TechMetrix research

HP Total-e-Transactions White Paper

A Technical White Paper by TechMetrix Research
Prepared for HP
Audience: Technical architects, developers

www.techmetrix.com

Table of Contents

1. Scope and purpose of the white paper

The transactions managed 30 years ago needed to be reliable – and the same applies to today's e-business transactions. However, the constraints of information systems have changed considerably. The transaction managers being built at the start of the '70s were constructed for a different system of computing and were based on standards and technologies available at the time. This paper will examine how the "new generation" transaction managers can help you to manage transactions involving components distributed on a corporate intranet.

A transaction manager is a core component of any modern distributed application architecture, and must be tightly integrated with the other components, such as application servers or databases. Total-e-Transactions is a JTS-compliant transaction manager package, and can either be used stand alone or within HP 's J2EE-compliant application servers, Total-e-Server and HP Application Server. We shall look at the implications of both of these options.

We shall then examine the positioning adopted by HP Total-e-Transactions, giving a precise description of the various components of the offering, and looking at the solution in the light of the main standards for e-business infrastructures today (in particular, J2EE and JTS). Particular emphasis shall be placed on how HP Total-e-Transactions 2.1 meets corporate intranet transaction requirements, and on its 100% Java implementation, which enables it to integrate tightly with existing IT systems.

Terminology:

- J2EE: Java 2 Enterprise Edition
- JTS: Java Transaction Service

2. Executive Summary

Many companies have long tried to automate their business processes, targeting efficiency and reliability, so as to improve their quality of service for customers. Business processes often involve multiple data sources (such as databases or flat files) and calculation components, which can span networks. But what happens if a failure occurs during execution of a business process? Data integrity can be compromised and this can lead to serious business problems.

These types of problems appeared very early on in the IT industry, and major software companies built transaction monitors to help solve them. These middleware products provided an environment for executing mission-critical business processes reliably and predictably, with what we call a "transaction paradigm."

The first transaction managers were designed during the '70s, helping many industries to solve transactional problems. However, at this time, enterprise computing was organized centrally and transaction monitors were designed in a proprietary fashion for host-centric computing.

Today, the challenge has shifted: reliability and predictability are still mandatory, but the data and operations involved in a transaction can be distributed across large corporate intranets and managed by heterogeneous systems. Traditional transaction managers lack the flexibility and standards to meet all of these new challenges in an efficient way.

The transaction paradigm has been very efficient in the past for centric systems, and new computer technologies and standards can bring this paradigm into the e-business context:

- **Component technology**: a paradigm for designing business processes that provides quality coupled with flexibility to enable businesses to cope with rapid changes.
- **Transaction standards**: based on the mature OMG OTS standard, Sun JTS provides a standard method for implementing transactions involving components from different data sources over large-scale corporate intranets.
- **Integration with existing systems**: J2EE standards allow existing systems to be seamlessly integrated in a transactional fashion.

To meet the new requirements for e-business transactions, HP has built Total-e-Transactions, the industry's first 100% pure Java JTS implementation. Because it is based on mature transaction standards and Java, HP provides a reliable object-based transactional infrastructure that seamlessly integrates data and applications within major corporate intranets.

Terminology:

- **J2EE**: Java 2 Enterprise Edition
- **JTS**: Java Transaction Service
- **OMG's OTS**: Object Management Group 's Object Transaction Service

3. Transaction processing concepts

In order to appreciate HP's Total-e-Transactions product, it is important to understand the benefits of transactions, the main transactional models, and the architectures that can support them. The purpose of this section is to describe these elements.

3.1. What is a transaction?

Let us take a simple example to illustrate the sort of problems that transactions are designed to solve. Let us imagine that a bank wishes to automate its business processes with an IT system, for example a transfer between two accounts (A and B) whose balances are respectively $200 and $300. This process will be made up of a series of operations, in the following order:

1- Balance of account A read ($200)
2- Solvency of account A checked before and after transfer (bank does not allow overdrafts)
3- Amount of transfer debited from account A ($100)
4- Amount of transfer credited to account B ($400)

This process may be affected by the following events:

- **Hardware or software failure**: If the IT system suffers a failure when a transfer process has completed step 3, account A will be debited but account B will not be credited. We would then say that data integrity has been compromised, which can be extremely harmful for the bank.

- **Network failure**: If the system is deployed over a network, a further risk to data integrity is added, in that the process can stall, for example at step three.

- **Concurrent access**: Let us imagine that a transfer of $100 from account A to another account, C, begins at the same time that a different transfer of $200 from account 1 to account B is just embarking on step 2. Both transfers will foresee a positive balance after the transfer, while in reality the account will be $100 overdrawn.

No matter how robust the hardware, software or networks, zero-risk is impossible, and an enterprise such as a bank cannot tolerate the slightest error, since it may result in substantial loss of revenue.

Ideally, our bank should have a system that is able to support its business processes while protecting data integrity under any circumstances. This system should display the following properties:

- *Atomicity:* a process must be carried out in its entirety, or not at all. If something occurs to stop it being fully executed, the system must return to its initial state.

- *Consistency:* guarantees that the transaction always produces correct results according to the semantics of the application.

- *Isolation:* every process must give the illusion of being executed independently from others, whether concurrently or not. This means that the system must use methods such as locks or timestamps to ensure that the data cannot be modified by multiple concurrent processes simultaneously, and to guarantee that a process cannot read data in a provisionally inconsistent state.

- *Durability:* the system must guarantee that the final state of a completed process exists beyond the lifetime of the application that created/modified it, despite non-catastrophic failures.

A transaction service provides a software infrastructure which ensures that transactions benefit from ACID properties.

3.2. Transactional architecture sub-components

The traditional components of a transactional architecture are illustrated in the schema below. This shows the transaction manager, in charge of directing transactions with the resource managers and the resources, acting under the orders of the application components.

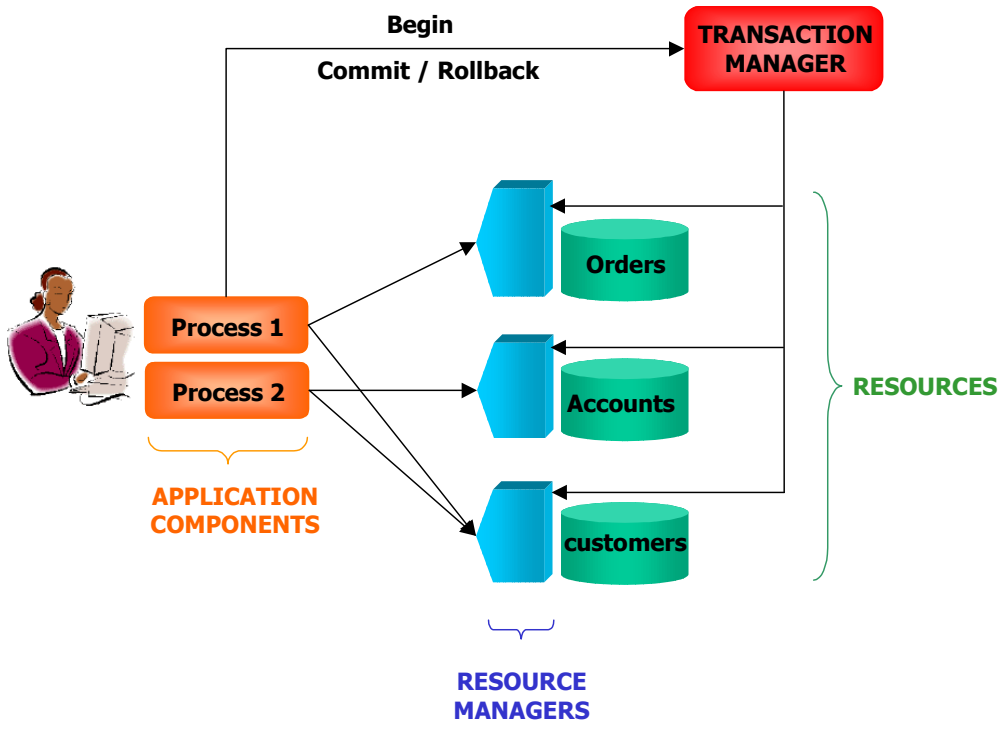


Figure 1: Transactional architecture sub-components

➲ The application components

Application components are in charge of implementing the business logic of processes, which brings into play the data from sources such as databases. They give explicit orders to 'commit' a transaction so as to ratify the final state of the system, or to cancel or 'roll back' in order to return to the system's initial state.

➲ The resource manager

The resource manager is a component that manages the traditional aspects of storage systems (e.g. a database). Generally, it is a driver that takes part in the process of registering resources with the transaction manager, so as to keep track of all the data in play in the transaction. This process is known as resource enlistment. Should one of the system components crash, the resource manager coordinates with the transaction manager to bring the application back to a consistent state.

➲ The transaction manager

The transaction manager is the core element of the transactional architecture. It creates a transaction and its context, and is governed by the application components. It coordinates resource enlistment. Should a serious incident occur during a transaction (network outage, software or hardware failure on one of the components…) causing data to be corrupted, the transaction manager must repair the system data in order to restore it to a consistent state. If a transaction is carried out correctly, the transaction manager sends a commit order to the resource managers involved.

3.3. Transaction models

We shall look here at the two principal models used today: the flat transaction model and the nested transaction model. Although other models do exist, we shall not be covering them in this particular document.

3.3.1. The flat transaction model

The flat transaction is the simplest model, and the most often used. It is based on the "all or nothing" paradigm, it being impossible to partially commit or roll back a flat transaction, even if it is made up of several operations. It forms an indivisible unit of work (UOW). The advantage of this flat transaction model lies in its simplicity, and Total-e-Transactions supports this model.

However, the flat transaction model reaches its limits as soon as transactions are required to be long-running and designed in a modular fashion. Let us imagine that a long flat transaction is made up of a series of cumbersome operations stops, for whatever reason (e.g. a software failure), when only 90% complete. The transaction manager will carry out a rollback and cancel all the work accomplished by the transaction thus far, even if valid. This can lead to reduced efficiency in certain cases.

Luckily, a more flexible model is available. This is the nested transaction model.

3.3.2. The nested transaction model

The nested transaction model was developed so as to offer finer granularity and makes it possible to nest transactions inside each other, hierarchically.

The nested transaction model features a top-level transaction, which contains several sub-transactions called child transactions. One sub-transaction may contain other sub-transactions. A hierarchy of transactions is called a transaction family.

A sub-transaction behaves like a flat transaction, in that it is either carried out completely or not at all. Conversely, a sub-transaction completes with a provisional commit which depends on the commitment of all the enclosing transactions. If a sub-transaction completes with a rollback, this is

permanent, and the enclosing transaction is free to commit or roll back, depending on the application context. If a transaction within a transaction family rolls back, all the sub-transactions will roll back.

This transaction model has the following advantages:

- *Modularity:* the nested transaction model enables transactions to be developed independently from the context in which they will be used. This nested model provides modularity, which is a very useful feature in distributed environments.

- *Failure containment:* If a sub-transaction fails, the top-level transaction has the option to fix the problem without having to roll back and cancel any work accomplished so far. The failure is therefore contained. If the business context of the application so permits, the top-level transaction can decide to commit. This will make all sub-transaction commits permanent.

Unlike most of the commercially available transaction managers, Total-e-Transactions fully supports the nested transaction model.

3.4. Standards involved in transactional architectures

Transactional computing has been the subject of a number of standardization efforts, which have responded to the functional and technical needs encountered over time. Distribution of critical data on networks is at the root of the X/Open DTP standard for distributed transactions that standardizes distributed transactional architectures. The OMG extended the transaction concept to distributed objects with the OTS standard.

J2EE is an n-tiered architecture standard whose arrival caused industry players to rush to develop e-business applications. JTA and JTS are standards that enable J2EE to support transactions. They are compatible with OTS standards for distributed transactions.

3.4.1. X/Open DTP

X/Open, a consortium of vendors, developed a standard called DTP – Distributed Transaction Processing. Its purpose is to standardize distributed transaction architectures. It specifies the API used to communicate between resource managers, transaction managers and applications.

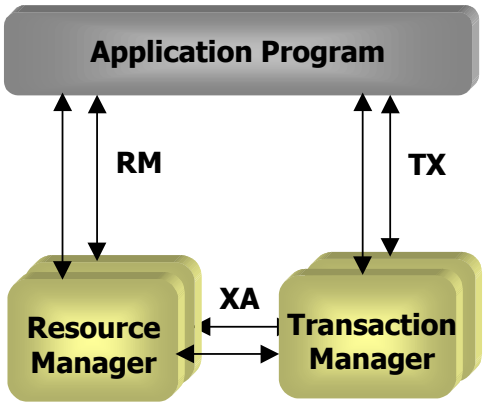


Figure 2: Standard X/Open DTP

The RM API is used by the application to deal with data handled by the resource manager. The XA API is used by the transaction manager to coordinate data updates via the resource managers. The TX API is used by applications to direct transactions via the transaction manager. The typical orders sent are start, stop and commit transactions.

It is important to note that the only transaction model supported by the DTP is the flat model.

3.4.2. CORBA OTS

The DTP standard was developed at a time when programs were mostly designed procedurally. The arrival of object technologies brought major changes to application development methodology and deployment architecture, providing new opportunities for easier, more modular application development and flexible deployment using a scalable architecture. Because the execution model for procedural programs is radically different from that of object-based programs, the need to develop standards for object transactions began to be felt. The OMG developed such a standard, called OTS (Object Transaction Service), as part of the CORBA services.

The OTS Specification has been built to be comparable with X/Open DTP. The XA protocol to coordinate data updates by resource managers is compatible between both standards. The main difference between the X/Open DTP Standard and the OTS specification is that OTS casts all of the basic operations as methods on objects, which use IIOP as the communication protocol.

An implementation of the OTS standard must support the flat transaction model, and support for nested transactions is optional.

3.4.3. J2EE Architecture Standard and EJB Component model

J2EE (Java 2 Enterprise Edition) is a standard designed to simplify problems linked to development, deployment and maintenance of multi-tier enterprise applications. J2EE was drawn up jointly by some of the main players in the middleware industry; its development is governed by Sun Microsystems.

The J2EE standard is based on Java, enriching it with a number of specifications which meet the requirements of enterprise applications (connection to databases, transactional systems, asynchronous middleware…). We can therefore say that Java is a subset of J2EE, and J2EE offers a standard framework for development of enterprise applications. The J2EE 1.2 standard is made up of the following specifications:

Mandatory components

API	Description
JDBC 2.0 (Java DataBase Connectivity)	Relational database connection API
RMI/IIOP 1.0 (Remote Method Invocation / Internet InterOrb Protocol)	EJB component access protocol
EJB 1.1 (Enterprise JavaBeans)	Component model
Servlet 2.2	Web server extension API
JSP 1.1 (Java Server Pages)	Server-side scripting API for the Web
JMS 1.0 (Java Message Services)	MOM connection API
JNDI 1.2 (Java Naming and Directory Interface)	Directory connection API
JTA 1.0.1 (Java Transaction Services)	Transactional service management API
JavaMail 1.1	E-mail management API
JAF 1.0 (Java Activation Framework)	API for management of messages attached to e-mail

Recommended components

API	Description
JTS 1.0 (Java Transaction Service)	Specification for transactional interoperability between EJB containers based on CORBA, OTS & JTA

Adopting J2EE for an enterprise application offers a number of advantages:

- Independence from the operating system used by the application
- Independence from the application server running the application
- Portability of J2EE competencies to various application deployment environments

The J2EE standard brought significant changes to the application server market. Most application server vendors have adhered to the standard, and can guarantee that clients remain independent from their platforms.

The EJB specification occupies an important place within the standard, since it offers an environment for modeling processes and business objects. There are two types of EJB:

- *Session EJBs* make it possible to model business processes (e.g. transferring a sum from account A to account B), and offer distribution, security, naming and transaction capabilities.

- *Entity EJBs* model business entities (e.g. a bank account). Entity EJBs offer the same features as Session EJBs and support persistence. This property makes it possible to save the state of the component in an appropriate facility (e.g. database).

➲ The role of JTA in J2EE Architectures

The J2EE standard was designed to simplify development of multi-tier applications and to make it lasting. The JTA is an API that developers can use to prompt transactional behavior in their applications. The main objective of JTA is to standardize and simplify the way in which developers use transactions in J2EE architectures.

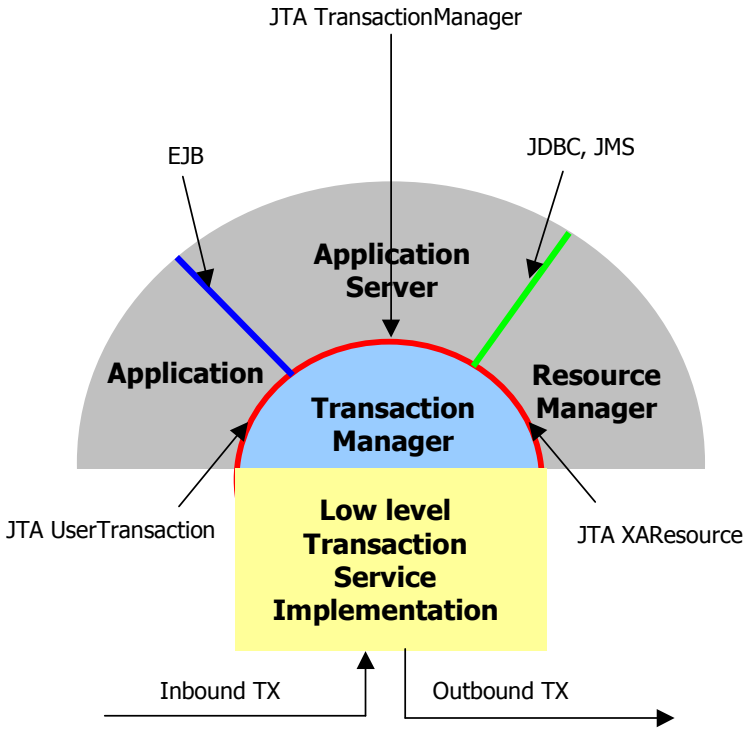


Figure 3: JTA Architecture

The above diagram illustrates how JTA standardizes the interfaces between the J2EE architecture components involved in the transactions.

- The *UserTransaction* interface enables the application to control the transaction (here we encounter the commit, rollback and begin orders, for example). This is the interface that is visible to developers.
- The *TransactionManager* interface enables the application server to control the perimeters of the transactions launched from the application.

- The *Transaction* interface is used to carry out a set of operations associated with the transaction. Enlistment and synchronization are two such operations.
- The *XAResource* interface is a Java mapping of the industry standard XA interface. This interface enables any XA resource manager to be involved in a transaction initialized by the Transaction Manager. This means an XA database or XA MOM can be included in a distributed transaction.

The JTA occupies a very important place in the J2EE architecture. Below are the specifications relating to it:

- EJB containers handle management of EJB components, offer transactional capacities via the JTA interface (more precisely, implementation of the UserTransaction interface)

- The extension of the JDBC 2.0 specification brings support for the XA standard, which enables JDBC 2.0 resource managers to allow databases to participate in distributed transactions, which may, for instance, be initiated by an EJB.

- The JMS specification includes XA support, and enables a JMS message to participate in a distributed transaction. This technique is important for uncoupling systems in a transactional fashion.

It is important to bear in mind that, at present, JTA is rooted in the DTP standard and operates with the XA interface, which limits it to the flat transaction model.

JTA does not, however, specify any standard for internal implementation of the transaction manager. This means that a transaction cannot involve more than one transaction manager. The JTS standard has been designed to overcome this problem in order to make transaction interoperability a possibility.

➲ The role of JTS in J2EE architectures

Figure 3 depicts the components of a transactional architecture, which comply with JTA standards. However, there is no specification to describe the internal operation of the transaction manager.

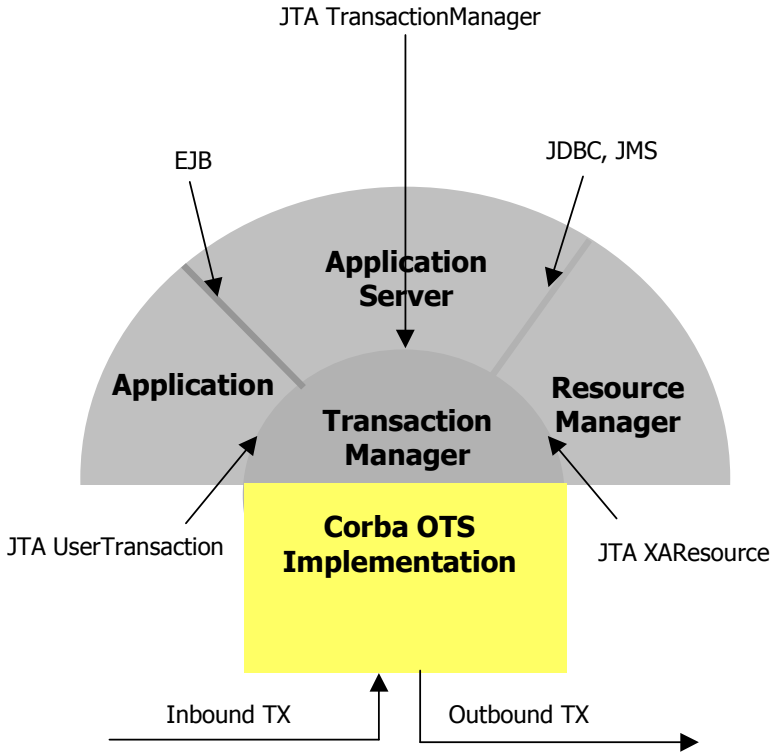


Figure 4: JTS Standard

A transaction manager that conforms to the JTA standard can be implemented internally in a proprietary fashion (communication protocol, identification of objects, conversations, etc.). IBM CICS and BEA Tuxedo are two transaction managers whose internal implementation is proprietary. JTA only covers high-level interfaces. Without a standard describing how transaction managers interoperate, two JTA transaction managers will not be able to communicate with each other or manage the same transaction. In fact, they will not be able to communicate with any other transactional system. The JTS standard therefore sought to standardize internal implementation of transaction managers by relying on the OTS standard.

JTS specifies the implementation of a JTA-compatible transaction manager for high-level interfaces, and relies on Java mapping of the CORBA OTS 1.1 specification at low level. A JTS transaction manager therefore offers transactional interoperability with other JTS managers, and more generally with any OTS-compliant transaction manager.

JTS is strongly recommended by the J2EE specification promoted by Sun. It is therefore important, when choosing a J2EE infrastructure for managing distributed transactions, to ensure that this standard is supported.

4. HP Total-e-transactions 2.1

4.1. Total-e-Transactions 2.1 overview

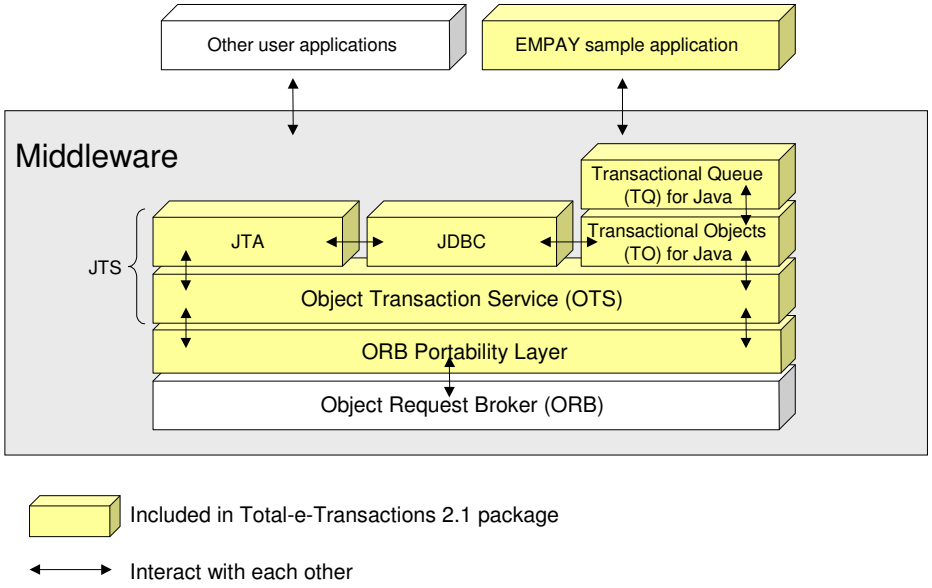


Figure 5: Total-e-Transactions 2.1 Architecture diagram

The above schema illustrates the HP Total-e-Transactions architecture. The yellow blocks stand for the different components supplied in the product.

The Total-e-Transactions transactional engine is fully compatible with J2EE. It is made up of the following elements:

- the ORB portability layer is a product-specific feature that gives all higher layers independence from the communicating infrastructure, which is the CORBA ORB.
- The JTS implementation is based on this last layer, providing low-level OTS compatibility and high-level JTA support.
- Transactional access to relational databases is based on the J2EE JDBC standard and Merant drivers. These components are J2EE-compliant.

Total-e-Transactions also offers non-J2EE-compliant components bringing additional functionality:

- The TO framework offers automatic object persistence and concurrent access management features, making it a high-level transactional application development environment.
- The TQ framework is built on TO for Java, and offers a transactional queue mechanism.
- Lastly, the Empay application is shipped with documentation which gives a practical illustration of what Total-e-Transactions can do.

In the next section, we shall give a more precise description of these elements.

4.2. Total-e-Transactions Components

4.2.1. Total e-Transaction core

The Total-e-Transactions core forms the heart of the infrastructure, and is where distributed transactions are managed. Below we list its main features, which the other components in the product offering also benefit from:

➲ ORB Portability Layer

HP has developed an interface to make its JTS implementation independent from the underlying communication infrastructure (CORBA ORB). This interface is the ORB portability layer which enables Total-e-Transactions to be ported to any CORBA-compliant ORB.

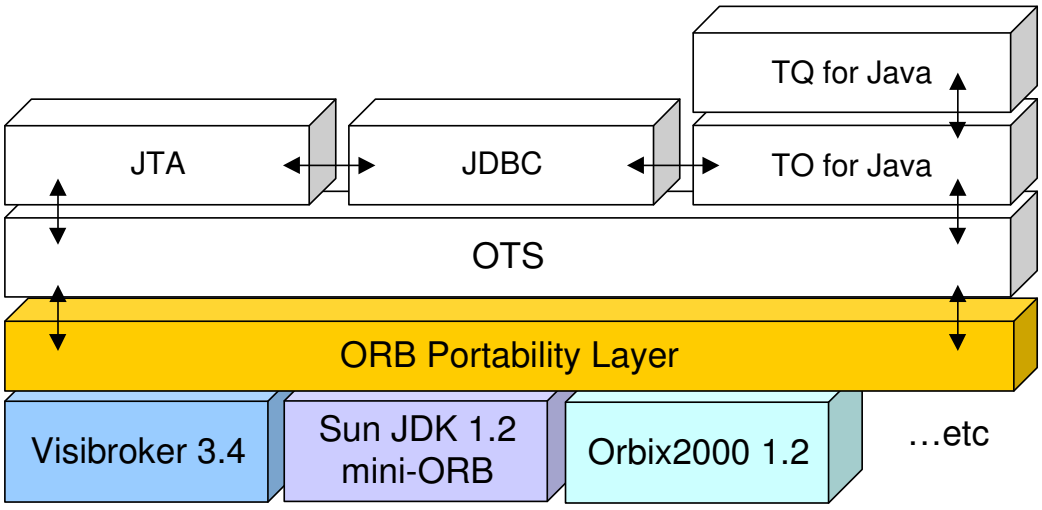


Figure 6: Orb portability layer

The ORBs supported in version 2.1 are Sun JDK 1.2 mini-ORB, Visibroker 3.4 and Orbix2000 1.2. HP is planning to extend this support to other ORBs.

The object adapter is an important component in CORBA infrastructures. When a CORBA client invokes a remote object, the Object adapter is responsible for activating and passing the remote call to the implementation object, and deactivating the object. The first object adapter introduced by the OMG was the BOA (Basic Object Adapter). However, the BOA description was not precise enough to provide for interoperability between different ORBs. So the OMG defined another object adaptor (Portable Object Adaptor)

with enhanced portability and flexibility features, which allows for real ORB interoperability.

The Total-e-Transactions ORB Portability layer is fully compatible with the BOA and the latest POA introduced in the CORBA 2.3 specification. This feature is necessary in order to ensure interoperability between CORBA OTS-compliant transaction managers and portable interceptors.

➲ Support for flat and nested transactions

Support for flat transaction models is provided by the JTA API, while low-level OTS support enables nested transactions to be handled (The JTA interface also allows nested transactions, if it is configured correctly). The OTS specification provides low-level APIs that are not suitable for a productive development. TO for Java and TQ for Java are higher-level frameworks that support better productivity for business objects and transactional processes.

➲ JTA API support

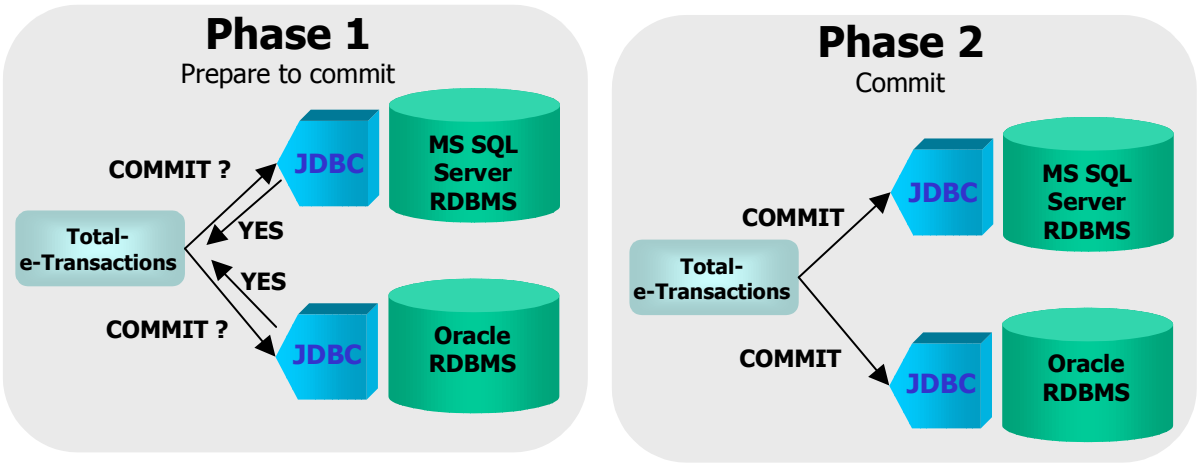
The flat transactions managed by Total-e-Transactions are controlled by the JTA standard API, which makes it possible to distribute transactions among XA-compatible sources (JDBC 2 and JMS data sources) using a two-phase commit protocol.

➲ Support for distributed transactions (using two-phase commit)

A transaction may involve resources that reside in different address spaces, such as two databases distributed over a network. It is important for the transaction to be fully committed, or else fully cancelled. In order for the ACID properties of distributed transactions to be guaranteed, the two-phase commit protocol must be used.

The example given below involves two databases taking part in a flat transaction initiated by Total-e-Transactions, which carries out an UPDATE on an Oracle database and an INSERT in an MS SQL Server database.

The API used by the transaction developer here is JTA. This means that the respective resource managers of Oracle and MS SQL Server (JDBC drivers) must be compatible with the XA interface.



Phase 1 : Prepare to Commit

Total-e-Transactions initiates the transaction with the resource managers involved (Oracle and MS SQL Server databases), and then determines whether they are able to commit.

If one of the resource managers is unable to commit its transaction, it uses its right to veto the distributed transaction. Total-e-Transactions then orders the rollback of all the resource managers taking part in the distributed transaction.

If the resource managers are able to commit, they send a confirmation order to the transaction manager, which registers the commit order for the distributed transaction in a stable storage system before moving to phase 2 of the protocol. The resource managers remain blocked as they wait for phase 2 of the protocol to be completed.

Phase 2 : Commit

Total-e-Transactions sends the commit order to the resource managers. If Total-e-Transactions crashes before it was able to send the commit order to one or more resource managers, then these remain blocked. A crash recovery mechanism or autonomous decision from the database is then required for the resources to be freed. (Please refer to part 4.2.2 for a complete description of the Total-e-Transactions crash recovery component.)

Figure 7: Managing a distributed transaction with a two-phase commit protocol

The distributed transaction covered in the example above is controlled by the JTA API and acts on XA resources. Therefore the transaction will, necessarily, be flat. For more modular development and a subtler degree of error management, the nested transaction model is more effective. Total-e-Transactions enables nested transactions involving different resource managers to execute a two-phase commit.

➲ Interposition

A distributed transaction involves resources residing in different address spaces, most often distributed over different machines in a network. Initializing the distributed transaction involves enlisting the resources with the transaction manager's root coordinator, which will require as many IIOP network calls as there are resources on remote machines. Distributed transaction coordination also requires IIOP Network calls between the resources and the transaction manager located on different machines.

A network call requires a set of operations (marshalling, demarshalling, object activation, IIOP communication…) that use up a lot of CPU and network bandwidth, and are subject to network outages. Optimization of IIOP calls may make it possible to improve application robustness and performances. The CORBA OTS specification provides for such a mechanism; it is called Interposition, and is supported by Total-e-Transactions.

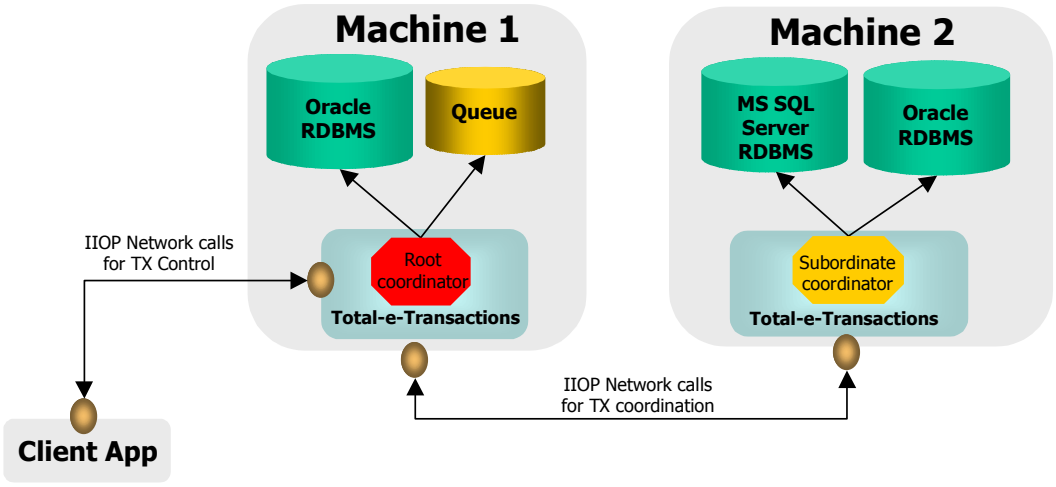


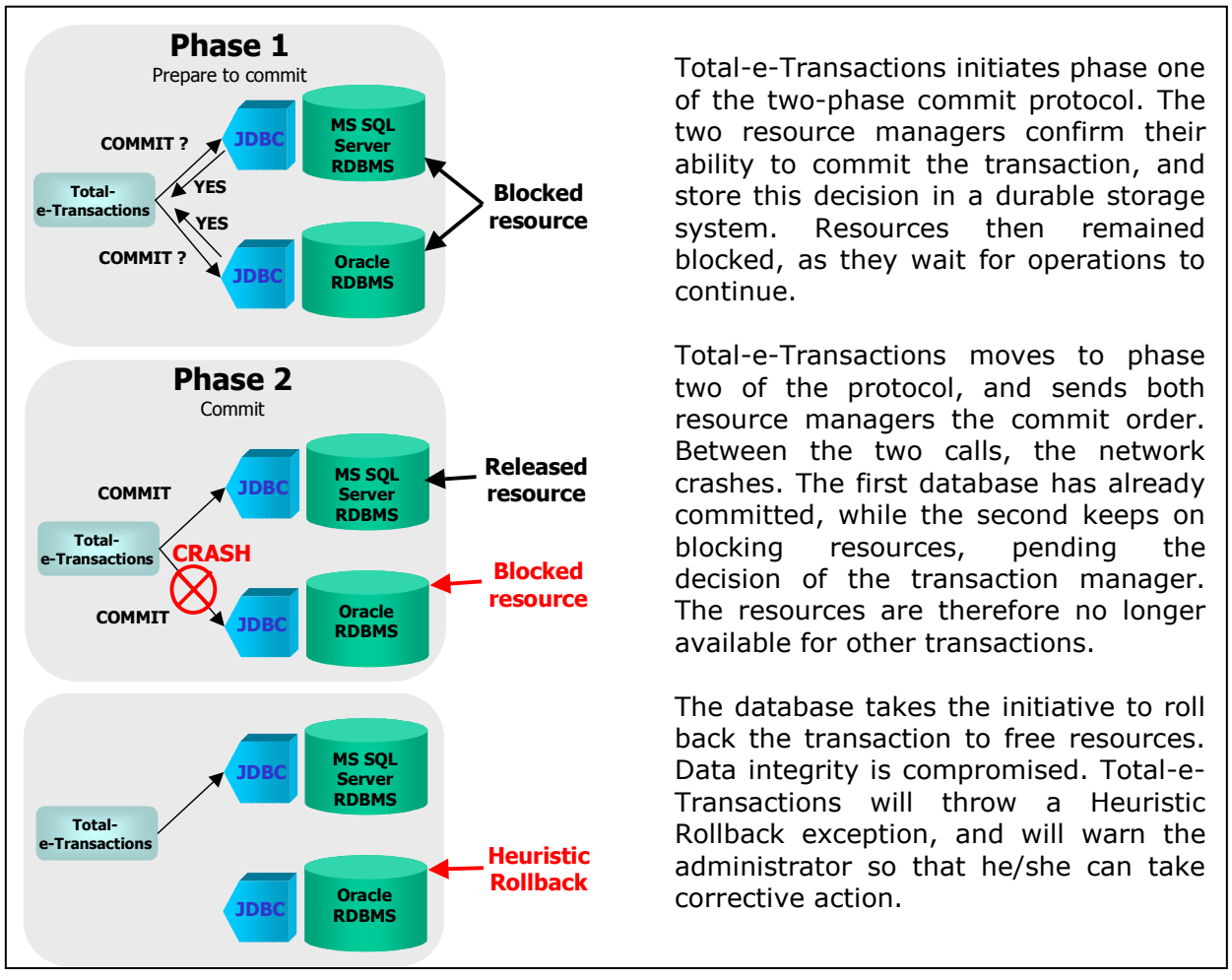
Figure 8: Interposition

The above example shows a client application initializing a transaction distributed over four resources located on two machines. The root coordinator of machine 1 must enlist the resources involved, two of which are found on machine 2, which means IIOP calls will be required. Interposition enables the enlistment of machine 2's resources to be delegated to a subordinate coordinator, which enlists with the root coordinator of machine 1. Transaction coordination is also more efficient, because the number of remote calls is kept to a minimum.

➲ Transaction heuristics

The OTS specification enables participants in a transaction to decide unilaterally to roll back, after the *prepare to commit* phase, without awaiting the decision of the transaction manager. This is called a heuristic decision. Heuristic decisions should be made with extreme caution and are assumed to be a relatively rare event, as they may differ from the decision made by the transaction manager, which can then cause the system to lose integrity.

These decisions can, however, be useful in certain cases; this is shown in the example below, which involves a transaction distributed over two databases, using a two-phase commit protocol and subject to a network outage during the post-prepare phase:



Total-e-Transactions initiates phase one of the two-phase commit protocol. The two resource managers confirm their ability to commit the transaction, and store this decision in a durable storage system. Resources then remained blocked, as they wait for operations to continue.

Total-e-Transactions moves to phase two of the protocol, and sends both resource managers the commit order. Between the two calls, the network crashes. The first database has already committed, while the second keeps on blocking resources, pending the decision of the transaction manager. The resources are therefore no longer available for other transactions.

The database takes the initiative to roll back the transaction to free resources. Data integrity is compromised. Total-e-Transactions will throw a Heuristic Rollback exception, and will warn the administrator so that he/she can take corrective action.

fig 9: Heuristic Rollback exception example

➲ Multi-thread aware

Unlike many other transaction managers, Total-e-Transactions enables a transaction to be managed by several threads, and enables the thread to manage several transactions.

➲ Explicit and Implicit transaction context propagation

Each transaction has its own context. This context is essential within distributed transactions. The OTS implementation of Total-e-Transactions is able to transfer the transactional context via the architecture components in a way that is transparent for the developer. However, it is possible to deactivate implicit propagation and to carry it out manually through programming. This technique provides a way to develop transactional applications more flexibly. In particular, it enables checked/unchecked transaction behavior to be supported, and allows transactional and non-transactional methods to be combined through a single interface.

➲ Checked/unchecked transaction behavior

Total-e-Transactions supports both Checked and Unchecked transaction behavior. The 'classical' transaction exhibits checked behavior whereby the transaction will not commit until all participants have completed their work, and only when the originator requests commit processing. Unchecked behavior allows relaxed models of atomicity to be implemented: the developer takes responsibility for ensuring that all outstanding work has completed before the transaction terminates.

➲ Direct / Indirect Transaction Management

The developer can choose to work the transaction directly by using the Control, Coordinator and Terminator services. Indirect transaction management is where the transaction control work is carried out through the Current object. This is similar to using the JTA, where transaction control and creation is abstracted from the user.

➲ Synchronization Interface

A synchronization interface is available for objects that are to be notified before a transaction is ready to start and/or after a transaction finishes its two-phase commit processing. Synchronizations are typically employed to flush the volatile state of objects or databases before the transaction commits.

➲ Support for CosTransaction::Current

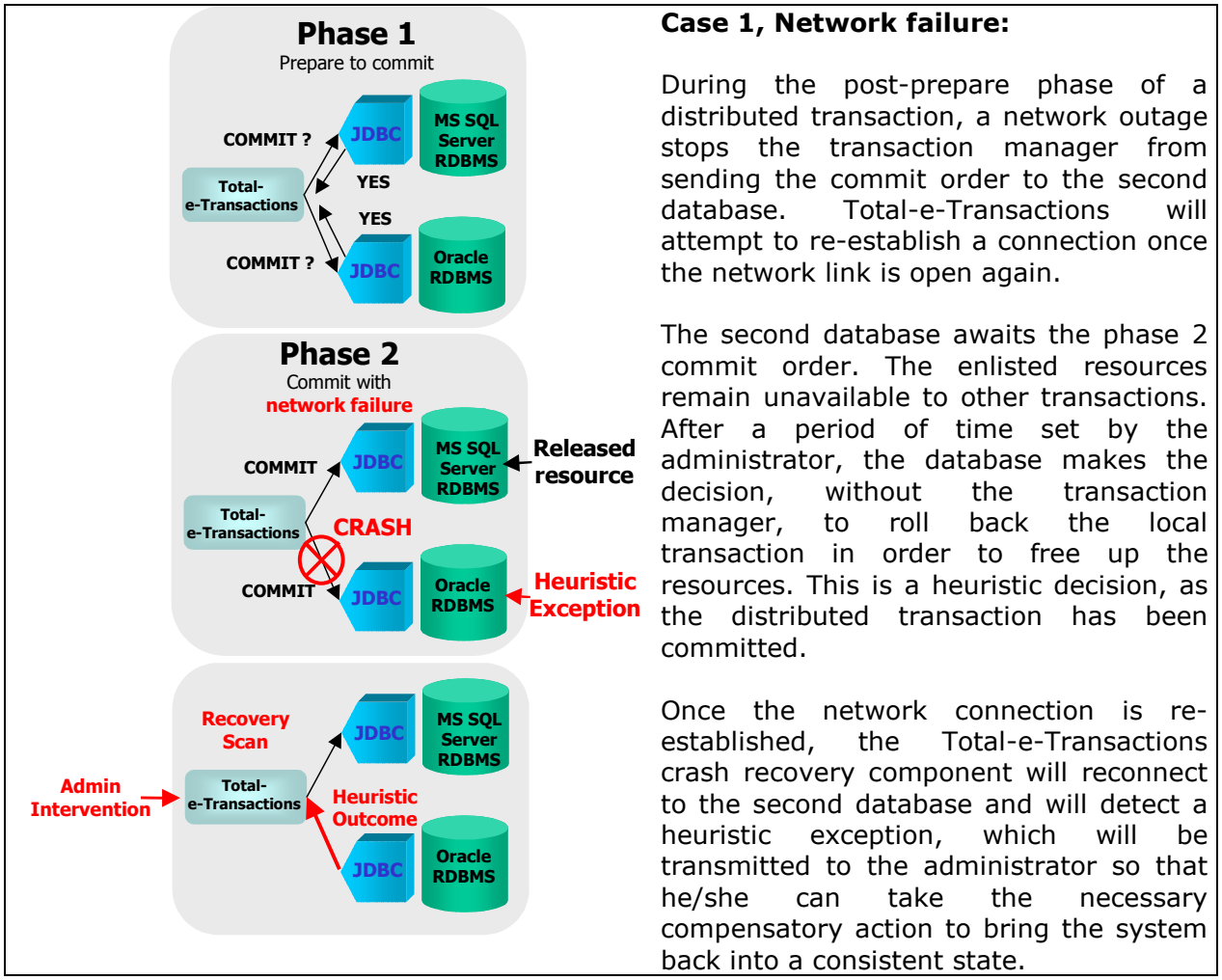
Total-e-Transactions supports the CosTransaction::Current object in addition to the JTA. This allows developers some flexibility in the way they wish to communicate with the Transaction Service.

The main advantage of CosTransaction::Current, with regard to the JTA UserTransaction interface for transaction control, is that it offers real management of multi-threading and nested transactions.

4.2.2. Crash Recovery

A component in a transactional architecture may fall prey to a failure (software, hardware or network) during a transaction. It is vital for the transactional infrastructure to carry out corrective action when the system is rebooted, in order to restart in a state that is consistent and compliant with ACID properties. Total-e-Transactions' Crash Recovery module handles data repair operations after one or more components crashes.

The diagrams in this section show how the crash recovery component brings the system back into a consistent state after two types of failure.



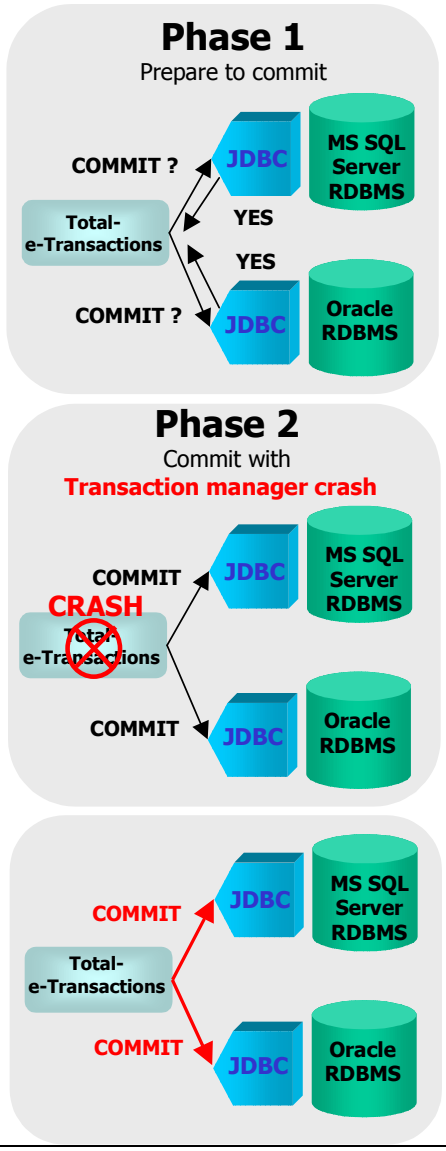
Case 1, Network failure:

During the post-prepare phase of a distributed transaction, a network outage stops the transaction manager from sending the commit order to the second database. Total-e-Transactions will attempt to re-establish a connection once the network link is open again.

The second database awaits the phase 2 commit order. The enlisted resources remain unavailable to other transactions. After a period of time set by the administrator, the database makes the decision, without the transaction manager, to roll back the local transaction in order to free up the resources. This is a heuristic decision, as the distributed transaction has been committed.

Once the network connection is re-established, the Total-e-Transactions crash recovery component will reconnect to the second database and will detect a heuristic exception, which will be transmitted to the administrator so that he/she can take the necessary compensatory action to bring the system back into a consistent state.

Fig 10: Crash recovery with Network failure



Case 2, Transaction manager failure:

At the end of the prepare phase, the transaction manager takes the decision to commit and record it in a stable storage system. The transaction manager fails. Just after rebooting, the transaction manager detects that a transaction in progress has been stopped irregularly, and will try and communicate with the resource managers involved. In our case, they are still on standby, and the crash recovery component will continue the transaction.

Fig 11: Crash recovery with Transaction manager failure

4.2.3. Transactional JDBC Drivers

Relational database management systems (RDBMSs) form one of the essential building blocks of any information system, and are generally used for storing data. Modern relational databases, such as Oracle or Microsoft SQL Server, are also able to manage transactions, but only for the data that they manage. This is referred to as a lightweight transaction.

A transaction that brings several relational databases into play is said to be distributed, and must be directed by a transaction manager such as Total-e-Transactions. The role of RDBMSs in distributed transactions is extremely important, since they help guarantee the ACIDity of transactions, coordinating with the transaction manager. RDBMSs offer internal mechanisms such as management of concurrent data access, which are often used for distributed transactions.

The connectivity of J2EE architectures with relational databases is covered by the JDBC standard, which offers a standard access API regardless of the RDBMS in question. The JDBC standard has enjoyed considerable success in the industry, and numerous implementations are available on the market. JDBC 2.0 supports X/Open DTP distributed transactions as it complies with the XA standard. JDBC 1.0 does not support distributed transactions.

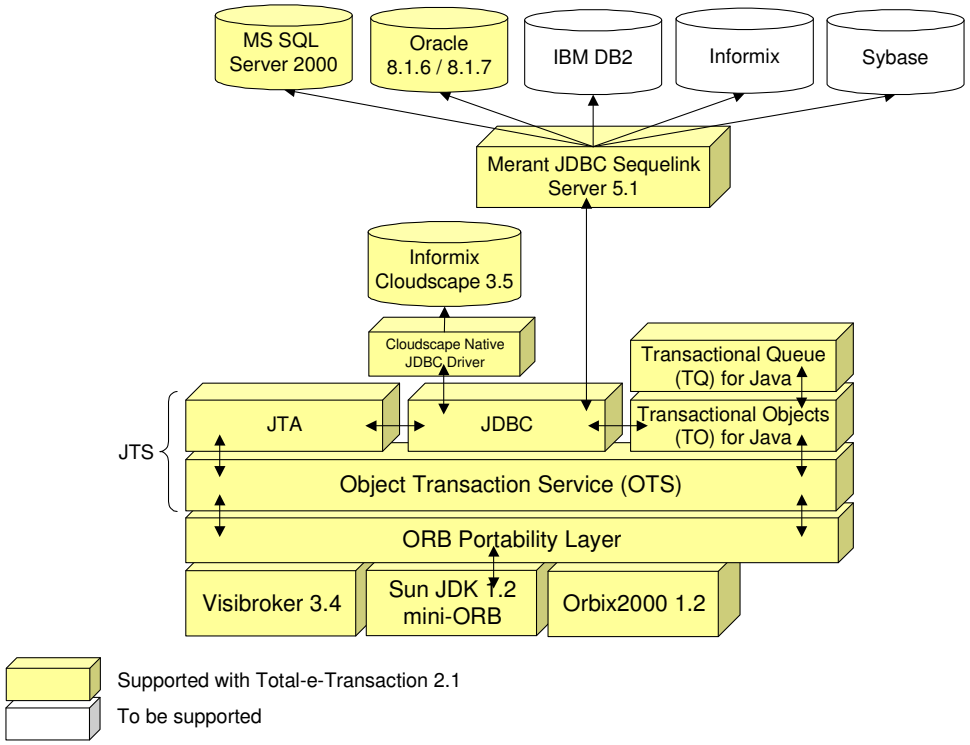


fig 12: Total-e-Transactions JDBC connectivity using Merant technology

Total-e-Transactions supports JDBC 1.0 and 2.0, thanks to the Merant Sequelink server 5.1 product which is included with Total-e-Transactions 2.1. Merant Sequelink Server 5.1 supports many JDBC 2.0-compliant RDBMSs (such as Oracle, MS SQL Server, IBM DB2, Informix and Sybase), enabling distributed transactions to be carried out. However, Total-e-Transactions only supports Oracle 8.1.6/8.1.7 and MS SQL Server 2000 connectivity through Merant Sequelink Server 5.1. Total-e-Transactions also supports the Informix Cloudscape 3.5 object relational database through its native JDBC 2.0-compliant driver. HP plans to extend support to more databases using Merant Technology.

As JDBC access is costly in terms of performances, Total-e-Transactions offers optimization mechanisms (JDBC pooling) so as to minimize the resources monopolized by the transactions using RDBMSs.

We note that as the JDBC transactional model is based on the DTP one, it is therefore limited to the flat transaction model. This means that the Total-e-Transactions distributed transactions involving JDBC resources will, necessarily, be flat. However, Total-e-Transactions does offer an alternative with a software layer for persistence of business objects in JDBC sources, which supports the nested transaction model. This software layer is described below.

4.2.4. Transactional Objects (TO) for Java

To avoid any confusion, we must point out that the Total-e-Transactions Transactional Objects for Java that we shall be describing in this section has no direct link with the transactional objects mentioned in the CORBA OTS specification.

The OTS implementation provided by Total-e-Transactions is used to develop business object-based transactional objects; however, OTS remains a low-level framework with less than optimum productivity. The developer may have to create and manage transactional contexts, enlist resources, and so on. The purpose of Transactional Objects for Java (TOs for Java) is to offer a high-level framework which can mask technical aspects, and a low-level development framework to enable the developer to concentrate on business objects and transactional processes. The functionalities of the TO for Java are as follows:

- *Object persistence:* TO for Java offer an automatic persistence facility for objects in a relational database or flat file (persistence here is limited to simple object models, and is not comparable to that offered by complex O/R mapping tools)
- *Concurrent access:* TO for Java offer a preconfigured locking mechanism for objects in read-only (shared) and in write (exclusive) mode.
- *Management of nested OTS transactions:* the TO for Java's persistence and concurrence management properties coordinate with the underlying JTS engine, to guarantee transaction ACIDity.

Access to concurrence control mechanisms uses interfaces that are fully extensible in order to respond to particular situations. This functionality is very important, since durability (the D in ACID), and isolation (the I in ACID) are features often provided by resource managers (e.g. databases) that must function in conjunction with the transaction manager, which guarantees atomicity and consistency – the A and the C in ACID.

The J2EE EJB model offers similar functionalities: an automatic, personalizable persistence mechanism, transactional capabilities and integration of asynchronous communications with EJB 2 (Message Driven Bean).

EJBs are limited in a number of areas, which can be problematic when it comes to implementing large-scale systems :

- Threads running through EJBs cannot re-enter,
- EJB transactions rely on the JTA standard and are therefore limited to the flat transaction model
- Only one thread can pass through an EJB instance
- Management of fine-grained objects is not yet definitive
- Access locking strategies cannot be personalized

The component model proposed by TO for Java does not impose such limitations, making it a viable alternative to EJBs for some architectures.

4.2.5. Transactional Queue (TQ) for Java

Applications developed with TO for Java can use nested transactions, but these are synchronous. This means that if an object that has to participate in a transaction is not available at the instant it is called, this will block the entire transaction, resulting in a long wait or rollback of the transaction.

The notion of the transaction queue is particularly useful when the systems taking part in a transaction are uncoupled. The situation is typical when business processes require ACIDity but the availability of the application involved cannot be guaranteed. The role of the transactional queue here becomes clear: when a transaction involves a remote application, a message is sent to a queue (enqueuing) and the transaction is suspended without blocking the initiating program; when the remote application becomes available, the message is taken from the queue (dequeuing) and the transaction continues to be executed.

As it is built upon TO for Java, TQ for Java is nested-transaction-aware. The next paragraph describes an example of an inter-bank transfer involving a transactional queue built with TQ for Java involved.

4.2.6. The Empay demonstration application

EMPAY is a real transaction and distributed object-based application shipped with documentation that demonstrates the product features of Total-e-Transactions, including TO for Java, TQ for Java, and both single and two-phase commit capabilities. EMPAY is CORBA based and implemented in pure Java.

The main components of EMPAY are two clerks, two bank servers and a clearing house.

Each bank server is made up of a database in which customer accounts and bank transactions are stored (transfers, account creation / deletion / modification). Bank transactions are carried out by clerks. The bank servers can transfer funds to accounts from different banks; these transfers may be direct or may pass via a clearing house, which uses a transactional queue to process payment operations.

Each bank server consists of a CORBA server implemented in Java, and a JDBC database. The CORBA servers feature four interfaces:

- The *access service* interface gives the clerk secure access to dedicated objects which perform the bank transactions.
- The *AccountMaintenance* interface is used to perform various back-end operations, not directly invoked by the clerks.
- The *MoneyTransfer* interface is transactional, and takes care of intra- or inter-bank payments.
- The *PaymentProcessor* interface is also transactional, and carried out transactions by passing via the clearing house.

The clearing house acts as intermediary between the two banks. Unlike the direct inter-bank transfer, the clearing house uses a transactional payment queue to enable transactions to be managed should one of the bank servers become unavailable. The payment interface is *paymentProcessor*.

The bank servers are controlled by the clerks, which can be operated via command line or GUI.

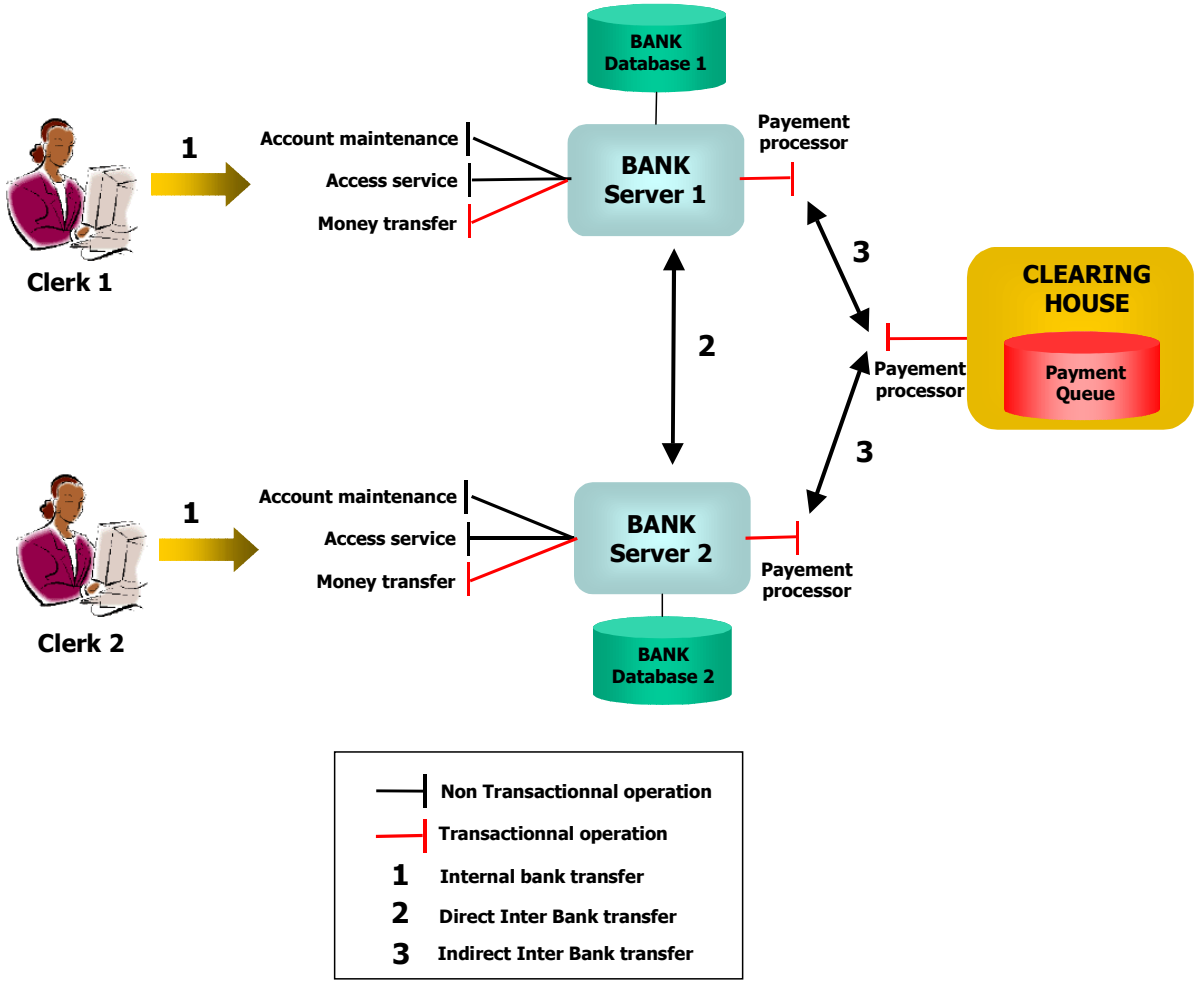


Fig 13: EMPAY architecture

Five bank transaction scenarios are provided to illustrate the transactional capabilities of Total-e-Transactions:

	Scenario Description	Total-e-Transactions Features used
Scenario 1	Account creation and intra-bank transfers using command lines	TO for Java
Scenario 2	Account creation and intra-bank transfers using GUI	TO for Java with a Graphical user interface
Scenario 3	Direct inter-bank transfers	TO for Java, 2PC (2 phase commit)
Scenario 4	Automatic transfer of funds by scripting	TO for Java, 2PC
Scenario 5	Inter-bank transfer using the clearing house	TQ for java, 2PC

5. Total-e-Transactions integration with HP's Total-e-Server and HP Application Server

Total-e-Transactions is a robust transactional architecture which can be used as a standalone application. However, it is sometimes necessary to open transactional applications up to different presentation tiers (Web browser, PDA…), to manage security, workload increases and availability. These needs are covered by J2EE application servers such as HP Total-e-Server and HP Application Server.

Total-e-Transactions is completely implemented in Java, which means it can be easily integrated with a J2EE application server; HP has gone ahead with this integration, endowing Total-e-Server and HP Application Server with the Total-e-Transactions transaction engine.

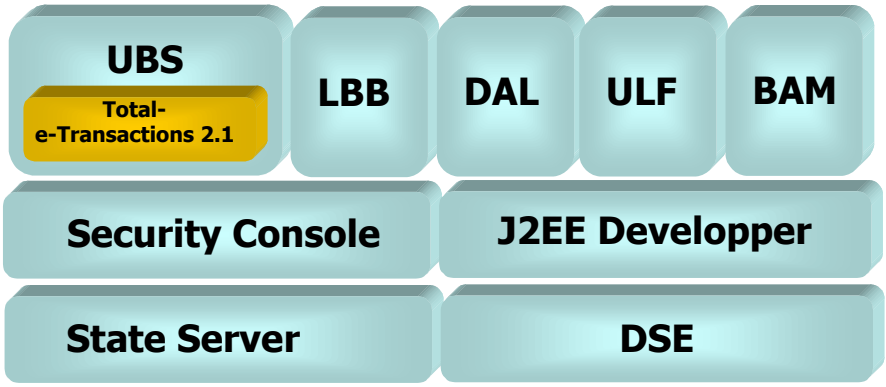


fig 14: Overview of Total-e-server

The diagram above gives an overview of the components in Total-e-Server:

- **UBS (Universal Business Server)** is made up of a servlet/JSP engine for integration with the Web, an EJB engine, an XML server and a **DSE (dynamic style sheet Engine)** which extends Total-e-Server's connectivity to mobile devices (PDA, WAP devices…)
- The **State server** and **LLB (Load Balance broker)** components provide the infrastructure with fail-over and load balancing capabilities, respectively.
- **DAL (Dynamic Application Launcher)** optimizes fail-over and lightens the administration task .
- **J2EE developer** offers a development environment for J2EE components (Java code, SQL code, EJB, XML/DTD)

- The **Security Console** is used to manage the lists for controlling access to J2EE components
- **ULF (Universal Listener Framework)** extends the connectivity of UBS to other protocols (TCP, MQ Series, e-mail, FTP…)
- **BAM (Bluestone Application Manager)** is the administration environment of Total-e-Server

Total-e-Server and Total-e-Transactions are products designed to work together, which enable two types of transactional applications to be developed:

- **100% J2EE Applications:** By conforming to the JTS standard, Total-e-Transactions enables Total-e-Server to develop transactional applications based on EJB components. Developments therefore benefit from the easy-to-use component models, while remaining 100% compatible with the J2EE standard.

- **Non-J2EE applications:** Total-e-Transactions extends the transactional capabilities of J2EE with its TO for Java and TQ for Java frameworks, providing features which are not covered by the JTA standard (support for nested transactions, subtler multi-threading management, interposition, and so on).

6. Summary

This document highlighted the importance of the transactional paradigm when managing business processes. The need for transactions is not a new one, and it becomes even more meaningful in the age of management of processes within large-scale corporate intranets. The way in which transactions were managed yesterday is far removed from today's way of doing things: information systems are distributed and heterogeneous. The aim of HP Total-e-Transactions is to provide an infrastructure able to manage distributed intra-enterprise processes transactionally.

Tomorrow's transactional architecture will have to comply with standards to manage transactions. Different standards have been developed over the history of transactional computing, in order to meet the needs which have emerged as time passes. Total-e-Transactions conforms to the essential object-based transactional standards, by being fully compatible with J2EE (JTA/JTS) 1.2 and the CORBA OTS specification, which ensure transactional interoperability between the other components of the architectures, such as ORBs and databases.

Total-e-Transactions is the industry's first 100% pure Java JTS implementation on the market, which secures it a singular position and lends it the following advantages:

- **JTA API support**, offers a high degree of productivity for developers and transactional compatibility with XA sources
- **CORBA OTS support** guarantees transactional interoperability with other transaction managers, an essential feature in the construction of a true transactional infrastructure
- **Independence from the operating system and hardware** is provided thanks to the 100% Java product, ensuring flexibility of deployment
- Total-e-Transactions can be used as a **standalone product**,
- Total-e-Transactions is also integrated **with the J2EE-compliant HP Total-e-Server and HP Application Server**, and combines the potential of a leading J2EE application server with unique transactional functionalities.

A white paper written by

TechMetrix Research

Author

Nicolas Farges
Yannick Bessy

Managing Editor & Translation

Hannah Riley

Published

RC01 - December 2001

USA

TechMetrix Research
76 Bedford Street, suite 33
Lexington, MA 02420

Tel.: +1.781.890.3900
Fax: +1.781.240.0502

EUROPEAN HEADQUARTERS

TechMetrix Research/SQLI
55/57 Rue Saint Roch
75001 PARIS
FRANCE

Tel.: + 33 1 44 55 40 00
Fax: + 33 1 44 55 40 01

SWITZERLAND

TechMetrix Research/SQLI
Chemin de la Rueyre
116-118
CH-1020 RENENS

Tel.: + 41 (0) 21 637 72 30
Fax : + 41 (0) 21 637 72 31

http://www.techmetrix.com
info@techmetrix.com

TechMetrix Research is a technically oriented analyst firm focused on e-business application development needs. TechMetrix Research has developed a unique evaluation approach to provide accurate information on software. Based in Waltham-MA (USA) and Paris (France), the firm publishes comparison reports and product reviews, which are real helpers when it comes to making decisions, or simply keeping pace with the fast moving e-business market.
As its parent company (SQLI) provides information system development and implementation services to major companies, TechMetrix Research also benefits from the feedback and experience acquired during large-scale, long-term development projects.
SQLI is a European global system integrator of 700 employees offering full service and continuing coaching to enable companies to move profitably toward an all-Internet solution.