reference to an object in that remote domain (although, of course, the reference is local to the domain of the factory).

Note: Applications are not aware of the domain of an object reference. Applications cannot find out what domain an object reference refers to. Thus, invocations on an object reference for a remote domain are transparent to the application. This transparency allows administrators the freedom to configure services in individual domains and to spread resources across multiple domains. If applications were to include information about domains, changing configurations would require that the applications be rewritten as well.

- FactoryFinders

  For a server in a local domain to obtain an object reference to an object in another domain, the application uses the same FactoryFinder pattern as it does for objects in the local domain. The application uses the same pattern because it is not aware that the factory finder returns a reference to a factory in another domain. The configuration files hide this fact.

  Once an object reference has been obtained via a FactoryFinder or factory, the object reference can be passed anywhere; that is, passed to objects in the local domain, returned to a client, or passed to another domain.

Configuring Multiple CORBA Domains

You use the following three configuration files to configure multiple domains:

- The main configuration UBBCONFIG file
- The domain configuration (DMCONFIG) file, and

- The FactoryFinder domain configuration file (`factory_finder.ini`).

The Configuration File

You must specify the following parameters in the `UBBCONFIG` file to configure multiple domains:

- Domain name
- Gateway group
- Gateway service

Domain Name

Though not required for single domains (that is, standalone domains), a domain that is connected to another domain must have a `DOMAIN ID`. You specify this parameter in the `RESOURCES` section of the `UBBCONFIG` file, as follows:

```
DOMAIN ID = <domain-name>
```

The `<domain-name>` must be 1 to13 characters long. For example:

```
DOMAIN ID = headquarters
```

`<domain-name>` is the name that will be referenced in the `DM_REMOTE_SERVICES` and `DM_LOCAL_SERVICES` sections of the related `DMCONFIG` file. In that file, the `<domain-name>` will be referenced as:

```
"//<domain-name>"
```

The quotes are part of the reference. The slashes (`//`) mean that the name applies to BEA Tuxedo CORBA domains, rather than to BEA Tuxedo ATMI domains. For example:

```
"//headquarters"
```

Note: Every domain in an enterprise must have a unique `<domain-name>`.

Gateway Group and Service

As with every other system service, there must be a group and a service name specified for a gateway. For example, the GROUPS section might contain:

```
LGWGRP      GRPNO=4      LMID=LDOM
```

In this example, LGWGRP is a name chosen by a user (perhaps an abbreviation for "Local Gateway Group").

The service name for a domain gateway is GWTDOMAIN and must be associated, like every other group, with a server group and a server ID. You specify the service name in the SERVERS section associated with the server group name chosen. For example:

```
GWTDOMAIN SRVGRP=LGWGRP SRVID=1
```

This tells the BEA Tuxedo CORBA server that a domain gateway is to be used and that additional information is found in the DMCONFIG file.

The Domain Configuration (DMCONFIG) File

There is one DMCONFIG file per domain. It describes the relationship between the local domain (the domain in which the DMCONFIG file resides) and remote domains (any other domains). The DMCONFIG file contains domain information for the core BEA Tuxedo domains and for BEA Tuxedo CORBA domains.

The sections below concentrate on the information that applies to BEA Tuxedo CORBA domains. In other documentation for the DMCONFIG file, the communication between local and remote domains is based on BEA Tuxedo ATMI services, a concept not used in BEA Tuxedo CORBA environments. For BEA Tuxedo CORBA environments, the "service" name is the name of another BEA Tuxedo domain that can service BEA Tuxedo CORBA requests.

The DMCONFIG file consists of up to eight parts, but one part, DM_ROUTING, does not apply to BEA Tuxedo CORBA environments. The other seven parts refer to BEA Tuxedo CORBA environments, but many of the BEA Tuxedo ATMI parameters are not used. Those seven parts are: DM_RESOURCES, DM_LOCAL_DOMAINS, DM_REMOTE_DOMAINS, DM_LOCAL_SERVICES, DM_REMOTE_SERVICES, DM_ACCESS_CONTROL, and DM_TDOMAIN.

The following sections refer to the sample DMCONFIG file shown in Listing 3-1.

Listing 3-1 Sample DMCONFIG File

```
#
# BEA Tuxedo CORBA DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES
VERSION=Experimental8.9

*DM_LOCAL_DOMAINS
LDOM GWGRP=LGWGRP TYPE=TDOMAIN DOMAINID="MUTT"

*DM_REMOTE_DOMAINS
TDOM1 TYPE=TDOMAIN DOMAINID="JEFF"

*DM_TDOMAIN
LDOM   NWADDR="//MUTT:2507"
TDOM1  NWADDR="//JEFF:3186"

*DM_LOCAL_SERVICES
"//MUTT"

*DM_REMOTE_SERVICES
"//JEFF"    RDOM=TDOM1
```

DM_RESOURCES

The DM_RESOURCES section can contain a single field, VERSION. It is not checked by software; it is provided simply as a place where users can enter a string that may have some documentation value to the application.

```
*DM_RESOURCES
VERSION=Experimental8.9
```

DM_LOCAL_DOMAINS

The DM_LOCAL_DOMAINS section specifies some attributes for gateways into the local domain from the outside. The section must have an entry for each gateway group defined in the UBBCONFIG fle that will provide access to the local domain from other domains. Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries have the form:

```
LDOM required-parameters [optional-parameters]
```

where LDOM is an identifier used to refer to the gateway to the local domain. LDOM must be unique among all LDOM and RDOM entries across the enterprise (that is, among the set of domains connected to each other). Note that LDOM is not the same name as the <domain-name> or the gateway group that is specified in the UBBCONFIG file. Rather, LDOM is a name used only within the DMCONFIG file to provide an extra level of insulation from potential changes in the UBBCONFIG file (changes in UBBCONFIG will affect only this one part of DMCONFIG).

The following are required parameters:

```
GWGRP = identifier
```

This parameter specifies the name of a gateway server group (the name provided in the UBBCONFIG file) representing this local domain.

```
TYPE = TDOMAIN
```

The TYPE parameter is required to specify the use of domains for BEA Tuxedo CORBA environments.

```
DOMAINID = string
```

The DOMAINID parameter is used to identify the local domain for the purposes of security. The gateway server group in GWGRP uses this string during any security checks. It has no required relationship to the <domain-name> found in the RESOURCES section of the UBBCONFIG file. DOMAINID must be unique across both local and remote domains. The value of string can be a sequence of characters (for example, "BA.CENTRAL01"), or a sequence of hexadecimal digits preceded by 0x (for example, "0x0002FF98C0000B9D6"). DOMAINID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing null).

For example, the lines

```
*DM_LOCAL_DOMAINS
LDOM  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="MUTT"
```

identify LDOM as an access point to the local domain. It is associated with the service group LGWGRP (as specified in the UBBCONFIG file). If the gateway is ever involved in a domain-to-domain security check, it goes by the name MUTT.

Optional parameters describe resources and limits used in the operation of domain gateways. For a description of these parameters, refer the dmconfig(5) reference page in the *File Formats, Data Descriptions, MIBs, and System Processes Reference.*

DM_REMOTE_DOMAINS

The DM_REMOTE_DOMAINS section specifies some attributes for gateways to remote domains. The section has an entry for each UBBCONFIG file-defined gateway group that will send requests to remote domains. Each entry specifies the parameters required for the domain gateway processes running in that group.

Entries have the form:

```
RDOM required-parameters
```

where RDOM is an identifier used to refer to the gateway providing access to the remote domain. RDOM must be unique among all LDOM and RDOM entries across the enterprise (that is, among the set of domains connected to each other). Note that RDOM is not the same name as the <domain-name> or the gateway group that is specified in the UBBCONFIG file. Rather, RDOM is a name used only within the DMCONFIG to provide an extra level of insulation from potential changes in UBBCONFIG (changes in UBBCONFIG will affect only this one part of DMCONFIG).

The required parameters are:

TYPE = TDOMAIN
: The TYPE parameter is required to specify the use of domains for BEA Tuxedo CORBA environments.

DOMAINID = string
: The DOMAINID parameter is used to identify the remote domain for the purposes of security. The gateway uses this string during any security checks. DOMAINID has no required relationship to the <domain-name> found in the RESOURCES section of the UBBCONFIG file. DOMAINID must be unique across both local and remote domains. The value of string can be a sequence of

characters (for example, "BA.CENTRAL01"), or a sequence of hexadecimal digits preceded by "0x" (for example, "0x0002FF98C0000B9D6").
DOMAINID must be 32 octets or fewer in length. If the value is a string, it must be 32 characters or fewer (counting the trailing null).
Entries associated with a remote domain can be specified more than once. The first one specified is considered to be the primary address, which means it is the first one tried when a connection is being attempted to a remote domain. If a network connection cannot be established using the NWADDR of the primary entry, the NWADDR associated with the secondary entry is used. (NWADDR is the physical address; see the DM_TDOMAIN section.)
For example, the lines

```
*DM_REMOTE_DOMAINS
TDOM1 TYPE=TDOMAIN DOMAINID="JEFF"
```

identify TDOM1 as the access point name of a gateway. If the gateway is ever involved in a domain-to-domain security check with a partner gateway, the gateway expects that partner to go by the name JEFF.

DM_TDOMAIN

The DM_TDOMAIN section defines the network addressing information for gateways implementing BEA Tuxedo CORBA domains. There should be one entry for each domain gateway that accepts requests from remote domains, and one entry for each domain gateway that sends requests to remote domains.

The format of each entry is:

```
DOM required-parameters [optional-parameters]
```

where DOM is an identifier value used to identify either a local domain access point (LDOM in the DM_LOCAL_DOMAINS section) or a remote domain access point (RDOM in the DM_REMOTE_DOMAINS section).

The following parameter is required:

NWADDR = string
: This parameter specifies the network address associated with a local domain or a remote domain. If the association is with a local domain, the NWADDR is used to accept connections from other domains. If the association is with a remote domain, the NWADDR is used to initiate a connection. This parameter specifies the network address to be used by the process as its listening address. The listening address for a domain gateway is the means by which it

is contacted by other gateway processes participating in the application. If `string` has the form `"0xhex-digits"` or `"\\xhex-digits"`, it must contain an even number of valid hex digits. These forms are translated internally into a character array containing TCP/IP addresses. The addresses may also be in either of the following two forms:

```
"//hostname:port_number"
"//#.#.#.#:port_number"
```

In the first of these formats, `hostname` is resolved to a TCP/IP host address at the time the address is bound, using the locally configured name resolution facilities accessed via `gethostbyname`(3c). The `"#.#.#.#"` is the dotted decimal format, where each `#` represents a decimal number in the range 0 to 255.

`Port_number` is a decimal number in the range 0 to 65535 (the hexadecimal representations of the string specified). For example:

```
*DM_TDOMAIN
  LDOM  NWADDR="//MUTT:2507"
  TDOM1 NWADDR="//JEFF:3186"
```

Continuing the example from above, the first entry specifies a gateway with the domain access name of `LDOM` (meaning that it corresponds to the local gateway group `LGWGRP`, specified in `UBBCONFIG`). Since `LDOM` was defined in `DM_LOCAL_DOMAINS`, that means the gateway is configured to accept requests from other domains. It listens on the address `"//MUTT:2507"`. Similarly, the second entry is for the domain access name `TDOM1`, which appears in `DM_REMOTE_DOMAINS`, transferring requests to a remote domain. In this case, the gateway associated with `TDOM1` sends requests to the address `"//JEFF:3186"`.

For a description of the optional parameters, refer to the `dmconfig(5)` reference page in the *File Formats, Data Descriptions, MIBs, and System Processes Reference.*

DM_REMOTE_SERVICES

The `DM_REMOTE_SERVICES` section specifies additional attributes for gateways to remote domains. The format of each entry is:

```
service  RDOM=<rdom-name>
         [LDOM=<ldom-name>]
         [TRAN_TIME=...]
```

where service is of the form:

"//<domain-name>"

This <domain-name> is the name that occurs RESOURCES section of the UBBCONFIG file as <domain-name>. Each entry specifies an rdom-name and, optionally, an ldom-name. The gateway uses the attributes for those entries for establishing a gateway pair for BEA Tuxedo CORBA domain communication. Gateways operate in pairs. At boot time, the local domain uses attributes of rdom-name (the address specified in the DM_TDOMAIN section) to establish a connection to a gateway in the other domain. If security is used, the other attributes of rdom-name and ldom-name are used for mutual authentication. At run time, when BEA Tuxedo determines that a request must travel to domain <domain-name>. It uses the gateway specified by rdom-name to send the request to another domain.

Most often, <domain-name> is the name of the domain specified in the address of the rdom-name. In that situation, when the request ends up at the other end of the gateway, it is served in that domain. For example:

```
*DM_REMOTE_SERVICES
  "//JEFF"    RDOM=TDOM1
```

In this case, the domain name JEFF is located at the address "//JEFF:3186". That address might or might not have a UBBCONFIG file that specifies its domain name as JEFF. If it does, the request can be serviced immediately.

It is possible to have entries that send requests for the specified domain-name to an intermediary domain that acts as a pass-through for routing purposes.

The remaining optional parameter, TRANTIME = integer, specifies the default timeout value, in seconds, for a transaction automatically started for the associated service. The value must be greater than or equal to 0 (zero) and less than 2147483648. The default is 30 seconds. A value of 0 (zero) implies the maximum timeout value for the machine.

DM_LOCAL_SERVICES

The DM_LOCAL_SERVICES section specifies additional attributes for gateways that accept requests into the local domain from the outside.

Lines within this section have the form:

```
service  [LDOM=<ldom-name>]
         [ACL=...]
```

where service is of the form:

```
"//<domain-name>"
```

This <domain-name> is the name that occurs in the RESOURCES section of the UBBCONFIG file as <domain-name>. Most likely this is the name of the domain in which the gateway resides, meaning that this (local) domain accepts BEA Tuxedo CORBA environment requests from other domains. It is also possible (but not necessary, except for purposes of security) to have an entry that accepts requests for a different domain name in the case where the local domain acts as a pass-through for routing purposes.

Notice that exported services inherit the properties specified for the service in an entry in the SERVICES section of the TUXCONFIG file, or their defaults. Some of the properties that may be inherited are LOAD, PRIO, AUTOTRAN, ROUTING, BUFTYPE, and TRANTIME.

The optional parameter, ACL = identifier, specifies the name of the access control list (ACL) to be used by the local domain to restrict requests made to this service by remote domains. The name of the ACL is defined in the DM_ACCESS_CONTROL section. If this parameter is not specified, access control is not performed for requests to this service.

For example, the lines:

```
*DM_LOCAL_SERVICES
"//MUTT"
```

state that this domain accepts requests destined for the domain with name MUTT.

DM_ACCESS_CONTROL

The DM_ACCESS_CONTROL section specifies the access control lists used by a local domain. Lines in this section are of the form:

```
ACL_NAME required parameters
```

where ACL_NAME is an (identifier) name used to identify a particular access control list; it must be 15 characters or less in length.

The only required parameter is:

```
ACLIST = identifier [,identifier]
```

where an ACLIST is composed of one or more remote domain names (RDOM) separated by commas. The wildcard character (*) can be used to specify that all the remote domains defined in the DM_REMOTE_DOMAINS section can access a local domain.

Note: The factory_finder.ini and DMCONFIG files must be coordinated; that is, if the factory_finder.ini file declares another domain to have accessible factories, there must be a way in DMCONFIG to get to that domain.

The factory_finder.ini File

Administrators are required to identify any factory objects that can be used in the current (local) /Domain, but that are resident in a different (remote) /Domain. You identify these factories in a FactoryFinder domain configuration file, also referred to as the factory_finder.ini file. This is an ASCII file that can be created and updated using a text editor.

The factory_finder.ini file can be used to identify remote CORBA factories that can be used in the local domain.

The format of the factory_finder.ini file is modeled after the syntax used to describe /Domains, and is shown below:

```
*DM_REMOTE_FACTORIES
    "local_factory_id.factory_kind"
    DOMAINID="domain_id"
    RNAME="remote_factory_id.factory_kind"
        ...

*DM_LOCAL_FACTORIES
    "factory_id.factory_kind"
    ...
```

Sample syntax for CORBA factory objects is as follows:

```
*DM_REMOTE_FACTORIES
    "AccountFactory.FactoryKind"
    "DOMAINID="MyAccountFactoryDomain"
    RNAME="MyAccountFactory.FactoryKind
```

where: AccountFactory is the name used to register the factory in the local domain's FactoryFinder, MyAccountFactoryDomain is the name of the remote domain, MyAccountFactory is the name used to register the factory in the remote domain's FactoryFinder.

The Master NameManager reads the factory_finder.ini file when the process is started. The reason for starting the Master NameManager affects which portions of the factory_finder.ini file are processed. If the Master NameManager is being started as part of booting an application, the initialization mode, the entire contents of the file is processed. As a result, the information in the DM_REMOTE_FACTORIES section results in entries being added for the factory objects being imported.

On the other hand, if the Master NameManager is being restarted as a result of a process failure, only the DM_LOCAL_FACTORIES section of the file is read. This section of the factory_finder.ini file must be re-read to reload the information that is used to restrict the exportation of certain factory objects into another domain.

Note: Since the Master NameManager reads the factory_finder.ini file only when the process is started, there is no way to update the Master NameManager (for example, when a new domain with factory objects to be imported needs to be added) without shutting down the Master NameManager.

A factory_finder.ini file applies to the domain in which it resides. It contains two sections: the DM_REMOTE_FACTORIES section and the DM_LOCAL_FACTORIES section. Either section can be absent or contain nothing.

The following sections provide more information on how to use the DM_REMOTE_FACTORIES section and the DM_LOCAL_FACTORIES section.

DM_REMOTE_FACTORIES

The DM_REMOTE_FACTORIES section provides information about the factory objects that are available in remote domains and that are imported so that applications in the local domain can use them. Identifiers for remote factory objects are listed in this section. The identifier, under which the object is registered, including a kind value of "FactoryInterface", must be listed in this section. For example, the entry for a remote factory object to be registered by the TP Framework with the identifier Teller in domain "Norwest" would be specified as:

```
*DM_REMOTE_FACTORIES
  "Teller.FactoryInterface"
    DOMAINID="Norwest"
    RNAME="BankTeller.FactoryInterface"
```

If the RNAME is not specified, the factory_kind must be specified in the factory name and the factory name must be enclosed in quotation marks; otherwise, the NameManager is not able to locate the appropriate factory. An entry that does not contain a factory_kind value is not defaulted with a value of "FactoryInterface". The following example shows a factory object to be registered with the identifier Teller in domain "Norwest". Note the absence of the RNAME specification, the specification of the factory_kind value, and the quotation marks around the factory name.

```
*DM_REMOTE_FACTORIES
 "Teller.FactoryInterface"
    DOMAINID="Norwest"
```

Because the identities of factories in a multidomain configuration may collide, the factory identifier and the RNAME parameters allow you to specify alternative identities, or "aliases," in the local domain for remote factories. Listing 3-2 shows two examples of a remote factory that is registered by the TP Framework with the identifier BankTeller in domain "Norwest". In both examples, the factory is made available in local domain with an alias of Teller.

Listing 3-2 Assigning an Alias to a Remote Factory

```
#EXAMPLE 1:

*DM_REMOTE_FACTORIES
   Teller
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"

#EXAMPLE 2:

*DM_REMOTE_FACTORIES
   "Teller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
```

You can also assign multiple aliases to the same remote factory. In the example shown in Listing 3-3, the remote factory will be registered in the local domain with two aliases: `Teller` and `BankTeller`.

Listing 3-3 Assigning Multiple Aliases to a Remote Factory

```
*DM_REMOTE_FACTORIES
   "Teller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
   "BankTeller.FactoryInterface"
     DOMAINID="Norwest"
     RNAME="BankTeller.FactoryInterface"
```

Usage Note: In multidomain configurations, factory object identifiers must be unique across domains in the enterprise.

In a multidomain configuration, two different domains must *not* have factory objects with the same `factory_id.factory_kind` identifier, for example: `"Teller.FactoryInterface"`.

If the same identifier, or name, is used in two domains, the software's behavior varies depending on whether BEA WebLogic Enterprise was used to configure the CORBA domain environment:

- In releases prior to BEA WebLogic Enterprise 5.1, the software allows the first server in a domain to register the factory without issuing an error message. If two factories with the same name are registered in a domain, the Master NameManager fails.
- In BEA WebLogic Enterprise release 5.1 or later and BEA Tuxedo release 8.0 or later, the software generates an error and writes it to the `ULOG`.

Note: In a single domain configuration, a BEA Tuxedo CORBA environment supports multiple factories objects with the same name. This type of configuration is allowed so as to achieve load-balancing.

There are two ways to ensure that your identifiers, or names, are unique across domains and thus avoid this problem:

1. Use unique identifiers throughout the enterprise. This may mean keeping a master list of all identifiers.
2. In the `factory_finder.ini` file, use the `RNAME` parameter so that an alias is used by the local NameManager. (This also means that local clients will have to be modified to use the alias to access the remote factory object.) Listing 3-2 shows an example of a `factory_finder.ini` file that uses the `RNAME` parameter to create an alias.

DM_LOCAL_FACTORIES

The `DM_LOCAL_FACTORIES` section specifies factory objects in the local domain that are available to be exported to other domains. This section can be used in the following ways:

- If the `DM_LOCAL_FACTORIES` section does not exist in a `factory_finder.ini`, or exists but is empty, all factory objects in the local domain are available to remote domains. This allows administrators an easy means to make local factory objects available to remote domains without having to provide an entry for every factory object in the local domain.
- If the `DM_LOCAL_FACTORIES` section exists in a `factory_finder.ini` file but contains the reserved keyword `"NONE"`, none of the factory objects in the local domain are available to remote domains. This allows administrators to restrict access without having to provide an entry for every factory object in the local domain.

The identifier, or name, under which the factory object is registered, including a kind value of `"FactoryInterface"`, must be listed in this section. For example, the entry for a factory object to be registered by the TP Framework with the identifier `Teller` would be specified as:

```
*DM_LOCAL_FACTORIES
  "Teller.FactoryInterface"
```

The `factory_kind` must be specified for the NameManager to locate the appropriate factory object. An entry that does not contain a `factory_kind` value is not defaulted with a value of `"FactoryInterface"`. This allows for the use of the CORBA NamingService.

The factory_finder.ini file specifies that the process of finding a factory can be exported to a remote domain by including a section beginning with "*DM_REMOTE_FACTORIES". In other words, including this section means that the local domain can find factories in a remote domain.

An entry into the file for domain A might be:

*DM_REMOTE_FACTORIES
fA.FactoryInterface DOMAINID=B

This means that a request in Domain A to find a factory with the identifier fA can be satisfied by the FactoryFinder in domain B. Of course, the UBBCONFIG and DMCONFIG files for the two domains must also be set up so that there are connected domain gateways between the two domains.)

An alternate form of the entry is:

CDE.FactoryInterface DOMAINID=B RNAME=fA.FactoryInterface

This means that a request in Domain A to find a factory with the identifier "CDE" will be satisfied by the FactoryFinder in domain B using the ID fA. This is sometimes called an alias.

Note: The factory ID must have `".FactoryInterface"` at the end. For simplicity, when talking about test configurations, we will leave that off, but it should appear in the file.

For more information about the `factory_finder.ini` file, see the description of the factory_finder.ini file in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Local Factories

A domain can specify which of its factories can be accessed by other domains. This is specified in a section beginning with `*DM_LOCAL_FACTORIES`. If the `factory_finder.ini` file does not exist, or if it exists and this section does not appear, or is empty, *all* local factories can be accessed by remote domains. If the section exists and contains the keyword `None`, none of the local factories are exportable; that is, none are allowed to be found by a remote FactoryFinder. If the section exists, it can contain a list of factories available to remote domains. For example,

```
*DM_LOCAL_FACTORIES
fA.FactoryInterface
fB.FactoryInterface
```

This specifies that factories `fA` and `fB` are findable from other domains. All factories other than factories explicitly listed are not findable. Unlike remote factories, there is no provision for an alias with local factories.

Note: The `factory_finder.ini` and `DMCONFIG` files must be coordinated, that is, if the `factory_finder.ini` file declares another domain to have accessible factories, there must be a way in `DMCONFIG` to get to that domain.

Types of CORBA Domain Configurations

When using the multiple domains feature, you can configure two types of configurations: directly connected domains and indirectly connected domains. You, as the administrator, configure both types using the domain configuration file, `DMCONFIG`.

Directly Connected Domains

It is possible for every domain in an enterprise to have a gateway to every other domain it might use. Such a configuration has the advantage that a request goes directly to the target domain, with the minimum of delay. Such an “n-way” configuration is quite reasonable when the number of domains is small, but each new domain requires two new gateways. At some point, an administrator may consider a different configuration, giving up speed of delivery for ease of management of domain connections. This is when the ability to configure indirectly connected domains becomes advantageous.

Indirectly Connected Domains

An administrator should consider what the likely traffic patterns are. Domains that have only occasional interactions are candidates for gateway removal. Since there will still be interactions, it must still be possible to reach the other domain. The technique used is to route the request through an intermediate domain that does have direct access

to the target domain. For example, we might have three domains, A, B, and C. Domains A and B are directly connected and Domains B and C are directly connected, but A and C are not directly connected (see Figure 3-2). For Domains A and C to communicate, they must use domain B as the intermediary. Therefore, the DMCONFIG file for Domain A must state that it is possible to connect to domain C by going through Domain B (and vice versa). That is, the connectivity is:

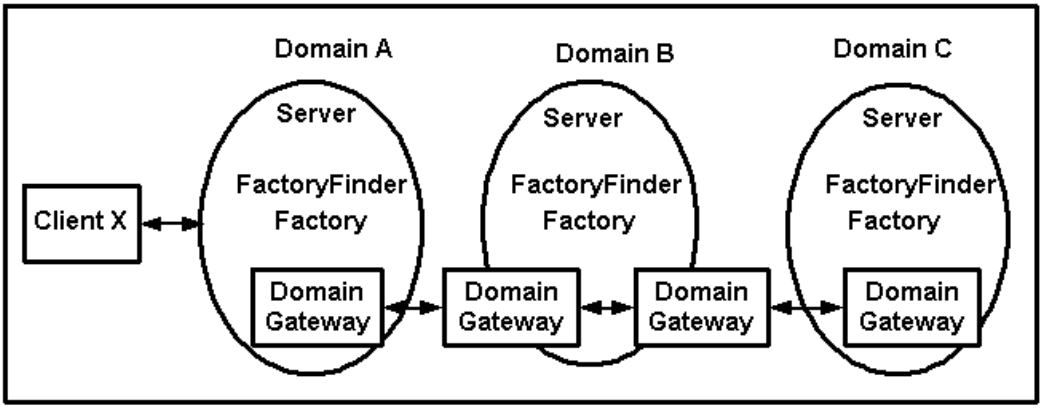
```
Domains        A      <->      B      <->     C
Gateways           GAB   GBA         GBC   GCB
```

Domain A has a gateway process, GAB (the Gateway from A to B), that connects to Domain B. The Domain A DMCONFIG file states that GAB acts as a gateway to two domains, Domains B and C. The DMCONFIG file for Domain C has a similar configuration, stating that GCB is connected to B and A. The DMCONFIG file for Domain B has two gateway processes, one which connects to A (GBA) and one which connects to C (GBC). This is called an indirect connection.

Given this indirect connection, when a server in A invokes a request on an object in C, BEA Tuxedo CORBA server knows that it can send the request to gateway GAB. The BEA Tuxedo gateway does not know that its partner gateway in B cannot service the request itself, but that is acceptable. Once the request is in domain B, it is routed through GBC to C, which can service the request. Thus, the request is serviced with one extra hop.

It is even possible for the two gateways in Domain B to be a single gateway, so that there is not an extra hop within B. In effect, the same processing occurs in Domain B, but it all occurs within a single gateway process.

Figure 3-2 Indirectly Connected Domains



Examples: Configuring Multiple CORBA Domains

The following sections provide examples of how to configure directly connected domains.

Note: These examples are provided for informational purposes only. If you want to use these examples, you will have to change the `APPDIR`, `TUXCONFIG`, and `TUXDIR` variables to match your environment. Also, you will have to substitute appropriate information wherever text is enclosed by left (<) and right (>) angle brackets (for example, <App Server Name>) and delete the angle brackets.

Sample UBBCONFIG Files

Listing 3-4, Listing 3-5, and Listing 3-6 show the `UBBCONFIG` files for three directly connected domains: Here, There, and Yonder.

Note: To use these files, you must replace *`host`* with the name of the local machine.

Listing 3-4 UBBCONFIG File for the Here Domain

```
#
#     Copyright (c) 1999 BEA Systems, Inc.
#     All rights reserved
#
#
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123312
    DOMAINID  HereD
    MASTER    LAPP
```

```
    MODEL     SHM
    LDBAL     N


#
# MACHINES
#
*MACHINES
    <host>
           LMID=LAPP
           APPDIR="/tst1/wle4.2/test_dom/t07:
                    /tst1/wle4.2/dec_unix/wlemdomai"
           TUXCONFIG="/tst1/wle4.2/test_dom/tuxconfig"
           TUXDIR="/lclobb/lc"
           MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT:   LMID=LAPP
    ICEGRP     GRPNO=11 OPENINFO=NONE
    GROUP1     GRPNO=21 OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEVT   SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
                   CLOPT="-A -- -N -M -f <FF ini file for Here>"
    TMFFNAME   SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME   SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>    SRVGRP=GROUP1 SRVID=2
    ISL        SRVGRP=GROUP1 SRVID=1
                    CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"

#
# SERVICES
#
*SERVICES
```

Listing 3-5 UBBCONFIG File for the There Domain

```
#
#      Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
# RESOURCES
#
*RESOURCES
    IPCKEY     133445
    DOMAINID   ThereD
    MASTER     LAPP1
    MODEL      SHM
    LDBAL  N
#
# MACHINES
#
*MACHINES
    <host>
         LMID=LAPP1
         APPDIR="D:\test_dom\t07;D:\Iceberg\qa\orb\bld\wlemdomain"
         TUXCONFIG="D:\test_dom\tuxconfig"
         TUXDIR="D:\Iceberg"
         MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT   LMID=LAPP1
    ICEGRP    GRPNO=11  OPENINFO=NONE
    GROUP1    GRPNO=21  OPENINFO=NONE
    LDMGRP    GRPNO=3
    LGWGRP    GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:   CLOPT="-A"
    DMADM      SRVGRP=LDMGRP SRVID=1
    GWADM      SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN  SRVGRP=LGWGRP SRVID=2
    TMSYSEV    SRVGRP=ICEGRP SRVID=1
    TMFFNAME   SRVGRP=ICEGRP SRVID=2
                 CLOPT="-A -- -N -M -f <FF ini file for There>"
    TMFFNAME   SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME   SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>     SRVGRP=GROUP1 SRVID=2
    ISL        SRVGRP=GROUP1 SRVID=1
```

```
                    CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

Listing 3-6 UBBCONFIG File for the Yonder Domain

```
#      Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
# RESOURCES
#
*RESOURCES
    IPCKEY    123334
    DOMAINID  YonderD
    MASTER    LAPP
    MODEL     SHM
    LDBAL     N
#
# MACHINES
#
*MACHINES
    <host>
          LMID=LAPP
          APPDIR="/tst1/wle4.2/test_dom/t07p:
                              /tst1/wle4.2/<host3>/wlemdomain"
          TUXCONFIG="/tst1/wle4.2/test_dom/<host3>/tuxconfig"
          TUXDIR="/lclobb/lc"
          MAXWSCLIENTS=10
#
# GROUPS
#
*GROUPS
    DEFAULT:   LMID=LAPP
    ICEGRP     GRPNO=11 OPENINFO=NONE
    GROUP1     GRPNO=21 OPENINFO=NONE
    LDMGRP     GRPNO=3
    LGWGRP     GRPNO=4
#
# SERVERS
#
*SERVERS
    DEFAULT:  CLOPT="-A"
    DMADM     SRVGRP=LDMGRP SRVID=1
```

```
    GWADM     SRVGRP=LGWGRP SRVID=1
    GWTDOMAIN SRVGRP=LGWGRP SRVID=2
    TMSYSEVT  SRVGRP=ICEGRP SRVID=1
    TMFFNAME  SRVGRP=ICEGRP SRVID=2
                  CLOPT="-A -- -N -M"
    TMFFNAME  SRVGRP=ICEGRP SRVID=3 CLOPT="-A -- -N"
    TMFFNAME  SRVGRP=ICEGRP SRVID=4 CLOPT="-A -- -F"
    <App Server Name>  SRVGRP=GROUP1 SRVID=2
    ISL       SRVGRP=GROUP1 SRVID=1
                  CLOPT="-A -- -d /dev/tcp -n //<host>:<port>"
#
# SERVICES
#
*SERVICES
```

Sample DMCONFIG File

Listing 3-7, Listing 3-8, and Listing 3-9 show the `DMCONFIG` files for three directly connected domains: Here, There, and Yonder.

Note: To use Listing 3-7 in a multidomain configuration, you must replace *`host1`* with the name of the local machine for the Here domain, replace *`host2`* with the name of the local machine for the There domain, and replace *`host3`* with the name of the local machine for the Yonder domain,

Listing 3-7 DMCONFIG File for the Local Machine in the Here Domain in a Three-Domain Configuration

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES

    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS
```

```
    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="HereG"

#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS

    TDOM1    TYPE=TDOMAIN DOMAINID="ThereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="YonderG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN

    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
    "//HereD"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES

    "//ThereD     "RDOM=TDOM1
    "//YonderD    "RDOM=TDOM2
```

Note: To use Listing 3-8 in a multidomain configuration, you must replace *host1* with the name of the local machine for the There domain, replace *host2* with the name of the local machine for the Here domain, and replace *host3* with the name of the local machine for the Yonder domain,

Listing 3-8 DMCONFIG File for the There Domain in a Three-Domain Configuration

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
```

```
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES

    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS

    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="ThereG"

#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS

    TDOM1    TYPE=TDOMAIN DOMAINID="HereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="YonderG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN

    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
    "//ThereD"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES

    "//HereD     "RDOM=TDOM1
    "//YonderD    "RDOM=TDOM2
```

Note: To use Listing 3-9 in a multidomain configuration, you must replace *host1* with the name of the local machine for the Yonder domain, replace *host2* with the name of the local machine for the Here domain, and replace *host3* with the name of the local machine for the There domain.

Listing 3-9 DMCONFIG File for the Yonder Domain in a Three-Domain Configuration

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#      All rights reserved
#
#
# Tuxedo DOMAIN CONFIGURATION FILE
#
*DM_RESOURCES

    VERSION=U22

#
# DM_LOCAL_DOMAINS
#
*DM_LOCAL_DOMAINS

    LDOM1  GWGRP=LGWGRP  TYPE=TDOMAIN  DOMAINID="YonderG"

#
# DM_REMOTE_DOMAINS
#
*DM_REMOTE_DOMAINS

    TDOM1    TYPE=TDOMAIN DOMAINID="HereG"
    TDOM2    TYPE=TDOMAIN DOMAINID="ThereG"


#
# DM_TDOMAIN
#
*DM_TDOMAIN

    LDOM1   NWADDR="//<host1>:<tcpport>"
    TDOM1   NWADDR="//<host2>:<tcpport>"
    TDOM2   NWADDR="//<host3>:<tcpport>"
#
# DM_LOCAL_SERVICES
#
*DM_LOCAL_SERVICES
```

```
    "//YonderG"
#
# DM_REMOTE_SERVICES
#
*DM_REMOTE_SERVICES

    "//HereD     "RDOM=TDOM1
    "//ThereD    "RDOM=TDOM2
```

Sample factory_finder.ini File

This section shows the `factory_finder.ini` files for the Here and There domains. The Yonder domain does not require a `factory_finder.ini` file.

Listing 3-10 factory_finder.ini File for the Here Local Domain

```
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
# Factory Finder Initialization file for Domain "Here".
# This is the local Domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES

    "AFactory.FactoryInterface"
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
       DOMAINID="YonderD"
       RNAME="AFactory.FactoryInterface"

    "BFactory.FactoryInterface"
       DOMAINID="YonderD"
```

Listing 3-11 factory_finder.ini File for the There Remote Domain

```
#
#Copyright (c) 1999 BEA Systems, Inc.
#All rights reserved
#
# Factory Finder Initialization file for Domain "There".
#This is a remote domain.
#
# DM_LOCAL_FACTORIES
#
*DM_LOCAL_FACTORIES
    "AFactory.FactoryInterface"
#
# DM_REMOTE_FACTORIES
#
*DM_REMOTE_FACTORIES
    "AFacYonder.FactoryInterface"
DOMAINID="YonderD"
RNAME="AFactory.FactoryInterface"
 "BFactory.FactoryInterface"
DOMAINID="YonderD"
```

CHAPTER

4 Administering Domains

This topic includes the following sections:

- Using Domains Run-time Administrative Commands
- Using the Administrative Interface, dmadmin(1)
- Using the Domains Administrative Server, DMADM(5)
- Using the Gateway Administrative Server, GWADM(5)
- Using the Gateway Process
- Managing Transactions in a Domains Environment

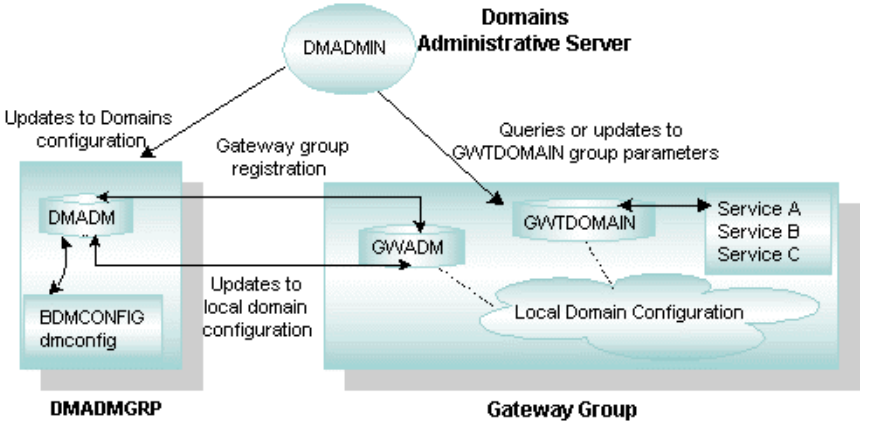
Using Domains Run-time Administrative Commands

To integrate the Domains component with an existing BEA Tuxedo application, add entries for domain gateway groups and gateway servers to the `TUXCONFIG` file. You can use either the `tmconfig(1)` (see `tmconfig, wtmconfig(1)`) or `tmadmin(1)` command to add a multiple-domain configuration to a running BEA Tuxedo application. You can also use `tmadmin` to list the information available in the bulletin board for Domain gateway groups and individual gateways.

Once your Domains environment is configured and integrated, you can administer it dynamically using a set of administrative tools provided by the Domains software. For example, you can specify and modify the list of services that are accessible across applications. The Domains software preserves the characteristics of the BEA Tuxedo programming interface (ATMI) and extends the scope of the ATMI so that clients can invoke services across domains. This functionality allows programmers to expand or partition applications without changing any application code.

The following figure shows the relationship between administrative commands and servers in the Domains administrative subsystem.

Figure 4-1 Domains Run-time Administration



Domains offers the following administrative commands:

- `dmadmin(1)` command, a generic administrative service—enables administrators to configure, monitor, and tune domain gateway groups dynamically, and to update the Domains configuration file (`BDMCONFIG`) while the BEA Tuxedo application is running. The command acts as a front-end process that translates administrative commands into service requests which it then sends to the `DMADMIN` service, a generic administrative service advertised by the `DMADM` server. The `DMADMIN` service invokes the validation, retrieval, or update of functions provided in the `DMADM` server to maintain the `BDMCONFIG` file.
- `DMADM(5)`, the gateway group administrative server—provides the administrative processing required for updating the Domains configuration. This server acts as a back-end to the `dmadmin` command. It provides a registration service to

gateway groups. This registration service is requested by GWADM servers as part of their initialization procedure. The registration service downloads the configuration information required by the requesting gateway group. The DMADM server maintains a list of registered gateway groups, and propagates to these groups any changes made to the configuration.

- GWADM(5), the gateway process—the GWADM server registers with the DMADM server to obtain the configuration information used by the corresponding gateway group. The GWADM accepts queries from DMADM to obtain run-time statistics or to change the run-time options of the corresponding gateway group. Periodically, the GWADM server sends an "I-am-alive" message to the DMADM server. If no reply is received from the DMADM server, the GWADM server registers again. This mechanism makes sure the GWADM server always has the latest copy of the Domains configuration for its group.

- GWTDOMAIN(5)—the gateway process, GWTDOMAIN, which provides connectivity to remote gateway processes, focuses on throughput of messages between BEA Tuxedo domains. Clients and servers send and receive messages across BEA Tuxedo domains via the GWTDOMAIN process.

  Note: For a gateway type other than GWTDOMAIN, an executable other than GWTDOMAIN must be used. Refer to the BEA eLink Adapter for Mainframe documentation and *Using the BEA Tuxedo TOP END Domain Gateway with ATMI Applications* for additional information.

- BDMCONFIG—the binary version of the Domains configuration file, which contains all the configuration parameters that the BEA Tuxedo software interprets to create a viable application.

Note: You can also specify gateway parameters when a gateway group is booted using the CLOPT parameter, when the GWADM server is defined in the SERVERS section of the TUXCONFIG file.

How to Migrate DMADM and a Domain Gateway Group

The migration of DMADM is possible. To migrate DMADM to a new machine, complete the following steps.

1. Copy DMCONFIG to the new machine and run dmloadcf.
2. Shut down all domain gateway groups (GWADM and a domain gateway, for example, GWTDOMAIN).

   Note: If the domain gateway groups are not shut down, they will continue to function, but after DMADM has been migrated, all MIB requests for them will fail.
3. Migrate the DMADM group to the new machine.

The migration of a domain gateway group is possible. However, when transactions are being used, the domain gateway group can be migrated only across machines of the same type. To migrate a domain gateway group, complete the following steps.

1. In the DMCONFIG file, add multiple listening addresses, in the following format, to the DM_TDOMAIN section:

   ```
   *DM_TDOMAIN
   LDOM NWADDR="//primary:port"
   LDOM NWADDR="//backup:port"
   ```

   Note: This step is unnecessary if third-party IP failover solutions are used.
2. If you are using transactions, you must copy the Domains transaction log manually to the backup machine.
3. The DMCONFIG files for the remote domains should include both network addresses as specified in step 1.
4. Migrate the domain gateway group to the new machine.

Using the Administrative Interface, dmadmin(1)

`dmadmin` is an administrative interface to the `DMADM` and `GWADM` servers. The communication between the two servers is done via FML typed buffers. Administrators can use the `dmadmin` command in the following ways:

- For the interactive administration of the information stored in the `BDMCONFIG` file and the different gateway groups running within a particular BEA Tuxedo application.
- To obtain statistics or other information gathered by gateway groups.
- To change gateway group parameters.
- To add (or update) information in the `BDMCONFIG` file.

Note: You can delete information from the `BDMCONFIG` file at run time only if the deletions do not involve an active gateway group.

See Also

- `dmadmin(1)` in the *BEA Tuxedo Command Reference*

Using the Domains Administrative Server, DMADM(5)

The Domains administrative server, DMADM(5), is a BEA Tuxedo-supplied server that performs the following functions:

- Supports run-time administration of the BDMCONFIG file
- Maintains the BDMCONFIG file
- Supports a list of registered gateway groups
- Propagates run-time configuration changes to the registered gateway groups

The DMADM server advertises two services:

- DMADMIN, which is used by the DMADMIN and the GWADM servers.
- A service called DMADM_*svrid*, where SRVID is the appropriate server ID for the service. Registered GWADM servers use DMADM_*svrid* for specific administrative functions (for example, to refresh the gateway group configuration information or to signal that a GWADM is still registered).

The DMADM server must be defined in the SERVERS section of the TUXCONFIG file as a server running within a group (for example, DMADMGRP). There should be only one instance of the DMADM server in this group and it must be defined with no reply queue (REPLYQ=N).

See Also

- DMADM(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Using the Gateway Administrative Server, GWADM(5)

The gateway administrative server, GWADM(5), is a BEA Tuxedo-supplied server that provides administrative functions for a Domains gateway group. The main functions of the GWADM server include the following:

- To get Domains configuration information from the DMADM server, and to accept queries from dmadmin. The GWADM server gets the gateway group configuration information by registering with the DMADM server. The GWADM server then makes the configuration available to gateways by storing the information in shared memory.
- To provide administrative functionality for a gateway group, for example, to accept queries from dmadmin for run-time statistics or to change the run-time parameters of the gateway group.
- To provide transaction logging functionality for a gateway group. The GWADM server determines which transactions need to be logged by reading information stored in shared memory. When the GWADM server is booted; scans the log to see whether any transactions need to be recovered; it then reconstructs the transaction information in shared memory. The gateway server scans the information in shared memory and performs recovery for the corresponding transactions. The recovery procedure is performed asynchronously with new incoming or outgoing requests received by the gateway group.

The GWADM server advertises a service name based on the local domain name (the value of the LDOM keyword in the BDMCONFIG). The dmadmin command uses this service to retrieve information from all active gateway groups or from a specific gateway group.

The GWADM server must be defined in the SERVERS section of the TUXCONFIG file. It should not be part of the MSSQ used by the gateways associated with the group and it must not have a reply queue, that is, REPLYQ=N must be specified. It must be the first server booted within the gateway group; that is, either (a) it must have a SEQUENCE number, or (b) it must be defined ahead of the gateway servers.

The GWADM server requires the existence of a DMADM server. Specifically, a DMADM server must be booted before that GWADM is booted.

The GWADM server must create the shared memory required by the gateway group to populate the configuration tables with information received from the DMADM server. The GWADM server uses IPC_PRIVATE with shmget and stores the ipckey returned in the shmid field of its registry entry in the bulletin board. Gateways can obtain the ipckey by retrieving the GWADM registry entry and checking the shmid field.

See Also

- GWADM(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*

Using the Gateway Process

A gateway process provides connectivity to remote gateway processes, and can communicate with one or more remote gateways simultaneously. A gateway advertises the services imported to a BEA Tuxedo application and controls access to the local services exported by the application. You define your application's exported and imported services in the Domains configuration file (DMCONFIG). Use dmadmin to dynamically configure, monitor, and tune domain gateway groups.

See Also

- "Types of Domain Gateways" on page 1-6

Managing Transactions in a Domains Environment

Application programmers can request the execution of remote services within a transaction. Also, users of remote domains can request local services to be executed within a transaction. Domains, therefore, coordinates the mapping of remote transactions to local transactions, and the sane termination (commitment or rollback) of these transactions.

The BEA Tuxedo system architecture uses a separate process, the Transaction Manager Server (`TMS`), to coordinate the commitment and recovery of transaction branches accessing a particular group. In a Domains environment, however, this architecture would require extra messages from the gateway to the `TMS` server to process a commitment for an incoming transaction. To simplify the Domains architecture and to reduce the number of messages, the `TMS` code is integrated with the gateway code. Thus, domain gateways can process the transaction protocol used by the BEA Tuxedo system. The BEA Tuxedo transaction protocol requires that the gateway group advertise the `TMS` service, which is done when the first gateway is booted. Once the `TMS` service is advertised, any transaction control messages directed to the gateway group are placed on the gateway's queue.

Domains gateway groups should be defined in the `TUXCONFIG` file without the `TMSNAME`, `TMSCOUNT`, `OPENINFO`, and `CLOSEINFO` parameters. These four parameters apply only to groups that use an XA-compliant resource manager, which Domains gateways do not use.

The commitment protocol across domains is strictly hierarchical. It is not possible to flatten the transaction tree because the structure of the transaction tree is not fully known by every domain; a superior knows only its immediately subordinate domains. Flattening the tree would also require the root domain to be fully connected to all domains participating in the transaction.

Transaction Management Capabilities

Domain gateways provide four capabilities that you can use to manage transactions. These capabilities are described in the following sections:

Using the TMS Capability Across Domains

In the BEA Tuxedo system, the TMS is a special server that is implicitly associated with server groups that use X/Open XA-compliant resource managers. The TMS server releases application servers from the delays associated with the distributed 2-phase commitment protocol. TMSs coordinate the commitment of a transaction via special service requests to the TMS service, which is offered by all TMS servers.

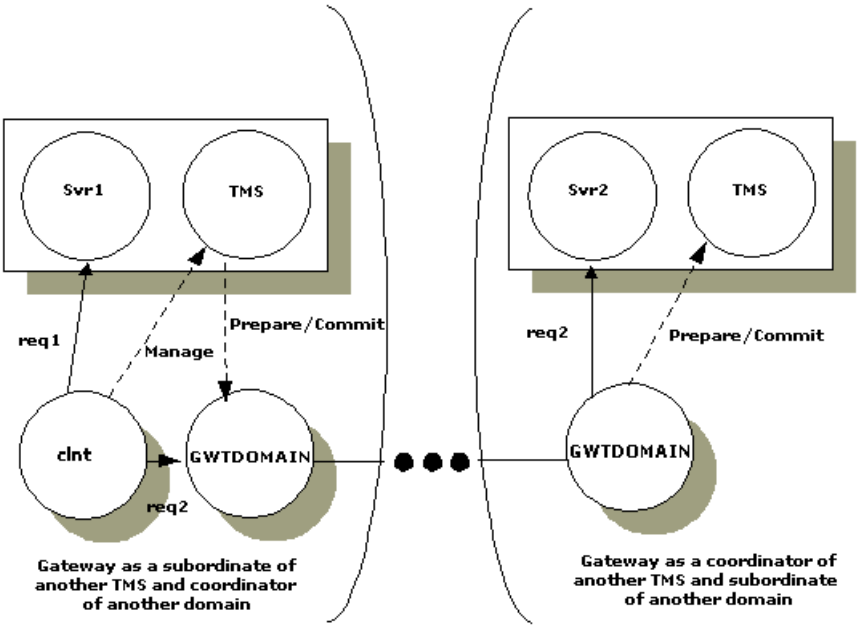
In a Domains environment, GWTDOMAIN gateways are not associated with an XA-compliant resource manager. The Transaction Processing Working Group (TPWG) of X/Open has proposed an advanced XA interface. This interface is not used in the BEA Tuxedo system because the interface does not match the highly asynchronous and non-blocking model required by the gateway. While Domains gateways do not use a separate TMS server, they do offer the Transaction Manager Servers (TMS) capability, which allows gateways to coordinate the 2-phase commitment of transactions executed across domains.

How Gateways Coordinate Transactions Across Domains

1. Domain gateways advertise the TMS service and perform all operations associated with that service. Messages sent to this service are placed on the queue used by the appropriate gateway group, and the gateways manage the transactions associated with the group.

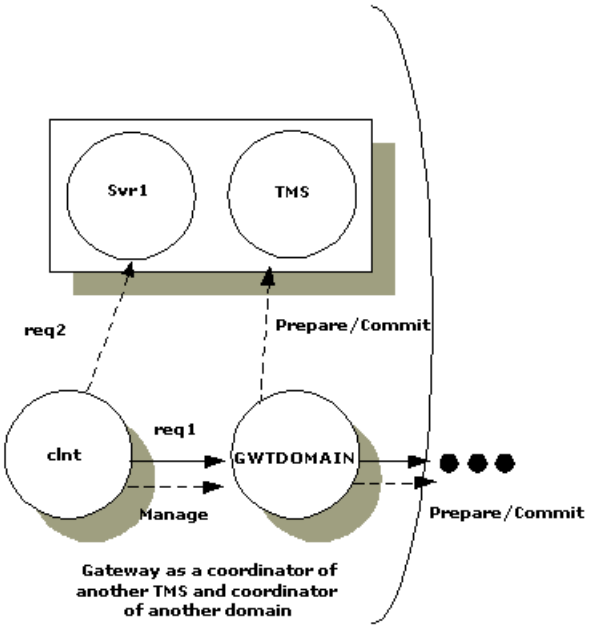
2. A gateway can act as a subordinate of transactions coordinated by another group within the domain. In this case, the gateway is a superior of the transaction branches executed in other remote domains. When acting as a subordinate of a transaction coordinated by a remote domain, the gateway also acts as the coordinator for all groups in the local domain accessed by the transaction. The gateway, acting as both subordinate and coordinator, is illustrated in the following figure.

Figure 4-2 The Gateway as Subordinate/Coordinator of Another Domain Group



3. As a coordinator of transactions within the domain, the gateway manages the commitment of a transaction for a particular client. This is illustrated in the following figure.

Figure 4-3 Client Commit Managed by a Gateway



4. Gateways manage transaction commitment for a particular client or for a server that uses the forwarding service with the AUTOTRAN capability. When this combination is used, the last server in the forward chain (the Domains gateway) issues the commit and becomes the coordinator of the transaction. (A domain gateway always acts as the last server in a forward chain.)

5. Gateways automatically start and terminate transactions for remote services specified with the AUTOTRAN capability. This capability is required when an the application administrator wants to enforce reliable network communication with remote services. Administrators can specify this capability by setting the AUTOTRAN parameter to Y in the corresponding remote service definition.

   For more information, refer to the DM_REMOTE_SERVICES Section of DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

6. Gateways map the BEA Tuxedo system transaction protocol to the networking transaction protocol used for interoperation with remote domains. How this mapping is done depends on which instantiation of Domains you are using: TDomains, SNA, or OSI TP.

Using GTRID Mapping in Transactions

In the BEA Tuxedo system, a transaction tree is a 2-level tree where the root is the gateway group coordinating a global transaction and branches are involved in the transaction. Each group performs its part of the global transaction independently from the parts performed by other groups. Each group, therefore, implicitly defines a transaction branch. The BEA Tuxedo system, through Transaction Manager Servers (TMSs), coordinates the completion of the global transaction, making sure each branch is completed.

A GTRID is a Global Transaction Identifier. GTRID mapping defines how to construct a transaction tree that crosses domain boundaries. You specify GTRIDs using the MAXGTT parameter in the RESOURCES section of the configuration file.

Defining Tightly-coupled and Loosely-coupled Relationships

In the X/Open DTP Model, a Transaction Manager Server can construct transaction trees by defining either *tightly-coupled* or *loosely-coupled* relationships with a resource manager (RM) by the way it interprets the transaction identifiers (XIDs) used by the XA interface.

A *tightly-coupled relationship* is one in which a single transaction identifier, XID, is used by all processes participating in a single global transaction, accessing a single RM. This relationship maximizes data sharing between processes; XA-compliant RMs expect to share locks for resources used by processes having the same XID. The BEA Tuxedo system achieves the tightly-coupled relationship via the group concept; that is, all work done by a group on behalf of a given global transaction belongs to the same transaction branch; all the processes executed by the group are given the same XID.

In a *loosely-coupled relationship*, the TMS generates a transaction branch for each part of the work in support of the global transaction. The RM handles each transaction branch separately; there is no sharing of data or of locks between the transaction branches. Deadlocks between transaction branches can occur and result in the rollback

of a global transaction. In the BEA Tuxedo application, when different groups participate in a single global transaction, each group defines a separate transaction branch, which results in a loosely-coupled relationship.

Global Transactions Across Domains

There are several differences between global transactions in a single BEA Tuxedo application and global transactions across domains. The first difference is that in the Domains framework, the transaction tree cannot be flattened to a 2-level tree. There are two reasons for this:

- The transaction may involve more domains than can be known from the root domain (where the transaction is controlled), so the structure of the transaction tree cannot be fully known.
- If a transaction tree is flattened to two levels, the root domain must be connected directly to all domains in the transaction.

This means that the commitment protocol across domains must be hierarchical. Even a loopback service request defines a new branch in the transaction tree.

Note: A loopback request goes to another domain and then comes back to be processed in the original domain. For example, Domain A requests a service of Domain B. The service in Domain B requests another service in Domain A. The transaction tree has two branches at the network level: a branch b1 from A to B and a branch b2 from B to A. Domain A cannot commit the work done on branch b2 before receiving commit instructions from B.

The structure of a transaction tree for global transactions across domains also depends on the distributed transaction processing protocol used by a relevant Domains instantiation. For example, in the OSI TP protocol each *dialogue* (the OSI TP word for a service request) is associated with a different transaction branch. In the BEA Tuxedo system, the OSI TP instantiation uses a dialogue for each service request, so each service request is mapped to a separate transaction branch. The XAP-TP interface hides this mapping and provides a mechanism by which an entire OSI TP subtree can be referenced by a user-defined identifier. (In the BEA Tuxedo implementation, this identifier is the `GTRID`.) The `GTRID` is used to instruct XAP-TP how a transaction tree must be constructed, that is, which dialogues must be included within a given OSI TP transaction. Therefore, from the BEA Tuxedo perspective, a whole OSI TP subtree can be managed as a single transaction branch.

This property, however, applies only to outgoing service requests (that is, service requests sent from the root domain to subordinate domains). It cannot be applied to incoming service requests. The OSI TP instantiation consequently implements a loosely-coupled relationship; each incoming service request is mapped to a new BEA Tuxedo global transaction.

The TDomain instantiation tries to optimize `GTRID` mapping by implementing a tightly-coupled relationship. In TDomain, multiple service requests issued on behalf of the same global transaction are mapped to the same network transaction branch. Therefore, incoming service requests can be mapped to a single BEA Tuxedo transaction. However, the hierarchical structure of interdomain communication and the interdomain transaction tree must still be maintained.

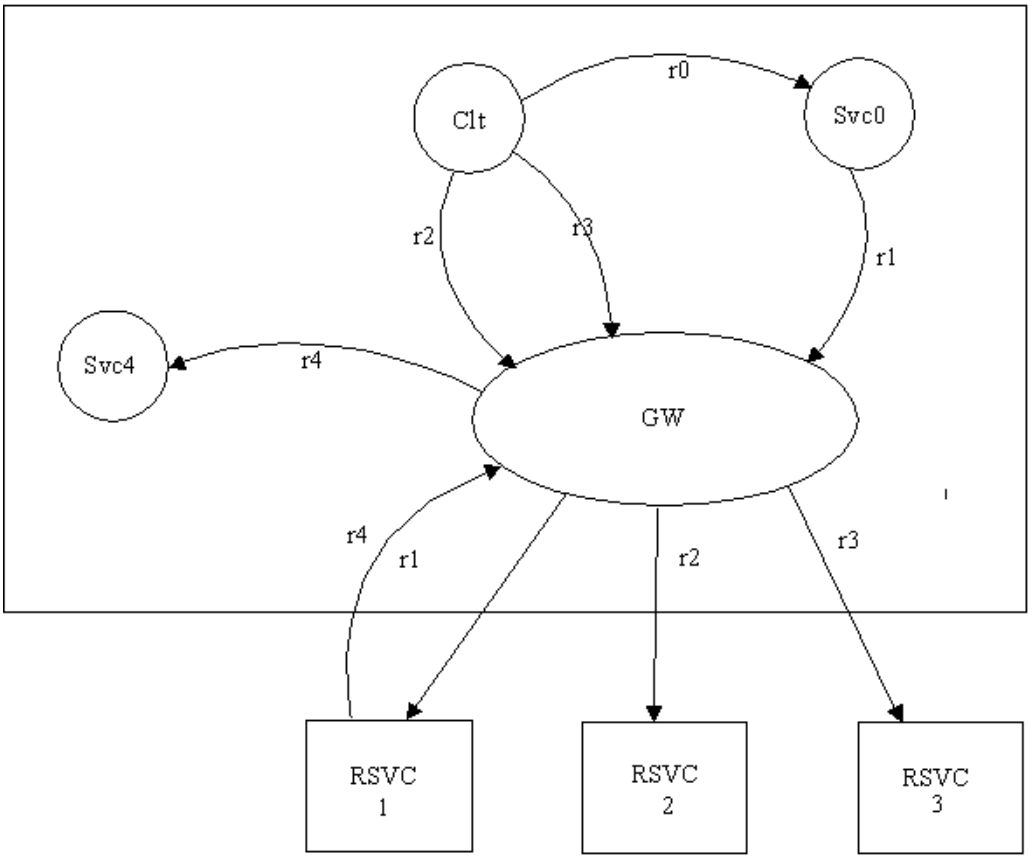
The optimization that TDomain introduces applies only to a single domain. When two or more domains are involved in a transaction, the network transaction tree contains at least one branch per domain interaction. Hence, across domains, the network transaction tree remains loosely-coupled. There are as many branches as there are domains involved in the transaction (even if all the branches access the same resource manager instance).

Domains gateway groups implement a loosely-coupled relationship because they generate different transaction branches for interdomain transactions.

Example of a Service Request Graph Generating Local and Remote Requests

The following figure shows the service request graph for a client that generates three service requests: one local request (`r0`) and two remote requests (`r2` and `r3`). Request `r0` goes to a local service (`Svc0`), which generates another remote service request (`r1`). Request `r1` goes to remote service `Rsvc1`, which issues a loopback service request `r4` to local service `Svc4`. `Svc0` and `Svc4` are executed in different groups (`G0` and `G4`). The domain gateway is executed within another group (`GW`), and the remote services `Rscv1`, `Rsvc2`, and `Rsvc3` are executed in another domain (Domain B).

Figure 4-4 Service Request Graph



Transaction Trees for BEA eLink OSI TP and BEA Tuxedo Domains

The following two figures show the transaction tree for BEA eLink OSI TP and the transaction tree for BA Tuxedo Domains. It is assumed, in these figures, that both Domains A and B are BEA Tuxedo system applications.

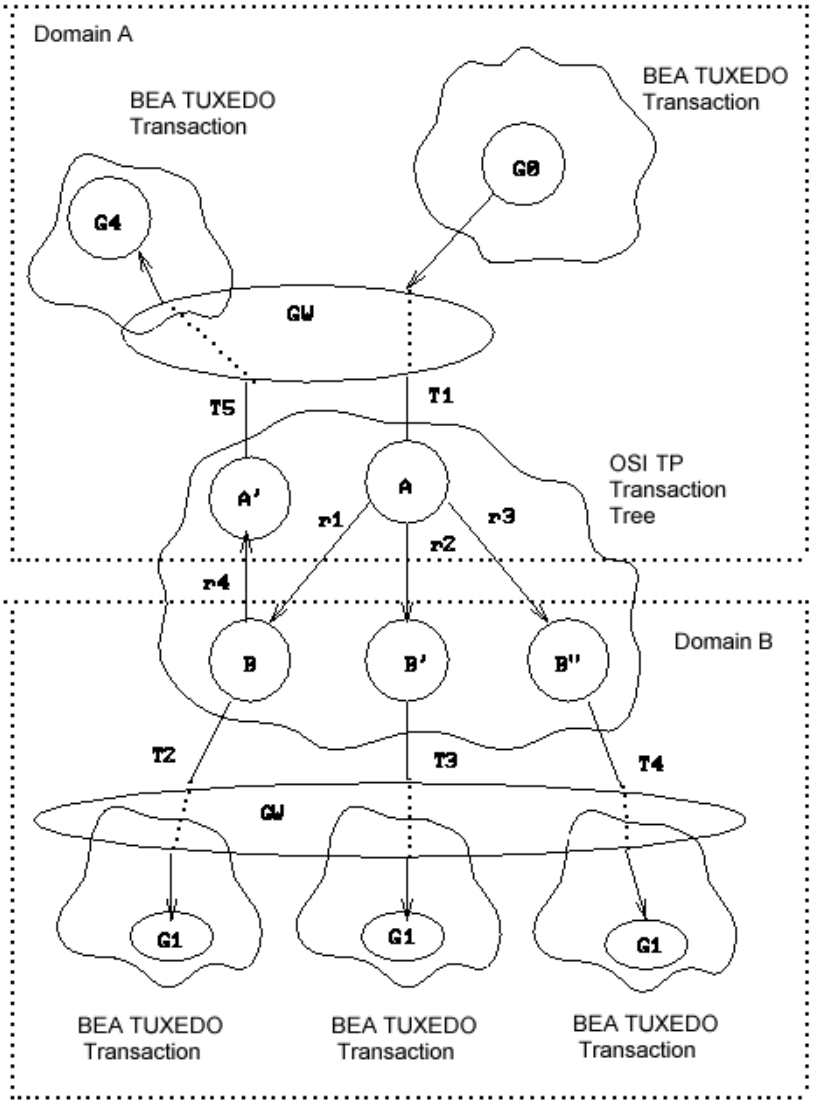
BEA eLink OSI TP is loosely-coupled because of the OSI TP protocol. The transaction tree for this instantiation shows group G0 in Domain A coordinating the global transaction started by the client. Group `G0` coordinates group `GW`. Requests `r1`, `r2`, and `r4` are mapped each to an OSI TP dialogue and therefore to an OSI TP transaction branch. However, OSI TP uses the XAP-TP feature that allows an entire OSI TP transaction to be referred by a unique identifier (`T1`) and uses this identifier for requests `r1`, `r2`, and `r3`. It is up to XAP-TP to generate OSI TP transaction identifiers and to construct the corresponding OSI TP transaction tree. The only function that must be performed by the generic Domains software is the mapping of service requests r1, r2, and r3 to the `T1` identifier.

In Domain B, OSI TP uses the rule that new transaction branches must be mapped to a new BEA Tuxedo transaction. Therefore, OSI TP transaction branches `r1`, `r2`, and `r3` get mapped to three different BEA Tuxedo transactions (the corresponding mapping is represented by identifiers `T2`, `T3`, and `T4`). The graph shows the gateway group `GW` in Domain B coordinating three BEA Tuxedo transactions on group `G1`.

Finally, there is the loopback service request `r4` that generates another branch in the transaction tree. OSI TP maps this request to identifier `T2`, but XAP-TP generates a new branch in its transaction tree (`r4`: B to A'). This is a new transaction branch on Domain A, and therefore, the gateway generates a new mapping `T5` to a new BEA Tuxedo transaction. Therefore, the transaction graph shows that gateway group GW on Domain A coordinates group `G4`.

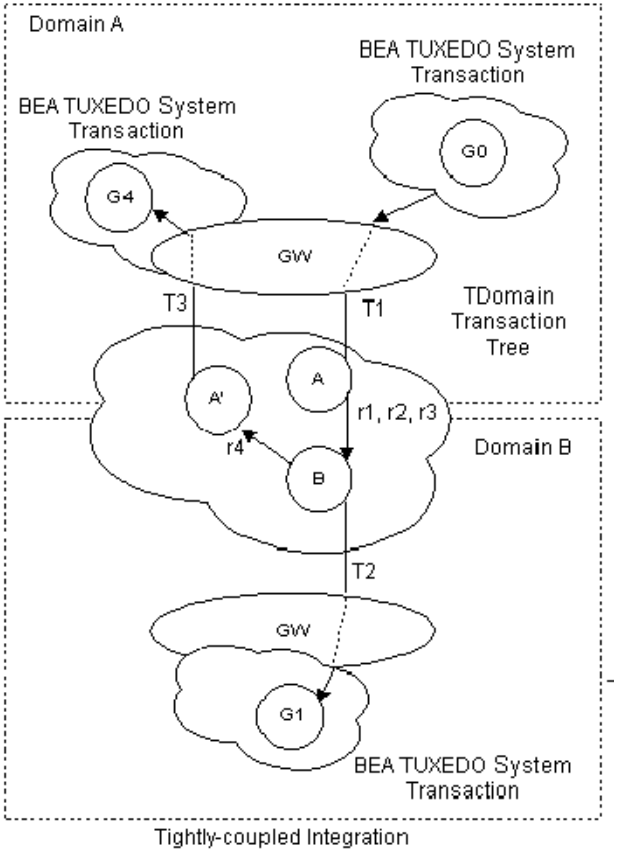
Notice that the hierarchical nature of the OSI TP protocol is fully enforced by these mappings. However, because these mappings introduce a loosely-coupled relationship, the probability of intratransaction deadlock is increased (for example, there are three BEA Tuxedo transactions accessing the RM represented by group `G1`).

Figure 4-5 Transaction Tree for BEA eLink OSI TP Environment



The TDomain instantiation provides a tightly-coupled integration that solves this deadlock problem by minimizing the number of transaction branches required in the interoperation between two domains. The corresponding transaction tree is shown in the following figure.

Figure 4-6 Transaction Tree for TDomain Environment



Notice that the gateway still must perform mappings between a BEA Tuxedo system transaction and a network transaction, and that the hierarchical nature of the communication between domains must be strictly enforced. The diagram shows that requests `r1`, `r2`, and `r3` are mapped to a single TDomain transaction branch. Therefore, on Domain B only one BEA Tuxedo system transaction needs to be generated; `T2` represents this mapping and the graph shows gateway group `GW` on Domain B

coordinating group G1. Request r4 is mapped to identifier T2 on Domain B, but TDomain will generate a new branch in its transaction tree (r4: B to A'). Because this is a new transaction branch on Domain A, the gateway generates a new mapping, T3, to a new BEA Tuxedo system transaction. The graph shows that gateway group GW on Domain A also coordinates group G4. Hence, the hierarchical nature of interdomain communication is fully enforced with this mapping: group G4 cannot commit before group G1.

Summary of Domains Transaction Management

Domains transaction management can be summarized as follows:

- Gateways generate mappings from a BEA Tuxedo system transaction to a network transaction. A new mapping is generated for each BEA Tuxedo system transaction and each incoming network transaction branch.
- Each instantiation of Domains (TDomains, SNA, or OSI TP) handles its own representation of the network transaction tree. All instantiations observe the hierarchical nature of the interdomain communication.

Using Logging to Track Transactions

Logging is used to keep track of the progress of a 2-phase commit protocol. The information stored in the log is used to make sure a transaction is completed in the event of a network failure or machine crash.

To ensure completion of transactions across domains, domain gateways log the mapping between local and remote identifiers. Along with this information, the Domains transaction management facility records the decisions made during different phases of the commitment protocol, and any information available about the remote domains involved in the transaction. In the OSI TP case, the XAP-TP interface logs the information required for the recovery of the OSI TP protocol machine. The information is referred to as a *blob* (binary large object) and is kept in the same log record as the commit information to make recovery easier.

Domains log records have a different structure from the log records stored in the BEA Tuxedo system TLOG. TLOG records are fixed in size and are stored in a single page. Domains log records vary in size; more than one page may be required to store the record. The Domains logging mechanism, DMTLOG, has the capability of storing variable-size log records.

When a TMS is the superior of a domain gateway group, the BEA Tuxedo TLOG is still required to coordinate the commitment.

How Logging Works

Logging is performed by the GWADM administrative server. The request for a log write is made by the GWTDOMAIN process, but the actual log write is performed by the GWADM process.

You must create a log called DMTLOG for each domain gateway group. The DMTLOG files are defined in the DM_LOCAL_DOMAINS section of the DMCONFIG file. To create a DMTLOG file, add an entry for the DMTLOGDEV parameter:

DMTLOGDEV=*string*

where *string* is the name of the log file. In addition, you cam set one or both of the two optional parameters:

- DMTLOGNAME=*identifier*
- DMTLOGSIZE=*numeric*

For more information, refer to DMCONFIG(5) in the *File Formats, Data Descriptions, MIBs, and System Processes Reference*.

Administrators also have the option of using the run-time administration utility (dmadmin) to create a DMTLOG. For more information, refer to dmadmin(1) in the *BEA Tuxedo Command Reference*.

If a DMTLOG has not been created when a domain gateway group is booted, the gateway server automatically creates the log, based on information in the BDMCONFIG file.

Until a logging device is specified in the BDMCONFIG file, a Domain gateway group cannot process requests in transaction mode and the gateway group cannot offer the TMS service.

To coordinate the commit protocol, Domains gateways require the following two log records:

- Ready record—a ready record is a file created by a gateway acting as a leaf or intermediate machine in a transaction tree. It records information about the superior and subordinate remote domains involved in the transaction. A ready record indicates that all subordinates of the domain gateway group logging the record have been prepared.
- Commit record—a commit record documents that a transaction has been committed. A domain gateway creates a commit record as the coordinator of a particular transaction tree.

When a transaction has been committed on all machines, these logs for the transaction are removed.

When the OSI TP protocol is being used, two types of heuristic records are logged:

- Log Heuristic record—this record holds the details of a heuristic decision in the domain until the outcome of the relevant transaction is known by the superior.
- Log Damage record—this record is created to indicate one of two conditions for a transaction branch: (run with `tmadmin(1)`) a *heuristic hazard* (when the outcome of the transaction branch for a subordinate is unknown) or a *heuristic mix* (when the transaction subtree has a mixed outcome).

Heuristic log records persist until they are explicitly removed by the administrator. This persistence is required to provide the correct information during recovery after a crash, and to provide diagnostic information for administrators.

The administrator uses the `forgettran` command (run with `tmadmin(1)`) to remove heuristic records when they are no longer needed.

Recovering Failed Transactions

When a domain gateway group is booted, the gateway server performs an automatic *warm-start* of the DMTLOG. The warm-start includes scanning the log to see if any transactions were not completed. If incomplete transactions are found, action is taken to complete them.

In OSI TP, any *blobs* stored in the DMTLOG with a transaction record are passed to the network access module, which uses the blobs to reconstruct its internal state and to recover any failed connections

In the case of heuristic decisions, if a domain gateway group is a subordinate of a local TMS and a heuristic decision has been indicated, the TMS generates a TMS_STATUS message to learn the final decision:

- If a gateway fails, then it cleans up after itself when it is restarted (this is called a *hot-start*). The gateway rolls back all undecided transactions in which it was involved.
- If a communication line failure occurs and the first phase of the commit has not been completed, the gateway rolls back the transactions associated with that connection.
- If OSI TP Domains is being used and a transaction fails in the second phase of the commit, recovery is managed by XAP-TP.