AG Heterogene Informationssysteme
Prof. Dr.-Ing. Stefan Deßloch
Fachbereich Informatik
Technische Universität Kaiserslautern

Middleware for Heterogeneous and Distributed Information Systems – Handout to Exercise Sheet 2

Wednesday, November 5, 2008 – 10:00 to 11:30 – Room 48-379

SQLJ Program Preparation

SQLJ is a standard for embedding SQL in the Java programming language. Unlike JDBC, which is a call-level interface (CLI), SQLJ is a language extension. The preparation of SQLJ programs involves a series of steps[1]. The preparation process is depicted in Figure 1. The straight lines indicate compile time dependencies while the dashed lines indicate runtime dependencies.

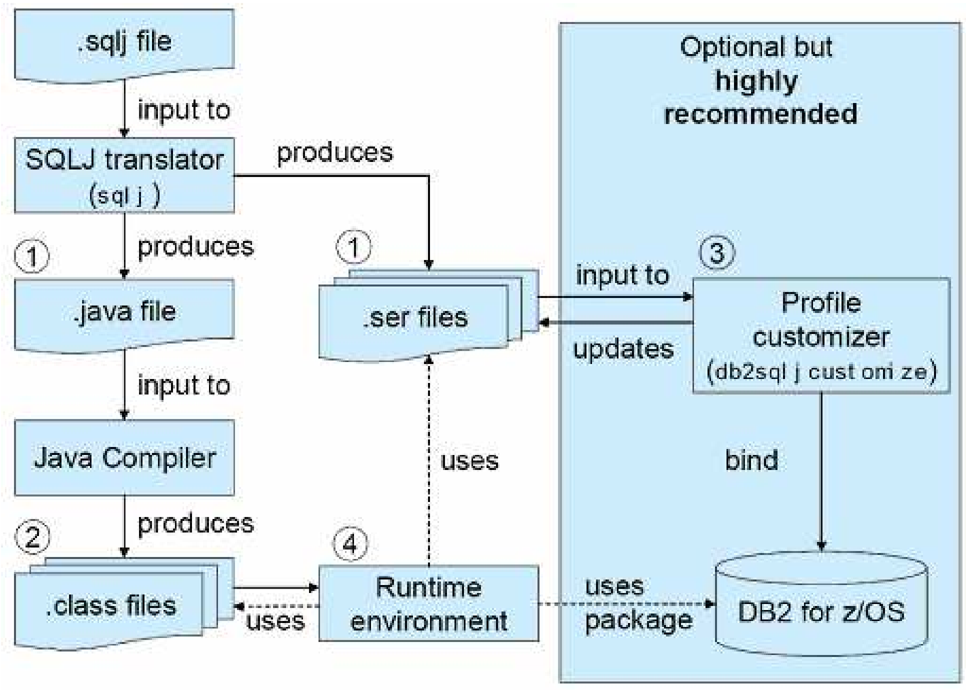

Figure 1: The SQLJ Program Preparation Process

- Create a SQLJ source file.
- Precompile the SQLJ source file using the SQLJ translator (1). The translator checks for correct Java and SQLJ syntax and replaces embedded SQL state-

[1] See IBM Redbook DB2 for z/OS and OS/390: Ready for Java chapters 9, 10, and 11 available at http://www.redbooks.ibm.com/abstracts/sg246435.html

ments in the SQLJ program with valid Java statements. The output of the preprocessing step is a compilable Java source file and one or more serialized profile files. Profiles contain information about all the SQL statements in the program. A profile consists of one or more profile entries, each of which describes one SQL statement.

```
sqlj -compile=false <sqlj_file>
```

- Optionally view the information stored in the serialized profile files. IBM DB2 is equipped with the *profile printer* utility that displays the information contained in a serialized profile in human-readable form.

  ```
  db2sqljprint <ser_file>
  ```

- Compile the java program (2). The compiled program is executable. However, up to this point *dynamic* SQL is used, i.e. SQL statements are prepared and executed dynamically using JDBC calls. An additional customization step is necessary to run *static* SQL instead.

  ```
  javac <java_file>
  ```

- Use the profile customizer to add customization information to the generated profiles (3). You may want to run the profile printer utility again to inspect the changes to the serialized profile file. Note that the customization process is DBMS specific. In case of DB2 so called *packages* are created during profile customization. Roughly speaking, packages are database objects that contain executable forms of SQL statements. When the Java program is run after profile customization, the SQLJ runtime uses the profile information to execute the packages instead of the source SQL statements. That is, after profile customization *static* SQL is used.

  ```
  db2sqljcustomize –url <jdbc_url> -user <db_user>
                   -password <db_ password> <ser_file>
  ```

- Run the java application (4).

  ```
  java <class_file>
  ```

X/Open DTP (Local Environment)

The X/Open Distributed Transaction Processing (DTP) model allows application programs to share resources provided by multiple resource managers and to coordinate their work in global transactions using the two phase commit protocol.

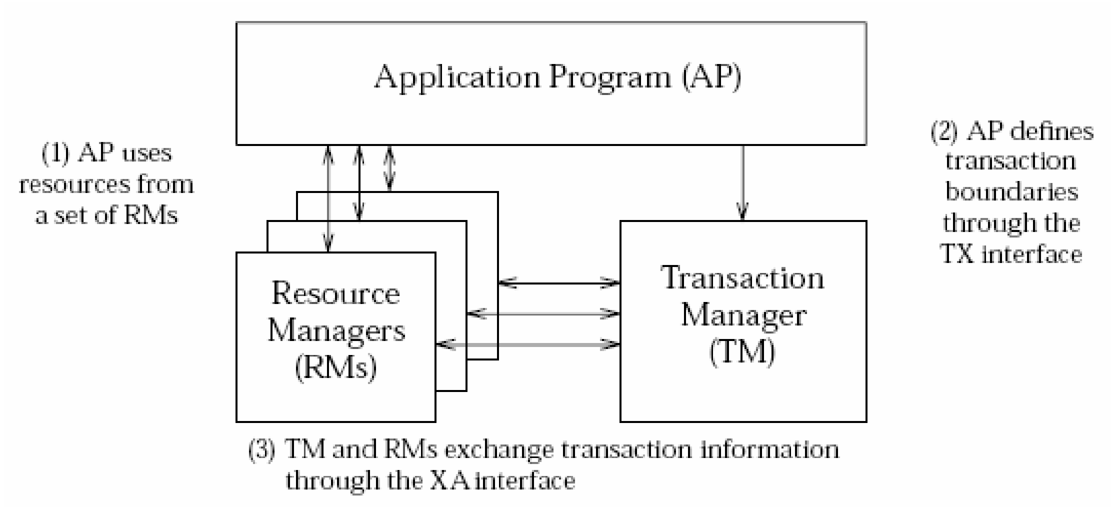
Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through the TX interface
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information through the XA interface

Figure 2: Functional Components and Interfaces

The X/Open DTP model distinguishes three functional components involved in distributed transaction processing, namely the Application Program (AP), the Transaction Manager (TM), and Resource Managers (RMs) as shown in Figure 2. The AP defines the start and end of global transactions, accesses resources within transaction boundaries, and normally makes the decision whether to commit or roll back the transaction. The transaction manager (TM) assigns unique identifiers (XID) to transactions, monitors their progress and coordinates participating Resource Managers during transaction completion (commit or rollback). A Resource Manager (RM) manages a shared resource that may be accessed by the AP using services that the RM provides. Examples of RMs are database management systems (DBMSs), file access methods such as X/Open ISAM, or print servers.

`tx_begin( )`	Begin a global transaction.
`tx_close( )`	Close a set of resource managers.
`tx_commit( )`	Commit a global transaction.
`tx_rollback ( )`	Roll back a global transaction.

Table 1: Functions of the TX interface (not complete)

`xa_commit( )`	Tell the RM to commit a transaction branch.
`xa_prepare( )`	Ask the RM to prepare to commit a transaction branch.
`xa_start( )`	Start or resume a transaction branch - associate an XID with future work that the thread requests of the RM.
`ax_reg( )`	Register an RM with a TM to join a transaction.

Table 2: Functions of the XA interface (not complete)

The components of the X/Open DTP model interact by means of standardized interfaces. The TX (Transaction Demarcation) interface[2] allows the AP to call the TM to demarcate global transactions and direct their completion. An (incomplete) list of the functions of the TX interface is found in Table 1.

The XA interface[3] is a bidirectional interface between the TM and the RM. It is used by the RM to join a transaction managed by the TM. It is further used by the TM to coordinate multiple RMs during transaction completion. An (incomplete) list of the functions of the XA interface is found in Table 2.

X/Open DTP (Distributed Environment)

The X/Open DTP model has been generalized to support transactions in distributed environments. That is, multiple TM domains may be involved in a global transaction. A dedicated component, referred to as Communication Resource Manager (CRM), is used to propagate transaction information across TM domains as shown in Figure 3.

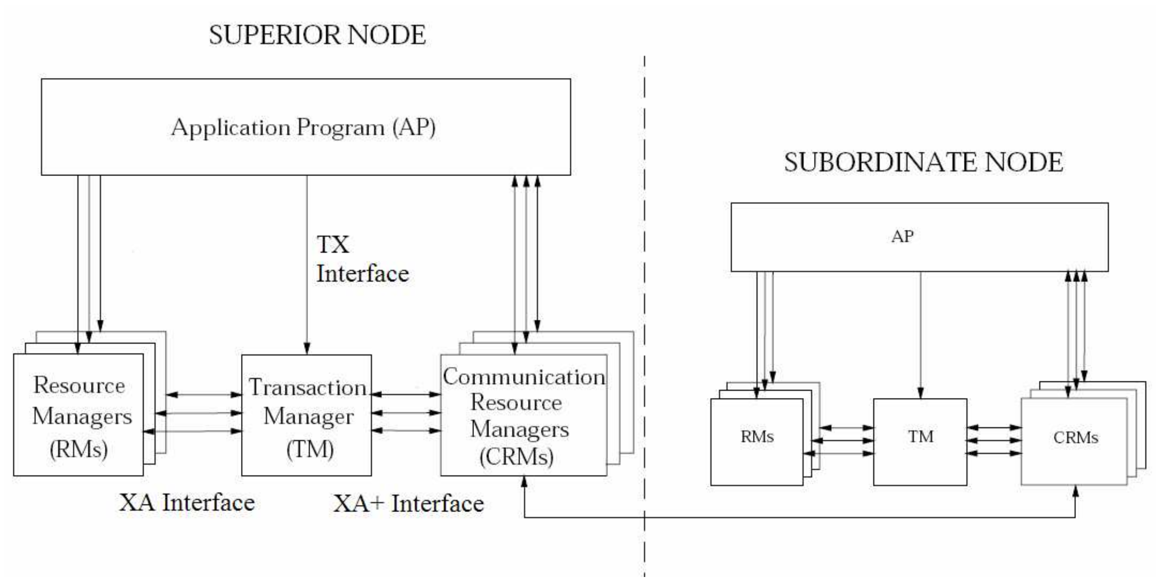

Figure 3: Functional Components and Interfaces

[2] See *Distributed Transaction Processing: The TX (Transaction Demarcation) Specification* available at http://www.opengroup.org/onlinepubs/9694999599/toc.pdf
[3] See *Distributed Transaction Processing: The XA Specification* available at http://www.opengroup.org/onlinepubs/009680699/toc.pdf

The XA interface has been extended to allow the CRM to propagate coordination information to subordinate TMs. The revised interface is referred to as XA+ interface[4]. An (incomplete) list of the functions of the XA+ interface is found in Table 2.

`ax_commit( )`	Propagate transaction branch commitment to a transaction manager.
`ax_prepare( )`	Propagate transaction branch prepare to commit to a transaction manager.
`ax_rollback ( )`	Propagate transaction rollback to a transaction manager.
`ax_start( )`	Notify the transaction manager to propagate or resume a transaction branch association with this thread of control.

Table 3: Functions of the XA+ interface (not complete)

Figure 4 illustrates the interactions between the functional components during transaction processing in a distributed environment. The application program begins a global transaction and accesses both, a local and a remote RM. The application then asks the local TM to commit the transaction. No errors occur during the 2PC and hence, the transaction completes successfully.

[4] See *Distributed Transaction Processing: The XA+ Specification Version 2* available at http://www.opengroup.org/onlinepubs/8095979699/toc.pdf

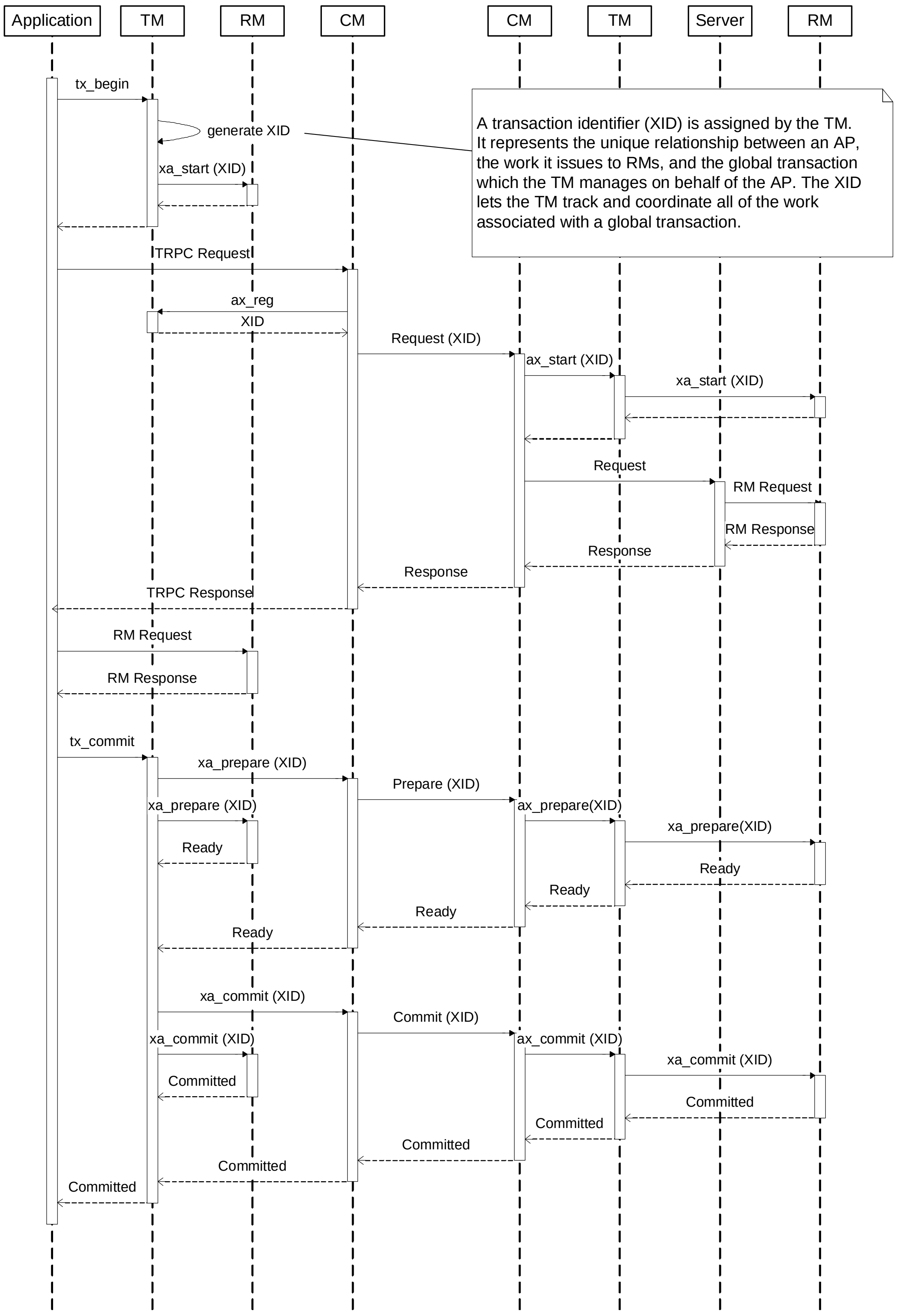

Figure 4: Sample distributed transaction involving two TM domains (simplified representation)