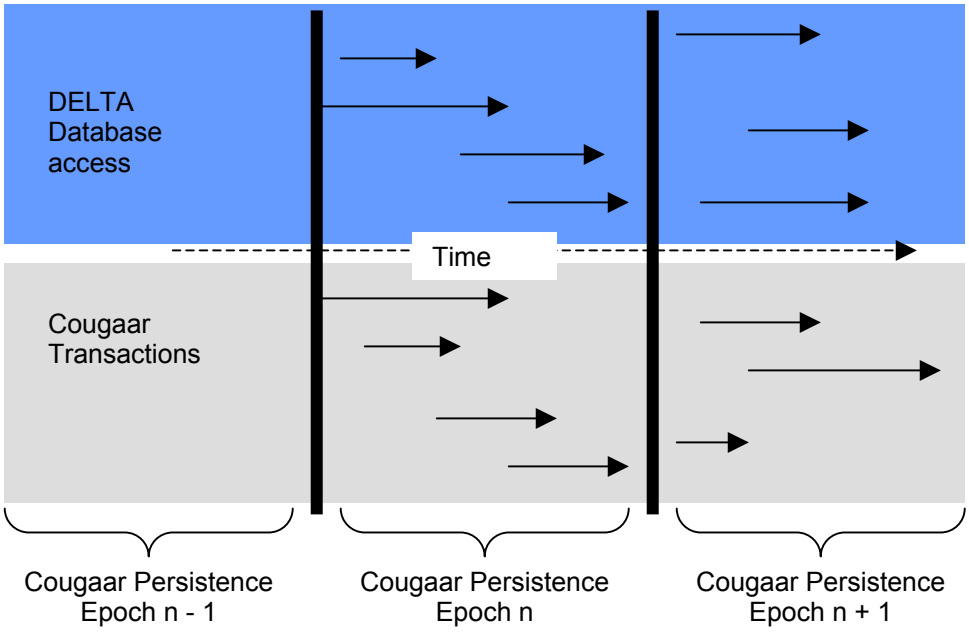
Persistence and Transaction Integrity in DELTA

The DLA/DARPA-sponsored DELTA system is a real-world example of how a Cougaar-based application can leverage Cougaar's persistence support to provide persistence and transaction integrity throughout the application. The DELTA application is continuously receiving supply requisitions which are processed via a sequence of steps: identifying possible suppliers, ranking and selecting the best supplier, updating contract status, communicating purchase requests with the selected supplier, and updating external accounting information. Since each step in the processing sequence results in state changes in either the application database instance or in file reads/writes, it is essential that DELTA have a mechanism for maintaining transaction integrity, even in the midst of unplanned application termination. The DELTA system contains state information of four types: 1) Java virtual machine memory, 2) DELTA application database, 3) Cougaar persistence database, and 4) files (reads and writes). At any given time, the state of the system must be consistent between these four. DELTA builds on Cougaar persistence and Oracle transaction integrity in order to maintain state consistency.

As described in the *Cougaar Architecture Guide*, Cougaar saves snapshots of the blackboard to the persistence database at the end of each persistence epoch. By using the Cougaar-provided database connection pools, DELTA uses the same database connections as the underlying Cougaar infrastructure. As a result, any database changes that result from DELTA processing are committed at the same time as the blackboard snapshots. Oracle's transaction integrity ensures that the entire set of changes succeeds or fails as a group. After a change is made but before it is committed, the change is only visible within the session that made the change. If the DELTA goes down in the middle of a persistence epoch, none of the database changes have been committed so the system state is restored from the last saved snapshot.

Cougaar Persistence Epochs and DELTA Database Access



Any system failure during epoch n results in a transaction rollback and system restoration to the state at the end of epoch n – 1. After epoch n is finished and a snapshot saved to the database, the state at the end of epoch n becomes the new rollback state for any system failure during epoch n + 1. The result is that each agent in the DELTA application is always internally consistent, both within its internal blackboard and with its file and database modifications.

File Reading

Example: MilstripFileReaderPlugIn

DELTA uses a database table to maintain the processing state of each input file. This file database is accessed via the common database connection as other database updates, ensuring transaction integrity. The FilePosterPlugIn polls the input directory and writes the file names of input files into the database. File reading plugins implement FGIFileReader and have a file reading thread, FGIFileReaderThread, which obtains the file names from a database table of files to read, processes the files, and marks them in the database as having been read. Files that have been marked as read are later removed/renamed and the entry is cleared from the database. In the file reading process, consistent state is maintained because for each batch of files to be read, the files are marked as read in the database and any effects from processing the files are committed at the same time. Effects from processing the files may include both database writes and changes to objects in the JVM that are recorded by Cougaar persistence. By synchronizing file processing through managed database connections (and shared transactions), DELTA is assured of properly processing each input file one, and only one time.

File Writing

Example: CSIAllocatorPlugInImpl

DELTA produces a number of output files that summarize the results of processed requisitions. Similar to the case of file input, the system uses a database table to maintain the processing state of each output file. This file database is accessed via the common database connection as other database updates, ensuring transaction integrity. A plugin uses the OutputFileTracker to create a new file, saving the new filename to the database in the table of files to be written. The file contents are generated and written, and the file entry in the database is updated to show that it has been written. Later the file entry is cleared from the database.

DELTA Processing Initiated Changes

Example: Requisition and contract status changes, item updates

During the normal course of requisition processing, a requisition's status and history are updated to provide a record of events. Similarly, during processing, some characteristics of contracts such as total amount sourced to contract will be updated. These changes must be recorded in the database so they are available to present and future DELTA user interfaces and to future DELTA agent instantiations. Through the LTAFactory, Java instances of contract objects are cached and shared so that changes to a contract object will be immediately visible anywhere in the system. Since DELTA processing initiated changes are committed to the database at the same time that the Cougaar blackboard snapshot is saved, the system will always be able to restart from a consistent state.

UI Initiated Changes

Example: Contract and rule edits

The DELTA user interfaces operate outside of the transactions involved with DELTA's Cougaar-based requisition processing. The user interfaces are implemented via a suite of servlets, using the same underlying set of factory classes (e.g., LTAFactory) as the agents, but with different database connections. Synchronization is maintained using version/timestamp information in the database (and in the DELTA LDM objects). Privileged users are able to edit details of the vendor contracts against which requisitions are sourced, and the business rules that determine sourcing decisions through a graphical user interface. While the user is making changes, DELTA agents may be processing requisitions and using the affected contract or rule information. There will be short a delay between the time the user saves changes (which makes changes to the database) and when DELTA agents begin to use the information, because requisitions which are currently being processed complete their processing with the old information. However, this delay is short and changes to rules and contracts are tracked by object versioning. For example, before the rules are applied to each requisition, they are re-cached from the database if necessary. Some apparent inconsistency may arise when viewing information through the UI since previous versions of contract and rule objects are not retained. UI screens linking requisition history to objects like contracts or rules display the most current version of the object. Some of this object information may have changed between the time it was used in requisition processing and the time it is viewed.