[MS-TPSO]:
Transaction Processing Services System Overview

This document provides an overview of the Transaction Processing Services System Overview Protocol Family. It is intended for use in conjunction with the Microsoft Protocol Technical Documents, publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts. It assumes that the reader is either familiar with the aforementioned material or has immediate access to it.

A Protocol Family System Document does not require the use of Microsoft programming tools or programming environments in order to implement the Protocols in the System. Developers who have access to Microsoft programming tools and environments are free to take advantage of them.

Abstract

Transaction processing is designed to maintain a computation system in a known, consistent state. It allows multiple individual operations to be linked together as a single, indivisible operation: an atomic transaction. Broadly speaking, **transaction** processing involves updating data, which may be distributed across multiple systems, so that either all the changes happen or none of the changes happen.

This document describes the intended functionality of the Transaction Processing Services System, how it interacts with systems or applications that need transaction processing, and how it interacts with management clients that need to configure and manage the system. Transaction Processing Services System supports multiple protocols for transaction processing and management. This document lists those supported protocols and how they interact in a combined system.

Revision Summary

Date	Revision History	Revision Class	Comments
06/20/2008	0.1	Major	Updated and revised the technical content.
07/25/2008	0.1.1	Editorial	Revised and edited the technical content.
08/29/2008	1.0	Major	Updated and revised the technical content.
10/24/2008	2.0	Major	Updated and revised the technical content.
12/05/2008	3.0	Major	Updated and revised the technical content.
01/16/2009	3.0.1	Editorial	Revised and edited the technical content.
02/27/2009	4.0	Major	Conversion to new template.
04/10/2009	4.1	Minor	Conversion to new template.
05/22/2009	4.1.1	Editorial	Conversion to new template.
07/02/2009	4.2	Minor	Updated the technical content.
08/14/2009	4.3	Minor	Updated the technical content.
09/25/2009	4.4	Minor	Updated the technical content.
11/06/2009	4.5	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
12/18/2009	5.0	Major	Updated and revised the technical content.
01/29/2010	5.1	Minor	Updated the technical content.
03/12/2010	5.1.1	Editorial	Revised and edited the technical content.
04/23/2010	5.1.2	Editorial	Revised and edited the technical content.
06/04/2010	5.1.3	Editorial	Revised and edited the technical content.
07/16/2010	5.1.3	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	5.1.3	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	5.1.3	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	6.0	Major	Significantly changed the technical content.
01/07/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	7.0	Major	Significantly changed the technical content.
03/25/2011	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	7.1	Minor	Clarified the meaning of the technical content.
09/23/2011	7.1	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	7.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	7.1	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	7.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	7.1	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	7.1	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This Protocol Family System Document (PFSD) is primarily intended to cover the Protocol Family as a whole. In conjunction with Member Protocol Technical Documents (TDs), which are intended to cover Member Protocols, it presents the rules for information exchange relevant to those Member Protocols and the Protocol Family that are used to interoperate or communicate with a Windows operating system in its various environments. This Protocol Family System Document describes how to interoperate with the Transaction Processing Services System, which provides distributed transaction services.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

abort request
application
commit request
recovery
security provider
subordinate transaction manager
superior transaction manager
transaction
transaction identifier
transaction manager
transaction propagation
two-phase commit

The following terms are defined in [MS-DTCO]:

abort outcome
commit outcome
enlistment
facet
outcome
participant
pull propagation
push propagation
resource
resource manager (RM)
rollback
root application
root transaction manager
subordinate participant

The following terms are defined in [MS-DTCLU]:

cold recovery
log
logical unit (LU)
LU name pair
remote LU

The following terms are specific to this document:

protocol family: A group of protocols designed to work together to accomplish common goals.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. Note that in [RFC2119] terms, most of these specifications should be imperative, to ensure interoperability. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

Any specification that does not explicitly use one of these terms is mandatory, exactly as if it used MUST.

1.2 References

This section contains normative and informative references relevant to the Transaction Processing Services System.

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[LU62Peer] IBM Corporation, "SNA LU 6.2 Peer Protocols SC31-6808-02", October 1996, http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SC31-6808-02

[LU62SPS] IBM Corporation, "SNA Sync Point Services Architecture References SC31-8134-00", August 1994, http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=SC31-8134-00

[LU62Verb] IBM Corporation, "SNA Transaction Programmer's Reference Manual for LU Type 6.2 GC30-3084-05", November 1993, http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=GC30-3084-05

[MC-DTCXA] Microsoft Corporation, "MSDTC Connection Manager: OleTx XA Protocol".

[MS-CMOM] Microsoft Corporation, "MSDTC Connection Manager: OleTx Management Protocol".

[MS-CMP] Microsoft Corporation, "MSDTC Connection Manager: OleTx Multiplexing Protocol".

[MS-CMPO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transports Protocol".

[MS-COM] Microsoft Corporation, "Component Object Model Plus (COM+) Protocol".

[MS-DTCLU] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension".

[MS-DTCM] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Internet Protocol".

[MS-DTCO] Microsoft Corporation, "MSDTC Connection Manager: OleTx Transaction Protocol".

[MS-MQSO] Microsoft Corporation, "Message Queuing System Overview".

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions".

[MS-TIPP] Microsoft Corporation, "Transaction Internet Protocol (TIP) Extensions".

[MS-WSRVCAT] Microsoft Corporation, "WS-AtomicTransaction (WS-AT) Version 1.0 Protocol Extensions".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2371] Lyon, J., Evans, K., and Klein, J., "Transaction Internet Protocol Version 3.0", RFC 2371, July 1998, http://www.ietf.org/rfc/rfc2371.txt

[WSAT10] Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft", Web Services Atomic Transaction (WS-AtomicTransaction)", August 2005, http://schemas.xmlsoap.org/ws/2004/10/wsat/

[WSAT11] OASIS, "Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1", July 2007, http://docs.oasis-open.org/ws-tx/wsat/2006/06

[WSC10] Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft, "Web Services Coordination (WS-Coordination)", August 2005, http://schemas.xmlsoap.org/ws/2004/10/wscoor/

[WSC11] OASIS, "Web Services Coordination (WS-Coordination) 1.1", March 2006, http://docs.oasis-open.org/ws-tx/wscoor/2006/06

[XOPEN-DTP] The Open Group, "Distributed Transaction Processing: The XA Specification", February 1992, http://www.opengroup.org/bookstore/catalog/c193.htm

1.2.2 Informative References

[GRAY] Gray, J. and Reuter, A., "Transaction Processing: Concepts and Techniques", San Mateo, CA: Morgan Kaufmann Publishers, 1993, ISBN: 1558601902.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

2 Overview

Section 1, "Introduction", describes this **Protocol Family** System Document. This section introduces the system that is being documented.

2.1 System Summary

It is sometimes necessary to ensure that a set of otherwise separate operations is either all complete, or that none of them succeed. These operations may be distributed in a computer network. In **application** programming, it is possible to achieve such semantics by using transactions. Systems that need transactions generally rely on a transaction processing service where the service handles the coordination of completing multiple operations all at the same time.

The Transaction Processing Services System described in this document provides transaction processing services for systems that require transactions to be coordinated in a distributed system. Very broadly, a transaction is an activity that makes changes to the state of some set of **resources** so that either all the changes happen or none of the changes happen. Resources may be data, such as rows in a database, or logical entities, such as the execution state of a program. Resources changed by a transaction may be in separate systems.

Achieving this under all expected and unexpected conditions is a difficult problem with a well-established solution, described in [GRAY]. The basis of the solution is to factor the execution of the transaction into three elements (application, **transaction manager**, and **resource manager**) that communicate with each other by using a well-defined protocol called the Two-Phase Commit protocol. These elements are the **participant**s in the transaction. The transaction manager and resource manager are usually provided as part of an operating system or other platform elements, such as databases, leaving most programmers with only the application itself to write.

Resource managers represent the resources involved in the transaction. A transaction manager coordinates the transaction, keeping all of the participants in step. All the changes to the resources involved in a transaction are made by applications via implementation-specific protocols outside the scope of the Two-Phase Commit protocol. Exactly one of the applications involved in the transaction initiates and completes the transaction, through communications with its transaction manager. This application is known as the **root application**. Other participants are added to the transaction progressively. Each participant is said to be enlisted on the transaction.

For more detailed descriptions of transaction processing concepts, see [GRAY] chapter 2.1, [MS-DTCO] section 1.3, and terms listed in the *Glossary*.

The Transaction Processing Services System consists of one or more transaction managers that communicate with each other by using protocols internal to the system, and that collectively provide external interfaces to applications and resource managers. Multiple transaction managers may be involved in a transaction for many reasons, for instance, because the applications and resources involved are distributed over a network, or because one of the resources involved is associated with its own specialized transaction manager.

The Transaction Processing Services System uses a set of internal protocols to allow the coordination between all transaction participants and also to allow the monitoring and management of the system. However, there are some other well-known transaction processing standards, as well. To provide interoperability through such standards, the Transaction Processing Services System provides specific external interfaces to enable some applications, resource managers, and transaction managers that do not support the internal protocols defined by the system, to participate in transactions. In this document, these are referred to as external applications, external resource managers, and external transaction managers.

Implementation of this system is required if there are other systems or applications that need transaction coordination services. For example, a message queuing system such as Message Queuing System, as described in [MS-MQSO], or a middle-tier application server system such as Component Object Model Plus (COM+) Protocol, as described in [MS-COM], use transaction services. [MS-MQSO] and [MS-COM] describe the details of how those systems interact with the Transaction Processing Services System.

The Transaction Processing Services System exists in the Windows operating system as an internal infrastructure to support application and systems that need transaction coordination services. Users are not expected to directly interact with the Transaction Processing Services System. Information about which Windows versions this system applies to is provided in section 9 of this document.

2.2 List of Member Protocols

The following table lists the Member Protocols of the Transaction Processing Services System.

Protocol name	Protocol purpose	Document short name
MSDTC Connection Manager: OleTx Transaction Protocol Specification	Enables the creation of, initiation of, distributed propagation of, and participation in transactions.	[MS-DTCO]
MSDTC Connection Manager: OleTx Management Protocol Specification	Enables management tools to obtain a list of transactions being processed by a transaction manager. Enables the changing of settings that are used by other Transaction Processing Services System protocols.	[MS-CMOM]
MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification	Enables the initiation of distributed **transaction propagation** via the TIP protocol.	[MS-DTCM]
Transaction Internet Protocol (TIP) Extensions	Enables distributed propagation of transactions by using the TIP protocol over TCP.	[MS-TIPP]
MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension	Enables an implementation of **logical unit** type 6.2 as defined by the IBM System Network Architecture (SNA) to participate in transactions coordinated by a transaction manager that does not implement SNA protocols.	[MS-DTCLU]
WS-AtomicTransaction Protocol	Enables distributed transaction processing and propagation by using the WS-AtomicTransaction protocol. The system supports both versions 1.0 and 1.1 of the protocol.	[WSAT10], [WSAT11]
WS-AtomicTransaction (WS-AT) Protocol Extensions	Enables external applications to query the system for system-specific transaction propagation information. It also describes how this information can be propagated by extending the WS-AtomicTransaction Protocol.	[MS-WSRVCAT]
MSDTC Connection Manager: OleTx XA Protocol Specification	Enables external transaction managers and external resource managers by using the protocol described on [XOPEN-DTP] to participate in transactions with the system.	[MC-DTCXA]
MSDTC Connection Manager:	Enables multiplexing multiple logical protocol	[MS-CMP]

Protocol name	Protocol purpose	Document short name
OleTx Multiplexing Protocol Specification	connections through a single MSDTC Connection Manager: OleTx Transports Protocol connection, which reduces the number of messages exchanged over the wire.	
MSDTC Connection Manager: OleTx Transports Protocol Specification	Provides negotiation of connections and sending of variable-length data for the MSDTC Connection Manager Protocols.	[MS-CMPO]

2.3 Relevant Standards

The system uses the standards listed below to allow interoperability with other external systems.

Protocol name	Specification reference	System use description
Transaction Internet Protocol (TIP)	[RFC2371]	Allows transaction propagation between the system and TIP transaction managers.
SNA LU Type 6.2 Protocol (LU 6.2)	[LU62Peer], [LU62Verb], [LU62SPS]	Allows resources with LU 6.2 implementation to participate in transactions.
Web Services Atomic Transaction (WS-AtomicTransaction)	[WSAT10], [WSAT11]	Allows distributed transaction processing and transaction propagation with systems implementing WS-AtomicTransaction.
Distributed Transaction Processing: The XA Specification (XA)	[XOPEN-DTP]	Allows distributed transaction processing and transaction propagation with systems implementing XA.

3 Foundation

This section describes the theoretical and practical information needed to understand this document and this system.

3.1 Background Knowledge and System-Specific Concepts

This section summarizes:

- Background knowledge required to understand this document.
- Concepts that are specific to this system.

It is assumed that the reader of this document has the following background knowledge as described in [GRAY]:

- ACID transactions.
- Transaction processing concepts.
- Concept of transaction managers, applications, and resource managers in transaction processing.
- Two-Phase Commit protocol.
- Transaction trees, **root transaction manager**, superior and **subordinate participants**.
- Transaction marshalling, transaction **pull** and **push propagation**, and transaction **recovery**.

The reader should also understand the following system-specific concepts, as described in sections 3.1.3, 3.1.4, and 3.1.5:

- Phase Zero
- Single-Phase Commit
- System Base and System External protocols

3.1.1 Transaction Trees

Multiple transaction managers and resource managers can participate in a transaction. Their individual responsibilities in the Two-Phase Commit protocol are defined by a transaction tree, shown in the following figure.

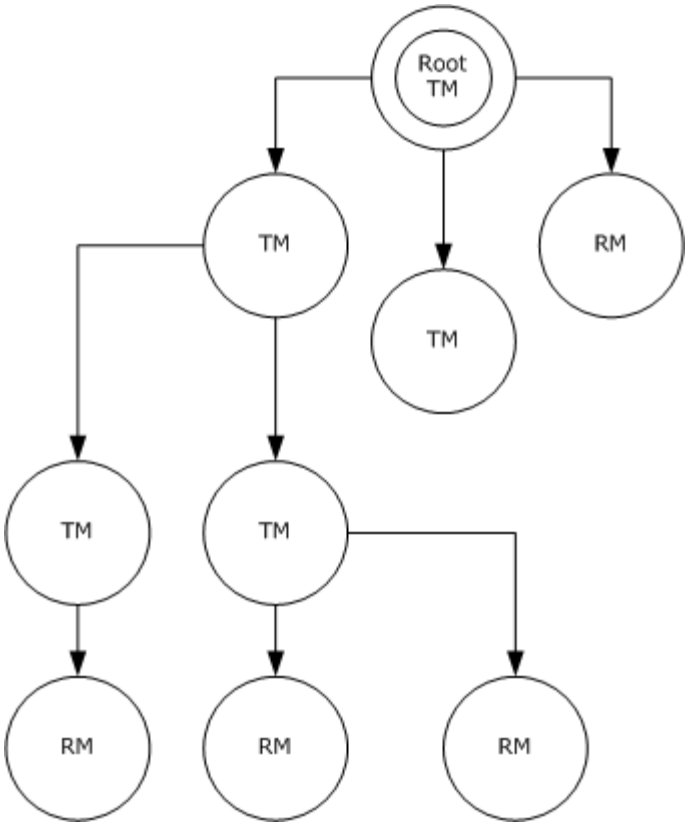

Figure 1: Transaction tree

The transaction manager at the root of the tree is the root transaction manager. This is the transaction manager with which the root application communicates. Any participant that **enlists** with a transaction manager is called a subordinate participant. Each transaction manager is a **superior transaction manager** if any of its subordinate participants are transaction managers. All transaction managers in the tree, apart from the root transaction manager, are **subordinate transaction managers**.

3.1.2 The Two-Phase Commit protocol

The Two-Phase Commit protocol, described in [GRAY], defines Phase One and Phase Two. These phases can be described informally as "are you ready" and "go / no go," respectively.

Phase One begins when all the required changes of resource state have been made, and the root application asks the transaction manager to complete it. Phase One ends when the transaction manager has decided the **outcome** of the transaction; that is, whether all the changes are to be made or whether none of them are to be made.

When the root application asks the root transaction manager to complete the transaction, it may make a **commit request**, asking that all the changes are to be made, or an **abort request**, asking that none of the changes are to be made. A commit request causes the root transaction manager to ask each of the subordinate participants involved in the transaction whether they are prepared to commit the changes made. This is called voting on the transaction outcome. Each subordinate participant must vote Read-Only, Prepared, or Aborted. Read-Only and Prepared are positive votes. Aborted is a negative vote. If any subordinate participant votes negatively, the root transaction

manager decides that the final outcome of the transaction as a whole is to make no changes. This is called the **abort outcome**. If every subordinate participant votes positively, then the root transaction manager decides that the final outcome of the transaction as a whole is to make all the changes. This is called the **commit outcome**. An abort request causes the root transaction manager to notify each subordinate participant to make no changes and to notify each of their respective subordinate participants, if any, to abort the transaction.

A subordinate transaction manager determines its vote by aggregating the votes of its subordinate participants. If a subordinate transaction manager has no subordinate participants, or if all of its subordinate participants vote Read-Only, then the subordinate transaction manager votes Read-Only. If at least one subordinate participant votes Prepared, and after collecting all votes no subordinate participant votes Aborted, then the subordinate transaction manager votes Prepared. In all other cases, the subordinate transaction manager votes Aborted, in which case it must also notify any of its subordinate participants that had voted Prepared that the transaction has aborted.

Until a participant votes on the outcome of a transaction, that participant can decide to unilaterally abort the transaction by issuing an abort request to its transaction manager. This is called the unilateral abort. Further details of unilateral abort are described in [MS-DTCO] section 1.3.2.1.

Phase Two begins after the root transaction manager decides the outcome of the transaction. In this phase, each subordinate participant that voted Prepared is sent either a request to commit the changes if the outcome was the commit outcome or a request to undo the changes (called a **rollback**) if the outcome was the abort outcome. The root transaction manager also sends the outcome of the transaction to the root application. A subordinate participant that voted Read-Only is not notified of the outcome of the transaction — for example, a resource manager might vote Read-Only if it made no changes as part of the transaction. A subordinate participant that voted Abort is also not notified of outcome of the transaction. Phase Two ends after the root transaction manager tells participants what the outcome is (commit or abort), and participants have notified the transaction manager that the operation is successfully completed.

This is the Two-Phase Commit protocol described in [GRAY]. The protocol specified by [MS-DTCO] adds an additional phase, Phase Zero. Phase Zero can be thought of as expanding the beginning of Phase One. It begins when the root application requests completion of the transaction, and it ends when all Phase Zero participants have voted that the phase is complete, after which Phase One proceeds as described above. The value of the additional phase is that, during Phase Zero, new participants can be enlisted on the transaction.

In the protocol described in [GRAY], the set of participants is fixed from the moment that Phase One begins. Phase Zero is a very useful extension in some scenarios. For example, a caching resource manager can be placed between an application and a database resource manager so that all requested changes are held in memory until the caching resource manager receives a request from the transaction manager to exit Phase Zero. Only then is the database resource manager itself enlisted on the transaction and are the changes made to the durable store, yielding potentially significant performance gains. Further details of Phase Zero are described in [MS-DTCO] section 1.3.1.1.

As an extension to the protocol described in [GRAY], the protocol described by [MS-DTCO] specifies a mechanism called Single-Phase Commit optimization. This optimization is performed when the root transaction manager has only one subordinate transaction manager. In this case, instead of Phase One, the root transaction manager sends a request to the subordinate transaction manager to perform Single-Phase Commit. If the subordinate transaction manager supports this operation, the responsibility to drive the outcome of the transaction is delegated to the subordinate transaction manager. When the outcome is determined, the subordinate transaction manager notifies the root transaction manager with the result. If the subordinate transaction manager does not support the Single-Phase Commit optimization, it rejects the initial request, and the root transaction manager resumes the normal Two-Phase Commit protocol. This is a useful optimization for reducing the total

time spent on driving the outcome of a transaction, especially in cases where the root transaction manager and the subordinate transaction manager are on separate computer systems on a network. Further details of Single-Phase Commit are described in [MS-DTCO] section 1.3.2.2.

3.1.3 Phase Zero

The system extends the Two-Phase Commit protocol by adding an additional phase, Phase Zero. Phase Zero can be thought of as expanding the beginning of Phase One. It begins when the root application requests completion of the transaction, and it ends when all Phase Zero participants have voted that the phase is complete, after which Phase One proceeds, as described above. The value of the additional phase is that during Phase Zero, new participants can be enlisted on the transaction.

In the **two-phase commit** protocol described in [GRAY], the set of participants is fixed from the moment that Phase One begins. Phase Zero is a very useful extension in some scenarios. For example, a caching resource manager can be placed between an application and a database resource manager so that all requested changes are held in memory until the caching resource manager receives a request from the transaction manager to exit Phase Zero. Only then is the database resource manager itself enlisted on the transaction and are the changes made to the durable store, yielding potentially significant performance gains. Further details of Phase Zero are described in [MS-DTCO] section 1.3.1.1.

3.1.4 Single-Phase Commit

As an extension to the Two-Phase Commit protocol, the system uses a mechanism called Single-Phase Commit optimization. Single-Phase Commit optimization is described in detail in [MS-DTCO]. This optimization is performed when the root transaction manager has only one subordinate transaction manager. In this case, instead of Phase One, the root transaction manager sends a request to the subordinate transaction manager to perform Single-Phase Commit. If the subordinate transaction manager supports this operation, then the root transaction manager gives the responsibility to coordinate the outcome of the transaction to the subordinate transaction manager. When the outcome is determined, the subordinate transaction manager notifies the root transaction manager with the result. If the subordinate transaction manager does not support the Single-Phase Commit optimization, it rejects the initial request, and the root transaction manager resumes the normal Two-Phase Commit protocol. Single-Phase Commit is a useful optimization in cases where the root transaction manager and the subordinate transaction manager are on separate networks.

3.1.5 System Base and System External Protocols

The system supports a set of System Base protocols and System External protocols, as described in section 5.3.2, to facilitate transaction coordination. System Base protocols are proprietary to the system and are used by default by applications, application services, and resource managers. System External protocols are intended to allow interoperability through transaction processing industry standards. Relevant industry standards are listed in section 2.3. Applications, application services, resource managers, and transaction managers communicating with the system over System External protocols are referred to as external applications, external application services, external resource managers, and external transaction managers. The system allows the possibility of processing a transaction using only a single System Base or System External protocol, or a combination of System Base and System External protocols.

3.2 System Purposes

The Transaction Processing Services System provides distributed transaction coordination services for applications, application services, resource managers, external applications, external application services, external resource managers, and external transaction managers. The system is also used

by clients that configure and manage the system. The following figure illustrates the Transaction Processing Services System and the components it interacts with.

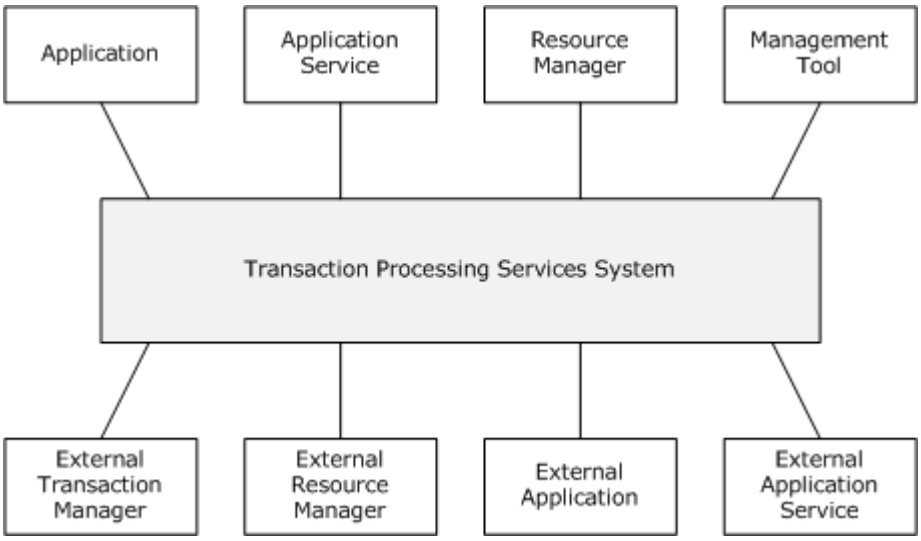

Figure 2: Components interacting with the Transaction Processing Services System

The purpose of the Transaction Processing Services System is to:

- Use the Two-Phase Commit protocol, described in [GRAY] and [MS-DTCO] 1.3.1 to coordinate the transaction participants.
- Enable applications, resource managers, and transaction managers distributed over networked computer systems to participate in a single transaction.
- Enable participating transaction managers and resource managers to recover from local failures by reestablishing a state consistent with that of the other participants in a distributed transaction. This is referred to as transaction recovery, and it is described in [MS-DTCO] section 1.3.4.
- Enable the participation of external transaction managers in coordinating a transaction.

Applications, application services, resource managers, external applications, external application services, external resource managers, and external transaction managers use a set of system interfaces to participate in a distributed transaction, and perform some transaction processing-specific operations, such as transaction marshalling, propagation, and recovery.

Applications and external applications use the system for the following purposes:

- Demarcating when a transaction begins and completes within a series of operations.
- Marshalling the transaction to other applications and resource managers.
- Propagating the transaction from one transaction manager to another.
- Performing administrative operations against a specific transaction or a transaction manager.

Resource managers and external resource managers use the system for the following purposes:

- Registering with a transaction manager and performing recovery operations.

- Enlisting for a specific transaction and participating in the corresponding Two-Phase Commit protocol notifications.
- Voting on transaction outcomes.

External transaction managers use the system for the following purposes:

- Enlisting with the system as a superior or subordinate transaction manager for a specific transaction.
- Participating in Two-Phase Commit protocol notifications.
- Coordinating recovery operations.

The system can also be used by applications or other systems to provide transaction coordination semantics to higher-level applications. For example, application programming frameworks, such as the .NET Framework, or middle-tier application server products, such as COM+ Services, provide transaction processing services to their clients by providing some set of high-level interfaces, but in the background can use the Transaction Processing Services System described in this document to fulfill the required transaction coordination semantics. This way, the complexity of interacting with the Transaction Processing Services System can be minimized.

Other systems that need transaction processing services can also use the system interfaces. For example, a message queuing system can use transaction processing to make sure that operations on two queues are both completed or that neither succeeded.

3.3 System Use Cases

3.3.1 Stakeholders and Interests Summary

The stakeholders in the Transaction Processing Services system are architects, developers, testers, and IT operations personnel.

3.3.1.1 Architects

An architect is responsible for the overall design of a system while managing the technical risks associated with it.

Many systems need to maintain consistency across a set of (possibly distributed) applications and resources or to give multiple applications concurrent access to a shared resource. Transactions provide an architect with an explicitly clear set of tools to solve the problems inherent in these requirements. Without the use of transactions, such a system would need to implement a complex and proprietary set of isolation and error-handling semantics. If the system involves distributed resources, the error and management scenarios become more complicated. Use of transactions and the Two-Phase Commit protocol reduces the design complexity for such systems by providing a standard and proven solution to the problems.

An architect can use the Transaction Processing Services System as an element of a system in the design process to provide proven, reusable support for distributed transactions.

The system also offers an architect the following additional advantages:

- The Transaction Processing Services System is based on the Two-Phase Commit protocol that has been implemented by many vendors. This allows the system to be extended to interact with other implementations of the Two-Phase Commit protocol, such as applications, resource

managers or other transaction managers. Several such extensions are provided as part of the system, and further extensions can be added.

- The Transaction Processing Services System focuses exclusively on providing support for transaction coordination. The system provides the necessary interfaces to allow any application or resource manager to participate in transactions. This contrasts with other systems where the transaction functionality is built directly into the resource, and therefore is usable only by applications that directly use the resource—inhibiting, for instance, the use of such a system in complex distributed scenarios.
- The Transaction Processing Services System factors out support for transactions, and it imposes a standard programming model for accessing this support, substantially simplifying the view that developers and other stakeholders have of transactions. This reduces the amount and complexity of the code that must be written, and therefore, in turn, mitigates the risks associated with development, testing, and support.

3.3.1.2 Developers

Developers implement the design. In order to build the correct implementation, developers need to understand the required functionality and the expected preconditions and post conditions.

Two kinds of developers build systems involving transaction processing. An application developer builds applications that update resources in a transaction. A resource manager developer builds resource managers that provide resources to applications in a transactional context.

For both kinds of developers, using the Transaction Processing Services System reduces the learning required and simplifies the code that they need to implement. This enables the developers to focus on the rest of the functional requirements. A developer who uses the Transaction Processing Services System also works with well-defined guidelines and goals, greatly increasing the chances of successful implementation.

Application developers who use the Transaction Processing Services System can safely assume that the transacted updates their application performs on a resource are not visible to other applications accessing the same resource until after the transaction is completed. Use of the Transaction Processing Services System also simplifies error handling because either all of the updates are completed or none of them are completed. No other states need to be considered.

Resource manager developers who use the Transaction Processing Services System can build a resource manager for an existing resource. This allows applications to use the resource as part of a transaction with all transactional guarantees, such as isolation, durability, and all-or-nothing semantics. It also allows the resource to be used in a wide variety of transactional scenarios.

Both kinds of developers will, however, need to understand the transactional model as it affects their code, and in some scenarios, the developer will need to control the flow of transactions; for example, when a transaction needs to be propagated from one application to another. This cost, however, is small compared to the cost of the large amount of complex code that would otherwise be necessary in order to achieve isolation, manage the state, and handle various error cases.

3.3.1.3 Testers and Quality Assurance (QA) Personnel

Testers and QA personnel are responsible for testing the implementation to determine whether it meets the design and functionality requirements.

Use of the Transaction Processing Services System standardizes communications between applications, resource managers, and transaction managers. It also defines the possible use case scenarios and the results to be expected in each scenario. This greatly reduces the amount of time

and risk associated with building test cases and analyzing test results. Also, the functionality delivered by the Transaction Processing Services System -- for instance, isolation and error handling -- does not need to be tested because it is a trusted system.

Use of the Transaction Processing Services System also makes it easier to measure test results. The system provides tracing features that can be used to monitor the progress of test cases. Because test programs written against the Transaction Processing Services System use a standard interface, they may be reused when new resource managers or applications are added to the system, or when new systems are being tested.

3.3.1.4 IT Operations Personnel

IT operations personnel are responsible for deploying, configuring, and monitoring a system.

The Transaction Processing Services System provides standard configuration options and the ability to perform these functions through a remote management tool. For example, the IT operations personnel can choose which computers are allowed to participate in transactions, easily enable or disable various features in the Transaction Processing Services System, and configure security settings so that communications are established over secure channels.

The Transaction Processing Services System also allows a management tool to monitor the health of the system at a per-transaction or system level. IT operations personnel may intervene, for instance, by resolving transaction states or by changing the **log** settings of a transaction manager. Another advantage of using the Transaction Processing Services System is that these interventions have well-defined characteristics so that IT operations personnel know what final results are possible.

3.3.2 Supporting Actors and System Interests Summary

The system does not have any supporting actors.

3.3.3 Use Cases

The Transaction Processing Services System provides support for distributed transactions. Distributed transactions are generally required in scenarios where a number of applications and resource managers cooperate to perform a set of related work items that require the ACID properties (atomic, consistent, isolated, and durable) of a distributed transaction, as described in [GRAY]. These properties are needed in order to make changes to persistent state in a deterministic, correct, and highly reliable manner.

As described in section 3.2, the Transaction Processing Services System provides distributed transaction coordination services for applications, application services, resource managers, external applications, external application services, external resource managers, and external transaction managers. Additionally, a management tool can be used to monitor and configure various settings in the system. This section describes a set of use cases that span the functionality of the Transaction Processing Services System.

The following table provides an overview for the groups of use cases which span the functionality of the Transaction Processing Services System. The sections which follow provide detailed descriptions of the use cases in each group.

Use case group	Use cases
Creating a transaction	Create a Transaction - Application Create a Transaction – External Application

Use case group	Use cases
Doing transacted work	Do Transacted Work – Application Do Transacted Work – External Application
Doing remote transacted work with pull propagation	Do Remote Transacted Work with Pull Propagation – Application Do Remote Transacted Work with Pull Propagation – External Application
Doing remote transacted work with push propagation	Do Remote Transacted Work with Push Propagation – External Application
Pulling a transaction	Pull Transaction – Transaction Manager Pull Transaction – External Transaction Manager
Pushing a transaction	Push Transaction – Transaction Manager
Enlisting in a transaction	Enlist in a Transaction – Resource Manager Enlist in a Transaction – External Resource Manager
Completing a transaction	Complete a Transaction – Application Complete a Transaction – External Application Drive Completion of a Transaction – Transaction Manager
Recovering transactions	Do Transaction Recovery – Resource Manager Do Transaction Recovery – External Resource Manager Do Transaction Recovery – Transaction Manager Do Transaction Recovery – External Transaction Manager
Managing transactions	Manage Transaction Managers – Management Tool Manage Transactions – Management Tool

The following figure provides an overview of Transaction Processing Services System use cases.

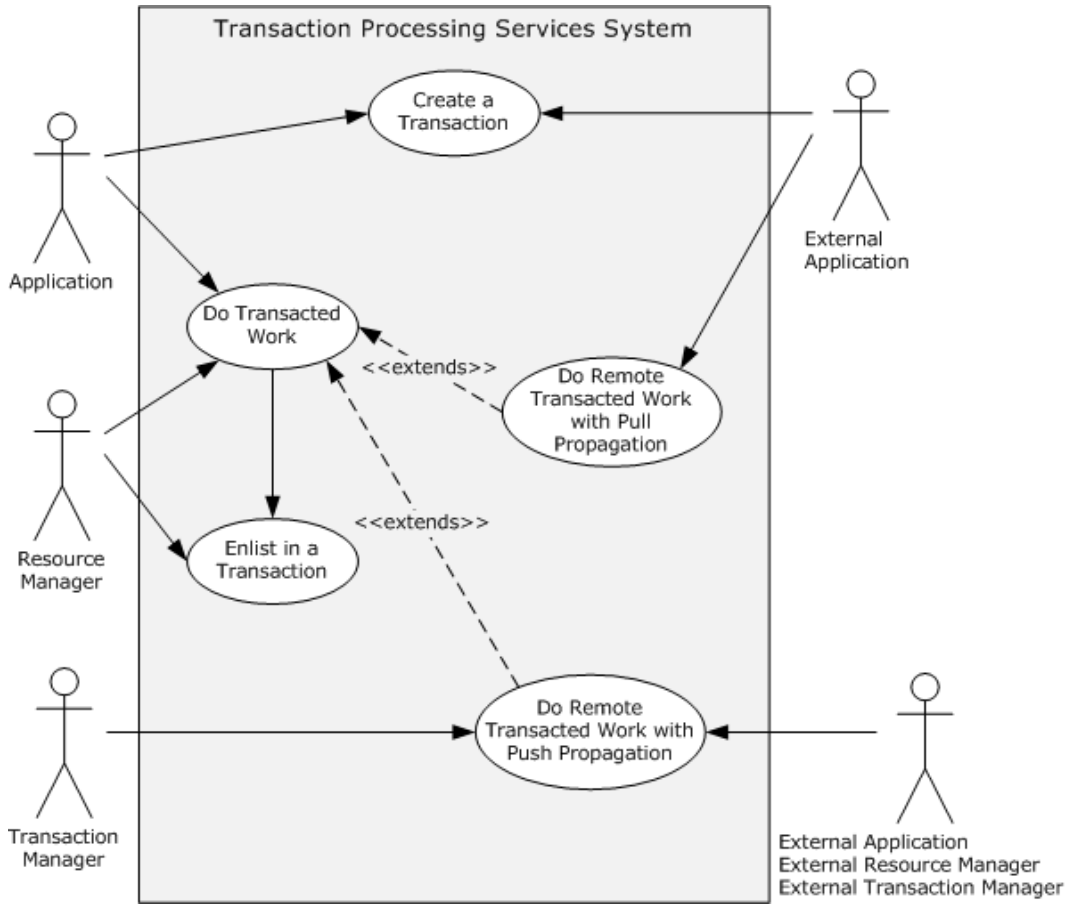

Figure 3: Creating a transaction and doing transacted work

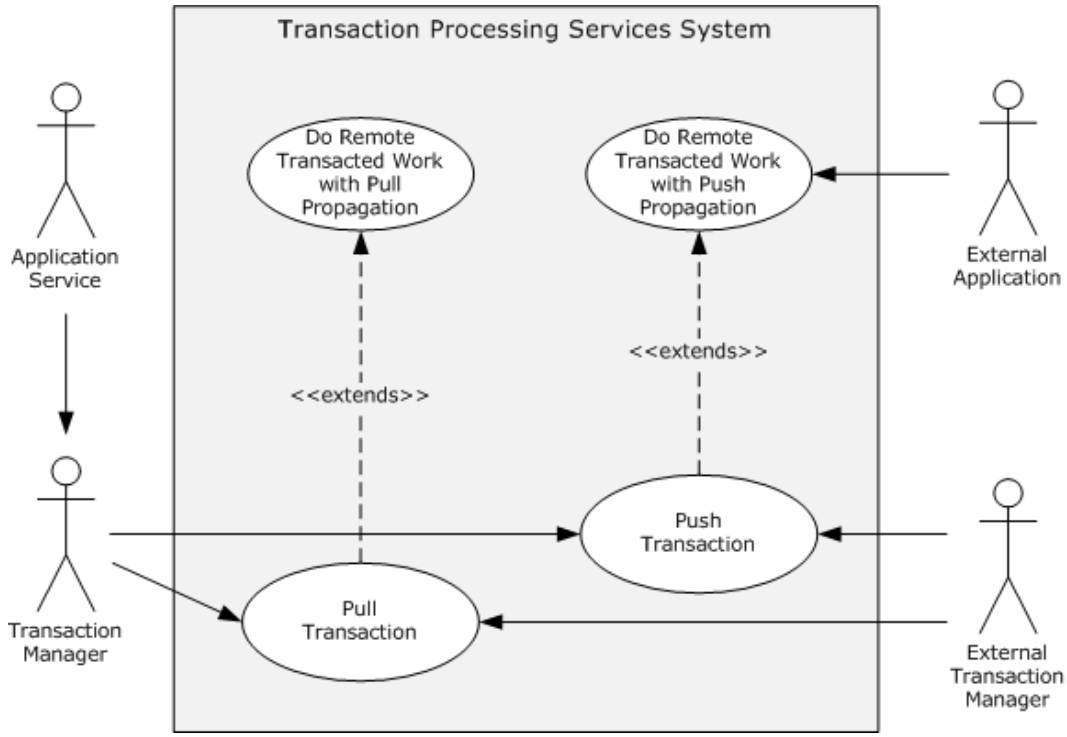

Figure 4: Pulling and pushing transactions

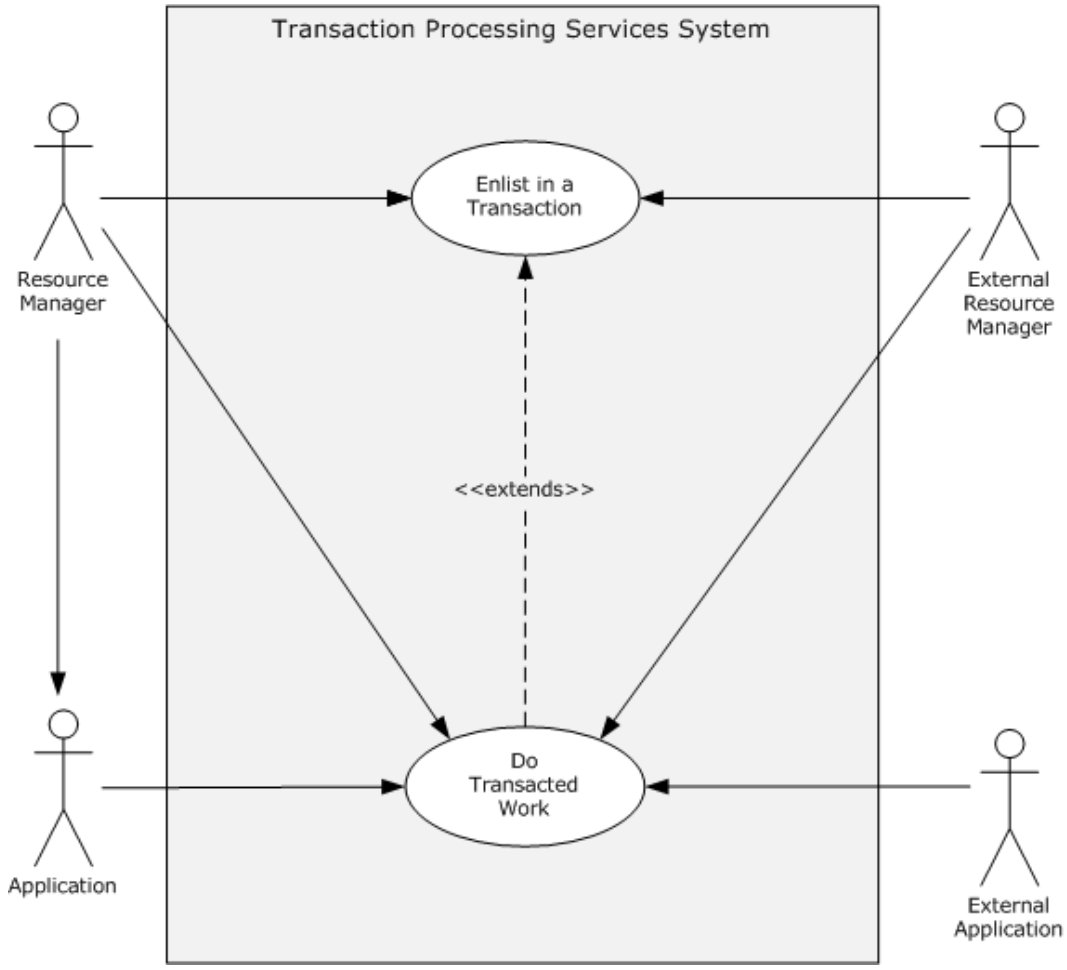

Figure 5: Enlisting in a transaction

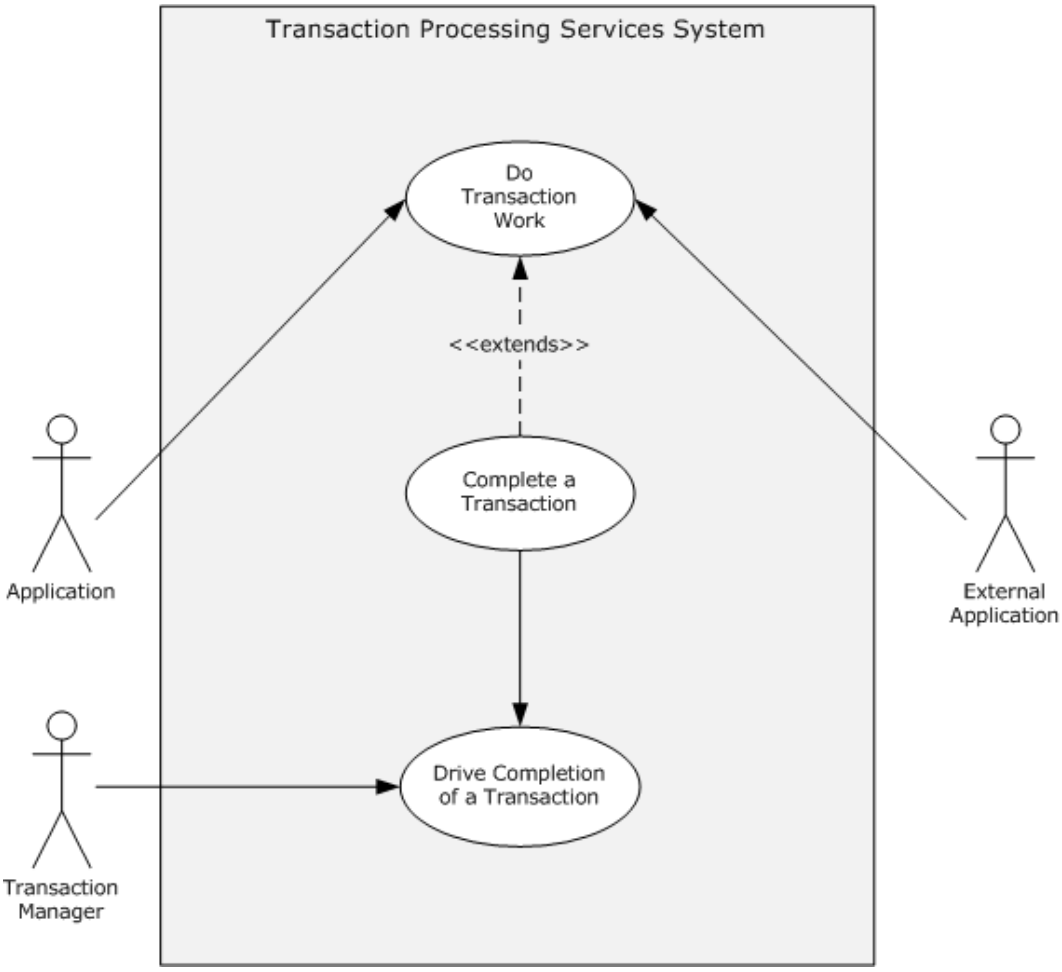

Figure 6: Completing a transaction

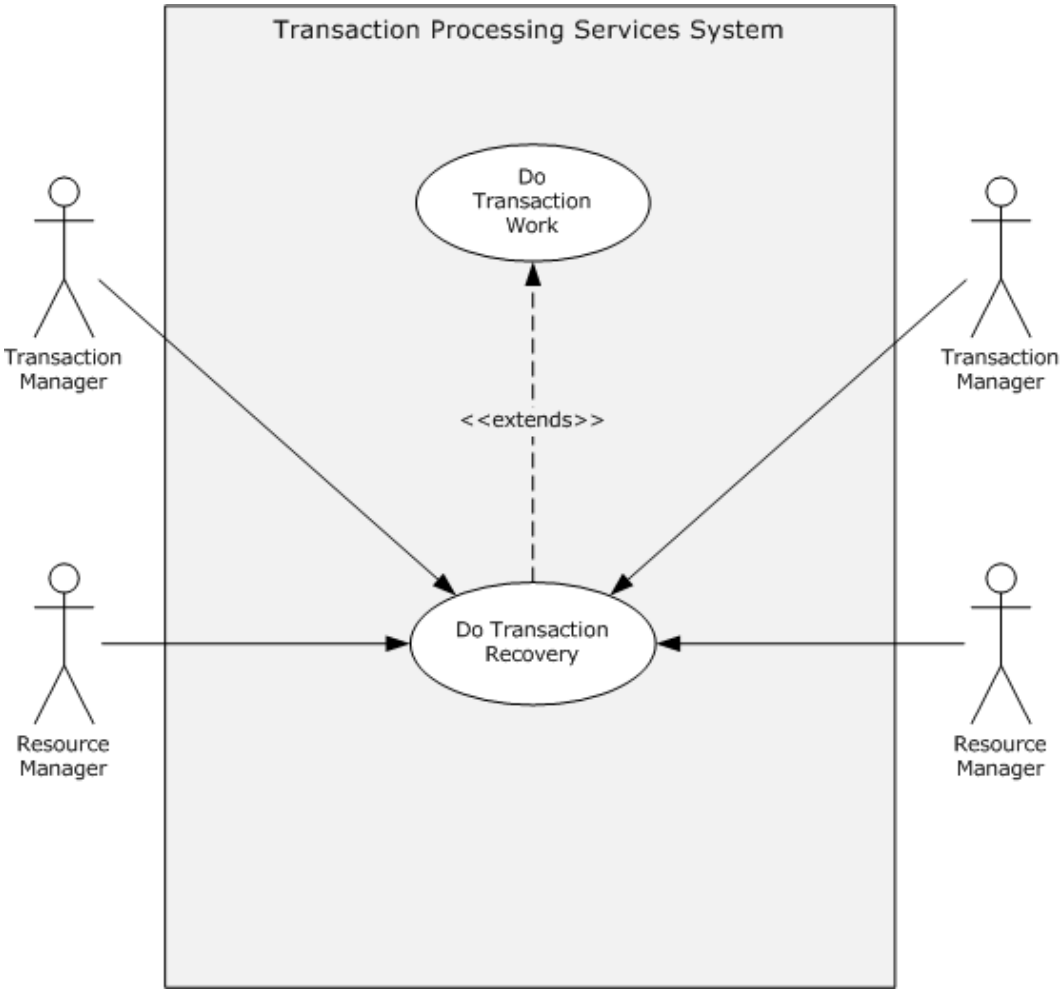

Figure 7: Recovering a transaction

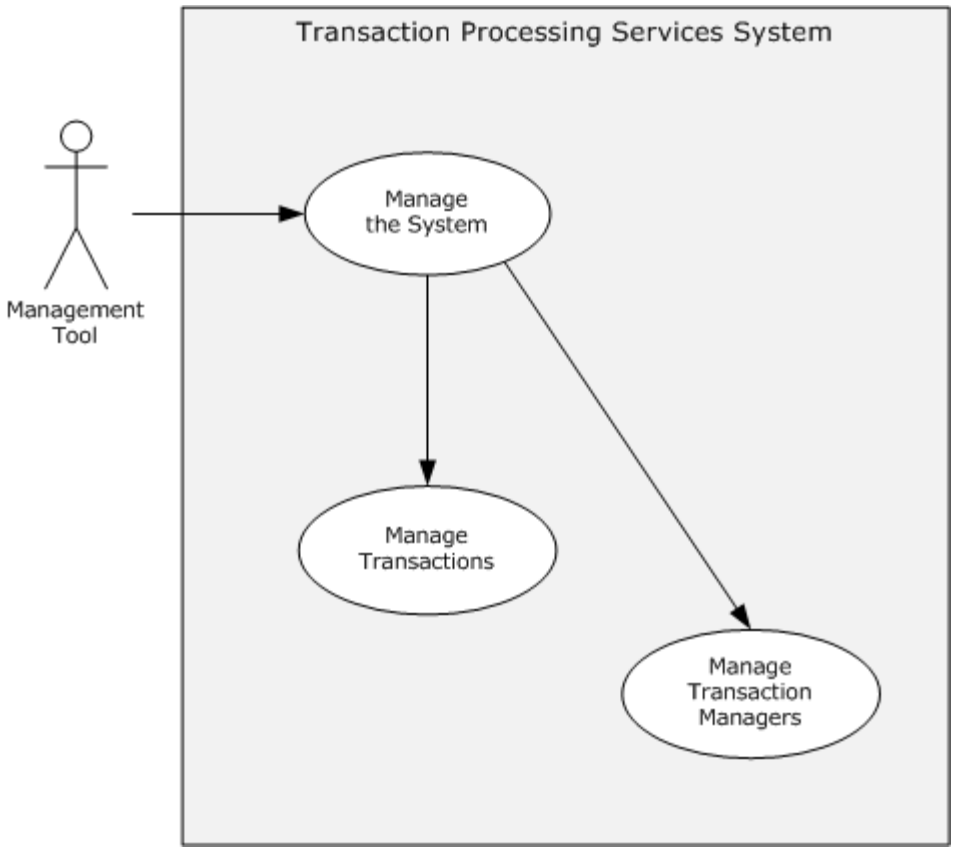

Figure 8: Managing transactions

The following sections provide detailed descriptions of the use cases in each group.

3.3.3.1 Create a Transaction – Application

Goal: The goal of the use case is to start a new transaction with a transaction manager in the system.

Context of use: This use case is initiated before doing any transacted work. The Direct Actor of the use case uses the transaction information obtained in this use case in consequent use cases where the transaction information is directly used while doing transacted work.

Direct Actor: The direct actor of this use case is an application.

Primary Actor: Same as direct actor.

Supporting Actors: None.

Stakeholders and Interests:

- Architects, as described in 3.3.1.1, design system architectures while assuming that this use case creates a new transaction with a transaction manager in the system. If the transaction is created, it is guaranteed that the transaction will not be disposed by the system without the application asking for it, or without administrative intervention. It is also guaranteed that the transaction is unique and belongs only to the application that has created it.

- Developers and testers, as described in 3.3.1.2 and 3.3.1.3, expect that when this use case is executed by an application, the use case successfully creates a new transaction, or if the creation of the transaction has failed, the failure will be communicated back to the application.

Preconditions:

- The Transaction Processing Services System is operational.
- The application can access a transaction manager in the system.

Minimal Guarantees: None.

Success Guarantee:

- If the creation of the transaction is successful, then a success message will be returned to the application.
- If the creation of the transaction is successful, then the transaction manager will dispose the transaction only if the application completes the transaction or an administrator intervenes.
- If the creation of the transaction is successful, then the transaction manager will guarantee transaction and application affinity for the lifetime of the system.

Trigger:

- The application triggers this use case.

Main Success Scenario:

1. The application asks the transaction manager to create a transaction.
2. The transaction manager creates a new transaction.
3. The transaction manager returns a reference to the transaction to the application.

Extensions: None.

3.3.3.2 Create a Transaction – External Application

This use case is the same use case as described in 3.3.3.1, with External Application as the direct actor.

3.3.3.3 Do Transacted Work – Application

Goal: The goal of the use case is perform a set of operations within an atomic transaction.

Context of use: This use case is initiated when the Create a Transaction use case (section 3.3.3.1) is successfully completed. As part of this use case the direct actor makes some updates on a resource within a transaction, using the transaction information obtained in the Create a Transaction use case (section 3.3.3.1).

Direct Actor: The direct actor of this use case is an application.

Primary Actor: Same as direct actor.

Supporting Actors: The resource manager, as described in the Enlist in a Transaction use case (section 3.3.3.11) is a supporting actor for this use case. Enlist in a Transaction use case is executed before successfully completing this use case.

Stakeholders and Interests:

- Architects, as described in section 3.3.1.1, design system architectures with this use case while assuming that all updates taking place in the transaction will be executed in accordance with the Two-Phase Commit and ACID semantics of transaction processing. If there are transient failures before the use case is completed, then the updates that took place in the transaction will appear as if none of them have happened.
- Developers and testers, as described in sections 3.3.1.2 and 3.3.1.3, assume that if the use case has not completed successfully, then all updates within the transaction appear as if none of them have happened. Developers and testers also assume that before this use case completes successfully, none of the updates within the transaction will be visible to other applications or systems.

Preconditions:

- The Transaction Processing Services System is operational.
- The application can access the system.

Minimal Guarantees:

- If the use case did not complete successfully, then the resource manager guarantees that all of the updates that took place in the transaction will appear as if they never happened.

Success Guarantee:

- If the use case is completed successfully, then the resource manager guarantees that the updates that took place in the use case will be executed in accordance with the Two-Phase Commit protocol semantics.

Trigger:

- The application triggers this use case after successfully completing the Create a Transaction use case (section 3.3.3.1).

Main Success Scenario:

1. The application asks the resource manager to make updates in its resource in the context of the transaction that was created in the Create a Transaction use case (section 3.3.3.1).
2. The resource manager initiates an Enlist in a Transaction use case (section 3.3.3.11).
3. After successful completion of the Enlist in a Transaction use case (section 3.3.3.11), the resource manager makes the requested updates to its resource in accordance with the semantics of the Two-Phase Commit protocol, such as isolation and durability.

Extensions: Do Remote Transacted Work with Pull Propagation use case (section 3.3.3.5) and Do Remote Transacted Work with Push Propagation use case (section 3.3.3.7) are extensions to this use case.

3.3.3.4 Do Transacted Work – External Application

This use case is the same use case as described in 3.3.3.3, with External Application as the direct actor.

3.3.3.5 Do Remote Transacted Work with Pull Propagation – Application

Goal: The goal of the use case is perform a set of operations in an atomic transaction on a remote resource, where the remote resource has a separate transaction manager. The transaction information is communicated to the remote transaction manager by using pull propagation.

Context of use: This use case is initiated when the Create a Transaction use case (section 3.3.3.1) is successfully completed. The transacted work that is performed in this use case uses the transaction reference obtained in the Create a Transaction use case (section 3.3.3.1).

Direct Actor: The direct actor of this use case is an application.

Primary Actor: Same as direct actor.

Supporting Actors: Enlist in a Transaction (section 3.3.3.11) and Pull Transaction (section 3.3.3.8) use cases are supporting actors for this use case.

Stakeholders and Interests: Same as Do Transacted Work use case (section 3.3.3.3).

Preconditions:

- The Transaction Processing Services System is operational.
- The application can access a transaction manager in the system.
- The resource manager of the resource and the application service are on a remote computer to the application and can access a transaction manager in the system.
- The two computers involved are networked together.
- The transaction managers in the system on each of the computers are operational.

Minimal Guarantees: Same as Do Transacted Work use case (section 3.3.3.3).

Success Guarantee: Same as Do Transacted Work use case (section 3.3.3.3).

Trigger: Same as Do Transacted Work use case (section 3.3.3.3).

Main Success Scenario:

1. Application sends the transaction reference received during the Create a Transaction use case (section 3.3.3.1), along with information about the work to be done to the application service.
2. Upon receiving the information about the transaction reference and the work to be done, the application service asks its transaction manager to pull the transaction, passing the transaction reference provided by the application.
3. The transaction manager pulls the transaction by executing the Pull Transaction use case (section 3.3.3.8).
4. The application service passes the information about the work to be done to the resource manager along with a reference to the transaction.
5. The resource manager executes the Enlist in a Transaction use case (section 3.3.3.11), asking the transaction manager to enlist it in the transaction.
6. The resource manager makes the requested updates to the resource in accordance with the Two-Phase Commit protocol semantics, such as isolation and durability.

7. The resource managers reports success to the application service, which in return, application service reports success to the application.

Extensions: None.

3.3.3.6 Do Remote Transacted Work with Pull Propagation – External Application

This use case is the same use case described in 3.3.3.5 with External Application as the direct actor.

3.3.3.7 Do Remote Transacted Work with Push Propagation – External Application

Goal: The goal of the use case is to perform a set of operations in an atomic transaction on a remote resource, where the remote resource has a separate transaction manager. The transaction information is communicated to the remote transaction manager by using push propagation.

Context of use: This use case is initiated when the Create a Transaction use case (section 3.3.3.1) is successfully completed. The transacted work that is performed in this use case uses the transaction reference obtained in the Create a Transaction use case (section 3.3.3.1).

Direct Actor: The direct actor of this use case is an external application.

Primary Actor: Same as Direct Actor.

Supporting Actors: Enlist in a Transaction (3.3.3.11) and Push Transaction (section 3.3.3.10) use cases are supporting actors for this use case.

Stakeholders and Interests: Same as Do Transacted Work use case (section 3.3.3.3).

Preconditions:

- The Transaction Processing Services System is operational.
- The external application and the external transaction manager can both access a transaction manager in the system.
- The external application and the external transaction manager are both on separate computers.
- The two computers involved are networked together.

Minimal Guarantees: Same as Do Transacted Work use case (section 3.3.3.3).

Success Guarantee: Same as Do Transacted Work use case (section 3.3.3.3).

Trigger: Same as Do Transacted Work use case (section 3.3.3.3).

Main Success Scenario:

1. The external application asks the external resource manager for the location information of the external transaction manager.
2. The external application asks the transaction manager to push the transaction to the external transaction manager.
3. The transaction manager initiates a Push Transaction use case (section 3.3.3.10) to push the transaction to the external transaction manager. As a result of this action, the external transaction manager is enlisted in the transaction.

4. The external application asks the external resource manager to make updates in the context of the transaction.

5. The external resource manager makes the requested updates to its resource in accordance with the Two-Phase Commit protocol semantics, such as isolation and durability.

6. The external resource manager reports success to the external application.

Extensions: None.

3.3.3.8 Pull Transaction – Transaction Manager

Goal: The goal of the use case is to pull the information about a transaction from a remote transaction manager.

Context of use: During the Do Remote Transacted Work with Pull Propagation use case (section 3.3.3.5), the application service asks the transaction manager to pull the transaction information from a remote transaction manager. When the transaction manager successfully receives the transaction information from the remote transaction manager, it is enlisted in the transaction.

Direct Actor: The direct actor of this use case is either a transaction manager or and external transaction manager, which will be referred to as the transaction manager in the remainder of this section.

Primary Actor: The primary actor for this use case is the Do Remote Transacted Work with Pull Transaction use case (section 3.3.3.5).

Supporting Actors: None.

Stakeholders and Interests:

- Architects, as described in section 3.3.1.1, design system architectures with this use case while assuming that upon successful completion of the use case, the transaction manager pulling the transaction is successfully enlisted in the transaction and it becomes the subordinate transaction manager of the remote transaction manager.
- Developers and testers, as described in sections 3.3.1.2 and 3.3.1.3, expect that upon completion of this use case, the transaction manager is enlisted in the transaction and it will be participating in the coordination of the transaction.

Preconditions:

- The Transaction Processing Services System is operational.
- The two transaction managers are on separate computers and can access each other.
- The two computers involved are networked together.

Minimal Guarantees: None.

Success Guarantee:

- If the use case is completed successfully, then the transaction manager pulling the transaction is enlisted in the transaction.

Trigger:

- The Do Remote Transacted Work with Pull Propagation use case (section 3.3.3.5) triggers this use case.

Main Success Scenario:

1. The transaction manager sends a transaction reference to the remote transaction manager asking to pull the transaction.
2. The remote transaction manager enlists the transaction manager in the transaction, and returns success.

Extensions: None.

3.3.3.9 Pull Transaction – External Transaction Manager

This use case is the same use case as described in 3.3.3.8, with External Transaction Manager as the direct actor.

3.3.3.10 Push Transaction – Transaction Manager

Goal: The goal of the use case is to push the information about a transaction to a remote transaction manager.

Context of use: During the Do Remote Transacted Work with Push Propagation use case (section 3.3.3.7), the external application asks the transaction manager to push the transaction information to a remote transaction manager. When the remote transaction manager successfully receives the transaction information, it is enlisted in the transaction.

Direct Actor: The direct actor of this use case is a transaction manager.

Primary Actor: The primary actor for this use case is the Do Remote Transacted Work with Push Transaction use case (section 3.3.3.10).

Supporting Actors: None.

Stakeholders and Interests:

- Architects, as described in section 3.3.1.1, design system architectures with this use case while assuming that upon successful completion of the use case, the transaction manager pulling the transaction is successfully enlisted in the transaction and it becomes the subordinate transaction manager of the remote transaction manager.
- Developers and testers, as described in sections 3.3.1.2 and 3.3.1.3, expect that upon completion of this use case, the transaction manager is enlisted in the transaction and it will be participating in the coordination of the transaction.

Preconditions:

- The Transaction Processing Services System is operational.
- The transaction manager and the external transaction manager are on separate computers.
- The two computers involved are networked together.

Minimal Guarantees: None.

Success Guarantee:

- If the use case is completed successfully, then the transaction manager pulling the transaction is enlisted in the transaction.

Trigger:

- The Do Remote Transacted Work with Pull Propagation use case (section 3.3.3.5) triggers this use case.

Main Success Scenario:

1. The transaction manager sends a transaction reference to the remote transaction manager asking to pull the transaction.
2. The remote transaction manager enlists the transaction manager in the transaction, and returns success to the transaction manager.

Extensions: None.

3.3.3.11 Enlist in a Transaction – Resource Manager

Goal: The goal of the use case is enlist a resource manager in a particular transaction. When it is enlisted, the resource manager will be able to participate in the coordination of the transaction.

Context of use: This use case is initiated when an application or an external application performs the Do Transacted Work use case (section 3.3.3.3). The use case allows the resource manager in the Do Transacted Work use case (section 3.3.3.3) to enlist in the transaction. Before doing any transacted work as part of a specific transaction, a resource manager must first enlist in that transaction, with the transaction manager coordinating the transaction.

Direct Actor: The direct actor of this use case is a resource manager.

Primary Actor: The primary actor of this use case is the Do Transacted Work use case (section 3.3.3.3).

Supporting Actors: None.

Stakeholders and Interests:

- Architects, as described in section 3.3.1.1, design system architectures with this use case while assuming that upon successful completion of the use case, the resource manager is successfully enlisted in the transaction.
- Developers and testers, as described in sections 3.3.1.2 and 3.3.1.3, expect that upon completion of this use case, the resource manager is enlisted in the transaction and it will be participating in the transaction.

Preconditions:

- The Transaction Processing Services System is operational.
- The resource manager can access the transaction manager where it needs to contact to enlist in the transaction.

Minimal Guarantees: None.

Success Guarantee:

- If the use case completes successfully, then the transaction manager guarantees that it will include the resource manager in the Two-Phase Commit protocol notifications related to the transaction on which the resource manager has enlisted.
- If the use case completes successfully, then the transaction manager guarantees that it will keep a durable copy of the state of the transaction for the lifetime of the transaction or the transaction manager, whichever is less. This state information may be used by the resource manager for recovery purposes.

Trigger:

- The Do Transacted Work use case (section 3.3.3.3) triggers this use case.

Main Success Scenario:

1. The resource manager asks the transaction manager to enlist in the transaction.
2. The transaction manager enlists the resource manager in the transaction and returns a success message to the resource manager.

Extensions: None.

3.3.3.12 Enlist in a Transaction – External Resource Manager

This use case is the same use case as described in 3.3.3.11, with External Resource Manager as the direct actor.

3.3.3.13 Complete a Transaction – Application

Goal: The goal of the use case is to complete a transaction with an abort or a commit outcome.

Context of use: This use case is initiated after completing the Do Transacted Work use case (section 3.3.3.3) to either commit or abort the transaction. This use case triggers the transaction manager to drive the completion of the transaction on all transaction participants.

Direct Actor: The direct actor of this use case is either an application or an external application, which will be referred to as the application in the remainder of this section.

Primary Actor: Same as Direct Actor.

Supporting Actors: The Drive Completion of a Transaction use case (section 3.3.3.15) is a supporting actor for this use case.

Stakeholders and Interests:

- Architects, developers, and testers, as described in 3.3.1.1, 3.3.1.2, and 3.3.1.3, expect that this use case completes an existing transaction with a commit or abort outcome. If the transaction is committed, then it is guaranteed that all operations that were executed within the transaction have been durably recorded. If the transaction is aborted, then it is guaranteed that the operations that were executed within the transaction are forgotten.

Preconditions:

- The Transaction Processing Services System is operational.
- The application can access a transaction manager in the system.

Minimal Guarantees:

- If the operation fails, then the transaction manager guarantees that the updates completed in the previous Do Transacted Work use case (section 3.3.3.3) will not be committed.

Success Guarantee:

- If the use case completed successfully, then the updates that took place in the preceding Do Transacted Work use case (section 3.3.3.3) are durably recorded.

Trigger:

- The application triggers this use case after successfully completing the Do Transacted Work.

Main Success Scenario:

1. The application asks the transaction manager to commit or abort a transaction.
2. The transaction manager makes a durable record for the result of the transaction and responds back to the application, indicating success.
3. The transaction manager initiates the Drive Completion of a Transaction use case (section 3.3.3.15) to notify all participants of the outcome of the transaction.

Extensions: None.

3.3.3.14 Complete a Transaction – External Application

This use case is the same use case as described in 3.3.3.13, with External Application as the direct actor.

3.3.3.15 Drive Completion of a Transaction – Transaction Manager

Goal: The goal of the use case is to inform the transaction participants of the outcome of the transaction and driving a consistent outcome.

Context of use: This use case is triggered by the Complete a Transaction use case (section 3.3.3.11). During this use case, the transaction manager informs all enlisted resource managers and subordinate transaction managers about the outcome of the transaction. Successful completion of this use case either aborts or commits the operations in the transaction.

Direct Actor: The direct actor of this use case is a transaction manager.

Primary Actor: The primary actor of this use case is the Complete a Transaction use case (section 3.3.3.11).

Supporting Actors: Transaction managers that are subordinate to the transaction manager executing this use case are supporting actors for this use case. Supporting actors execute a new instance of the Drive Completion of a Transaction use case on resource managers and transaction managers that are enlisted in the transaction through them.

Stakeholders and Interests:

- Architects, developers, and testers as described in section 3.3.1.1, 3.3.1.2, and 3.3.1.3, expect that this use case will handle the coordination of all transaction participants, and that, upon successful completion, the transaction will reach the desired outcome.

Preconditions:

- The Transaction Processing Services System is operational.

- The transaction manager can access the participants in the transaction.

Minimal Guarantees:

- If the operation completed partially because of a transient failure, then only a subset of the participants may have been notified of the outcome of the transaction. In this case, the transaction manager guarantees to keep a record of the transaction outcome until the unnotified participants are informed of the outcome of the transaction through recovery.

Success Guarantee:

- If the use case completed successfully, then it is guaranteed that all participants known to the transaction manager have been durably notified of the outcome of the transaction.

Trigger:

- The Complete a Transaction use case (section 3.3.3.11) triggers this use case.
- The Drive Completion of a Transaction use case can also trigger this use case.

Main Success Scenario:

1. The transaction manager drives the Two-Phase Commit notifications on each participant enlisted in the transaction.
2. Each transaction manager that is subordinate to this transaction manager initiates a new Drive Completion of a Transaction use case for their participants.
3. Transaction manager returns success to the initiating use case.

Extensions: None.

3.3.3.16 Do Transaction Recovery – Resource Manager

Goal: The goal of the use case is to discover the outcome of transactions.

Context of use: This use case represents the recovery of the state of a transaction after a transient system failure. As described in [GRAY], during the Two-Phase Commit protocol, if a participant has completed phase one, but experienced a failure before completing phase two, the transaction stays in in-doubt state in the participant's log. The participant uses this use case to recover the outcome of such transactions.

Direct Actor: The direct actor of this use case is either a resource manager or an external resource manager, which will be referred to as the resource manager in the remainder of this section.

Primary Actor: Same as direct actor.

Supporting Actors: None.

Stakeholders and Interests:

- Architects as described in section 3.3.1.1 expect that upon successful completion of this use case, a resource manager will be able to correctly recover an in-doubt transaction.
- Developers and testers, as described in sections 3.3.1.2 and 3.3.1.3, expect that if there is an in-doubt transaction in the resource manager log, it can be recovered with this use case.

- This use case also interests the IT Operations Personnel, as described in section 3.3.1.4. If there are transactions in in-doubt state in a resource manager log, then the resource manager must execute this use case to recover the affected transactions. Similarly, if a transaction manager has any transactions in failed-to-notify state, then a resource manager must execute this use case to receive the outcomes of those transactions. Both of these operations may require manual intervention by the IT Operations Personnel to trigger the recovery, or to force the affected resource managers and transaction managers to forget the transactions in in-doubt and failed-to-notify states.

Preconditions:

- The Transaction Processing Services System is operational.
- The resource manager can access a transaction manager in the system.
- The resource manager has transactions in in-doubt state in its log.

Minimal Guarantees:

- The transaction manager guarantees that the transactions will remain in failed-to-notify-state unless the use case is completed successfully.

Success Guarantee:

- If the use case completed successfully, then the transaction manager forgets the transaction and the resource manager durably updates its records, according to the outcome received from the transaction manager.

Trigger:

- The resource manager triggers this use case on startup if it has any in-doubt transactions in its log, as described in [MS-DTCO] section 1.3.4.2.

Main Success Scenario:

1. The resource manager asks the transaction manager for the outcome of the transactions in in-doubt state in its log.
2. The system returns the state of each transaction if it has a record of the transaction in its own log. Otherwise, the transaction manager indicates to the resource manager that it does not have a record for the transaction.
3. The resource manager either aborts or commits each transaction on the basis of the outcome information received from the transaction manager. If the transaction manager indicated that it does not have a record for a transaction, the resource manager assumes that the transaction has been aborted.

Extensions: None.

3.3.3.17 Do Transaction Recovery – External Resource Manager

This use case is the same use case as described in 3.3.3.16, with External Resource Manager as the direct actor.

3.3.3.18 Do Transaction Recovery – Transaction Manager

This use case is the same use case as described in 3.3.3.16, with Transaction Manager as the direct actor.

3.3.3.19 Do Transaction Recovery – External Transaction Manager

This use case is the same use case as described in 3.3.3.16, with External Transaction Manager as the direct actor.

3.3.3.20 Manage Transaction Managers – Management Tool

Goal: The goal of the use case is to monitor and administer a transaction manager in the system.

Context of use: This use case is used by a management tool to monitor and administer a transaction manager.

Direct Actor: The direct actor of this use case is a management tool.

Primary Actor: Same as direct actor.

Supporting Actors: None.

Stakeholders and Interests:

- Architects and IT operations personnel, as described in sections 3.3.1.1 and 3.3.1.4, expect that upon successful completion of this use case, correct statistical information about the managed transaction manager is returned, and the transaction manager is correctly configured.

Preconditions:

- The Transaction Processing Services System is operational.
- The management tool can access the transaction manager in the system.

Minimal Guarantees:

- Unless the use case is successfully completed, no configuration changes are applied on the transaction manager.

Success Guarantee:

- If the management tool is used for collecting statistical information about a transaction manager, then the use case guarantees that correct statistical information is returned to the management tool.
- If the management tool is used for configuring a transaction manager, then the use case guarantees that upon successful completion, the transaction manager configuration is updated.

Trigger:

- The management tool triggers this use case.

Main Success Scenario:

1. The management tool either asks for statistical information, or requests to update the configuration of the transaction manager.

2. The correct statistical information is returned or the configuration of the transaction manager is updated.

Extensions: None.

3.3.3.21 Manage Transactions – Management Tool

Goal: The goal of the use case is to monitor and administer a transaction manager in the system.

Context of use: This use case is used by a management tool to monitor and administer a transaction manager.

Direct Actor: The direct actor of this use case is a management tool.

Primary Actor: Same as direct actor.

Supporting Actors: None.

Stakeholders and Interests:

- Architects and IT operations personnel, as described in sections 3.3.1.1 and 3.3.1.4, expect that upon successful completion of this use case, the transaction state is correctly updated.

Preconditions:

- The Transaction Processing Services System is operational.
- The management tool can access the transaction manager in the system.

Minimal Guarantees:

- Unless the use case is successfully completed, no configuration changes are applied on the transaction manager.

Success Guarantee:

- The transaction state is correctly updated.

Trigger:

- The management tool triggers this use case.

Main Success Scenario:

1. The management tool asks the transaction manager for a list of existing transactions.
2. The transaction manager returns a list of existing transactions and their known states.
3. The management tool asks the transaction manager to update the state of a transaction. For example, it may force the transaction to abort.
4. The transaction manager successfully updates the state of the transaction.

Extensions: None.

4 System Context

This section describes the relationship between this system and its environment.

4.1 System Environment

The system is designed to provide transaction coordination services for applications and other systems. These applications and other systems may be local to the computer where the system components are installed, or they may be located on a remote machine. This requires the system to be reachable by the applications and other systems that may require transaction coordination services.

Transactions are used by applications and other systems to maintain data coherency in the event of transient failures. To satisfy this requirement, the system guarantees atomicity through transactions. Transactions require the use of a log in a durable storage system. The log is used to hold important state information. Following a transient failure, the system can access the log to recall the last known state and continue its processing from that point.

A transaction may require updates in multiple resources, where the resources are spread across a computer network. The system is designed to provide transaction coordination services for such cases, and it is assumed that computers involved in the transaction can access each other.

There are various transaction coordination standards that either implement the Two-Phase Commit protocol, or provide means for disparate applications to propagate transaction information. The list of these standards is provided in section 2.3. The system is designed to allow multiple participants participate in one transaction while using different transaction coordination standards.

4.2 System Assumptions and Preconditions

Given the environment described in section 4.1, the system has the following assumptions and preconditions:

- The system must have access to a durable storage system where it can keep a log. The system holds state information for each running transaction in the log. Depending on the number of running transactions at a given time, the log size requirement may differ because the size of the log will grow with the number of transaction states that it must store.
- If the system spans across a computer network, the system must be installed on all the computers involved.
- The system must be configured so that participants can access its services locally or remotely.
- If the system components are going to span across a computer network, the computers in question must be connected to each other via the durable network described above.
- It is assumed that each transaction participant is trusted by the system. As discussed in section 8, it is possible that a malicious participant may start several new transactions and never complete them, resulting in a filled log. Such a case would force the system to stop responding to new incoming transaction requests until enough log space is available again.

Member protocols supported by the system, as listed in section 2.2, may have additional assumptions and preconditions when that protocol is being used. Please see the relevant member protocol specification for details.

4.3 System Relationships

This section describes the relationships between the system and external components, system dependencies, and other systems influenced by this system.

4.3.1 Black Box Relationship Diagram

This section describes the externally visible view of the system and the components within the system.

The conceptual framework for the Transaction Processing Services System is defined in terms of roles specified in [MS-DTCO] section 1.3.3. These roles are as follows:

Roles that use System Base protocols:

- Application: A client application that wants to perform transacted work on a number of resource managers. The application creates a transaction, and therefore, only that application has the right to commit the transaction.
- Application Service: A service that accepts requests to perform transacted work on local resource managers. An application service does not have the right to commit transactions.
- Transaction Manager: A service that coordinates the lifetime of transactions, providing functionality for resource managers to enlist in these transactions, and provides functionality to enlist in transactions that are coordinated by remote transaction managers.
- Resource Manager: A participant that is responsible for coordinating the state of a resource with the outcome of transactions. For a specified transaction, a resource manager enlists with exactly one transaction manager to vote on that transaction outcome and to obtain the final outcome.
- Management Tool: An application that monitors the health of a transaction manager and configures settings related to transaction coordination.

Roles that use System External protocols:

- External Application: An application that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.
- External Application Service: An application service that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.
- External Transaction Manager: A transaction manager that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.
- External Resource Manager: A resource manager that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.

The most basic role interaction scenario is illustrated in the following figure. The Application performs work on a local Resource Manager. There is no propagation necessary because the Resource Manager and the Application share a common local Transaction Manager. All communications between the Application and the Transaction Manager, between the Resource Manager and the Transaction Manager, and between the Management Tool and the Transaction Manager are based on System Base protocols. Communications between the Application and the Resource Manager are implementation-specific.

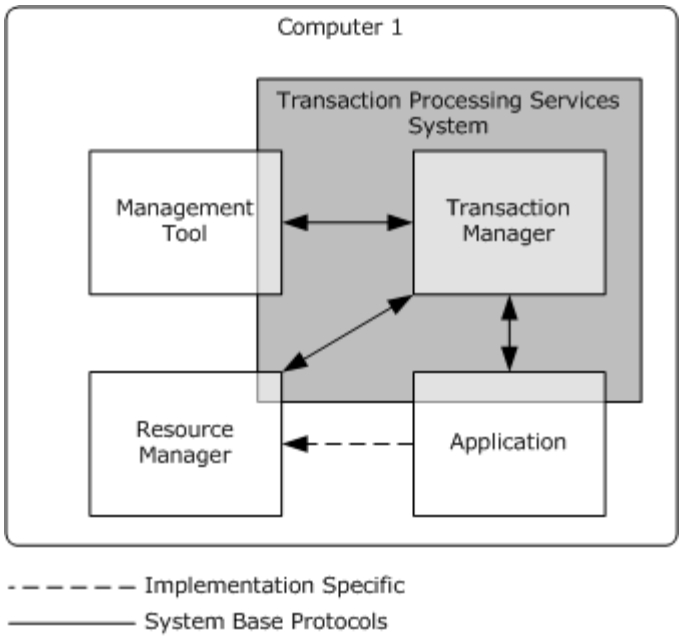

Figure 9: Basic communication between the roles defined in the transaction life cycle

The following figure illustrates a distributed scenario. The Application performs work on a local Resource Manager and a remote Resource Manager. It is necessary for the transaction to be propagated from the Application's local Transaction Manager to the remote Resource Manager's Transaction Manager.

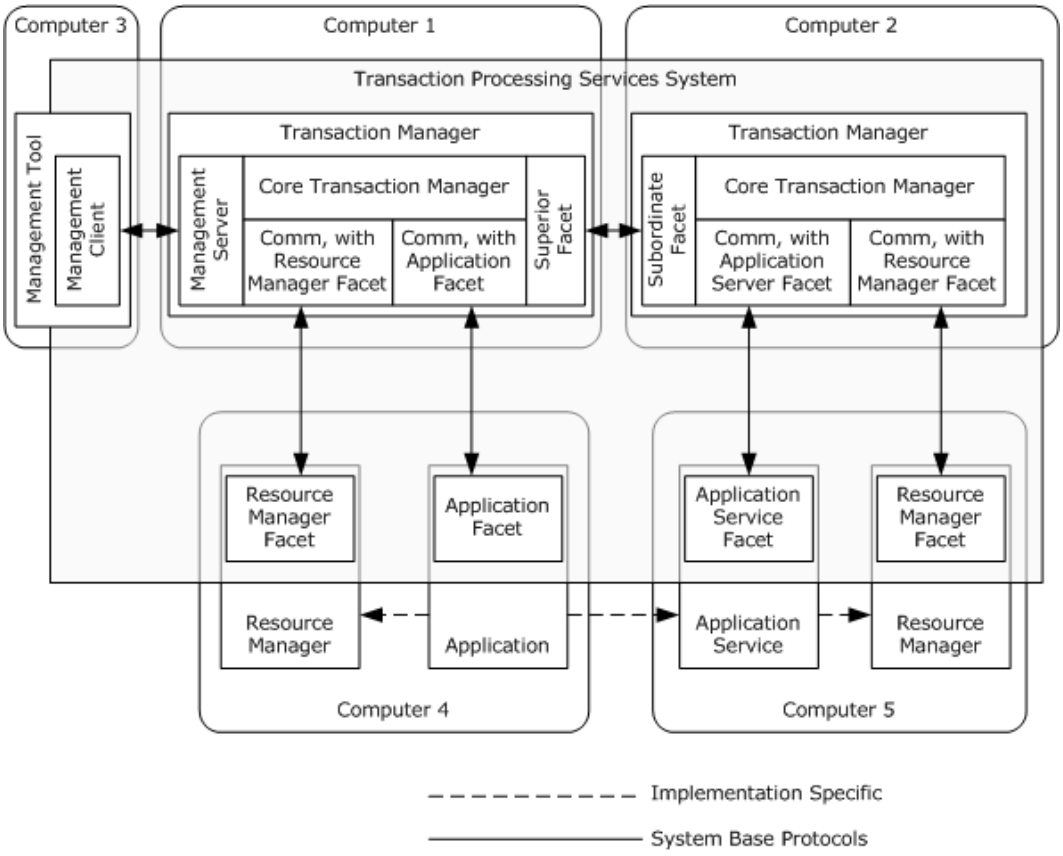

Figure 10: Distributed communication between the roles defined in the transaction life cycle

As illustrated in the preceding figure, the system uses various facets to enable the communication between different roles. Specific details about these facets and their use are discussed in section 5.

The communications between the Application and Application Service, between the Application and the Resource Manager, and between the Application Service and the Resource Manager are implementation-specific. The expectation is that this communication will consist of a request for work to be done, along with all information that is necessary to enlist in the transaction, including the **transaction identifier**. Otherwise, all other communication is based on System Base protocols.

4.3.2 System Dependencies

The system depends on a durable storage system to maintain the state that is used when recovering from failures. The storage that holds this state is referred to as a log. The log is a crucial component of the system. Without the log, following a transient failure where all in-memory state is lost, it is not possible for the system to determine the last known state of a given transaction and whether the transaction outcome has been communicated to the corresponding participants. If recovery is needed, but the log that has the recovery information is lost, it is not possible to recover the corresponding transactions. As a result, data corruption or data loss may occur on the affected resources.

The Transaction Processing Services System depends on a networking system to connect the computers involved in the system together, if the system spans multiple computers. The system has

no specific requirements regarding the type of network that needs to be used for this purpose. The system internal components may span across multiple computers, or some transaction participants may be remotely communicating with the system over a network. In either case the components on separate computers rely on the networking system to discover and communicate with each other.

The Transaction Processing Services System depends on a security identity management system to authenticate identities and group them. The system uses the security identity management system to restrict access to its assets and functionality to specified groups.

4.3.3 System Influences

Transaction Processing Services System influences the following systems:

- **Message Queuing System:** The Message Queuing System, as described in [MS-MQSO], depends on the Transaction Processing Services System to allow its queues to be treated as resources in the context of a distributed transaction. Without the Transaction Processing Services System, the Message Queuing System would need to either extend its internal transaction manager to support distributed transactions or rely on another transaction processing system to achieve this.
- **Component Object Model Plus (COM+):** The Component Object Model Plus Protocol, as described in [MS-COM], depends on the Transaction Processing Services System for implementing its transactional features. Without the Transaction Processing Services System, the Component Object Model Plus Protocol would need to either implement an internal transaction processing system, or to rely on another transaction processing system, to achieve the same functionality.

4.4 System Applicability

The Transaction Processing Services system is applicable in scenarios where atomic transaction processing is required, and where the participants may be local on the same computer or distributed to a network of computers, and each participant may be configured to use a different transaction processing protocol.

4.5 System Versioning and Capability Negotiation

The system does not define any versioning and capability negotiation beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

4.6 System Vendor-Extensible Fields

The system does not define any vendor-extensible fields beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

5 System Architecture

This section describes the basic structure of the system and the interrelationships among its parts, consumers, and dependencies.

5.1 Abstract Data Model

This section describes the conceptual data organization the system maintains to provide its functionality. The data model described in this section can be implemented in a variety of ways. This specification does not prescribe any specific implementation technique.

As described in section 4.3.1, the system consists of one or more transaction managers. Each transaction manager in the system may communicate with applications, application services, resource managers, external applications, external applications services, external resource managers, and external transaction managers, using the protocols listed in section 2.2. If a transaction is distributed across multiple computers, multiple transaction managers in the system may also communicate with each other by using the protocol described in [MS-DTCO]. Such an example is described and illustrated in section 4.3.1. Each transaction manager in the system and each transaction participant that the system communicates with maintains a set of abstract data models, depending on the protocols in effect. The following sections describe those abstract data models from the perspective of a transaction manager, and each transaction participant.

5.1.1 Transaction Manager

Each transaction manager within the system must maintain the following abstract data models at all times.

- Common Details abstract data model as described in [MS-DTCO] section 3.1.1.
- Core Transaction Manager Facet Details abstract data model as described in [MS-DTCO] section 3.2.1.

The Common Details abstract data model described in [MS-DTCO] section 3.1.1 represents the conceptual data organization for all active transactions known to the transaction manager, and for communications using the protocols described in [MS-CMP] and [MS-CMPO]. The Core Transaction Manager Facet Details abstract data model described in [MS-DTCO] section 3.2.1 specifies a set of data elements that represent the configuration of the transaction manager. The abstract data model described in [MS-DTCO] section 3.2.1 also builds on top of the Common Details abstract data model described in [MS-DTCO] section 3.1.1, extending the representation of each active transaction known to the system.

As a transaction manager starts interactions with transaction participants and other transaction managers within the system, as the transaction becomes distributed, the transaction manager creates and maintains additional abstract data models for each interaction. The protocol used for the interaction determines which abstract data model must be implemented by its participants, depending on the roles of the participants. All abstract data models maintained by the transaction manager share state through the Common Details and Core Transaction Manager Facet Details abstract data models, as described in [MS-DTCO], sections 3.1.1 and 3.2.1. Details about how an abstract data model specified by a protocol corresponds to the Common Details and Core Transaction Manager Facet Details abstract data models, as described in [MS-DTCO], sections 3.1.1 and 3.2.1, is described in the corresponding protocol technical documentation. The following figure illustrates a transaction manager maintaining all possible abstract data models, and the relationships between the abstract data models.

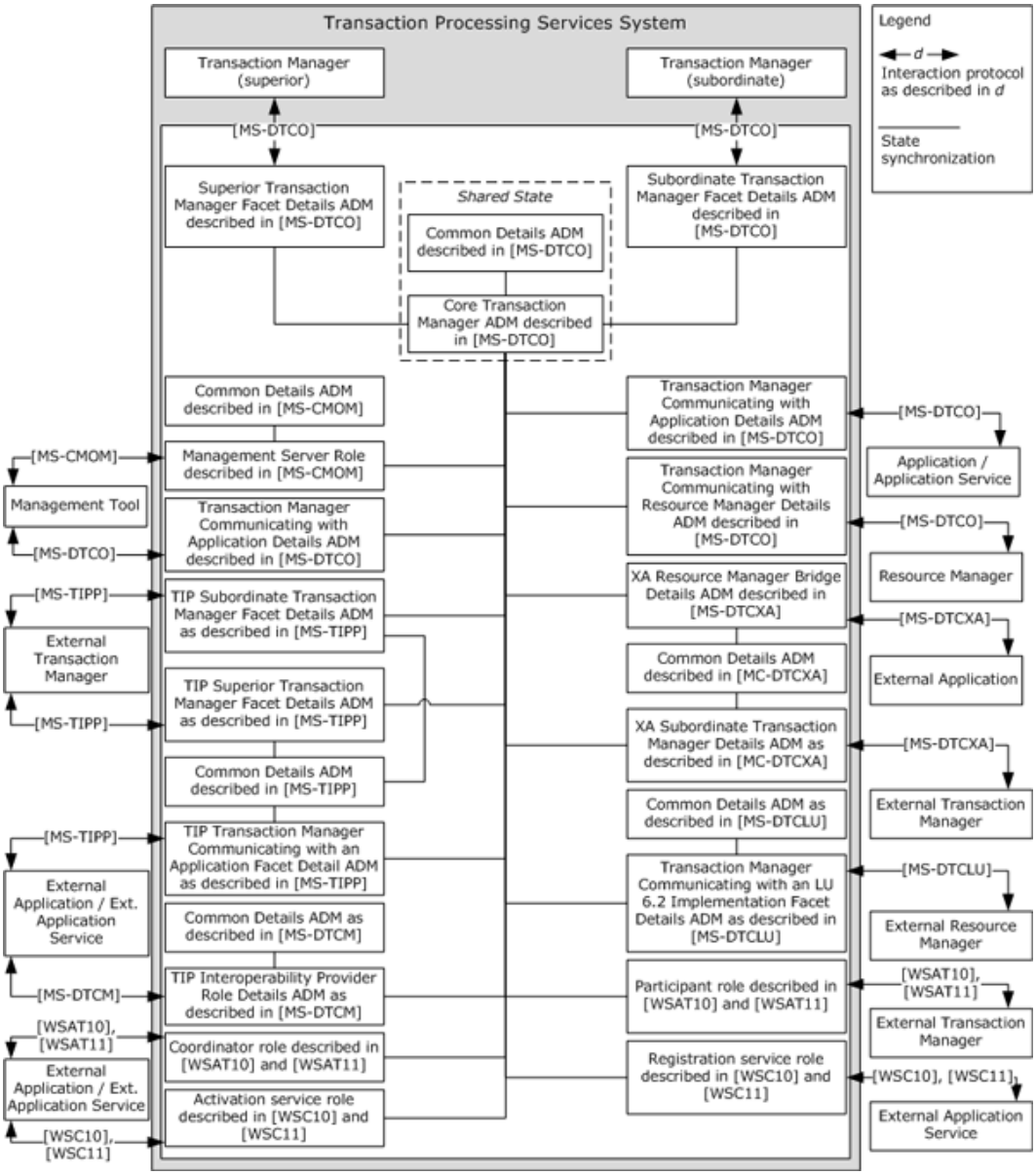

Figure 11: Transaction Manager Abstract Data Model

When multiple transaction managers are involved in coordinating a transaction in the system, each transaction manager implements an additional abstract data model for the duration of the transaction lifetime, depending on the subordinate-superior relationship between the transaction managers:

- The superior transaction manager implements the Superior Transaction Manager Facet Details abstract data model as described in [MS-DTCO] section 3.7.1.
- The subordinate transaction manager implements the Subordinate Transaction Manager Facet Details abstract data model as described in [MS-DTCO] section 3.8.1.

When communicating with an application or application service using System Base protocols, the transaction manager maintains an instance of Transaction Manager Communicating with Application Details abstract data model as described in [MS-DTCO] section 3.4.1. The application or the application service maintains an instance of Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Application Details abstract data model as described in [MS-DTCO] section 3.3.1.

When communicating with a resource manager using System Base protocols, the transaction manager maintains an instance of the Transaction Manager Communicating with Resource Manager Facet Details abstract data model as described in [MS-DTCO] section 3.6.1. The resource manager maintains an instance of Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Resource Manager Details abstract data model as described in [MS-DTCO] section 3.5.1.

When communicating with a management tool, the transaction manager maintains an instance of the Common Details abstract data model as described in [MS-CMOM] section 3.1.1, and an instance of the Management Server Role Details abstract data model as described in [MS-CMOM] section 3.3.1, and an instance of Transaction Manager Communicating with Application Details abstract data model as described in [MS-DTCO] section 3.4.1. The management tool maintains an instance of Common Details abstract data model as described in [MS-CMOM] section 3.1.1, and an instance of Management Client Role Details abstract data model as described in [MS-CMOM] section 3.2.1, an instance of Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Application Details abstract data model as described in [MS-DTCO] section 3.3.1.

When communicating with an external application or external application service using the protocol described in [MS-DTCM], the transaction manager maintains an instance of the Common Details abstract data model as described in [MS-DTCM] section 3.1.1, and an instance of the TIP Interoperability Provider Role Details abstract data model as described in [MS-DTCM] section 3.3.1. The external application or the external application service maintains an instance of Common Details abstract data model as described in [MS-DTCM] section 3.1.1, and an instance of the TIP Interoperability Application Role Details abstract data model as described in [MS-DTCM] section 3.2.1.

When communicating with an external application using the protocol described in [MS-TIPP], the transaction manager maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1, and an instance of the TIP Transaction Manager Communicating with an Application Facet Details abstract data model as described in [MS-TIPP] section 3.4.1. The application maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1.

When communicating with an external transaction manager using the protocol described in [MS-TIPP], the transaction manager maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1. Additionally, if the transaction manager is subordinate to the external transaction manager, the transaction manager maintains an instance of TIP Subordinate Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.3.1. If the transaction manager is superior to the external transaction manager, the transaction manager maintains an instance of the TIP Superior Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.2.1. The external transaction manager maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1. Additionally, if the external transaction manager is subordinate to the transaction manager, the external transaction manager maintains an instance of TIP Subordinate Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.3.1. If the external transaction manager is superior to the transaction manager, the external transaction manager maintains an instance of the TIP Superior Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.2.1.

When communicating with an external application using the protocol described in [MC-DTCXA], the transaction manager maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Resource Manager Bridge Facet Details abstract data model as described in [MC-DTCXA] section 3.4.1. The external application maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Resource Manager Bridge Details abstract data model as described in [MC-DTCXA] section 3.5.1.

When communicating with an external transaction manager using the protocol described in [MC-DTCXA], the transaction manager maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Subordinate Transaction Manager Details abstract data model as described in [MC-DTCXA] section 3.2.1. The external transaction manager maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Superior Transaction Manager abstract data model as described in [MC-DTCXA] section 3.3.1.

When communicating with an external resource manager using the protocol described in [MS-DTCLU], the transaction manager maintains an instance of the Common Details abstract data model described in [MS-DTCLU] section 3.1.1, and an instance of the Transaction Manager Communicating with an LU 6.2 Implementation Facet Details abstract data model as described in [MS-DTCLU] section 3.3.1. The external resource manager maintains an instance of the Common Details abstract data model described in [MS-DTCLU] section 3.1.1, and an instance of the LU 6.2 Implementation Details abstract data model described in [MS-DTCLU] section 3.2.1.

When communicating with external applications, external application services, and external transaction managers, using the WS-AtomicTransaction protocol described in [WSAT10] and [WSAT11], the system also uses the WS-Coordination protocol described in [WSC10] and [WSC11], because the WS-AtomicTransaction protocol directly depends on the WS-Coordination protocol as described in section 5.3. The details about how a combination of these protocols is used for transaction coordination are described in [WSAT10] and [WSAT11]. Example scenarios describing how the system uses these protocols are provided in [MS-WSRVCAT] section 1.3. When using WS-AtomicTransaction and WS-Coordination protocols, a transaction manager implements a set of the roles described by each protocol and maintains an instance of the corresponding data models specified by those roles.

When communicating with an external application using the WS-AtomicTransaction and WS-Coordination protocols, the transaction manager implements the Coordinator role described in [WSAT10] and [WSAT11], and the Activation Service role described in [WSC10] and [WSC11], while the external application implements the Application role described in [WSC10] and [WSC11], and the Initiator role described in [WSAT10] and [WSAT11]. When communicating with an external application service using the WS-Coordination protocol, the transaction manager implements the Registration service role, while the external application service implements the Requesting service role described in [WSC10] and [WSC11]. When communicating with an external transaction manager using the WS-AtomicTransaction and WS-Coordination protocols, the transaction manager implements the Registration service role described in [WSC10] and [WSC11], and the Participant role described in [WSAT10] and [WSAT11], while the external transaction manager implements the Requesting service role described in [WSC10] and [WSC11], and the Coordinator role described in [WSAT10] and [WSAT11].The member protocol described in [MS-WSRVCAT] is used as a helper protocol to facilitate transaction propagation between an external application and external application service or between an external application and external transaction manager. Therefore, the abstract data models described in [MS-WSRVCAT] are not implemented by the system.

5.1.2 Application

An application communicates with a transaction manager in the system by using the protocol described in [MS-DTCO]. The application maintains an instance of Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Application Details abstract data model as described in [MS-DTCO] section 3.3.1.

The following figure illustrates the abstract data model maintained by an application.

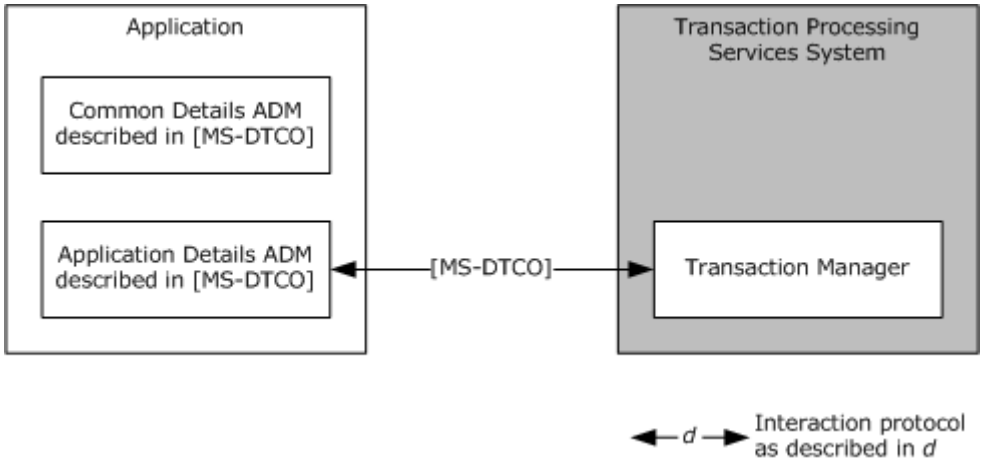

Figure 12: Application abstract data model

5.1.3 Application Service

An application service communicates with a transaction manager in the system by using the protocol described in [MS-DTCO]. The application maintains an instance of Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Application Details abstract data model as described in [MS-DTCO] section 3.3.1.

The following figure illustrates the abstract data model maintained by an application service.

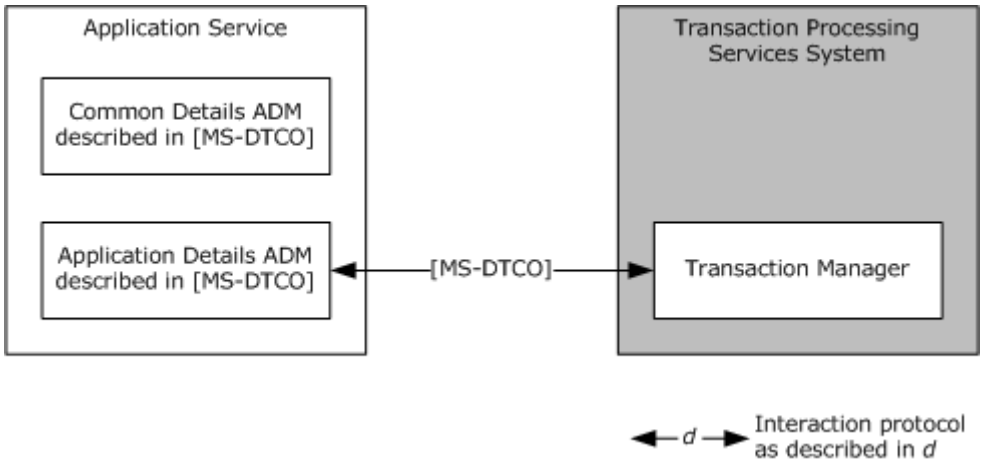

Figure 13: Application Service abstract data model

5.1.4 Resource Manager

A resource manager communicates with a transaction manager in the system by using the protocol described in [MS-DTCO]. The resource manager maintains an instance of the Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Resource Manager Details abstract data model as described in [MS-DTCO] section 3.5.1.

The following figure illustrates the abstract data model maintained by a resource manager.

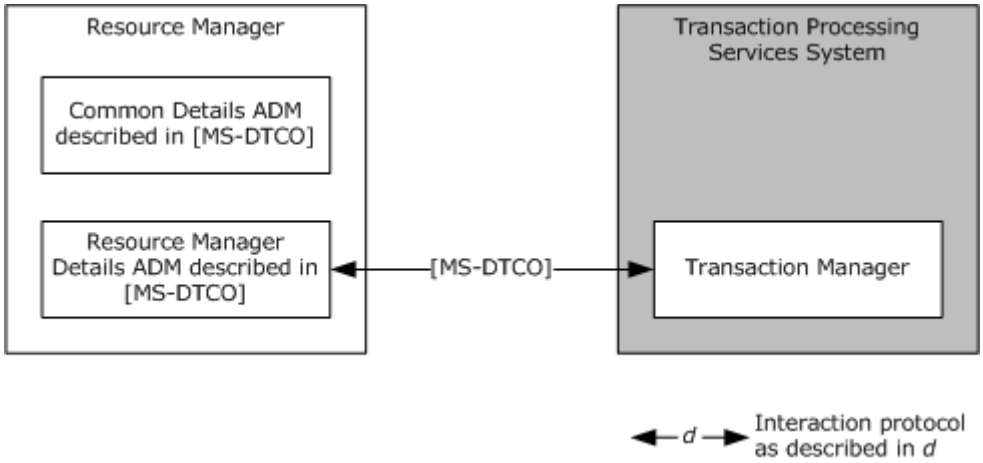

Figure 14: Resource Manager abstract data model

5.1.5 External Application

An external application communicates with a transaction manager in the system by using the protocols described in [MS-DTCM], [MS-TIPP], [MC-DTCXA]. The external application may also use WS-AtomicTransaction protocol, as described in [WSAT10] and [WSAT11]. When using the WS-AtomicTransaction protocol, the external application also uses the WS-Coordination protocol, as described in [WSC10] and [WSC11], because the WS-AtomicTransaction protocol directly depends on the WS-Coordination protocol as explained in section 5.3. Example scenarios describing how the system uses WS-AtomicTransaction and WS-Coordination protocols are provided in [MS-WSRVCAT] section 1.3.

When using the protocol described in [MS-DTCM], the external application maintains an instance of the Common Details abstract data model as described in [MS-DTCM] section 3.1.1, and an instance of the TIP Interoperability Application Role Details abstract data model as described in [MS-DTCM] section 3.2.1.

When using the protocol described in [MS-TIPP], the external application maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1.

When using the protocol described in [MC-DTCXA], the external application maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Resource Manager Bridge Details abstract data model as described in [MC-DTCXA] section 3.5.1.

When using the protocol the WS-AtomicTransaction protocol as described in [WSAT10] and [WSAT11], and the WS-Coordination protocol as described in [WSC10] and [WSC11], the external application implements the Application role described in [WSC10] and [WSC11], and the Initiator role described in [WSAT10] and [WSAT11].

When the external application starts a transaction using either of the protocols mentioned above, it uses the same protocol with the transaction manager for all consecutive operations. For example when a transaction is started using one protocol, the external application completes the same transaction by using the protocol that it started with. This means that neither of the abstract data models required by the protocols that the external application uses share any state.

The following figure illustrates the abstract data model maintained by an external application.

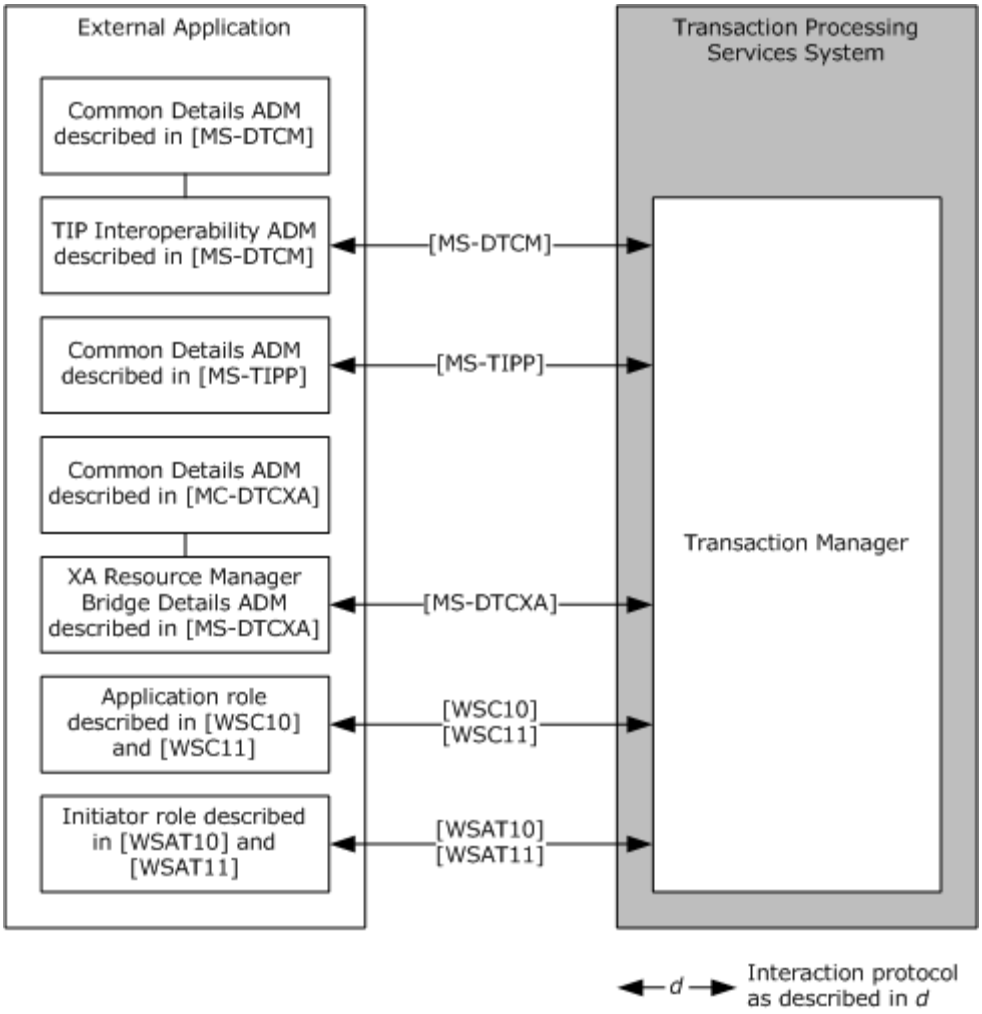

Figure 15: External Application abstract data model

5.1.6 External Application Service

An external application service communicates with a transaction manager in the system by using the protocol described in [MS-DTCM]. The external application service may also use WS-AtomicTransaction protocol as described in [WSAT10] and [WSAT11]. When using the WS-AtomicTransaction protocol, the external application service also uses the WS-Coordination protocol, as described in [WSC10] and [WSC11], because the WS-AtomicTransaction protocol directly depends on the WS-Coordination protocol as explained in section 5.3. Example scenarios describing

how the system uses WS-AtomicTransaction and WS-Coordination protocols are provided in [MS-WSRVCAT] section 1.3.

When using the protocol described in [MS-DTCM], the external application service maintains an instance of the Common Details abstract data model as described in [MS-DTCM] section 3.1.1, and an instance of the TIP Interoperability Application Role Details abstract data model as described in [MS-DTCM] section 3.2.1.

When using the protocol the WS-AtomicTransaction protocol as described in [WSAT10] and [WSAT11], and the WS-Coordination protocol as described in [WSC10] and [WSC11], the external application service implements the Requesting service role described in [WSC10] and [WSC11].

Abstract data models required by the protocol described in [MS-DTCM] do not share any state with the abstract data models required by the WS-AtomicTransaction and WS-Coordination protocols.

The following figure illustrates the abstract data model maintained by an external application.

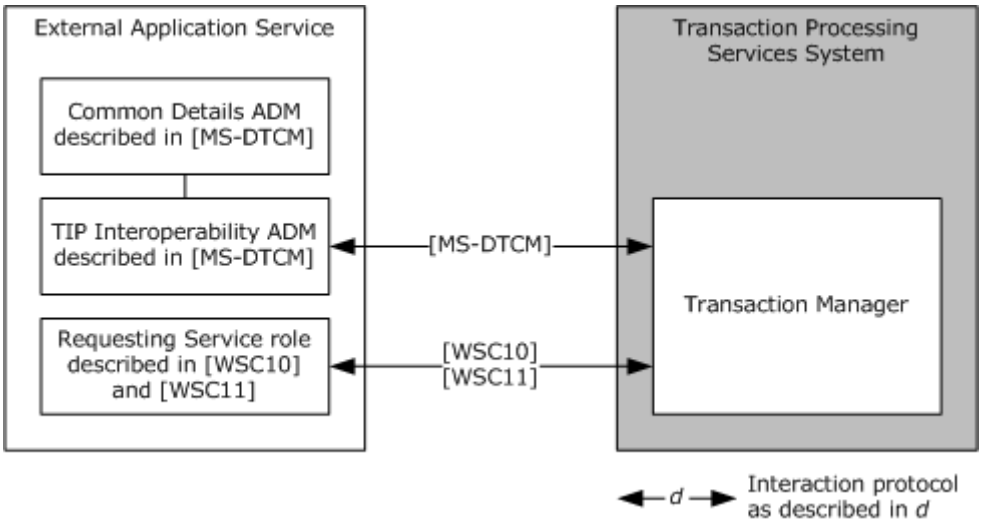

Figure 16: External Application Service abstract data model

5.1.7 External Resource Manager

An external resource manager communicates with a transaction manager in the system by using the protocol described in [MS-DTCLU].

When using the protocol described in [MS-DTCLU], the external resource manager maintains an instance of the Common Details abstract data model described in [MS-DTCLU] section 3.1.1, and an instance of the LU 6.2 Implementation Details abstract data model described in [MS-DTCLU] section 3.2.1.

The following figure illustrates the abstract data model maintained by an external resource manager.

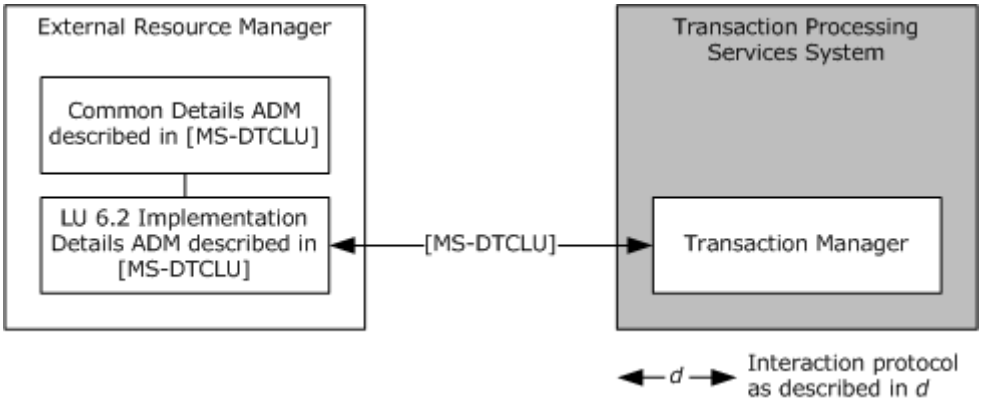

Figure 17: External Resource Manager abstract data model

5.1.8 External Transaction Manager

An external transaction manager communicates with a transaction manager in the system by using the protocols described in [MS-TIPP] and [MC-DTCXA]. The external transaction manager may also use WS-AtomicTransaction protocol, as described in [WSAT10] and [WSAT11], to communicate with a transaction manager in the system. When using the WS-AtomicTransaction protocol, the external transaction manager also uses the WS-Coordination protocol, as described in [WSC10] and [WSC11], because the WS-AtomicTransaction protocol directly depends on the WS-Coordination protocol as explained in section 5.3. Example scenarios describing how the system uses WS-AtomicTransaction and WS-Coordination protocols are provided in [MS-WSRVCAT] section 1.3.

When using the protocol described in [MS-TIPP], the external transaction manager maintains an instance of the Common Details abstract data model as described in [MS-TIPP] section 3.1.1. Additionally, if the external transaction manager is subordinate to the transaction manager, the external transaction manager maintains an instance of TIP Subordinate Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.3.1. If the external transaction manager is superior to the transaction manager, the external transaction manager maintains an instance of the TIP Superior Transaction Manager Facet Details abstract data model as described in [MS-TIPP] section 3.2.1.

When using the protocol described in [MC-DTCXA], the external transaction manager maintains an instance of the Common Details abstract data model as described in [MC-DTCXA] section 3.1.1, and an instance of the XA Superior Transaction Manager abstract data model as described in [MC-DTCXA] section 3.3.1.

When using the protocol the WS-AtomicTransaction protocol, as described in [WSAT10] and [WSAT11], and the WS-Coordination protocol, as described in [WSC10] and [WSC11], the external transaction manager implements the Requesting service role described in [WSC10] and [WSC11], and the Coordinator role described in [WSAT10] and [WSAT11].

The abstract data models listed above extend the abstract data models specified in [MS-DTCO] section 3.1.1, and 3.2.1.

The following figure illustrates the abstract data model maintained by an external transaction manager.

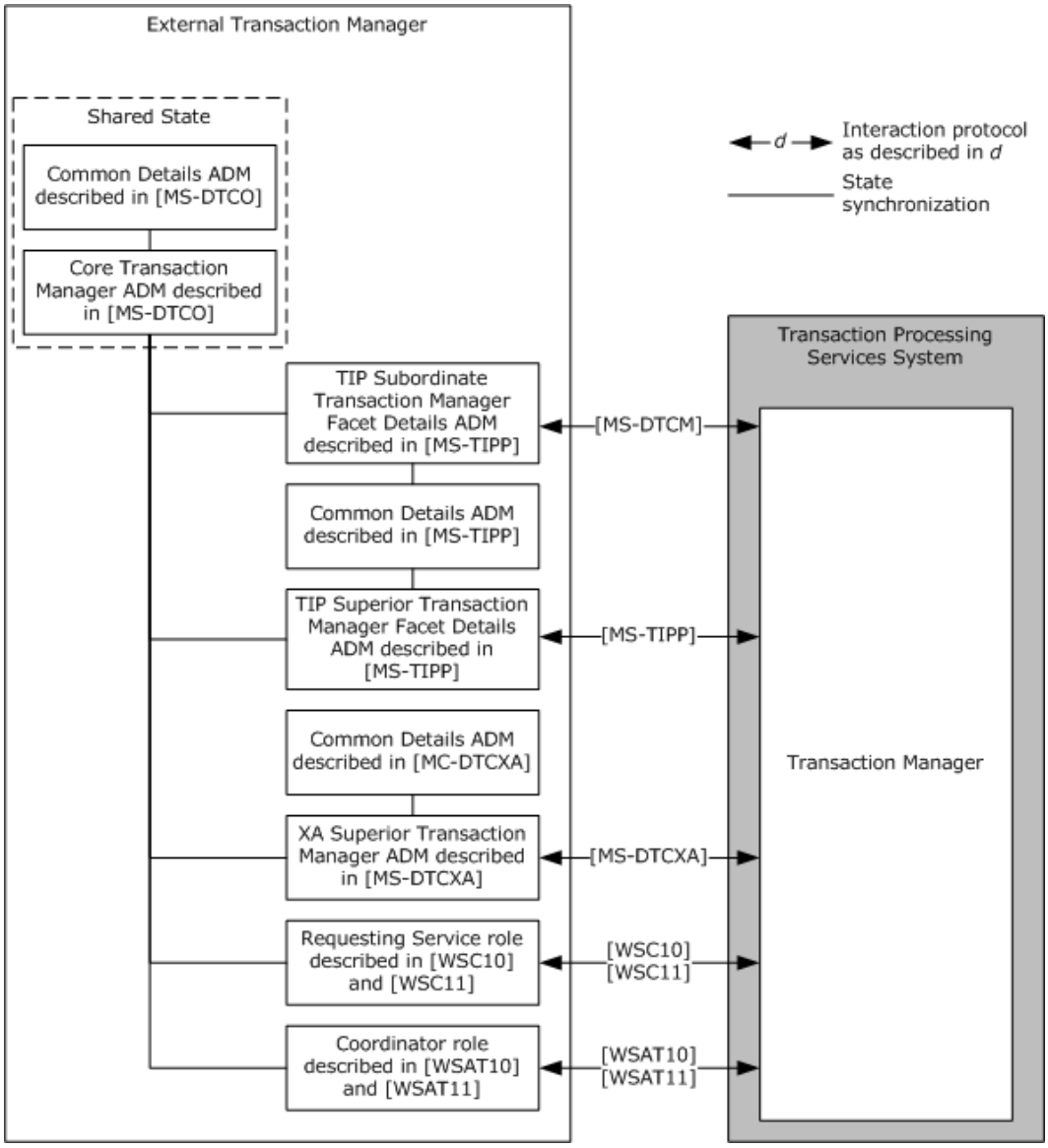

Figure 18: External Transaction Manager abstract data model

5.1.9 Management Tool

A management tool communicates with a transaction manager in the system by using the protocols described in [MS-DTCO] and [MS-CMOM].

When using the protocol described in [MS-DTCO], the management tool maintains an instance of the Common Details abstract data model as described in [MS-DTCO] section 3.1.1, and an instance of Application Details abstract data model as described in [MS-DTCO] section 3.3.1.

When using the protocol described in [MS-CMOM], the management tool maintains an instance of Common Details abstract data model as described in [MS-CMOM] section 3.1.1, and an instance of Management Client Role Details abstract data model as described in [MS-CMOM] section 3.2.1.

An implementation of a management tool also maintains the following data element:

- **Transaction List:** An enumeration of the transactions tracked by the transaction manager. Specifically, this is an enumeration DTCUITransListElement as described in [MS-CMOM] 2.2.2.2.1.

The protocol described in [MS-CMOM] is used to populate the Transaction List data element, as explained in [MS-CMOM] section 3.2. The management tool performs management operations on the transactions listed in Transaction List by using the protocol described [MS-DTCO].

The following figure illustrates the abstract data model maintained by a management tool.

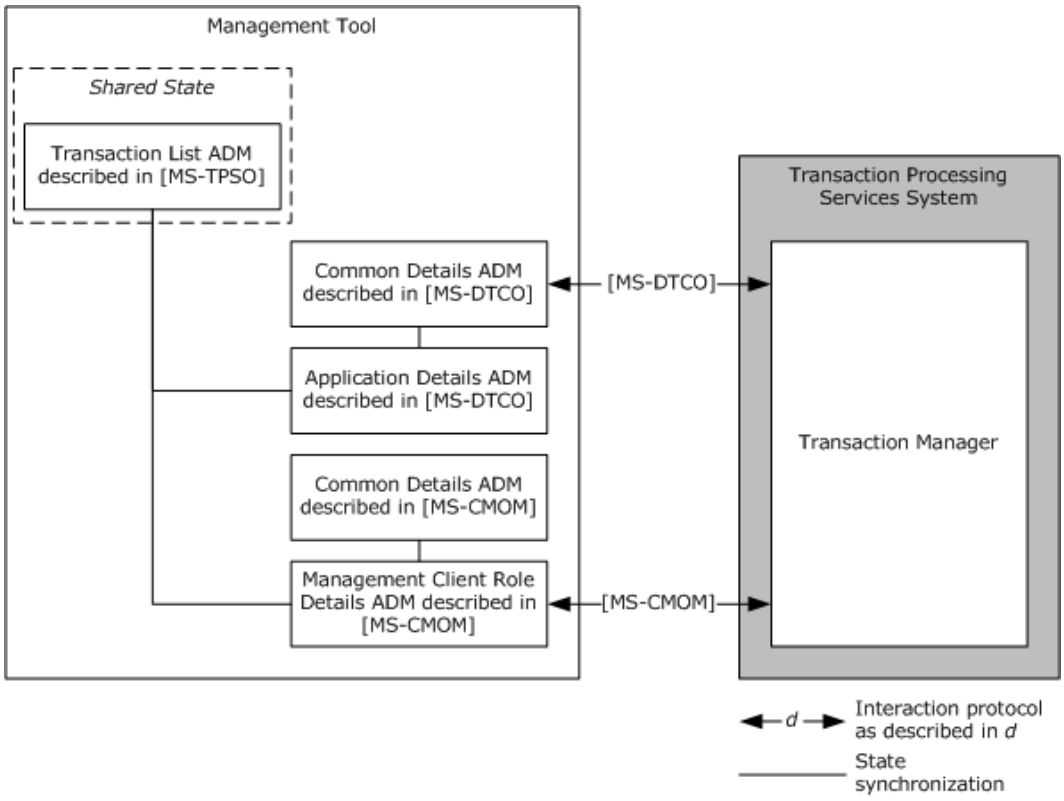

Figure 19: Management Tool abstract data model

5.2 White Box Relationships

The following diagram illustrates the roles that the system takes when communicating with external entities by using the system member protocols listed in section 2.2.

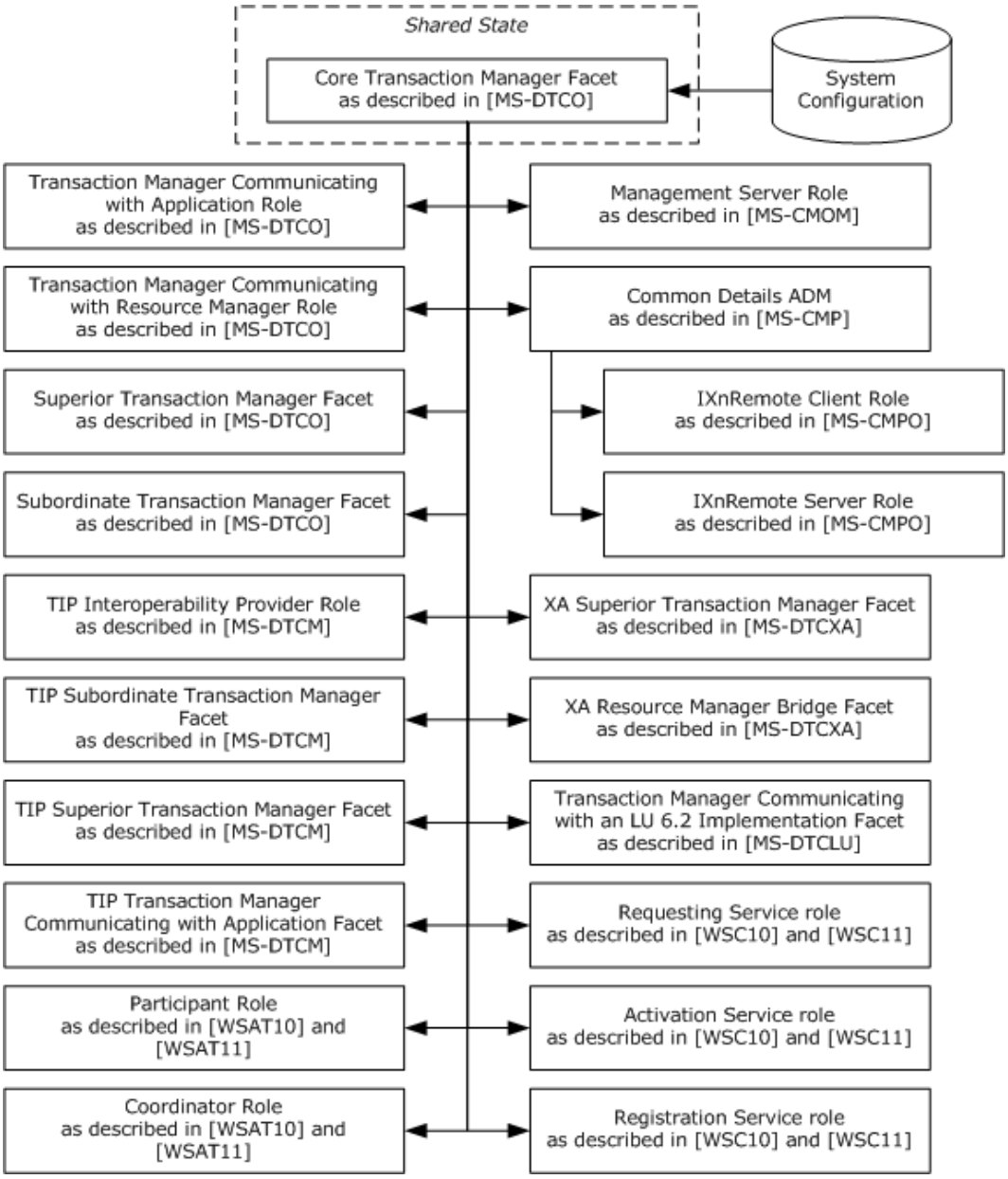

Figure 20: Internal System Roles

The Core Transaction Manager Facet is the central piece of the system. It maintains a comprehensive data model to track the current list of transactions, transaction states, security settings, enlistment information, and system configuration data as described in [MS-DTCO] section 3.2.1.

The Core Transaction Manager Facet reads the registry keys and values generated by the Management Client to populate the data model for System Configuration.

All other roles and **facet**s implemented by the system use the data model maintained by the Core Transaction Manager Facet to store shared state where needed, as described in section 5.1.1.

Management Server role uses the transaction list data available in the Core Transaction Manager Facet data model to build the transaction statistics when needed.

TIP Interoperability Provider Role establishes itself as a protocol extension with the Core Transaction Manager Facet by using the mechanism described in [MS-DTCO] section 3.2.1.5. Specific scenarios on how the TIP Interoperability Provider Role interacts with the Core Transaction Manager Facet are described in [MS-DTCM] section 1.3.

TIP Subordinate Transaction Manager Facet, TIP Superior Transaction Manager Facet, and TIP Transaction Manager Communicating with Application Facet are used for associating system internal transactions with TIP Transactions as described in [MS-TIPP]. The protocol described in [MS-TIPP] establishes itself as a protocol extension with the Core Transaction Manager Facet, by using the mechanism described in [MS-DTCO] section 3.2.1.5.

XA Superior Transaction Manager Facet and XA Resource Manager Bridge Facet as described in [MC-DTCXA] are used by the system to enable applications participate in transactions coordinated by XA Transaction Managers, and enable XA Resource Managers to participate in transactions coordinated by the system. Specific interactions between these facets and the Core Transaction Manager Facet are described in [MC-DTCXA] section 1.3.

Transaction Manager Communicating with an LU 6.2 Implementation Facet establishes itself as a protocol extension to the Core Transaction Manager Facet by using the mechanism described in [MS-DTCO] section 3.2.1.5. Specific scenarios how this Facet interacts with the Core Transaction Manager Facet are described in [MS-DTCLU] section 1.3. Section 3.3.1 in [MS-DTCLU] describes which abstract data model elements map to that of the Core Transaction Manager abstract data model.

Requesting Service Role, Activation Service Role, and Registration Service Role as described in [WSC10] and [WSC11] interact with the Core Transaction Manager Facet for transaction propagation and Two-Phase Commit protocol notifications. Participant Role and Coordinator Role as described in [WSAT10] and [WSAT11] interact with the Core Transaction Manager Facet for creating and completing transactions.

When interacting with protocols that depend on [MS-CMP], the system maintains the Common Details abstract data model specified in [MS-CMP] section 3.1.1. This data model maps to the IXnRemote Server and IXnRemote Client roles described in [MS-CMPO]. Details of how the data models described in [MS-CMP] interact with IXnRemote Server and IXnRemote Client are described in [MS-CMP] section 3.1.

5.3 Member Protocol Functional Relationships

5.3.1 Member Protocol Roles

This section provides details about how each member protocol is used by the system for specific roles, and also describes the interrelationships and dependencies between the protocols.

The following figure represents the protocol dependencies of the protocols used by the Transaction Processing Services System.

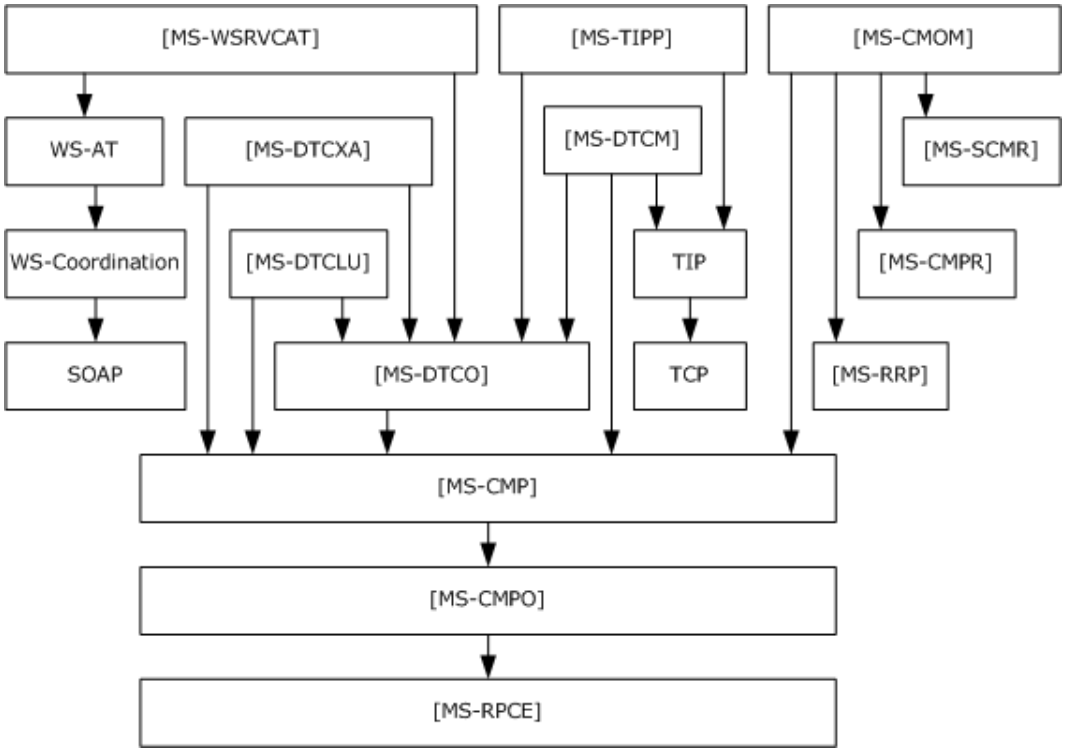

Figure 21: Transaction Processing Services Protocol dependencies

This section describes the roles played by each member protocol in the overall function of the system.

- MSDTC Connection Manager: OleTx Transaction Protocol, as described in [MS-DTCO], supports all the communications between the components described in section 3.2, except those between the management tool and the transaction manager, between the application and the application service, between the application and the resource manager, and between the application service and the resource manager. The abstract state machine that drives the transaction life cycle that is described in [MS-DTCO] section 1.3.1 is defined only in [MS-DTCO]. An implementation of this state machine is necessary for any implementation of a Transaction Manager, and therefore, any implementation of the protocols described in [MS-DTCM], [MS-TIPP], [MS-DTCLU], [MS-CMOM], [WSAT10], [WSAT11], [MS-WSRVCAT], and [MC-DTCXA] requires an MSDTC Connection Manager: OleTx Transaction Protocol implementation.
- MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as described in [MS-DTCLU], supports External Resource Manager to Transaction Manager communications. The system uses this protocol to provide transactional support to implementations of LU 6.2.
- MSDTC Connection Manager: OleTx Transaction Internet Protocol, as described in [MS-DTCM], supports External Application to Transaction Manager communications and External Application Service to Transaction Manager communications. The system uses this protocol to allow External Application and External Application Services to request the system to pull a transaction from, or push a transaction to, an External Transaction Manager that implements TIP.
- MSDTC Connection Manager: OleTx Transaction Internet Protocol, Transaction Internet Protocol (TIP) Extensions as described in [MS-TIPP] supports External Application to Transaction Manager

communications, External Application Service to Transaction Manager communications, and External Transaction Manager to Transaction Manager communications. This protocol represents an extension to TIP, as specified in [RFC2371]. It provides mechanisms to associate TIP transactions and the transactions internal to the system. It also provides mechanisms for driving a single atomic outcome, coordinating the distribution of this outcome, and transaction propagation.

- WS-AtomicTransaction protocol, as described in [WSAT10] and [WSAT11], is an alternative transaction coordination protocol. It supports External Application to Transaction Manager communications, External Application Service to Transaction Manager communications, and External Transaction Manager to Transaction Manager communications.
- WS-AtomicTransaction (WS-AT) Protocol Extensions, as described in [MS-WSRVCAT], supports External Application to External Transaction Manager communications, and External Application to External Application Service communications. The system uses this protocol to provide support for external applications to exchange system specific transaction propagation information with external application services. By using the data structures described by the WS-AtomicTransaction (WS-AT) Extensions and also by using the protocol described in [MS-DTCO], external applications can query system specific transaction propagation information from the system. External applications can then include this information in WS-AtomicTransaction messages when communicating with external application services. If the external application service also supports the protocols described in [MS-WSRVCAT] and [MS-DTCO], then for performance reasons, it may choose to communicate with the system by using System Base protocols rather than by using WS-AtomicTransaction protocol. Further details about this protocol and its usage scenarios are provided in [MS-WSRVCAT].
- MSDTC Connection Manager: OleTx XA Protocol Specification, as described in [MC-DTCXA], supports External Transaction Manager to Transaction Manager communications, External Application to Transaction Manager communications, and External Resource Manager to Transaction Manager communications. The system uses this protocol to provide transactional support for external transaction managers and external resource managers implementing the protocol specified in [XOPEN-DTP].
- MSDTC Connection Manager: OleTx Management Protocol Specification as described in [MS-CMOM] is used for communications between the management tool and the transaction manager and performs administration and configuration operations on the system.
- MSDTC Connection Manager: OleTx Transports Protocol as described in [MS-CMPO] is a framing and message transport protocol. It implements remote procedure call (RPC) interfaces as described in [MS-RPCE] for establishing duplex sessions between two partners and for exchanging messages between them. [MS-CMPO] describes specific restrictions on the use of RPC interfaces. Specific details are provided in [MS-CMPO] sections 1.3, 1.7, and 2.
- MSDTC Connection Manager: OleTx Multiplexing Protocol as described in [MS-CMP] supports both multiplexing multiple logical sessions over a single MSDTC Connection Manager: OleTx Transports Protocol session, and multiplexing multiple protocol messages into a single MSDTC Connection Manager: OleTx Transports Protocol message.

5.3.2 Member Protocol Groups

This section describes the member protocol groups that are used together accomplish a conceptually separate subgoal of the system.

Details about which external entities interact with the system by using these protocols, and specific interaction details, are provided in section 5.4.

5.3.2.1 System Base Protocols

The Transaction Processing Services System consists of one or more transaction managers that communicate with each other by using protocols internal to the system and that collectively provide external interfaces to applications and resource managers. All of this communication uses a base set of system-defined protocols that are referred to as the System Base protocols.

System Base protocols consist of the following:

- MSDTC Connection Manager: OleTx Transaction Protocol Specification, as described in [MS-DTCO]
- MSDTC Connection Manager: OleTx Management Protocol Specification, as described in [MS-CMOM]

5.3.2.2 System External Protocols

In addition to the System Base protocols, the Transaction Processing Services System provides specific external interfaces to enable applications and transaction managers that support protocols other than the System Base protocols to participate in transactions. These protocols are referred to as the System External protocols, and the participants that use System External protocols are referred to as external applications, external resource managers, and external transaction managers in this document.

System External protocols consist of the following:

- MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification, as described in [MS-DTCM]
- Transaction Internet Protocol (TIP) Extensions, as described in [MS-TIPP]
- MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as described in [MS-DTCLU]
- WS-AtomicTransaction Protocol, as described in [WSAT10] and [WSAT11]
- WS-AtomicTransaction (WS-AT) Protocol Extensions, as described in [MS-WSRVCAT]
- MSDTC Connection Manager: OleTx XA Protocol Specification, as described in [MC-DTCXA]

5.3.2.3 Communication Protocols

The following member protocols are used in conjunction to provide the underlying communications functionality for the System Base protocols and the protocols described in[MS-DTCM], [MS-DTCLU], and [MC-DTCXA]:

- MSDTC Connection Manager: OleTx Multiplexing Protocol Specification, as described in [MS-CMP]
- MSDTC Connection Manager: OleTx Transports Protocol Specification, as described in [MS-CMPO]

5.3.2.4 Protocols to Support TIP Transactions

The following protocols are used in conjunction to provide support for TIP transactions:

- MSDTC Connection Manager: OleTx Transaction Internet Protocol Specification, as described in [MS-DTCM]

- Transaction Internet Protocol (TIP) Extensions, as described in [MS-TIPP]

5.3.2.5 Protocols to Support WS-AtomicTransaction Transactions

The following protocols are used in conjunction to provide support for WS-AtomicTransactions:

- WS-AtomicTransaction Protocol, as described in [WSAT10] and [WSAT11]
- WS-AtomicTransaction (WS-AT) Protocol Extensions, as described in [MS-WSRVCAT]

5.4 System Internal Architecture

The conceptual framework for the Transaction Processing Services System is defined in terms of roles participating in the Two-Phase Commit protocol, as described in section 4.3.1. These roles are as follows:

- Roles that use System Base protocols:

Application: A client application that performs transacted work on a number of resource managers. The application creates a transaction, and therefore, only that application has the right to commit the transaction.

Application Service: A service that accepts requests to perform transacted work on local resource managers. An application service does not have the right to commit transactions.

Transaction Manager: A service that coordinates the lifetime of transactions, providing functionality for resource managers to enlist in these transactions, and that provides functionality to enlist in transactions that are coordinated by remote transaction managers.

Resource Manager: A participant that is responsible for coordinating the state of a resource with the outcome of transactions. For a specified transaction, a resource manager enlists with exactly one transaction manager to vote on that transaction outcome and to obtain the final outcome.

Management Tool: An application that monitors the health of a transaction manager and configure settings related to transaction coordination.

- Roles that use System External protocols:

External Application: An application that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.

External Application Service: An application service that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.

External Transaction Manager: A transaction manager that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.

External Resource Manager: A resource manager that uses a protocol other than a System Base protocol to communicate with the Transaction Processing Services System.

5.4.1 Communications within the System

The Transaction Processing Services System may use more than one transaction manager when coordinating a transaction, as described in 4.3. When there are multiple transaction managers in the system, they need to communicate with each other to collectively coordinate the transaction life cycle.

A transaction manager's role is to coordinate the lifetime of transactions and to provide functionality to enlist in transactions coordinated by a remote transaction manager. This allows applications to enlist local or remote resource managers in transactions coordinated by their local transaction manager. Each transaction manager maintains a transaction table of all currently incomplete transactions. When two transaction managers are involved in coordinating a transaction, one of them takes the superior transaction manager role, while the other takes the subordinate transaction manager role.

A transaction manager communicates with other transaction managers within the system to perform the following actions:

- **Push Transaction**: Performs push propagation as described in [MS-DTCO] section 1.3.5. This action requests that the remote transaction manager add the transaction to its transaction table and enlist in the transaction as a subordinate transaction manager. As a result of this, the remote transaction manager is subordinate to the transaction manager originating this action. This allows a subsequent Deserialize Transaction action by an application service local to the remote transaction manager to retrieve the propagated transaction's transaction identifier.
- **Pull Transaction**: Performs pull propagation as described in [MS-DTCO] section 1.3.5. This action requests that the remote transaction manager enlist the transaction manager originating this action in the transaction. If the request is successful, the originating transaction manager is enlisted in the transaction as a subordinate transaction manager. The originating transaction manager is subordinate to the remote transaction manager. The transaction is added to the originating transaction manager's transaction table so that a subsequent Deserialize Transaction action by an application service local to the originating transaction manager will retrieve the propagated transaction's transaction identifier.
- **Two-Phase Commit Notifications**: Request that a subordinate transaction manager go through the two-phase commit states.
- **Query Transaction Outcome**: Requests that the superior transaction manager provide the outcomes of a list of transactions.
- **Recover Transactions**: Request that a subordinate transaction manager either Abort or Commit each of a supplied list of transactions, as specified by the request.

When performing the Push Transaction, Two-Phase Commit Notifications, and Recover Transactions actions, the initiating transaction manager maps to the superior transaction manager facet, as specified in [MS-DTCO], and the receiving transaction manager maps to the subordinate transaction manager facet, as specified in [MS-DTCO].

When performing the Pull Transaction and Query Transaction Outcome actions, the initiating transaction manager maps to the subordinate transaction manager facet, as specified in [MS-DTCO], and the receiving transaction manager maps to the superior transaction manager facet, as specified in [MS-DTCO].

When a transaction is propagated to a transaction manager, it is enlisted in the transaction, as described in the Two-Phase Commit protocol, and it receives notifications, coordinates the outcome with its enlistments, and votes on the outcome of the transaction. When the protocol specified by the MSDTC Connection Manager: OleTx Transaction Protocol, as described in [MS-DTCO], is used to propagate a transaction, the subsequent Two-Phase Commit communication occurs through the protocol that was used to propagate the transaction.

5.4.2 Communications with External Systems

The following figure illustrates all possible external communications that the Transaction Processing Services System supports:

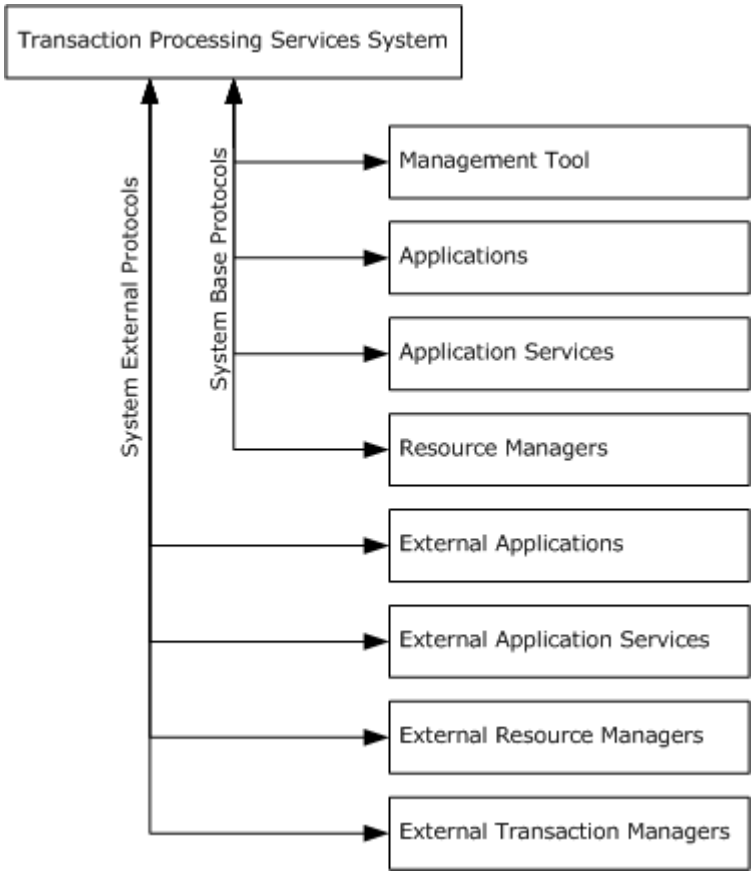

Figure 22: Components with which the Transaction Processing Services System communicates

Each component is explained as a role in terms of the conceptual actions in the transaction life cycle that it performs. Each communication path between the Transaction Processing Services System and a component precisely maps to existing protocol or protocols and roles described by those protocols. Sections 5.4.3 and 5.4.4 provide those details from the perspective of each component.

5.4.3 Incoming Interfaces

This section describes the incoming interfaces of the Transaction Processing Services System. These interfaces are described in terms of the components that interact with the Transaction Processing Services System, and in terms of the conceptual actions in the transaction life cycle that each component performs. A conceptual action may represent the multiple roundtrip communications that are specified in a protocol. Each protocol specifies these interactions in terms of roles that are defined by that protocol.

The following figure summarizes the incoming interfaces supported by System Base protocols, and the corresponding roles defined by the protocol involved.

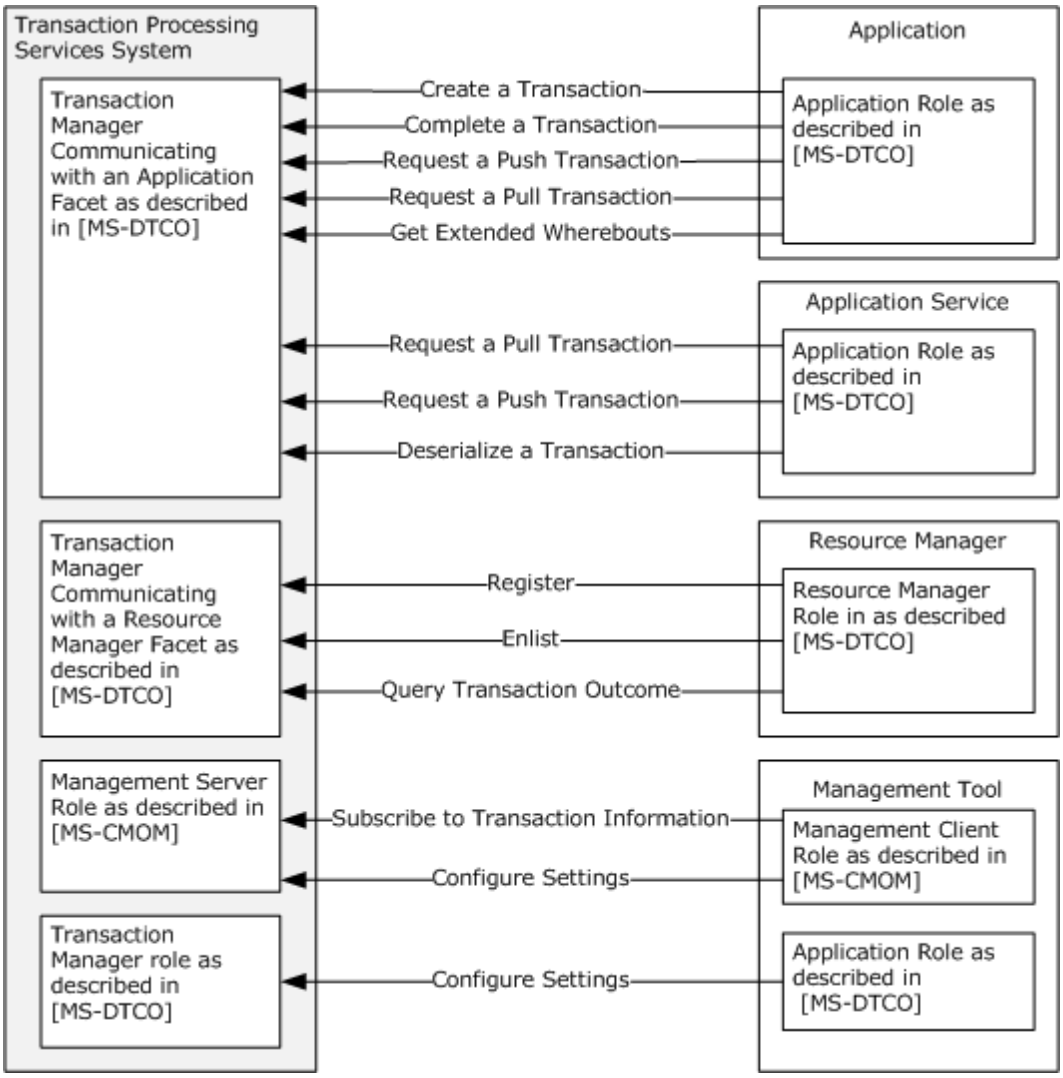

Figure 23: Incoming interfaces supported by System Base protocols

The following figure summarizes the incoming interfaces supported by System External protocols, and the corresponding roles defined by the protocol involved.

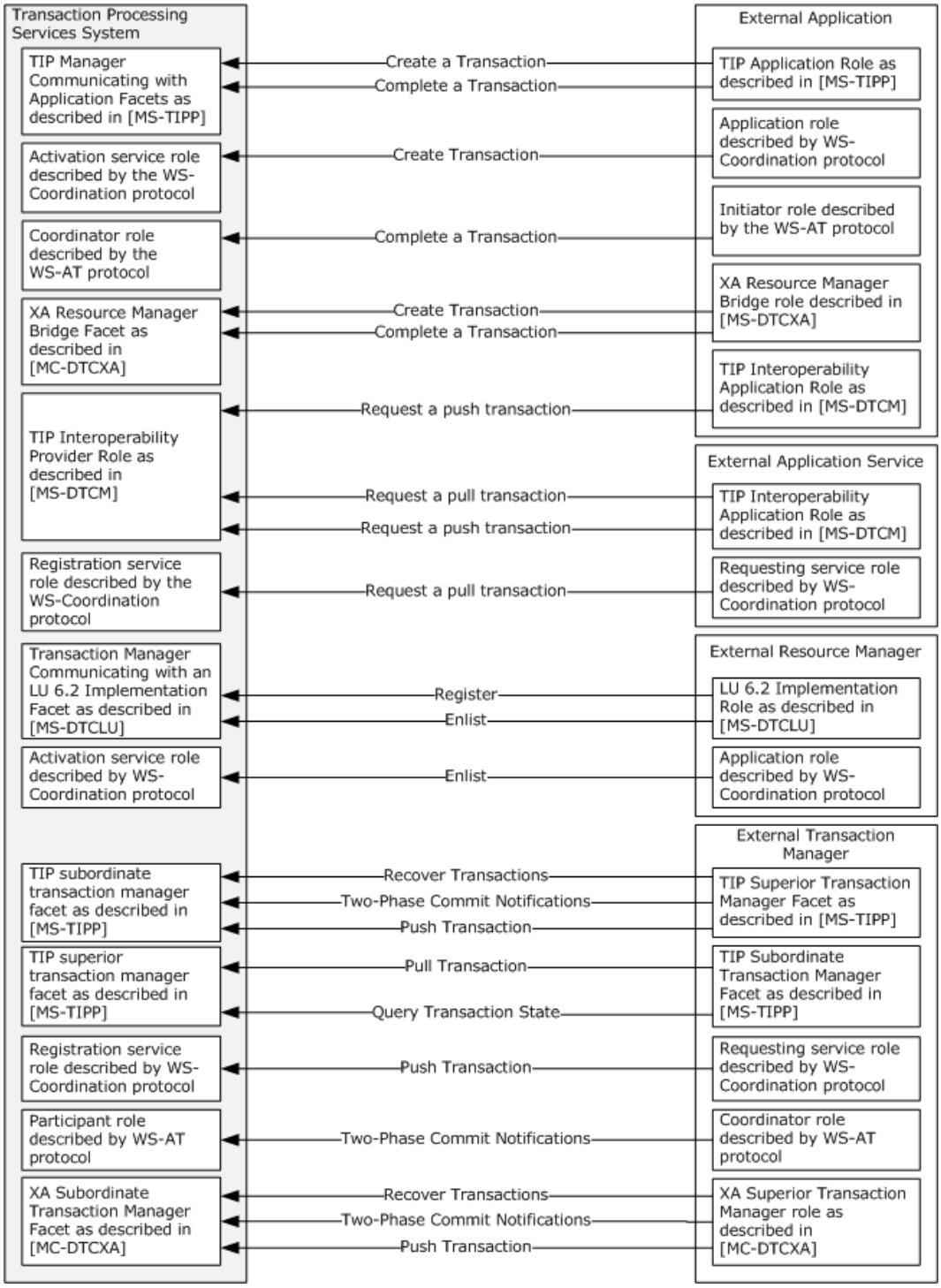

Figure 24: Incoming interfaces supported by System External protocols

5.4.3.1 Application

The application component represents a client application that performs transacted work on a number of resource managers. To perform this function, the application takes responsibility for the initial creation of a transaction at the start of the transaction life cycle, as defined in [MS-DTCO] section 1.3.1. It performs the following actions when communicating with the Transaction Processing Services System:

- **Create a Transaction**: Requests that the local transaction manager create a transaction and provide the application with the transaction identifier of the created transaction.
- **Request a Push Transaction**: Requests that the local transaction manager perform a Push Transaction action to propagate the transaction associated with the transaction identifier to a remote transaction manager.
- **Complete a Transaction**: Indicates to the transaction manager that it can commit or abort the transaction by using the Two-Phase Commit protocol.
- **Get Extended Whereabouts**: Requests that the local transaction manager provide the application with the network location and security configuration of the local transaction manager's WS-AtomicTransaction protocol endpoint.

This component maps to the Application role in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The system's communications with the application component are defined by the Transaction Manager Communicating with an Application Facet role in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The data types used for the Get Extended Whereabouts action are specified by the protocol described in [MS-WSRVCAT]. Applications that implement the protocol described in [MS-WSRVCAT] perform this action by using the protocol specified in [MS-DTCO].

5.4.3.2 Application Service

The application service component represents an application that accepts requests to perform transacted work on local resource managers. It performs the following actions when communicating with the Transaction Processing Services System:

- **Request a Pull Transaction**: Requests that the local transaction manager perform a Pull Transaction action to propagate the transaction associated with the transaction identifier from a remote transaction manager.
- **Request a Push Transaction**: Requests that the local transaction manager perform a Push Transaction action to propagate the transaction associated with the transaction identifier to a remote transaction manager.
- **Deserialize a Transaction**: Requests that the local transaction manager provide the transaction identifier of the transaction associated with the serialized information if it is present in the transaction table, or return an error if it is not.

This component maps to the Application role in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The system's communications with the application service component are defined by the Transaction Manager Communication with an Application Facet role in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

5.4.3.3 Resource Manager

The resource manager component represents a resource that can accept requests to perform transacted work. It does this to ensure consistency between its resources and other resources enlisted in a transaction. The canonical example of a resource manager is a database. It performs the following actions when communicating with the Transaction Processing Services System:

- **Register**: Registers the resource manager with the local transaction manager. This is necessary to ensure durability, because when the resource manager is recovering from a transient failure, the local transaction manager must be able to identify the resource manager.
- **Enlist**: Requests that the transaction manager register the resource manager to participate in the transaction by using the Two-Phase Commit protocol.
- **Query Transaction Outcome**: Requests that the transaction manager provide the outcomes of a list of transactions.

This component maps to the Resource Manager role in MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The system's communications with this component are defined by the Transaction Manager Communication with a Resource Manager Facet in MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

5.4.3.4 Management Tool

The management tool component represents an application that wants to monitor the health of a transaction manager and configure settings related to transaction coordination. It performs the following actions when communicating with the Transaction Processing Service System:

- **Subscribe to Transaction Information**: Requests that the transaction manager periodically provide a list of all transactions in its transaction table that are not currently complete, along with information about the state of the transactions returned.
- **Configure Settings**: Configure the transaction manager's settings or perform administrative operations on a specific transaction.

This component maps to the Management Client role in MS-DTC Connection Manager: OleTx Management Protocol Specification as specified in [MS-CMOM], and for administrative operations on a specific transaction, to the Application role in MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The system's communications with this component are defined by the Management Server role in MS-DTC Connection Manager: OleTx Management Protocol Specification, as specified in [MS-CMOM], and for administrative operations on a specific transaction, to the Transaction Manager role in MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

5.4.3.5 External Application

The Transaction Processing Services System provides incoming interfaces for external applications, which implement the following protocols:

- The MSDTC Connection Manager: OleTx Transaction Internet Protocol, as specified in [MS-DTCM]. This protocol supports the Request a Push Transaction action. Details are provided in section 5.4.3.9.1.

- The Transaction Internet Protocol (TIP) Extensions protocol, as specified in [MS-TIPP]. This protocol supports the Create a Transaction and Complete a Transaction actions. Details are provided in section 5.4.3.9.2.
- WS-AtomicTransaction Protocol, as specified in [WSAT10] or in [WSAT11]. This protocol supports the Create a Transaction and Complete a Transaction actions. Details are provided in section 5.4.3.9.4.
- MSDTC Connection Manager: OleTx XA Protocol Specification, as specified in [MC-DTCXA]. This protocol supports the Create a Transaction and Complete a Transaction actions. Details are provided in section 5.4.3.9.6.

5.4.3.6 External Application Service

The Transaction Processing Services System provides incoming interfaces for external application services, which implement the following protocols:

- MSDTC Connection Manager: OleTx Transaction Internet Protocol, as specified in [MS-DTCM]. This protocol supports the Request a Pull Transaction and Request a Push Transaction actions. Details are provided in section 5.4.3.9.1.
- WS-AtomicTransaction Protocol, as specified in [WSAT10] or in [WSAT11]. This protocol supports the Request a Pull Transaction action. Details are provided in section 5.4.3.9.4.

5.4.3.7 External Resource Manager

The Transaction Processing Services System provides incoming interfaces for external resource managers, which implement the following protocols:

- MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as specified in [MS-DTCLU]. This protocol supports the Register and Enlist actions. Details are provided in section 5.4.3.9.3.
- WS-AtomicTransaction Protocol, as specified in [WSAT10] or in [WSAT11]. This protocol supports the Enlist action. Details are provided in section 5.4.3.9.4.

5.4.3.8 External Transaction Manager

The Transaction Processing Services System provides incoming interfaces for External Transaction Managers, which implement the following protocols:

- Transaction Internet Protocol (TIP) Extensions protocol, as specified in [MS-TIPP]. This protocol supports the Push Transaction, Pull Transaction, Query Transaction Outcome, Recover Transactions, and Two-Phase Commit Notifications actions. Details are provided in section 5.4.3.9.2.
- WS-AtomicTransaction Protocol, as specified in [WSAT10] or in [WSAT11]. This protocol supports the Pull Transaction and Two-Phase Commit Notifications actions. Details are provided in section 5.4.3.9.4.
- MSDTC Connection Manager: OleTx XA Protocol Specification, as specified in [MC-DTCXA]. This protocol supports the Push Transaction, Two-Phase Commit Notifications, and Recover Transactions actions. Details are provided in section 5.4.3.9.6.

5.4.3.9 Incoming Interface Variations through System External Protocols

This section lists the System External protocols supported by the Transaction Processing Services System and describes the alternative incoming interfaces supported by each protocol for corresponding components participating in a transaction.

5.4.3.9.1 MSDTC Connection Manager: OleTx Transaction Internet Protocol

The MSDTC Connection Manager: OleTx Transaction Internet Protocol, as specified in [MS-DTCM], provides an alternative application to transaction manager and application service to transaction manager communication protocol. The protocol provides some of the same functionality as the MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the required actions are performed by using a Transaction Internet Protocol (TIP) implementation. This protocol provides an alternative protocol used by the following actions from the following components:

External Application:

- Request a Push Transaction

External Application Service:

- Request a Pull Transaction
- Request a Push Transaction

Both of these components map to the TIP Interoperability Application Role of the protocol specified by [MS-DTCM]. When communicating with these components, the system maps to the TIP Interoperability Provider Role specified in [MS-DTCM].

If an External Application or External Application Service uses the protocol specified in [MS-DTCM] to perform a Request Push Transaction action, then the system must perform the requested Push Transaction action by using the Transaction Internet Protocol (TIP) Extensions protocol specified in [MS-TIPP], as described in section 5.4.3.9.2.

If an External Application or External Application Service uses the protocol specified in [MS-DTCM] to perform a Request Pull Transaction action, then the system must perform the requested Pull Transaction action by using the Transaction Internet Protocol (TIP) Extensions protocol specified in [MS-TIPP], as described in section 5.4.3.9.2.

5.4.3.9.2 Transaction Internet Protocol (TIP) Extensions

The Transaction Internet Protocol (TIP) Extensions protocol, as specified in [MS-TIPP], provides an alternative transaction manager to transaction manager communication protocol. The protocol provides the same functionality as the transaction manager to transaction manager protocol specified by an MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the communication is performed by using TIP messages. This protocol provides an alternative protocol used by the following actions from the following components:

External Application:

- **Create a Transaction**
- **Complete a Transaction**

External Transaction Manager:

- Push Transaction

- Two-Phase Commit Notifications
- Recover Transactions
- Pull Transaction
- Query Transaction Outcome

When performing the Create a Transaction and Complete a Transaction actions, the system maps to the TIP Transaction Manager Communicating with Application Facet, as specified in [MS-TIPP], and the External Application maps to the TIP Application Role, as specified in [MS-TIPP].

When communicating with an External Transaction Manager, the system can take either a subordinate transaction manager role or a superior transaction manager role.

When performing the Push Transaction, Two-Phase Commit Notifications, and Recover Transactions actions, the system maps to the TIP Subordinate Transaction Manager Facet, and the External Transaction Manager maps to the TIP Superior Transaction Manager Facet, as specified in [MS-TIPP].

When performing the Pull Transaction and Query Transaction Outcome actions, the system maps to the TIP Superior Transaction Manager Facet, and the External Transaction Manager maps to the TIP Subordinate Transaction Manager Facet, as specified in [MS-TIPP].

The protocol interchanges defined by [MS-TIPP] are performed by using TIP over TCP.

5.4.3.9.3 MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension

IBM's System Network Architecture (SNA) defines Logical Unit type 6.2 (LU 6.2) to support peer communications between LU 6.2 programs. LU 6.2 includes support for coordinating transactions between the LU 6.2 programs so that either both complete or both abort. The MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension as specified in [MS-DTCLU] allows an implementation of LU 6.2 to delegate much of the responsibility for managing a local transaction and coordinating its outcome with a transaction owned by a **remote LU** 6.2 implementation to a transaction manager. The transaction manager manages and coordinates transaction state, while the LU 6.2 implementation remains responsible for all LU to LU communications on the wire. To achieve this behavior, the protocol specified by [MS-DTCLU] provides an alternate resource manager to transaction manager communication protocol. The protocol provides the same conceptual functionality as the MSDTC Connection Manager: OleTx Transaction Protocol (as described in [MS-DTCO]) implementation, though its implementation differs. This corresponds to the following actions from the following component:

External Resource Manager:

- Register
- Enlist

The External Resource Manager component maps to the LU 6.2 Implementation Role specified in [MS-DTCLU]. When communicating with this component, the system is mapped to the Transaction Manager Communicating with an LU 6.2 Implementation Facet specified in [MS-DTCLU]. This has the effect of enlisting the LU 6.2 transacted work in the transaction as a resource. This, in turn, means that in the course of a Perform Transacted Work action, a transaction identifier can be serialized in a form understandable by an LU 6.2 Application Service.

This allows an LU 6.2 Application Service to perform work in the same transactional context as any other application that uses the protocols specified by [MS-DTCO].

5.4.3.9.4 WS-AtomicTransaction (WS-AT) Protocol

The WS-AtomicTransaction Protocol, as specified in [WSAT10] or [WSAT11], provides an alternative application to transaction manager, application service to transaction manager, resource manager to transaction manager, and transaction manager to transaction manager communication protocol. The protocol provides some of the same functionality as the MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the required actions are performed by the WS-AtomicTransaction Protocol. This corresponds to the following actions from the following components:

External Application:

- Create a Transaction
- Complete a Transaction

External Application Service:

- Request a Pull Transaction

External Resource Manager:

- Enlist

External Transaction Manager:

- Pull Transaction
- Two-Phase Commit Notifications

The WS-AtomicTransaction protocol builds on top of WS-Coordination protocol, as specified in section 5.3. When communicating by using the WS-AtomicTransaction protocol, each party may assume a role from either the WS-AtomicTransaction or the WS-Coordination protocol, depending on the action being performed.

When performing the Create a Transaction action, the External Application maps to the Application role, and the system maps to the Activation service role in the Coordinator component as described by the WS-Coordination protocol. This action maps to the Create Coordination Context and Create Coordination Context Response messages described by the WS-Coordination protocol.

When performing the Complete a Transaction action, the External Application maps to the Initiator role, and the system maps to the Coordinator role as described by the WS-AtomicTransaction protocol. This action maps to messages exchanged in the Completion Protocol as described by the WS-AtomicTransaction protocol.

When performing the Request a Pull Transaction action, the External Application Service maps to the Requesting service role in the Application component, and the system maps to the Registration service role in the Coordinator component described by the WS-Coordination protocol. This action maps to the Create Coordination Context, Create Coordination Context Response, Register, and Register Response messages as described by the WS-Coordination protocol.

When performing the Enlist action, the External Resource Manager maps to the Application role, and the system maps to the Activation service role in the Coordinator component as described by the

WS-Coordination protocol. This action maps to the Register and Register Response messages as described by the WS-Coordination protocol.

When performing the Pull Transaction action, the External Transaction Manager maps to the Requesting service role in the Application component, and the system maps to the Registration service role in the Coordinator component as described by the WS-Coordination protocol. This action maps to the Create Coordination Context, Create Coordination Context Response, Register, and Register Response messages described by the WS-Coordination protocol.

When performing the Two-Phase Commit Notifications action, the External Transaction Manager maps to the Coordinator role, and the system maps to the Participant role as described by the WS-AtomicTransaction protocol. This action maps to messages exchanged in the Two-Phase Commit protocol, as described by the WS-AtomicTransaction protocol.

5.4.3.9.5 WS-AtomicTransaction (WS-AT) Protocol Extensions

The WS-AtomicTransaction (WS-AT) Protocol Extensions, as specified in [MS-WSRVCAT], describes the data structure to be exchanged in the messages when performing the Get Extended Whereabouts action.

By using this protocol, external applications can embed transaction propagation information specific to the system into the WS-AtomicTransaction messages. This gives the option to the external application service to communicate with the system by using System Base protocols rather than by using the WS-AtomicTransaction protocol. Using System Base protocols results in increased performance in transaction processing. Further details about this protocol and its usage scenarios are provided in [MS-WSRVCAT].

5.4.3.9.6 MSDTC Connection Manager: OleTx XA Protocol

The MSDTC Connection Manager: OleTx XA Protocol, as specified in [MC-DTCXA], provides an alternative application to transaction manager and transaction manager to transaction manager communication protocol. The protocol provides some of the same functionality as the MSDTC Connection Manager: OleTx Transaction Protocol implementation, but the required actions are performed by using the protocol described in [XOPEN-DTP]. This corresponds to the following actions from the following components:

External Application:

- Create a Transaction
- Complete a Transaction

External Transaction Manager:

- Push Transaction
- Two-Phase Commit Notifications
- Recover Transactions

When performing the Create a Transaction and Complete a Transaction actions External Application maps to the XA Resource Manager Bridge role, and the system maps to the XA Resource Manager Bridge Facet as specified in [MC-DTCXA].

When performing the Push Transaction, Two-Phase Commit Notifications, and Recover Transactions actions External Transaction Manager maps to the XA Superior Transaction Manager role, and the system maps to the XA Subordinate Transaction Manager Facet, as specified in [MC-DTCXA].

5.4.4 Outgoing Interfaces

This section describes the outgoing interfaces of the Transaction Processing Services System. These interfaces are described in terms of the components that interact with the Transaction Processing Services System, and in terms of the conceptual actions in the transaction life cycle that each component performs. A conceptual action may represent the multiple roundtrip communications that are specified in a protocol. Each protocol specifies these interactions in terms of roles that are defined by that protocol.

The following figure summarizes the outgoing interfaces supported by System Base protocols, and the corresponding roles defined by the protocol involved.

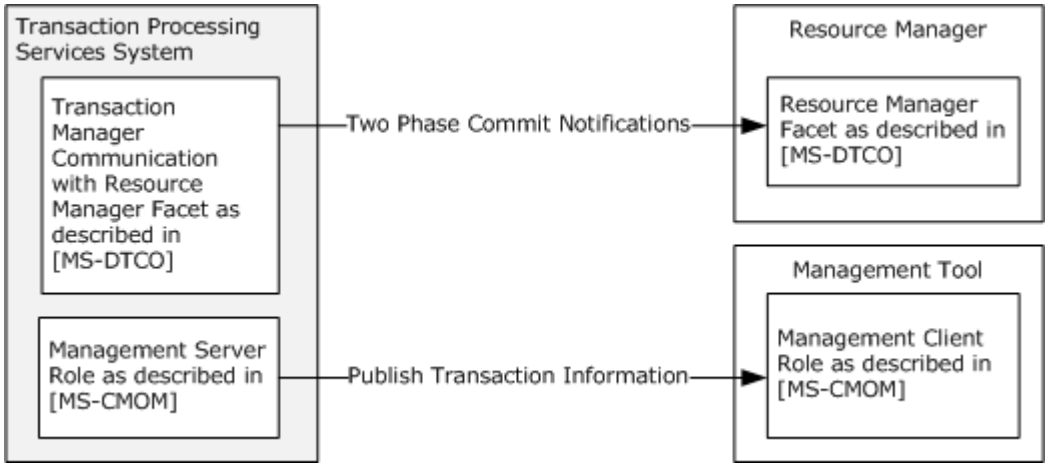

Figure 25: Outgoing interfaces supported by System Base protocols

The following figure summarizes the outgoing interfaces supported by System External protocols, and the corresponding roles defined by the protocol involved.

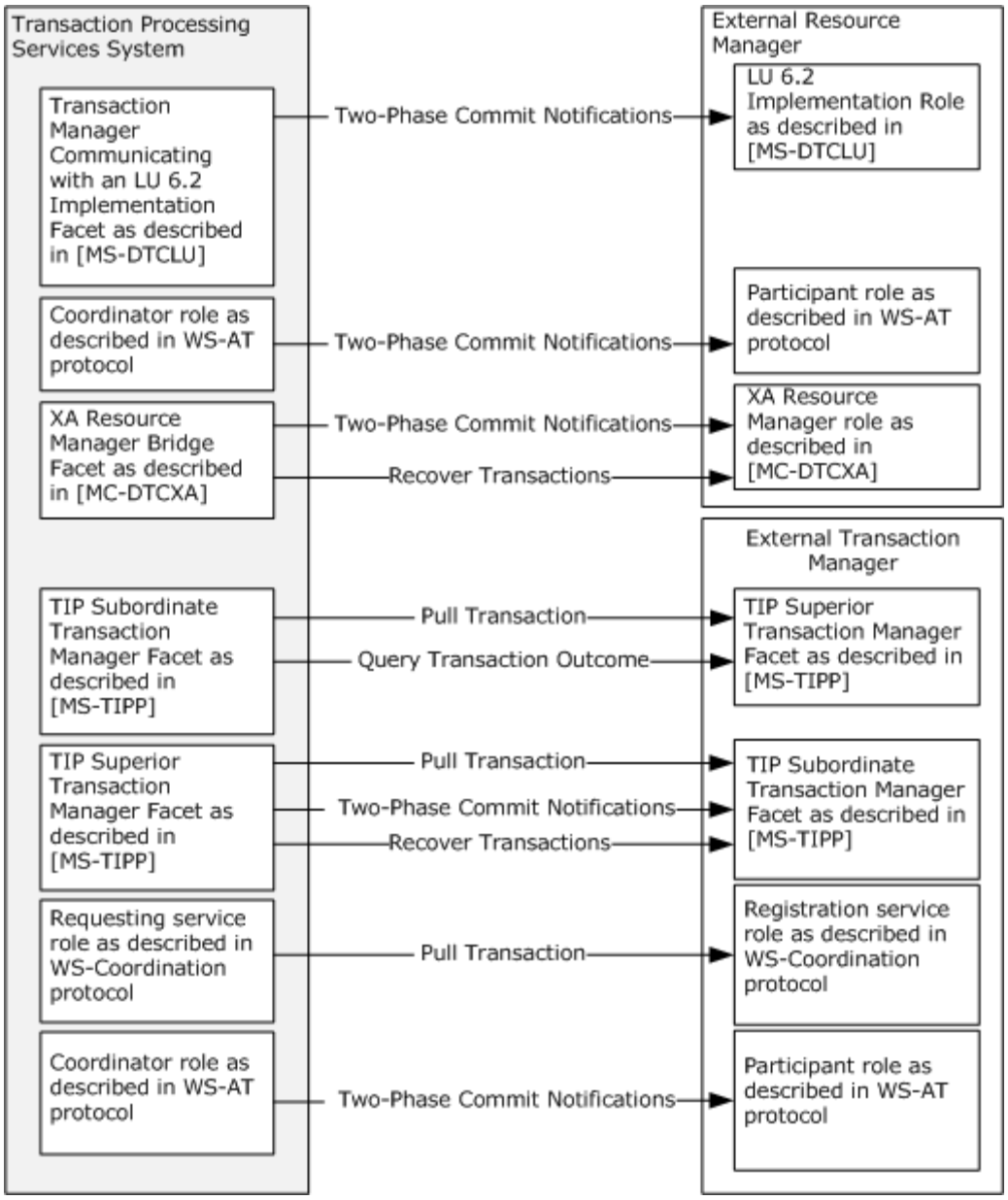

Figure 26: Outgoing interfaces supported by System External protocols

5.4.4.1 Application

The Transaction Processing Services System does not have any outgoing interfaces to an application.

5.4.4.2 Application Service

The Transaction Processing Services System does not have any outgoing interfaces to an application.

5.4.4.3 Resource Manager

The Transaction Processing Services System performs the following actions when communicating with the resource manager component:

Two-Phase Commit Notifications: Requests that the resource manager go through the Two-Phase Commit states.

When communicating with a resource manager, the Transaction Processing Services System maps to the Transaction Manager Communicating with Resource Manager Facet in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

The resource manager's communications with the Transaction Processing Services System are defined by the Resource Manager Facet described in the MSDTC Connection Manager: OleTx Transaction Protocol, as specified in [MS-DTCO].

5.4.4.4 Management Tool

The Transaction Processing Services System performs the following actions when communicating with the management tool component:

Publish Transaction Information: Publishes transaction monitoring information to the management tool. The Transaction Processing Services System does this only for the management tools that have previously subscribed to receive this information.

The publish transaction information action maps to the Management Server Role in MSDTC Connection Manager: OleTx Management Protocol, as specified in [MS-CMOM].

The management tool's communication with the Transaction Processing Services System is defined by the Manager Client Role described in the MSDTC Connection Manager: OleTx Management Protocol, as specified in [MS-CMOM].

5.4.4.5 External Application

Transaction Processing Services System does not have any outgoing interfaces to an external application.

5.4.4.6 External Application Service

Transaction Processing Services System does not have any outgoing interfaces to an external application service.

5.4.4.7 External Resource Manager

The Transaction Processing Services System provides outgoing interfaces for external resource managers that implement the following protocols:

- MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension protocol, as specified in [MS-DTCLU]. This protocol supports the Two-Phase Commit Notifications action. Details are provided in section 5.4.4.9.2.
- WS-AtomicTransaction (WS-AT) Protocol, as specified in [WSAT10]. This protocol supports the Two-Phase Commit Notifications action. Details are provided in section 5.4.4.9.3.
- WS-AtomicTransaction (WS-AT) Protocol, as specified in [WSAT11]. This protocol supports the Two-Phase Commit Notifications action. Details are provided in section 5.4.4.9.3.

- MSDTC Connection Manager: OleTx XA Protocol Specification, as specified in [MC-DTCXA]. This protocol supports the Two-Phase Commit Notifications action, and also a Recover Transactions action. The Recover Transactions action is similar to the Recover Transactions action described in 5.4.1; however, this one is initiated from the system to the external resource manager. Details are provided in section 5.4.4.9.4.

5.4.4.8 External Transaction Manager

The Transaction Processing Services System provides outgoing interfaces for external transaction managers that implement the following protocols:

- Transaction Internet Protocol (TIP) Extensions protocol, as specified in [MS-TIPP]. This protocol supports the Push Transaction, Pull Transaction, Query Transaction Outcome, Recover Transactions in Failed to Notify State, and Two-Phase Commit Notifications actions. Details are provided in section 5.4.4.9.1.
- WS-AtomicTransaction (WS-AT) Protocol, as specified in [WSAT10]. This protocol supports the Pull Transaction and Two-Phase Commit Notifications actions. Details are provided in section 5.4.4.9.3.
- WS-AtomicTransaction (WS-AT) Protocol, as specified in [WSAT11]. This protocol supports the Pull Transaction and Two-Phase Commit Notifications actions. Details are provided in section 5.4.4.9.3.

5.4.4.9 Outgoing Interface Variations through System External Protocols

This section lists the System External protocols supported by the Transaction Processing Services System and describes the alternative outgoing interfaces supported by each protocol for corresponding components participating in a transaction.

5.4.4.9.1 Transaction Internet Protocol (TIP) Extensions

When communicating with an external component by using the Transaction Internet Protocol (TIP) Extensions protocol as specified in [MS-TIPP], the Transaction Processing Services System performs the following actions on the following component:

External Transaction Manager:

- Pull Transaction
- Query Transaction Outcome
- Push Transaction
- Two-Phase Commit Notifications
- Recover Transactions

When communicating with an External Transaction Manager, the system can take either a subordinate transaction manager role or a superior transaction manager role.

When performing the Pull Transaction and Query Transaction Outcome actions, the Transaction Processing Services System maps to the TIP Subordinate Transaction Manager Facet, and the External Transaction Manager maps to the TIP Superior Transaction Manager Facet, as specified in [MS-TIPP].

When performing the Push Transaction, Two-Phase Commit Notifications, and Recover Transactions actions, the Transaction Processing Services System maps to the TIP Superior Transaction Manager Facet, and the External Transaction Manager maps to the TIP Subordinate Transaction Manager Facet, as specified in [MS-TIPP].

The protocol interchanges defined by [MS-TIPP] are performed by using TIP over TCP.

5.4.4.9.2 MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension

When communicating with an external component by using the MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as specified in [MS-DTCLU], the Transaction Processing Services System performs the following action on the following component:

External Resource Manager:

- Two-Phase Commit Notifications

The Two-Phase Commit Notifications action of the external resource manager component maps to the LU 6.2 Implementation Role specified in [MS-DTCLU]. When communicating with this component, the Transaction Processing Services System is mapped to the Transaction Manager Communicating with an LU 6.2 Implementation Facet specified in [MS-DTCLU].

5.4.4.9.3 WS-AtomicTransactions (WS-AT) Protocol

When using WS-AtomicTransaction (WS-AT) Protocol to communicate with an external component, the Transaction Processing Services System performs the following actions on the following components:

External Resource Manager:

- Two-Phase Commit Notifications

External Transaction Manager:

- Pull Transaction
- Two-Phase Commit Notifications

WS-AtomicTransaction protocol builds on top of WS-Coordination protocol, as specified in section 5.3. When communicating by using the WS-AtomicTransaction protocol, each party may assume a role from either the WS-AtomicTransaction or the WS-Coordination protocol, depending on the action being performed.

When performing the Two-Phase Commit Notifications action with the External Resource Manager, the External Resource Manager maps to the Participant role, and the system maps to the Coordinator role described by the WS-AtomicTransaction protocol. This action maps to messages exchanged in the Two-Phase Commit Protocol described by the WS-AtomicTransaction protocol.

When performing the Pull Transaction action, the External Transaction Manager maps to the Registration service role in the Coordinator component, and the system maps to the Requesting service role in the Application component described by the WS-Coordination protocol. This action maps to the Register and Register Response messages described by the WS-Coordination protocol.

When performing the Two-Phase Commit Notifications action with the External Transaction Manager, the External Transaction Manager maps to the Participant role, and the system maps to the

Coordinator role described by the WS-AtomicTransaction protocol. This action maps to messages exchanged in the Two-Phase Commit protocol, as described by the WS-AtomicTransaction protocol.

5.4.4.9.4 MSDTC Connection Manager: OleTx XA Protocol Specification

When using MSDTC Connection Manager: OleTx XA Protocol to communicate with an external component, the Transaction Processing Services System performs the following actions on the following components:

External Resource Manager:

- Two-Phase Commit Notifications
- Recover Transactions

When performing the Two-Phase Notifications and Recover Transactions actions, the system maps to the XA Resource Manager Bridge Facet, and the External Resource Manager maps to the XA Resource Manager role as specified in [MC-DTCXA].

The Recover Transactions action mentioned here is similar to the Recover Transactions action described in section 5.4.1. The difference is that, in this model, the recovery process is initiated by the transaction manager. This action is described in detail in [MC-DTCXA] in section 1.3.1.2.1.

5.4.5 Administration and Configuration

Administration and configuration interfaces for the Transaction Processing Services System are specified in MSDTC Connection Manager: OleTx Management Protocol Specification, as described in [MS-CMOM].

Both System Base Protocols and System External Protocols require the availability of some registry keys to successfully initialize and maintain their operations. Registry keys and their relationship to the protocols described in [MS-DTCO], [MS-DTCM], [MS-DTCLU], [MS-CMP], [MS-CMPO], [MS-TIPP], [MC-DTCXA], and [MS-DTCLU], are described in sections 2.2.3 and 3.3.1.2 of [MS-CMOM].

5.4.5.1 Configuration of WS-AtomicTransactions

The system requires HTTPS connections when communicating over WS-AtomicTransactions, as described in section 7.3. An external entity is allowed to establish this HTTPS connection only if the system has been previously configured to recognize the X509 certificate used by the external entity for the secure connection. The system uses an implementation-specific mechanism to acquire and store the list of X509 certificates that are allowed to communicate with the system by using the WS-AtomicTransactions protocol.

5.5 Failure Scenarios

This section describes the common failure scenarios and specifies the system behavior in such conditions.

5.5.1 Connection Disconnected

A common failure scenario is an unexpected connection breakdown between the system and external entities, or between transaction managers within the system. A disconnection can be caused by the network not being available, or by one of the communicating participants becoming unavailable. In the case where the network is not available, both participants remain active and expect the other party to continue the communication pattern specified by the protocol being executed at the time of the failure. Similarly, in the case where one of the participants is not

available, the active participant expects the communication to proceed as specified by the protocol being executed.

Generally, a protocol detects a connection breakdown failure through either of the following methods:

- By using a timer object that generates an event if the corresponding participant has not responded within a reasonable time span.
- By being notified by the underlying protocol that the connection is disconnected.

When a connection disconnected event is detected, it causes the protocol to tear down all related communications and update any necessary data structures to maintain the system state.

Details about how each protocol detects a connection disconnected event, and how it behaves under this scenario, are provided in the specifications of the member protocols, as listed in section 2.2.

5.5.2 Internal Failures

It is possible that the system or a transaction participant may detect an unrecoverable internal state at any point during the lifetime of a transaction. In such a scenario, if the system or the participant experiencing the internal failure decides that it cannot continue the transaction any more for any reason, it can issue an abort on the existing transactions that are not yet in the second phase of the Two-Phase Commit protocol. The Two-Phase Commit protocol is designed to handle unilateral termination of transactions so that all participants are rolled back to their states before the transaction started. For the transactions that are in the second phase, that means the transaction information is persisted, which in return means that it is recoverable. When the participant comes back to a state where it can resume its operations, it can recover the transaction. Detailed descriptions of unilateral abort and recovery scenarios are provided in [MS-DTCO] sections 1.3.2.1 and 1.3.4, respectively.

5.5.3 System Configuration Corruption or Unavailability

The system relies on the availability and consistency of its configuration data. Configuration consists of the data that determines the behavior of the system under specific conditions or for specific functionality. For example, the configuration can be used to enable or disable certain protocols, or whether the system can span across a network of computers.

If the configuration data is not available, the protocol that needs the configuration data may assume a default value. The protocol specified in [MS-CMOM] describes the system configuration data in section 3.3.1.2 and how it maps to the abstract data models in [MS-CMOM] section 3.3.1, [MS-CMPO] section 1.3.2, [MS-DTCO] section 3.2.1, and [MC-DTCXA] section 3.1.1.

5.5.4 Log Corruption or Unavailability

The log is where the system keeps the transaction state information. Availability and consistency of the log is crucial to guaranteeing atomicity in transaction processing. The system may use implementation-specific mechanisms to make sure the data in the log is reliable. If the log is corrupt, or if it is not available at all, the system cannot process any new durable transactions or respond to recovery requests.

If the log is not recoverable or if it is lost, a new log must be created, which means that any transaction information that was in the previous log is lost. This means that the data or application state that was dependent on the transaction information from the lost log may become corrupt.

6 System Details

This section contains the details that complete the descriptions in earlier sections of the document. These details are needed to understand and implement this system.

6.1 Architectural Details

This section provides a series of examples illustrating the use of the Transaction Processing Services System. The examples are:

- Basic Transaction Life Cycle
- Two-Phase Commit
- Message Is Aborted
- Connection to Resource Manager Breaks Down
- Transaction Manager Recovers after a Connection Failure
- Distributed Transaction Coordination with External Components

Each example is described in detail in the following sections as a set of conceptual actions interchanged between the system and transaction participants. These actions correspond to the incoming and outgoing interfaces described in sections 5.4.3 and 5.4.4. To provide better clarity, some of those actions may be broken into subactions. For instance, the Two-Phase Commit Transactional Message Exchanges example in section 6.1.6.3, individually calls out the prepare, commit, and other Two-Phase Commit protocol messages when when demonstrating the outgoing Two-Phase Commit Notifications action from the system to the Resource Manager as described in section 5.4.4.3.

6.1.1 Basic Transaction Life Cycle

This example shows how transaction life-cycle roles, as indicated in the following figure, interact with each other in the life time of the transaction.

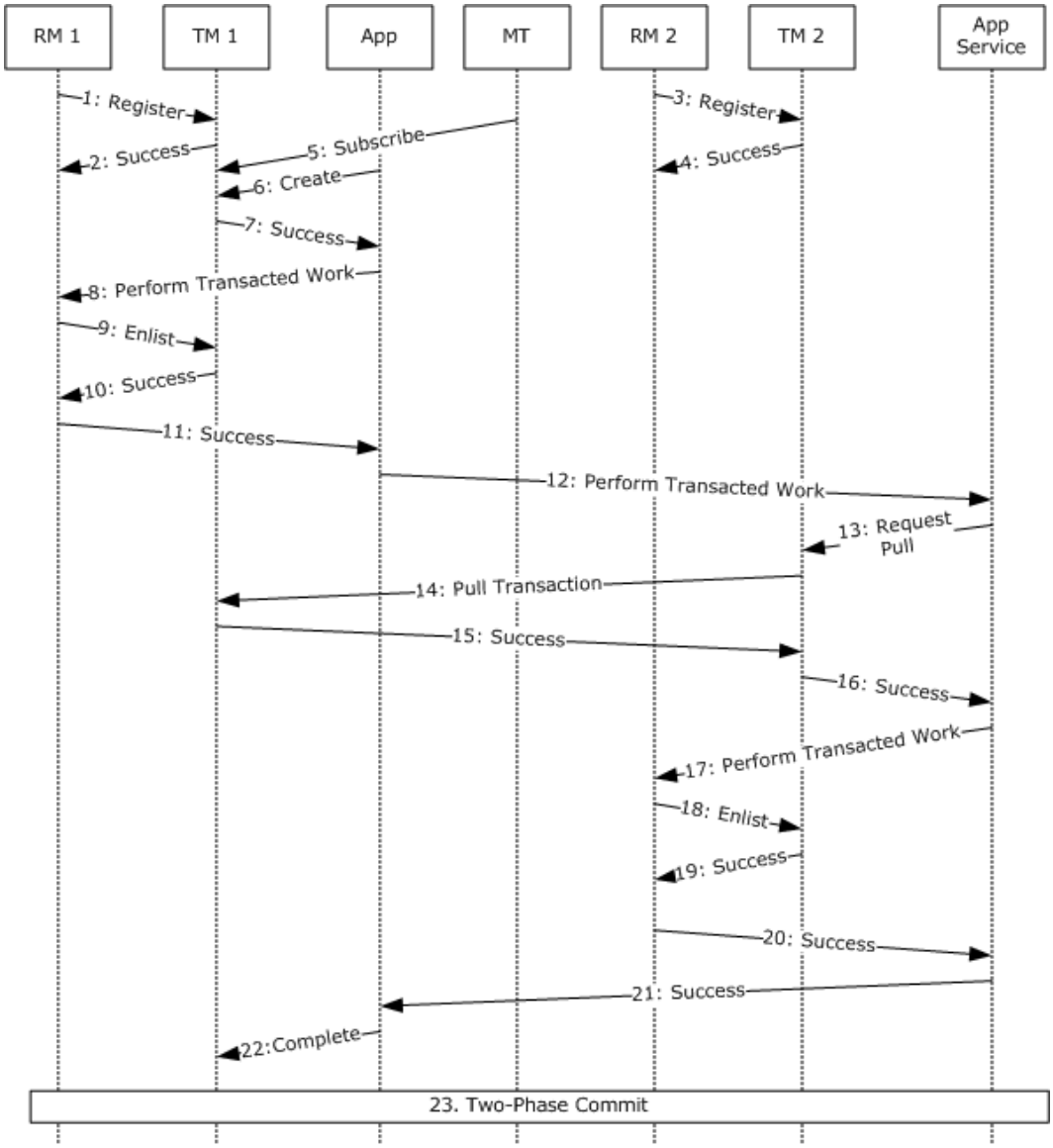

Figure 27: Example 1: Lifetime of a transaction

The message flow in this example is as follows:

1. Resource Manager 1 (RM1) performs a Register action against Transaction Manager 1 (TM1).
2. TM1 acknowledges that RM1 is registered with TM1 as a resource manager.
3. Resource Manager 2 (RM2) performs a Register action against Transaction Manager 2 (TM2).
4. TM2 acknowledges that RM2 is registered with TM2 as a resource manager.
5. The Management Tool (MT) performs a Subscribe for transaction information action against TM1 to monitor the progress of the Two-Phase Commit protocol and to resolve the transaction if it reaches an error state.

6. The Application initiates a Create a Transaction action against TM1.

7. TM1 creates the transaction and returns the transaction identifier to Application, completing the Create a Transaction action initiated in step 6.

8. The Application initiates a Perform Transacted Work action against RM1.

9. RM1 initiates an Enlist action against TM1.

10. TM1 acknowledges that RM1 is enlisted in the transaction, completing the Enlist action initiated in step 9.

11. RM1 reports successful completion of transacted work, completing the Perform Transacted Work action initiated in step 8.

12. The Application initiates a Perform Transacted Work action against Application Service, passing a serialized transaction identifier, which includes the transaction propagation information.

13. Application Service initiates a Request Pull Transaction action against TM2, using the transaction information and propagation information.

14. TM2 initiates a Pull Transaction action against TM1, specifying the serialized transaction identifier.

15. TM1 acknowledges that TM2 is now enlisted in the transaction, completing the Pull Transaction action initiated in step 14.

16. TM2 returns success to the application service, completing the Request Pull Transaction action initiated in step 13.

17. The Application Service initiates a Perform Transacted Work action against RM2.

18. RM2 initiates an Enlist action against TM2.

19. TM2 acknowledges that RM2 is enlisted in the transaction, completing the Enlist action initiated in step 18.

20. RM2 reports successful completion of transacted work, completing the Perform Transacted Work action initiated in step 17.

21. The Application Service responds to the application, completing the Perform Transacted Work action initiated in step 12.

22. The Application performs a Complete transaction action against TM1.

The transaction now begins Phase 1 of the Two-Phase Commit protocol. The transaction tree established by the interactions explained in this example is illustrated in the following figure. TM1 is the root transaction manager.

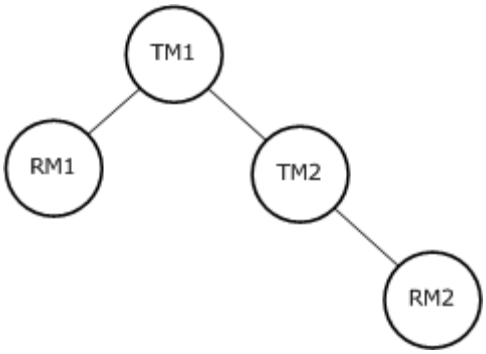

Figure 28: Transaction tree for Example 1

6.1.2 Two-Phase Commit

This example is the continuation of Example 1 and shows how transaction life-cycle roles interact with each other during the Two-Phase Commit protocol. Transaction Manager 1 (TM1) is the root transaction manager in the transaction tree and it coordinates the Two-Phase Commit protocol for all participants.

Interactions explained in this example are illustrated in the following figure.

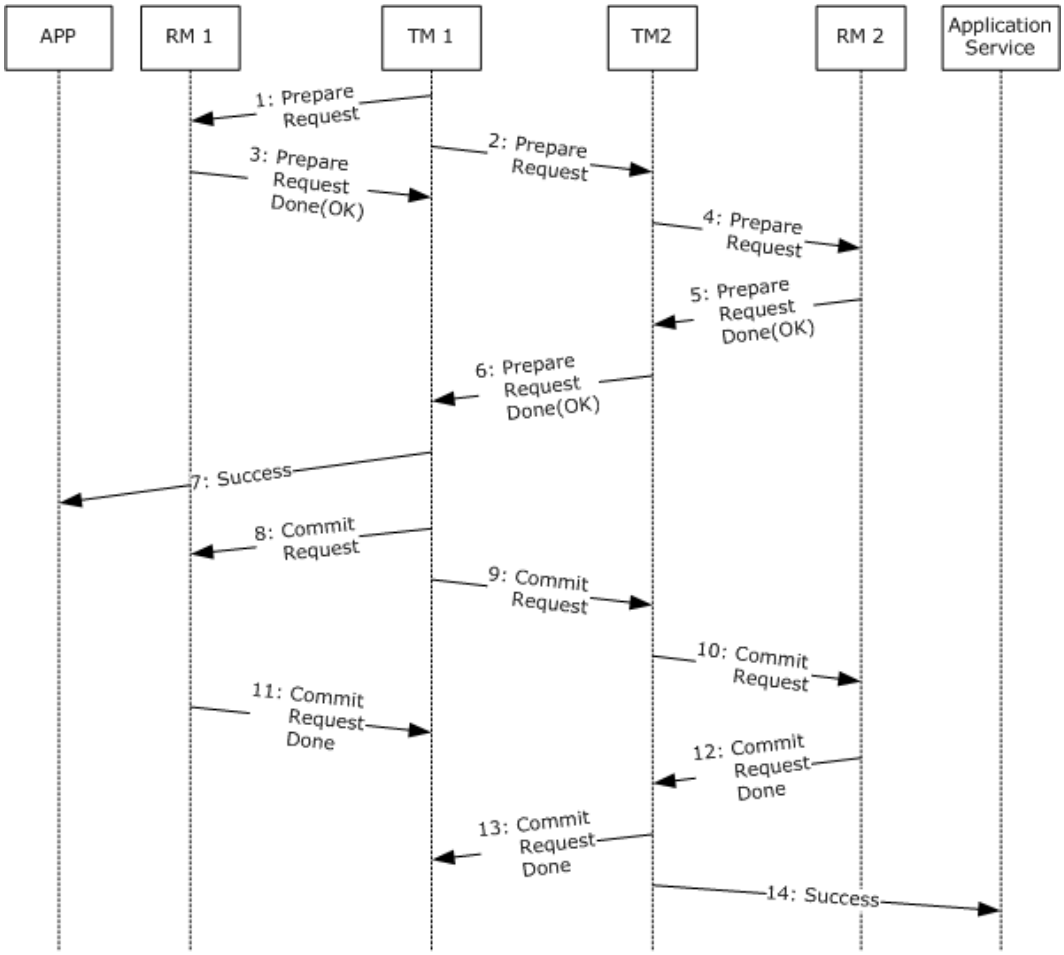

Figure 29: Example 2: Two-Phase Commit protocol sequence diagram

The messages exchanged in this example are contained within the Two-Phase Commit Notifications action between the system and participating roles.

The message flow shown in Example 2 is as follows:

1. Transaction Manager 1 (TM1) sends a PrepareRequest message to Resource Manager 1 (RM1).
2. TM1 sends a PrepareRequest message to Transaction Manager 2 (TM2).
3. RM1 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 1.
4. TM2 sends a PrepareRequest message to Resource Manager 2 (RM2).
5. RM2 sends a PrepareRequestDone message to TM2 indicating that the prepare request was finished successfully (OK), completing step 4.
6. TM2 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 2.

7. TM1 decides the outcome as Commit and sends a Success message to APP indicating that the transaction is committed, completing step 22 in Example 1.

8. TM1 sends a CommitRequest message to RM1.

9. TM1 sends a CommitRequest message to TM2.

10. TM2 sends a CommitRequest message to RM2.

11. RM1 sends a CommittedRequestDone message to TM1, completing step 8.

12. RM2 sends a CommittedRequestDone message to TM2, completing step 10.

13. TM2 sends a CommittedRequestDone message to TM1, completing step 9.

14. TM2 sends a Success message to the Application Service, notifying it of the completion of the Two-Phase Commit sequence.

6.1.3 Transaction Is Aborted

This example is the continuation of Example 1 and shows how transaction life-cycle roles interact with each other during the Two-Phase Commit protocol where one of the participants aborts the transaction.

The transaction tree for this example is illustrated in Example 1. Transaction Manager 1 (TM1) is the root transaction manager in the transaction tree and it coordinates the Two-Phase Commit protocol for all participants.

Interactions explained in this example are illustrated in the following figure.

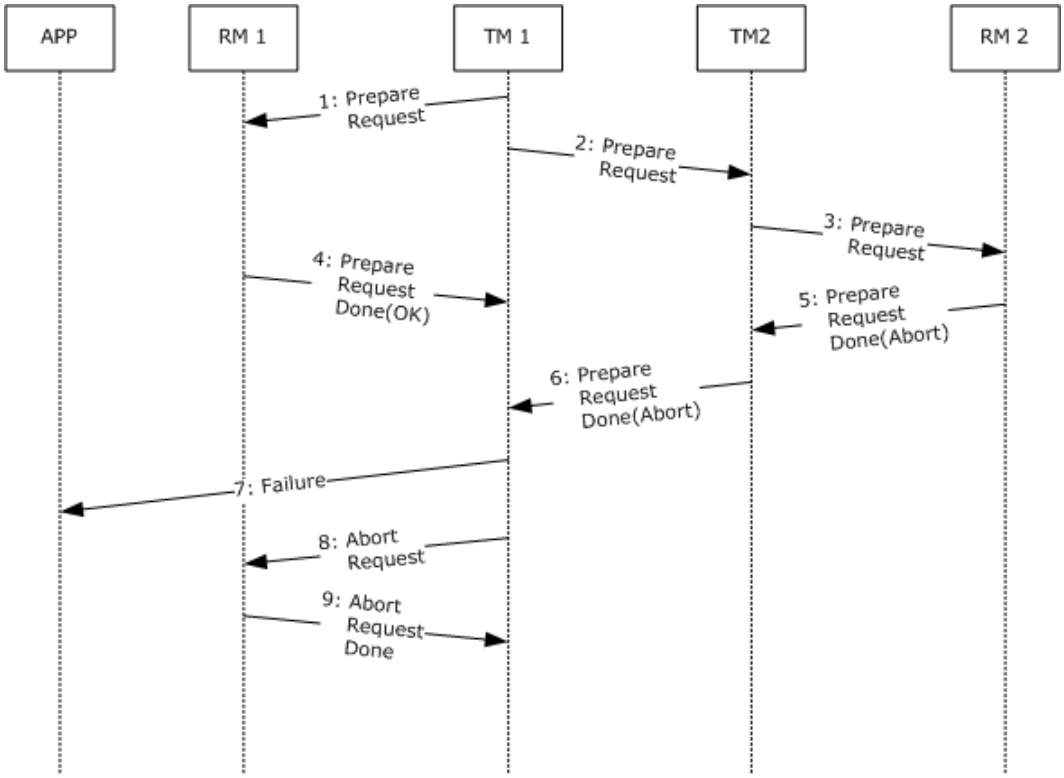

Figure 30: Example 3: Transaction is aborted during Two-Phase Commit protocol

The messages exchanged in this example are contained within the Two-Phase Commit Notifications action between the system and participating roles.

The message flow shown in this example is as follows:

1. Transaction Manager (TM1) sends a PrepareRequest message to Resource Manager 1 (RM1).
2. TM1 sends a PrepareRequest message to Transaction Manager 2 (TM2).
3. TM2 sends a PrepareRequest message to Resource Manager 2 (RM2).
4. RM1 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 1.
5. RM2 sends a PrepareRequestDone message to TM2 indicating that the prepare request was finished unsuccessfully (Abort), completing step 3.
6. TM2 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished unsuccessfully (Abort), completing step 2.
7. TM1 determines the outcome as Abort and sends a Failure message to APP indicating that the transaction is aborted, completing step 22 in Example 1.
8. TM1 sends an AbortRequest message to RM1.
9. RM1 sends an AbortRequestDone to TM1, completing step 8.

6.1.4 Connection to Resource Manager Breaks Down

This example is the continuation of Example 1 and shows how transaction life-cycle roles interact with each other during the Two-Phase Commit protocol where connection to a resource manager breaks down, and how the resource manager drives recovery afterwards.

The transaction tree for this example is illustrated in Example 1. Transaction Manager 1 (TM1) is the root transaction manager in the transaction tree and it coordinates the Two-Phase Commit protocol for all participants. The following figure illustrates the broken connection to Resource Manager 2.

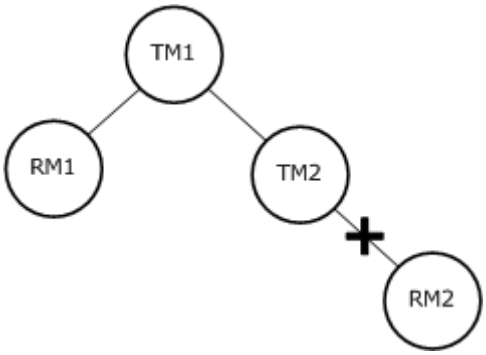

Figure 31: Connection to Resource Manager 2 breaks down

Interactions explained in this example are shown in the following figure.

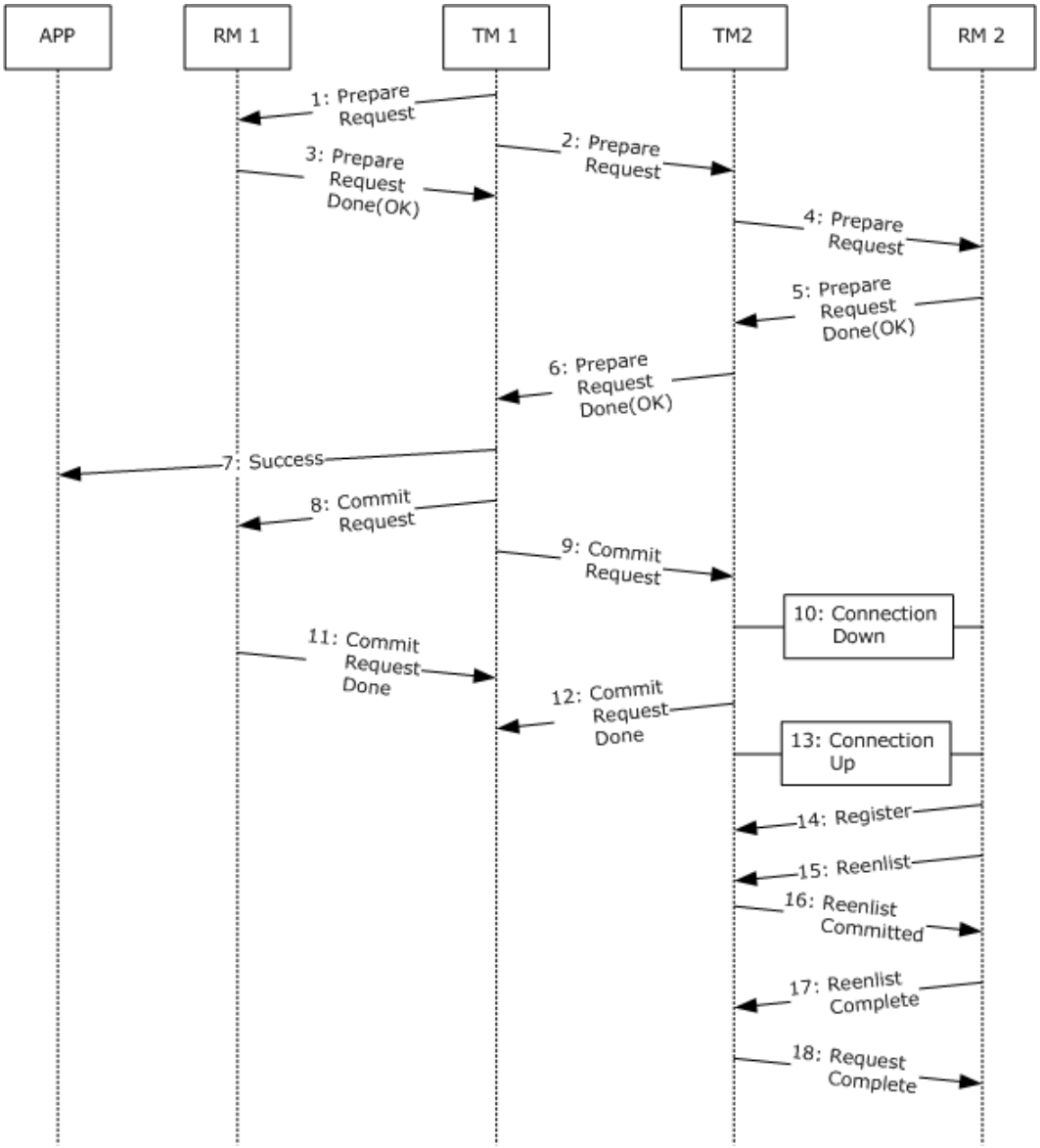

Figure 32: Example 4: Resource Manager connection breaks down during Two-Phase Commit protocol

The PrepareRequest, PrepareRequestDone, CommitRequest and CommitRequestDone messages exchanged in this example are contained within the Two-Phase Commit Notifications action between the system and participating roles.

The message flow shown in this example is as follows:

1. Transaction Manager 1 (TM1) sends a PrepareRequest message to Resource Manager 1 (RM1).

2. TM1 sends a PrepareRequest message to Transaction Manager 2 (TM2).

3. RM1 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 1.

4. TM2 sends a PrepareRequest message to Resource Manager 2 (RM2).

5. RM2 sends a PrepareRequestDone message to TM2 indicating that the prepare request was finished successfully (OK), completing step 4.

6. TM2 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 2.

7. TM1 determines the outcome as Commit and sends a Success message to APP indicating that the transaction is committed, completing step 22 in Example 1.

8. TM1 sends a CommitRequest message to RM1.

9. TM1 sends a CommitRequest message to TM2.

10. Connection from TM2 to RM2 goes down. As a result, TM2 cannot send a CommitRequest message to RM2.

11. RM1 sends a CommitRequestDone message to TM1, completing step 8.

12. TM2 sends a CommitReguestDone message to TM1, completing step 9.

13. RM2 comes back up and finds the transaction in the in-doubt state.

14. RM2 performs a Register action against TM2.

15. RM2 performs a Query Transaction Outcome action against TM2 by sending a Reenlist message.

16. TM2 sends a ReenlistCommitted message to RM2 to indicate the outcome of the transaction is committed.

17. RM2 sends a ReenlistComplete message to TM2 to indicate it has recovered its transactions.

18. TM2 sends a RequestComplete message to RM2 to confirm the completion of recovery.

6.1.5 Transaction Manager Recovers after a Connection Failure

This example is the continuation of Example 1 and shows how transaction life-cycle roles interact with each other during the Two-Phase Commit protocol where connection to a subordinate transaction manager breaks down, and how the transaction manager drives recovery afterwards.

The transaction tree for this example is illustrated in Example 1. Transaction Manager 1 (TM1) is the root transaction manager in the transaction tree and it coordinates the Two-Phase Commit protocol for all participants. The following figure illustrates the broken connection to Transaction Manager 2.

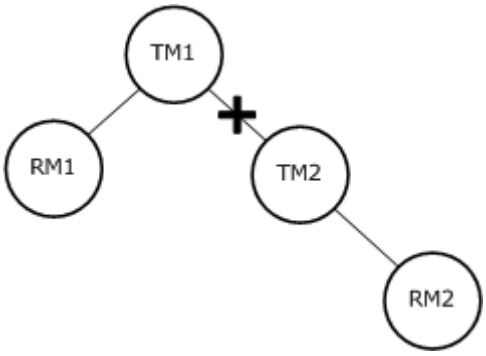

Figure 33: Connection to Transaction Manager 2 breaks down

Interactions explained in this example are illustrated in the following figure.

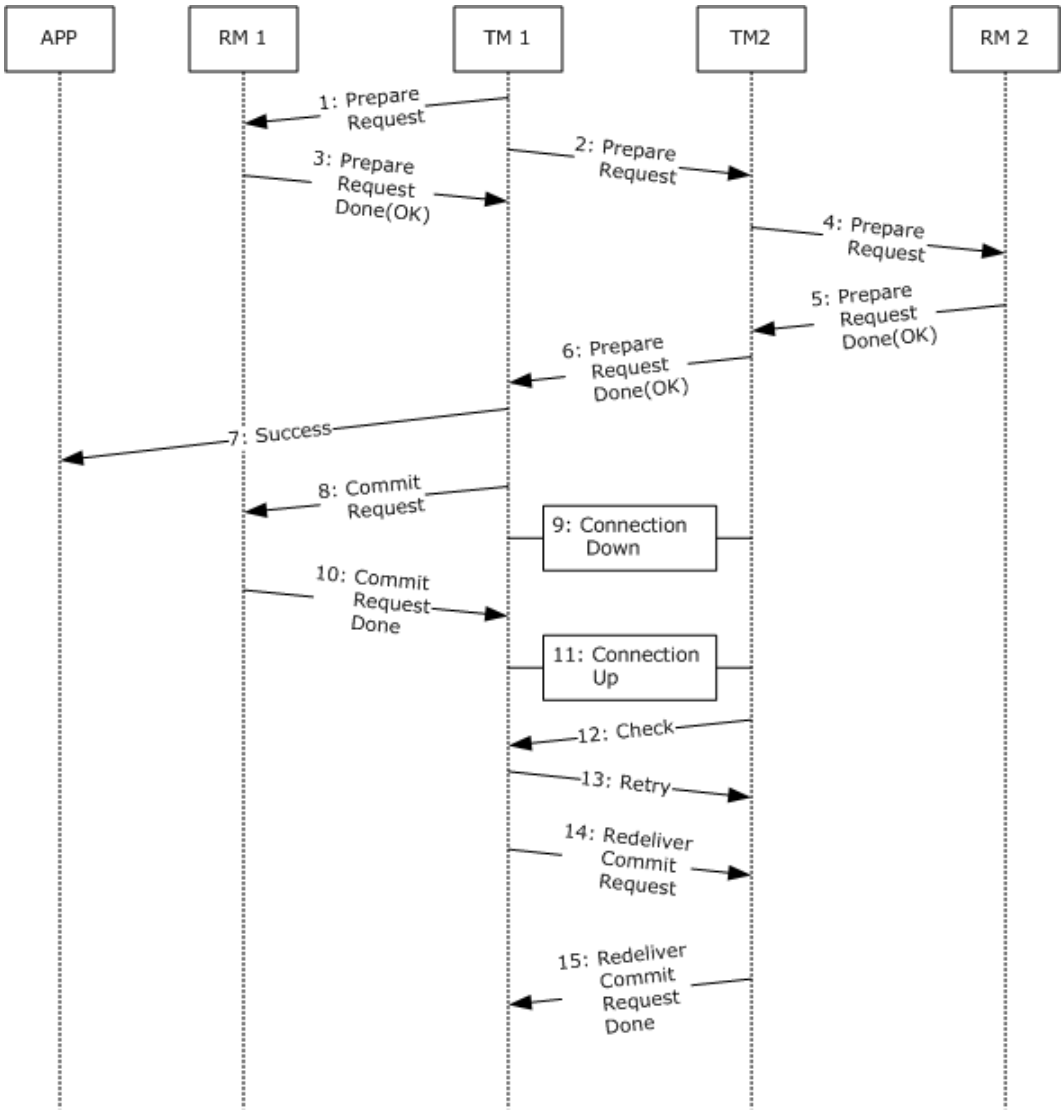

Figure 34: Example 5: Transaction Manager recovery after a connection failure during Two-Phase Commit protocol

The PrepareRequest, PrepareRequestDone, CommitRequest and CommitRequestDone messages exchanged in this example are contained within the Two-Phase Commit Notifications action between the system and participating roles.

The message flow for Example 5 is as follows:

1. Transaction Manager (TM1) sends a PrepareRequest message to Resource Manager 1 (RM1).

2. TM1 sends a PrepareRequest message to Transaction Manager 2 (TM2).

3. RM1 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 1.

4. TM2 sends a PrepareRequest message to Resource Manager 2 (RM2).

5. RM2 sends a PrepareRequestDone message to TM2 indicating that the prepare request was finished successfully (OK), completing step 4.

6. TM2 sends a PrepareRequestDone message to TM1 indicating that the prepare request was finished successfully (OK), completing step 2.

7. TM1 determines the outcome as Commit and sends a Success message to APP indicating that the transaction is committed, completing step 22 in Example 1.

8. TM1 sends a CommitRequest message to RM1.

9. Connection from TM1 to TM2 goes down. As a result TM1 cannot send a CommitRequest message to TM2.

10. RM1 sends a CommitRequestDone message to TM1 completing step 8.

11. Connection from TM1 to TM2 gets reestablished.

12. TM2 initiates a CheckAbort connection with TM1, and sends a Check message to TM1 on that session to determine whether the transaction is aborted.

13. TM1 sends a Retry message to TM2 indicating that the transaction is not aborted. TM2 waits for commit request.

14. TM1 sends a RedeliverCommitRequest message to TM2.

15. TM2 sends a RedeliverCommitRequestDone message to TM1 completing step 14.

This sequence causes the TM2 to make a record of this transaction as committed. At a later time, RM2 will drive its own recovery sequence. As described in [MS-DTCO] section 1.3.4.2, the resource manager is always responsible for initiating recovery with its transaction manager.

6.1.6 Distributed Transaction Coordination with External Components

6.1.6.1 Distributed Transaction Coordination with External Components Precursory Message Exchanges

This example shows the message exchanges that occur at the initialization time of the participants.

In these precursory message exchanges, a Management Tool (MT) subscribes for transaction information with its Transaction Manager (TM1), a Resource Manager (RM1) registers with its Transaction Manager (TM1), and an LU 6.2 Implementation (LU1) registers with its Transaction Manager (TM3).

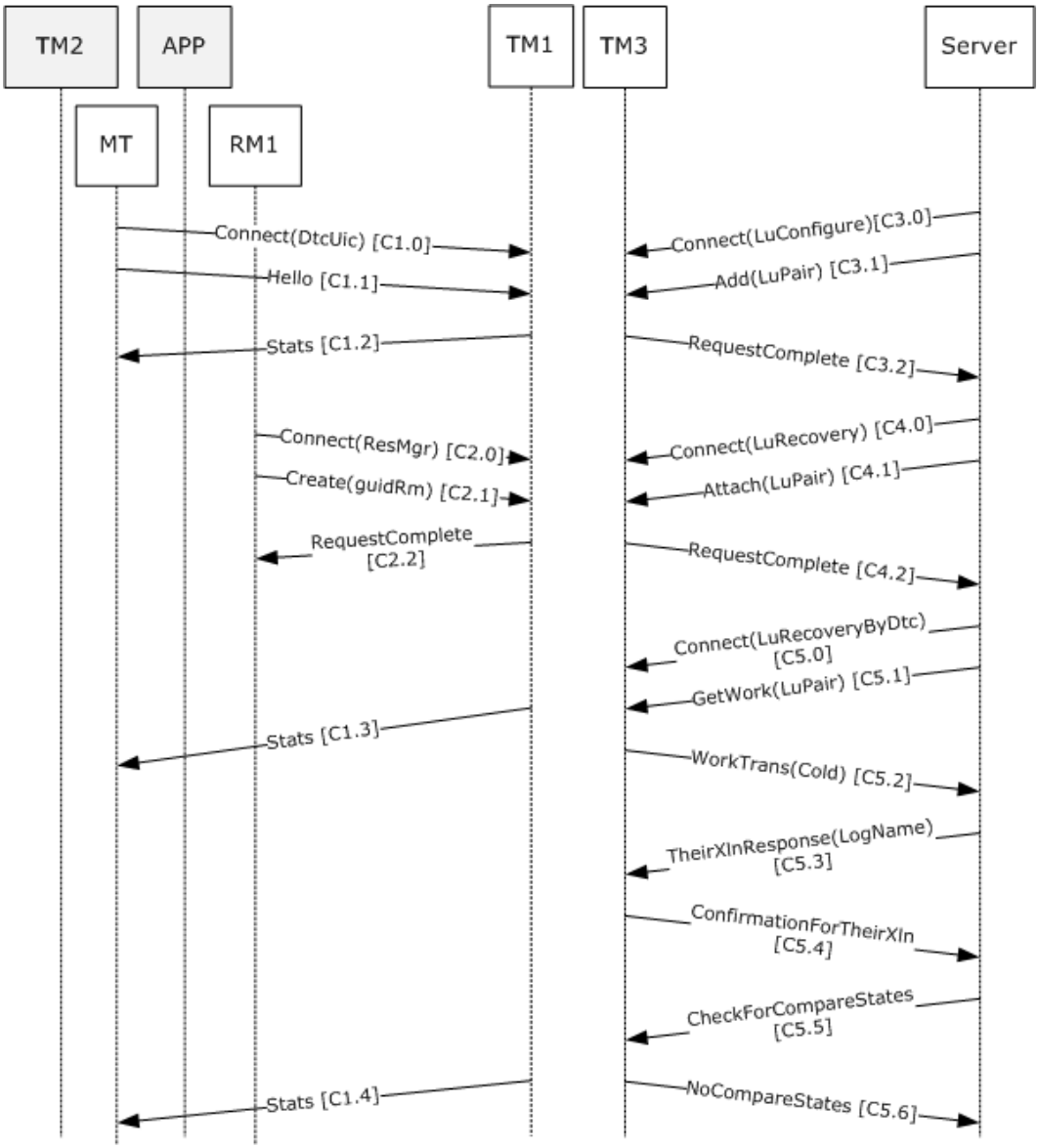

Figure 35: Example 6 sequence diagram: precursory message exchanges in a distributed transaction with external components

6.1.6.1.1 Subscribing a Management Tool for Transaction Information

This message exchange demonstrates how a Management Tool (MT) subscribes for transaction information from a Transaction Manager (TM1) by using protocols specified in the MSDTC Connection Manager: OleTx Management Protocol, as specified in [MS-CMOM]. The message flow is as follows:

1. [C1.0] Connect(DtcUic): MT initiates a CONNTYPE_TXUSER_DTCUIC connection on an MSDTC Connection Manager: OleTx Management Protocol session with TM1.

2. [C1.1] Hello: MT sends an MTAG_HELLO message to TM1.

3. [C1.2] Stats: When TM1 receives the connection request, it starts a timer (if one does not exist) and adds MT to its list of Management Client Role connections. Each time the timer expires, TM1 sends a MSG_DTCUIC_STATS message to MT. If TM1 is tracking any active transactions, then TM1 also sends an MSG_DTCUIC_TRANLIST message. In this example, no MSG_DTCUIC_TRANLIST message is sent.

MT continues to receive these messages from TM1 until it closes the connection by initiating the disconnect sequence [C1.3; C1.4].

6.1.6.1.2 Registering a Resource Manager

This message exchange demonstrates how a Resource Manager (RM1) registers with its Transaction Manager (TM1) by using protocols as specified in [MS-DTCO]. The message flow is as follows:

1. [C2.0]Connect(ResMgr): RM1 initiates a CONNTYPE_TXUSER_RESOURCEMANAGER connection on an MSDTC Connection Manager: OleTx Transports Protocol (as specified in [MS-CMPO]) session with TM1.

2. [C2.1]Create(guidRm): RM1 sends a TXUSER_RESOURCEMANAGER_MTAG_CREATE message specifying a GUID that uniquely identifies the Resource Manager (guidRm) to TM1.

3. [C2.2]RequestComplete: When TM1 receives the TXUSER_RESOURCEMANAGER_MTAG_CREATE message, TM1 adds RM1 to its list of registered Resource Managers and sends a TXUSER_RESOURCEMANAGER_MTAG_REQUEST_COMPLETE message to RM1.

RM1 continues to maintain this connection to enable the creation of new enlistments in transactions and its participation in two-phase commit processing.

6.1.6.1.3 Registering an LU 6.2 Implementation

This message exchange demonstrates how an LU 6.2 Implementation (LU1) registers with its Transaction Manager (TM3) by using protocols specified in the MSDTC Connection Manager: OleTx Transaction Protocol Logical Unit Mainframe Extension, as specified in [MS-DTCLU]. Registering involves three separate message exchanges as follows:

- Configuring an LU Name Pair.
- Registering as the recovery process for the LU Name Pair.
- Performing **cold recovery** for the LU Name Pair.

6.1.6.1.4 Configuring an LU Name Pair

This message exchange demonstrates how LU1 configures an LU Name Pair with TM3. The message flow is as follows:

1. [C3.0]Connect(LuConfigure): LU1 initiates a CONNTYPE_TXUSER_DTCLUCONFIGURE connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

2. [C3.1]Add(LuPair): LU1 sends a TXUSER_DTCLURMCONFIGURE_MTAG_ADD message specifying the LU Name Pair (LuPair) to TM3.

3. [C3.2]RequestComplete: When TM3 receives the TXUSER_DTCLURMCONFIGURE_MTAG_ADD message, TM3 adds the LU Name Pair to its table of LU Name Pairs, and sends a TXUSER_DTCLURMCONFIGURE_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMCONFIGURE_MTAG_REQUEST_COMPLETED response from TM3, LU1 initiates the disconnect sequence.

6.1.6.1.5 Registering as the Recovery Process for an LU Name Pair

This message exchange demonstrates how LU1 registers as the recovery process for an LU Name Pair with TM3. The message flow is as follows:

1. [C4.0]Connect(LuRecovery): LU1 initiates a CONNTYPE_TXUSER_DTCLURECOVERY connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

2. [C4.1]Attach (LuPair): LU1 sends a TXUSER_DTCLURMRECOVERY_MTAG_ATTACH message to TM3 specifying an LU Name Pair (LuPair) which was previously configured with TM3.

3. [C4.2]RequestComplete: When TM3 receives the TXUSER_DTCLURMRECOVERY_MTAG_ATTACH message, TM3 registers the connection's MS-CMPO session for all Recovery Processing associated with the LU Name Pair, and sends a TXUSER_DTCLURMRECOVERY_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMRECOVERY_MTAG_REQUEST_COMPLETED message from the TM3, LU1 continues to maintain the connection to enable recovery processes to be initiated and to enable the creation of new enlistments in the transactional work associated with the LU Name Pair.

6.1.6.1.6 Performing Cold Recovery for an LU Name Pair

This message exchange demonstrates how LU1 performs cold recovery for an LU Name Pair with TM3. The message flow is as follows:

1. [C5.0]Connect(LuRecoveryByDtc): LU1 initiates a CONNTYPE_TXUSER_DTCLURECOVERYINITIATEDBYDTC connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

2. [C5.1]GetWork (LuPair): LU1 sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_GETWORK message to TM3 specifying the LU Name Pair (LuPair) for which LU1 registered as the recovery process.

3. [C5.2]WorkTrans (Cold): When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_GETWORK message, TM3 determines that it needs to perform cold recovery (Cold) for the LU Name Pair, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_WORK_TRANS message to LU1.

4. [C5.3]TheirXlnResponse (LogName): When LU1 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_WORK_TRANS message, LU1 exchanges log names with the Remote LU, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_THEIR_XLN_RESPONSE message specifying the Remote LU Log Name (LogName) to TM3.

5. [C5.4]ConfirmationForTheirXln: When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_THEIR_XLN_RESPONSE message, TM3 verifies that the reported state of the Remote LU is consistent with TM3's state, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CONFIRMATION_FOR_THEIR_XLN message to LU1.

6. [C5.5]CheckForCompareStates: When LU1 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CONFIRMATION_FOR_THEIR_XLN message,

LU1 sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CHECK_FOR_COMPARESTATES message to TM3 to query whether recovery work is required for any LU 6.2 transactional work involving the LU Name Pair.

7. [C5.6]NoCompareStates: When TM3 receives the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_CHECK_FOR_COMPARESTATES message, TM3 checks the local and remote transactional state, and sends a TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_NO_COMPARESTATES message to LU1.

When LU1 has received the TXUSER_DTCLURECOVERYINITIATEDBYDTC_MTAG_NO_COMPARESTATES message, no further messages are to be sent using this connection, and LU1 initiates the disconnect sequence.

6.1.6.2 Application-Driven Transactional Message Exchanges

This example shows the message exchanges that occur when an application creates a transaction and other participants enlist in the transaction. The following figure provides further illustration.

In these sets of message exchanges: an Application (APP) begins a transaction, enlists a Resource Manager (RM1), pushes the transaction to a TIP Transaction Manager (TM2), and propagates the transaction to an LU 6.2 Implementation (LU1), which enlists the transaction, all in preparation for committing the transaction. These message exchanges induce related message exchanges between the three Transaction Managers (TM1, TM2, and TM3), and the effects of these message exchanges are reflected in the message exchanges between Transaction Manager (TM1) and the Management Tool (MT).

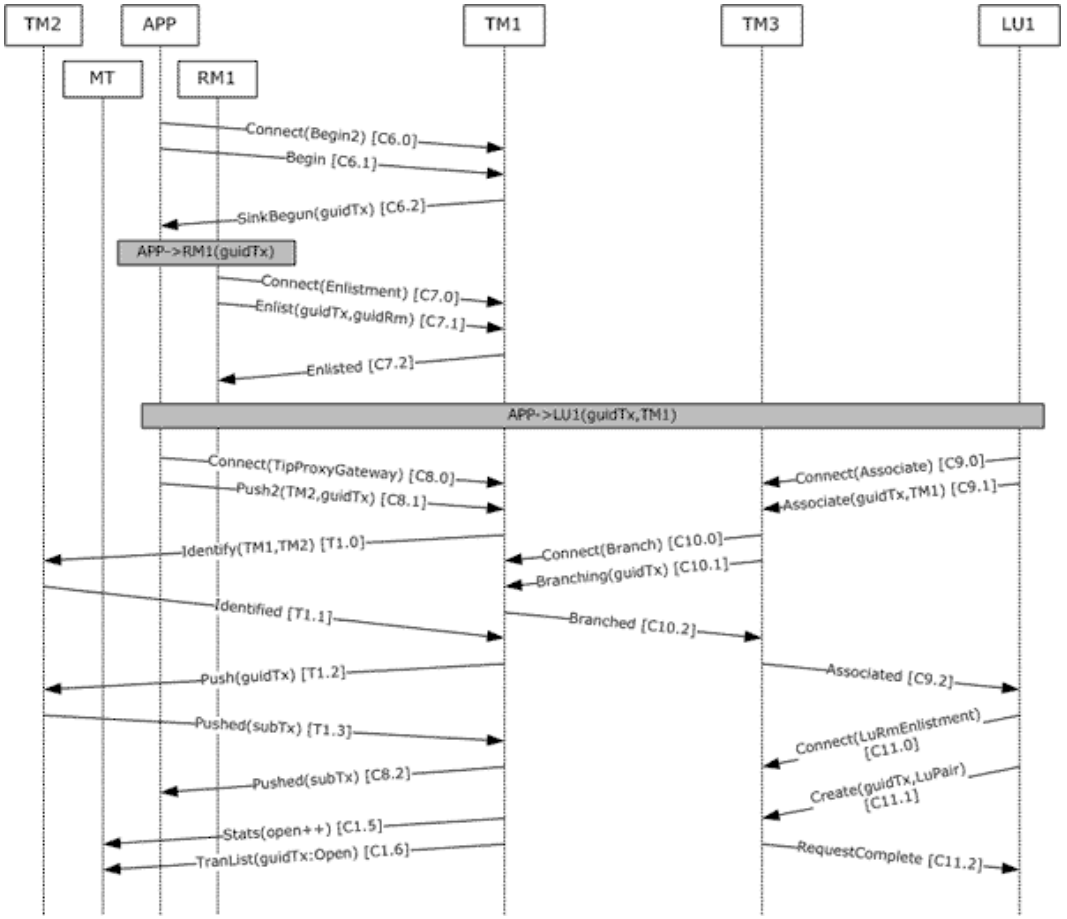

Figure 36: Example 6 sequence diagram: transactional message exchanges before Two-Phase Commit in a distrubted transaction

6.1.6.2.1 Beginning an MSDTC Connection Manager: OleTx Transaction Protocol Transaction

This message exchange demonstrates how an Application (APP) creates a transaction with its Transaction Manager (TM1) by using protocols specified in [MS-DTCO]. The message flow is as follows:

1. [C6.0]Connect(Begin2): APP initiates a CONNTYPE_TXUSER_BEGIN2 connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.

2. [C6.1]Begin: APP sends a TXUSER_BEGIN2_MTAG_BEGIN message to TM1 specifying the isolation level, timeout, transaction description, and isolation flag.

3. [C6.2]SinkBegun(guidTx): When TM1 receives the TXUSER_BEGIN2_MTAG_BEGIN message, TM1 creates a transaction object with a globally unique identifier (guidTx) and sends a TXUSER_BEGIN2_MTAG_SINK_BEGUN message to APP, and adds the transaction to its list of known transaction objects.

When APP receives the TXUSER_BEGIN2_MTAG_BEGUN message from TM1, the transaction (guidTx) has begun. APP is now free to propagate this transaction to Transaction Managers,

Resource Managers, and Application Services to perform work as part of the transaction, as long as it maintains the CONNTYPE_TXUSER_BEGIN2 connection. Eventually, APP determines whether to commit or abort the transaction. If APP disconnects the connection before committing or aborting the transaction, then TM1 will abort the transaction.

6.1.6.2.2 Enlisting a Resource Manager by Using Protocols Specified in [MS-DTCO]

This message exchange demonstrates how a Resource Manager (RM1) enlists on a transaction with its Transaction Manager (TM1) by using protocols specified in [MS-DTCO]. This message exchange assumes that APP has used an out-of-band mechanism (for example, application API) to request RM1 to perform work as part of the transaction [APP->RM1(guidTx)]. This message exchange also assumes that RM1 has registered with TM1 as shown in section 6.1.6.1.2. The message flow is as follows:

1. [C7.0]Connect(Enlistment): RM1 initiates a CONNTYPE_TXUSER_ENLISTMENT connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.

2. [C7.1]Enlist(guidTx, guidRM): RM1 sends a TXUSER_ENLISTMENT_MTAG_ENLIST message to TM1 specifying the transaction GUID (guidTx), and the GUID that uniquely identifies itself (guidRm).

3. [C7.2]RequestComplete: When TM1 receives the TXUSER_ENLISTMENT_MTAG_ENLIST message, TM1 enlists RM1 in the requested transaction, adds RM1 to its list of subordinates for the transaction, and sends a TXUSER_ENLISTMENT_MTAG_ENLISTED message to RM1.

RM1 continues to maintain the connection and waits for two-phase commit notifications from TM1.

6.1.6.2.3 Propagating the Transaction to a TIP Transaction Manager by Using Protocols Specified in [MS-DTCM] and [MS-TIPP] via Push Semantics

This message exchange demonstrates how an Application (APP) requests its Transaction Manager (TM1) by using protocols specified in [MS-DTCM] to propagate the transaction from the Transaction Manager (TM1) to the TIP Transaction Manager (TM2) by using the TIP protocol.

This message exchange assumes that the Application (APP) has acquired through some out-of-band mechanism (for example, application API) the TIP URL of the TIP Transaction Manager (TM2). The message flow is as follows:

1. [C8.0]Connect(TipProxyGateway): APP initiates a CONNTYPE_TXUSER_TIPPROXYGATEWAY connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.

2. [C8.1]Push2(guidTx,TM2): APP sends a TXUSER_TIPPROXYGATEWAY_MTAG_PUSH2 user message specifying the transaction GUID (guidTx), and the TIP URL of TM2.

3. [T1.0]Identify(TM1,TM2): When TM1 receives the TXUSER_TIPPROXYGATEWAY_MTAG_PUSH2 message, TM1 locates the transaction and creates a new TIP connection with TM2 in the INITIAL state by using the protocol specified in [MS-TIPP]. TM1 uses the TIP URL specified in the message to create the TIP connection over the TCP transport session established with TM2, and sends an IDENTIFY command to TM2 specifying TM1's primary Transaction Manager address and TM2's secondary Transaction Manager address.

4. [T1.1]Identified: When TM2 receives the IDENTIFY command, TM2 sends an IDENTIFIED command to TM1, and the state of the TIP connection is changed to IDLE.

5. [T1.2]Push(guidTx): When TM1 receives the IDENTIFIED command, TM1 sends a PUSH command to TM2 specifying the Primary's Transaction Identifier (guidTx).

6. [T1.3]Pushed(subTx): When TM2 receives the PUSH command, TM2 adds the transaction to its list of transaction objects with a newly created transaction identifier (subTx), sends a PUSHED command to TM1, and the state of the TIP connection changes to ENLISTED.

7. [C8.2]Pushed(subTx): When TM1 receives the PUSHED command, TM1 sends a TXUSER_TIPPROXYGATEWAY_MTAG_PUSHED message APP specifying TM2's Transaction Identifier (subTx).

When APP receives the TXUSER_TIPPROXYGATEWAY_MTAG_PUSHED message, APP initiates the disconnect sequence on the CONNTYPE_TXUSER_TIPPROXYGATEWAY connection.

6.1.6.2.4 Propagating the Transaction to an MSDTC Connection Manager: OleTx Transaction Protocol Application Service by Using Pull Semantics

This message exchange demonstrates how an Application (APP) propagates the transaction to an LU 6.2 Implementation (LU1) by using protocols specified in [MS-DTCO] via pull semantics; this enables the LU1 to subsequently enlist on the transaction by using protocols specified in [MS-DTCLU].

APP starts this exchange by sending an out-of-band message (for example, application API) to LU1 requesting it to pull the transaction (guidTx) from TM1 [APP->LU1(guidTx,TM1)]. The message flow is as follows:

1. [C9.0]Connect(Associate): LU1 initiates a CONNTYPE_TXUSER_ASSOCIATE connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

2. [C9.1]Associate(guidTx,TM1): LU1 sends a TXUSER_ASSOCIATE_MTAG_ASSOCIATE message to TM3 specifying the Transaction Identifier (guidTx) and sufficient information (TM1's Machine Name, Endpoint GUID) to establish an MSDTC Connection Manager: OleTx Transports Protocol session with TM1, if a session is not already established.

3. [C10.0]Connect(Branch): When TM3 receives the TXUSER_ASSOCIATE_MTAG_ASSOCIATE message, TM3 attempts to locate the transaction in its list of transaction objects by using the Transaction Identifier (guidTx). Because the transaction object is not located, TM3 attempts to pull the transaction from the TM1 by using information contained in the message, and therefore TM3 initiates a CONNTYPE_PARTNERTM_BRANCH connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM1.

4. [C10.1]Branching(guidTx): TM3 sends a PARTNERTM_BRANCH_MTAG_BRANCHING message with the transaction identifier (guidTx) of the requested transaction to TM1.

5. [C10.2]Branched: When TM1 receives the PARTNERTM_BRANCH_MTAG_BRANCHING message, TM1 creates a subordinate branch and sends a PARTNERTM_BRANCH_MTAG_BRANCHED message to TM3.

6. [C9.2]Associated: When TM3 receives the PARTNERTM_BRANCH_MTAG_BRANCHED message, TM3 keeps the connection open in order to process two-phase commit notifications from TM1, and sends a TXUSER_ASSOCIATE_MTAG_ASSOCIATED message to LU1 on the CONNTYPE_TXUSER_ASSOCIATE connection to inform LU1 that the pull operation was successful. TM3 continues to maintain the CONNTYPE_PARTNERTM_BRANCH connection with TM1, and waits for two-phase commit processing.

When LU1 receives the TXUSER_ASSOCIATE_MTAG_ASSOCIATED message, LU1 initiates the disconnect sequence on the CONNTYPE_TXUSER_ASSOCIATE connection.

6.1.6.2.5 Enlisting an LU 6.2 Implementation in a Transaction by Using Protocols Specified in [MS-DTCLU]

After the LU 6.2 Implementation (LU1) has pulled the transaction to its Transaction Manager (TM3), it enlists on the transaction by using protocols specified in [MS-DTCLU]. The message flow is as follows:

1. [C11.0]Connect(LuRmEnlistment): LU1 initiates a CONNTYPE_TXUSER_DTCLURMENLISTMENT connection on an MSDTC Connection Manager: OleTx Transports Protocol session with TM3.

2. [C11.1]Create(guidTx,LuPair): LU1 sends a TXUSER_DTCLURMENLISTMENT_MTAG_CREATE message to TM3 specifying the transaction identifier (guidTx) and the LU Name Pair (LuPair).

3. [C11.2]RequestComplete: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_CREATE message, TM3 creates an LU enlistment on the transaction, and sends a TXUSER_DTCLURMENLISTMENT_MTAG_REQUEST_COMPLETED message to LU1.

When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_REQUEST_COMPLETED message, LU1 continues to maintain the connection and waits for two-phase commit processing.

6.1.6.2.6 Monitoring the Transaction by Using Protocols Specified in [MS-CMOM]

This message exchange demonstrates TM1 broadcasting transaction information at this point in the life cycle of the transaction (guidTx). The message flow is as follows:

1. [C1.5]Stats(open++): Because the transaction (guidTx) is active but not yet committing or aborting, TM1 sends a MSG_DTCUIC_STATS message to MT with the number of open transactions incremented by one (open++).

2. [C1.6]TranList(guidTx:Open): TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the open state (XACTSACT_OPEN).

6.1.6.3 Two-Phase Commit Transactional Message Exchanges

This example shows the message exchanges that occur when an application requests its Transaction Manager to drive the transaction to a conclusion by using the Two-Phase Commit protocol.

In these sets of message exchanges, the Application (APP) commits the transaction started in section 6.1.6.2.1 by using protocols specified in [MS-DTCO]. At this point, the message exchanges are driven by the Root Transaction Manager (TM1), which has three (3) subordinates—the Resource Manager (RM1), the TIP Transaction Manager (TM2), and the Subordinate Transaction Manager (TM3) and all message exchanges follow the Two-Phase Commit protocol. All messages occur on the existing TIP connection and existing connections over MSDTC Connection Manager: OleTx Transports Protocol sessions.

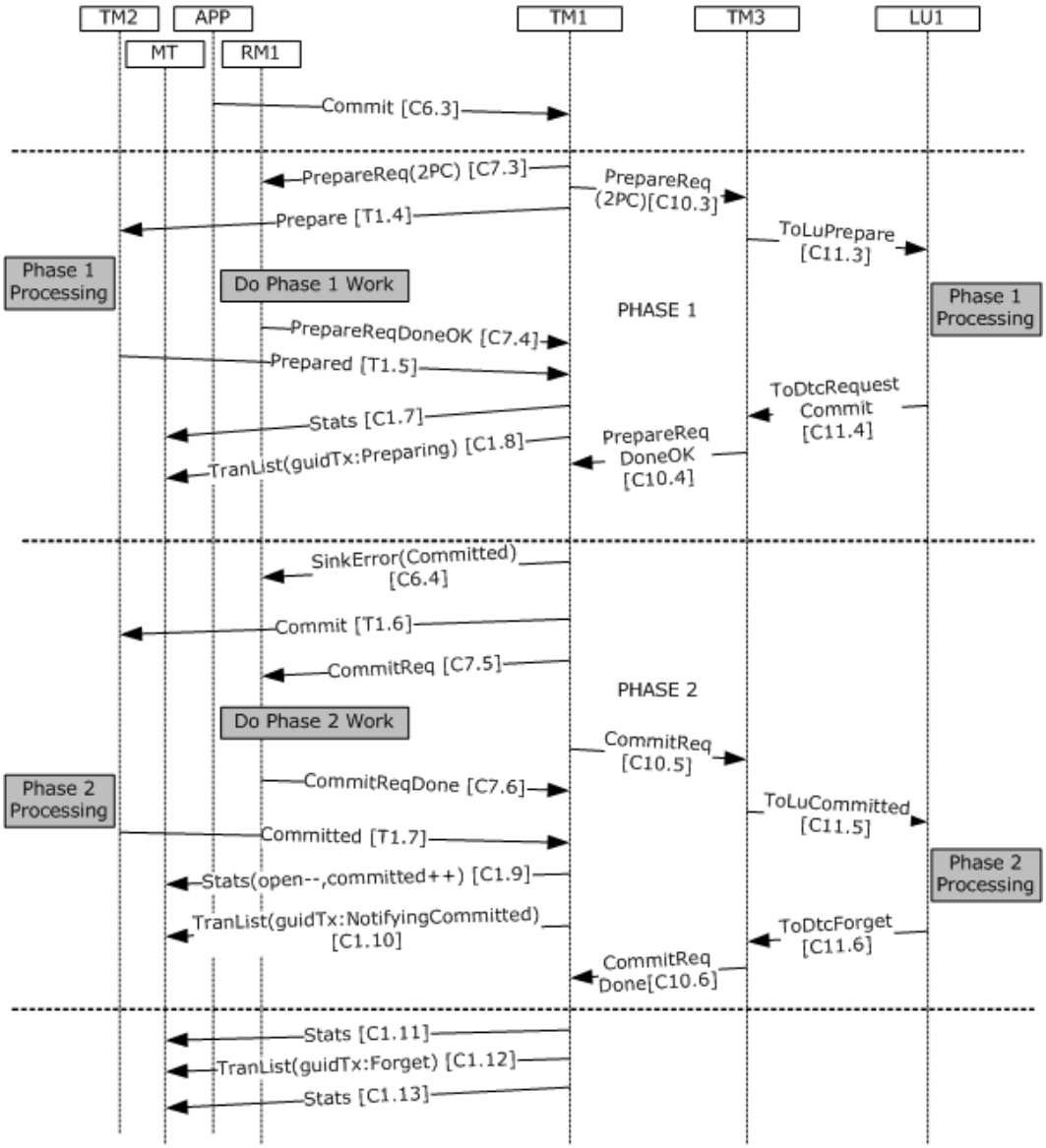

Figure 37: Example 6 sequence diagram: Two-Phase Commit protocol message exchanges in a distributed transaction

6.1.6.3.1 Committing the Transaction by Using Protocols Specified in [MS-DTCO]

This message exchange demonstrates the Application (APP) committing the transactions (guidTx)as follows:

- [C6.3]Commit: APP sends a TXUSER_BEGIN2_MTAG_COMMIT message to TM1.

APP maintains the connection and waits for the outcome of the transaction from TM1.

6.1.6.3.2 Phase One

When TM1 receives the TXUSER_BEGIN2_MTAG_COMMIT message, the transaction's state enters phase one and TM1 attempts to commit the transaction by iterating through its list of subordinate branches, notifying the subordinates that phase one processing has begun. TM1 then waits for reply notifications from each of the subordinates before determining the outcome of the transaction. The message flow is as follows:

- Phase One—Subordinate Resource Manager

1. [C7.3]PrepareReq(2PC): TM1 sends a TXUSER_ENLISTMENT_MTAG_PREPAREREQ message to RM1 over the CONNTYPE_TXUSER_ENLISTMENT connection, indicating that this is a two-phase commit (2PC).

2. [C7.4]PrepareReqDoneOK: When RM1 has successfully completed its phase one work, RM1 sends a TXUSER_ENLISTMENT_MTAG_PREPAREREQDONE message to TM1 specifying TXUSER_ENLISTMENT_PREPAREREQDONE_OK.

RM1 maintains the connection and waits for the transaction outcome from TM1.

- Phase One—Subordinate TIP Transaction Manager

1. [T1.4]Prepare: TM1 sends a PREPARE command over the TIP connection associated with the transaction to TM2.

2. [T1.5]Prepared: When TM2 has successfully completed its phase one processing, TM2 sends a PREPARED command to TM1 over the TIP connection.

The state of the TIP connection is now PREPARED and TM2 waits for transaction outcome from TM1.

- Phase One—Subordinate Transaction Manager

1. [C10.3]PrepareReq(2PC): TM1 sends a PARTNERTM_PROPAGATE_MTAG_PREPAREREQ message to TM3 over the CONNTYPE_PARTNERTM_BRANCH connection, indicating that this is a two-phase commit (2PC).

2. [C11.3]ToLuPrepare: When TM3 receives the PARTNERTM_PROPAGATE_MTAG_PREPAREREQ message, TM3 iterates through each of its subordinate branches to send out phase one notifications and sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_PREPARE message to LU1 over the CONNTYPE_TXUSER_DTCLURMENLISTMENT connection.

3. [C11.4]ToDtcRequestCommit: When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_PREPARE message, LU1 completes its phase one work, sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_REQUESTCOMMIT message to TM3, and waits for the transaction outcome from TM3.

4. [C10.4]PrepareReqDoneOK: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_REQUESTCOMMIT message, TM3 sends a PARTNERTM_PROPAGATE_MTAG_PREPAREREQDONE message to TM1.

TM3 maintains the connection and waits for transaction outcome from TM1.

- Phase One—Monitoring the Transaction by using Protocols Specified in MS-CMOM

This message exchange demonstrates TM1 broadcasting transaction information during phase one of the transaction (guidTx).

1. [C1.7]Stats: Because the transaction's outcome is not yet known, TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.5] related to this transaction.

2. [C1.8]TranList(guidTx:Preparing): TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the preparing state (XACTSACT_PREPARING).

6.1.6.3.3 Phase Two

When TM1 receives successful phase-one notification replies from each of its subordinate branches, TM1 commits the transaction and begins phase-two processing by sending commit verification to APP and commit notifications to each of its subordinate branches (RM1, TM2, and TM3). Phase two message flow is as follows:

- Phase Two—Application

- [C6.4]SinkError(Committed): TM1 sends a TXUSER_BEGIN2_MTAG_SINK_ERROR message to APP over the CONNTYPE_TXUSER_BEGIN2 connection, specifying that the transaction has committed (TRUN_TXBEGIN_ERROR_NOTIFY_COMMITTED).

When APP receives the TXUSER_BEGIN2_MTAG_SINK_ERROR message, APP initiates the disconnect sequence.

- Phase Two—Subordinate Resource Manager

1. [C7.5]CommitReq: TM1 sends a TXUSER_ENLISTMENT_MTAG_COMMITREQ message to RM1 over the CONNTYPE_TXUSER_ENLISTMENT connection.

2. [C7.6]CommitReqDone: When RM1 has completed its commit work, RM1 sends a TXUSER_ENLISTMENT_MTAG_COMMITREQDONE message to TM1 and initiates the disconnect sequence on the CONNTYPE_TXUSER_ENLISTMENT connection with TM1.

- Phase Two—Subordinate TIP Transaction Manager

1. [T1.6]Commit: TM1 sends a COMMIT command over the TIP connection associated with the transaction to TM2.

2. [T1.7]Committed: When TM2 has successfully completed its phase two processing, TM2 sends a COMMITTED command to TM1 over the TIP connection.

The state of the TIP connection with TM1 is now IDLE, and the current transaction of the TIP connection is cleared.

- Phase Two—Subordinate Transaction Manager

1. [C10.5]CommitReq: TM1 sends a PARTNERTM_PROPAGATE_MTAG_COMMITREQ message to TM3 over the CONNTYPE_PARTNERTM_BRANCH connection.

2. [C11.5]ToLuCommitted: When TM3 receives the PARTNERTM_PROPAGATE_MTAG_COMMITREQ message, TM3 iterates through each of its subordinate branches to send out commit notifications and sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_COMMITTED message to LU1 over the CONNTYPE_TXUSER_DTCLURMENLISTMENT connection.

3. [C11.6]ToDtcForget: When LU1 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_LU_COMMITTED message, LU1 completes its phase-two processing, sends a TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_FORGET message to TM3, and initiates the disconnect sequence.

4. [C10.6]CommitReqDone: When TM3 receives the TXUSER_DTCLURMENLISTMENT_MTAG_TO_DTC_FORGET message, TM3 sends a PARTNERTM_PROPAGATE_MTAG_COMMITREQDONE user message to TM1, and initiates the disconnect sequence.

- Phase Two—Monitoring the Transaction by using Protocols Specified in [MS-CMOM]

This message exchange demonstrates TM1 broadcasting transaction information during phase two of the transaction (guidTx).

1. [C1.9]Stats(open--,committed++): Because the transaction is now committed, TM1 sends a MSG_DTCUIC_STATS message to MT with the number of open transactions decremented by one (open--) and the number of committed transactions incremented by one (committed++).

2. [C1.10]TranList(guidTx:NotifyingCommitted): TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the notifying committed state (XACTSACT_NOTIFYING_COMMITTED).

- Phase Two – The Root Transaction Manager

After TM1 receives all phase-two replies from each of its subordinates, the transaction life cycle is complete and TM1 removes the transaction from its list of known transactions.

- Post-Phase Two Messages – Monitoring the Transaction by using Protocols Specified in MS-CMOM

This message exchange demonstrates TM1 broadcasting transaction information after phase two of the transaction (guidTx) is completed.

1. [C1.11]Stats: TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.9] related to this transaction.

2. [C1.12]TranList(guidTx:Forget): TM1 sends a MSG_DTCUIC_TRANLIST message to MT listing the transaction (guidTx) in the forget state (XACTSACT_FORGET). Any future MSG_DTCUIC_TRANLIST messages will not include this transaction.

3. [C1.13]Stats: TM1 sends a MSG_DTCUIC_STATS message to MT with no changes from its previous message [C1.11] related to this transaction. Because there are no active transactions that TM1 is tracking, no MSG_DTCUIC_TRANLIST message is sent.

6.2 Communication Details

The system does not define any communication constraints or additional message types beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

6.3 Transport Requirements

The system does not define a transport beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

6.4 Timers

The system does not define any timers beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

An implementation of the system MUST provide the timers described in the specifications of the protocols supported by the system.

6.5 Non-Timer Events

This section describes the externally visible events that the Transaction Processing Services System generates or receives.

6.5.1 Local Events

There is one local event: when a connection is disconnected.

6.5.1.1 Connection Disconnected

This event occurs when a connection with the Transaction Processing Services System that uses the MSDTC Connection Manager: OleTx Transports Protocol Specification, as described in [MS-CMPO], is disconnected. This event impacts both the system and parties communicating with it.

Protocols that depend on the protocol specified in [MS-CMPO] are described in section 5.3.

The party receiving the event must perform the actions specified in [MS-DTCO], section 3.1.8.

6.5.1.2 External Event

There is one external event.

6.5.1.2.1 Recover

This event occurs when the system is initiated. When the system receives this signal it performs the actions specified in [MS-DTCO], section 3.2.3.3.

When the system is distributed across multiple computers on a network, the system must be installed on each computer. Each installation of the system receives the recover event individually at initiation.

6.6 Initialization and Reinitialization Procedures

The system does not define any initialization requirements beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

An implementation of the system MUST comply with the initialization requirements described in the specifications of the supported protocols.

The system does not require the supported protocols to be initialized in a specific sequence.

6.7 Status and Error Returns

The system does not define any error handling requirements beyond those described in the specifications of the protocols supported by the system, as listed in section 2.2.

Various kinds of errors may occur impacting the system. More precisely, an error condition may impact one or more protocols supported by the system. Such error conditions and the resulting protocol semantics are described under section 3 of the corresponding protocol specifications.

7 Security

This section documents the system wide security issues that are not otherwise described in the Technical Documents for the Member Protocols. It does not duplicate what is already in the Member Protocol Technical Documents unless there is some unique aspect that applies to the system as a whole.

The Transaction Processing Services System is designed to protect the following assets:

- Transaction Information
- System Configuration
- Messages
- Events

This is illustrated in the following figure, where the system is shown communicating with a Resource Manager and an Application Service.

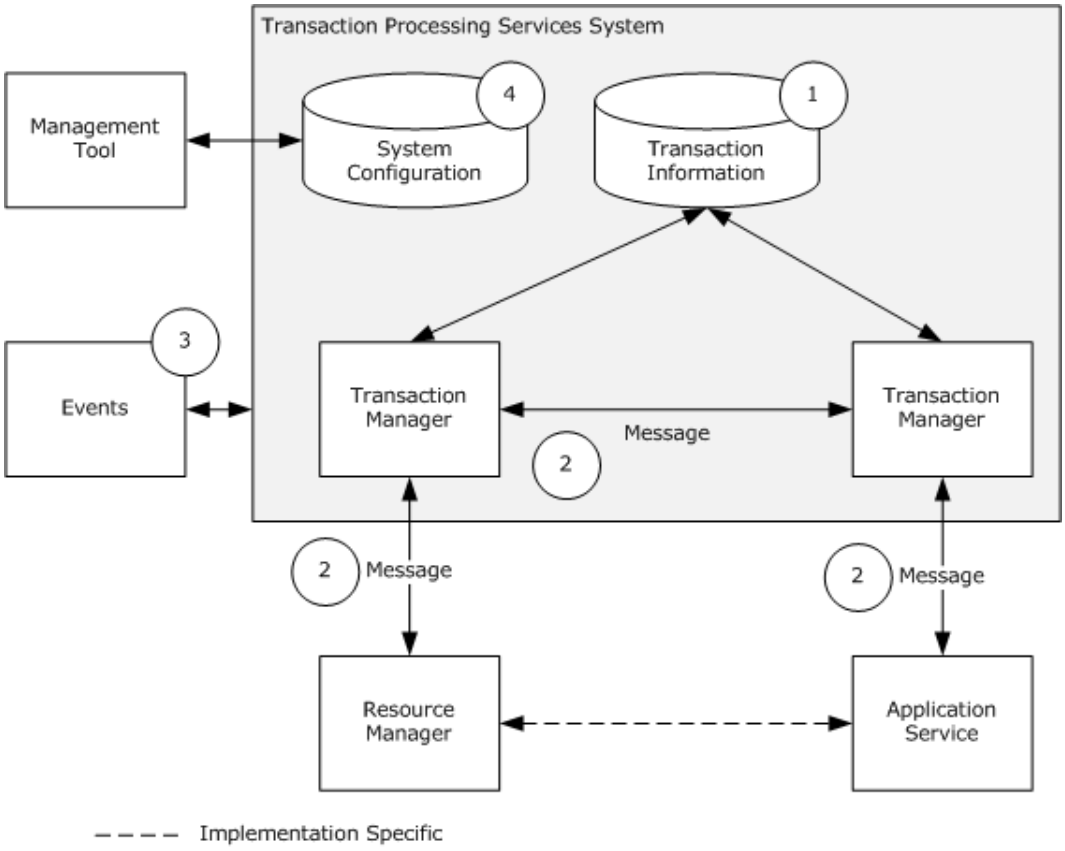

Figure 38: Transaction Processing Services System assets

7.1 Transaction Information Security

The Transaction Information asset consists of the state of the transaction, the identity, and the locations of the participants, and other data about the transaction, such as the transaction

description. The Transaction Information is held in memory, and also in a log supported by a durable storage system.

The system relies on the durable storage system to maintain the integrity of this log and to restrict access to it.

The system accesses and modifies its Transaction Information as a result of events and messages that it receives. Therefore, the security and integrity of the Transaction Information is also dependent on the system's ability to secure these events and messages, which is described in sections 7.3 and 7.4.

7.2 System Configuration Security

The System Configuration asset consists of all the configuration data required by the system. Examples are security identities and associated credentials used by the system, and feature enablement settings, such as the setting that allows a transaction to span multiple computers. The System Configuration data is kept in a durable storage system. The system relies on the durable storage system to enforce the access restrictions specified by the system.

The system accesses and modifies its system Configuration data as a result of messages that it receives, for instance, from a management tool, as described in [MS-CMOM]. Therefore, the security and integrity of the System Configuration is also dependent on the system's ability to secure these messages, which is described in section 7.3.

7.3 Message Security

The Messages asset consists of the messages received and sent by the system, and messages received and sent within the system. The system protects the privacy and integrity of these messages, and also ensures that they are sent to and received from an authorized party.

The messages that the system receives and sends are specified by the system protocols (see section 2.2). Most of these protocols, in turn, depend on the protocol specified in [MS-CMPO], which requires that an RPC session must be established before exchanging any messages. The MSDTC Connection Manager: OleTx Transports Protocol uses the **security provider** security model, as specified in [MS-RPCE] section 2.2.1.1.7, and an authentication level, as specified in [MS-RPCE] section 2.2.1.1.8, to configure protection of messages; for instance full encryption for privacy and integrity, or by requiring mutual authentication for authorization. See [MS-CMPO] section 2.1.3 for more details. Some system protocols do not depend on the protocol specified in [MS-CMPO], but they may use, depend on, or extend other industry standard protocols, as described in section 5.3. When communicating over protocols that do not depend on CMPO, the system adopts the security requirements and semantics specified by the industry standard protocol.

When communicating over the WS-AtomicTransaction protocol, the system fully adheres to the security requirements and semantics specified by the WS-AtomicTransaction protocol. Additionally, the system requires that all WS-AtomicTransaction communication is done over an HTTPS connection. All entities that participate in transaction coordination with the system via WS-AT protocol must use a valid X509 security certificate when communicating with the system. The system keeps a list of X509 security certificate thumbprints in its System Configuration so that it can authorize whether an entity can participate in transaction coordination with the system by using the WS-AtomicTransaction protocol.

7.4 Event Security

The Events asset consists of the events raised and handled by the system. These are limited to events received from the network system reporting a change of connection state, and events

received from the operating environment of the system when the system is initialized. Both of these event sources and their connection to the system are trusted by the system, and no additional protections are applied.

7.5 Connection Type and Feature Restriction

The system also restricts access to certain features to specified groups of security identities. This restriction is applied at the level of connection type. A connection type specifies a set of messages. The system protocols specify these connection types and the related messages. The system protocols use connection types to group messages by functionality, and most messages are members of exactly one connection type. Therefore, the functionality associated with a message can be restricted by restricting access to the connection type, and by sending or receiving a message only if the communicating party has access to the connection type.

Connection types related to transaction state changes are restricted to sessions authenticated as administrator, and connection types related to transaction manager to transaction manager communication are restricted to parties known to be transaction managers, as described in [MS-DTCO] section 5.

The system also restricts the set of supported connection types through configuration, as described in [MS-DTCO] section 5. For example, the system can be configured to not allow connection types related to network transactions.

When using the protocol described in [MS-TIPP], the system can be configured to restrict the use of specific functionalities related to that protocol through configuration, as described in [MS-TIPP] section 5.

The system can be configured to restrict the use of the protocol specified in [MC-DTCXA]. Further details of this configuration are described in [MS-CMOM].

The system can also be configured to restrict the use of the WS-AtomicTransaction protocol, as described in section 5.4.5.

7.6 Internal Security

The Transaction Processing Services System applies the security mechanisms described in 7.1, 7.2, 7.3, 7.4, and 7.5 to ensure its internal security.

Other systems interacting with the Transaction Processing Services System SHOULD take the following steps to ensure the security of this system:

- Support the mutual authentication feature of the protocol specified in [MS-CMPO].
- Correctly execute the Two-Phase Commit protocol, so that other transaction participants experience well-regulated progress towards a common transaction outcome.
- Always complete transactions after creating them, to avoid filling up the system log and requiring administrative intervention.
- Do not allow transactions to stay in in-doubt state for a longer period than the higher layer business logic allows.

7.7 External Security

The Transaction Processing Services System applies the following security measures to ensure the security of other entities that it interacts with:

- Support the mutual authentication feature of the protocol specified in [MS-CMPO] when communicating over that protocol.
- When using WS-AT, establish all communication over HTTPS connections.
- Correctly execute the Two-Phase Commit protocol, so that all transaction participants experience well-regulated progress towards a common transaction outcome.
- Do not allow transactions to stay in in-doubt state for a longer period than the higher layer business logic allows.

The other entities that interact with this system SHOULD apply the following security measures to ensure their own security during interactions with this system:

- If the other entity is a resource manager or a transaction manager, it takes security measures similar to those described in the sections related to transaction information security, system configuration security, message security, and event security.
- Support the mutual authentication feature of the protocol specified in [MS-CMPO] where applicable, when communicating with the Transaction Processing Services System.
- When using WS-AT, establish all communication over HTTPS connections.
- Correctly execute the Two-Phase Commit protocol, so that other transaction participants experience well-regulated progress towards a common transaction outcome.
- Do not allow transactions to stay in in-doubt state for a longer period than the higher layer business logic allows.

8 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows 2000 operating system
- Windows XP operating system
- Windows XP operating system Service Pack 1 (SP1)
- Windows Server 2003 operating system
- Windows Server 2003 operating system with Service Pack 1 (SP1)
- Windows Server 2003 R2 operating system
- Windows Vista operating system
- Windows Vista operating system with Service Pack 1 (SP1)
- Windows Server 2008 operating system
- Windows 7 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

9 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

10 Index

E

F

G

I

L

M

N

O

T

U

V

W