Microsoft Application TP Project

The purpose of this project is to gain an understanding of how to build a transaction processing (TP) application using a commercial application server and related products. Your goal is to use Microsoft's products to construct a distributed TP application that implements a travel reservation system.

The core TP services of Microsoft's product suite are called COM+ (formerly known as Microsoft Transaction Server (MTS)) and ship as part of Windows 2000. COM+ has a wide variety of features, including integration with many related products, such as Microsoft Message Queue, IIS, and SQL Server. The main grading metric is the number and complexity of transaction-related features of MTS and related products that you exercise and get working.

Background material:

- Documentation for COM+ can be found in MSDN under
  Platform SDK / Component Services / COM+
  especially the COM+ Services Primer and Services Provided by COM+.
- You will need to know a modest amount of SQL for this project, e.g., to create a simple database, to read and write a database in SQL, and to issue those SQL calls from an application (probably via ADO (Active Data Objects)).

Complexity of the application itself is *not* a goal and will not be rewarded, grading-wise. To start, the reservation system's functions should be as similar as possible to those of the Java/C# project. You should add application functionality only insofar as it helps you demonstrate some technical capability of the system.

A sample application that illustrates some of COM+'s basic features is described in the COM+ Services Primer. You should build your travel reservation system to illustrate the following basic features before embarking on other ones:

- Database connection pooling
- Shared property (resource) manager. For example, you could maintain the total amount of money that all customers have spent during today's sessions.
- Context object related to a Transaction Server object
- Transaction attributes (requires transaction, requires new transaction, supports transactions)
- Transactional RPC, which allows a component running a transaction to call another component to do further work on behalf of the same transaction.
- Stateful vs. stateless objects

Here are some other features to consider:

- Run one or more components in IIS and access them from a web browser.
- Run components in separate servers, possibly on separate machines, against a single database
- Run components in separate servers with different databases. For example, customer data (which includes a list of the customer's reservations) could be maintained in a separate database than flight, auto, and hotel data.
- Put the administrative functions into a separate component, so that you can protect them using a separate security role. Possibly add programmatic security.
- Force a transaction manager failure during the uncertainty period. Show how COM+ displays the transaction state, and show the state being resolved when the failure is repaired.
- Use Microsoft Message Queue for a queued transaction type, such as processing a request to join the airline's Airport Lounge Club.
- Use BizTalk Orchestration to run a multi-transaction workflow.
- Configure a Windows 2000 cluster, replicate one of the application servers, and show how it continues to run after a node fails.
- Write scripts that automate installation, so you application can be installed to run on another machine with minimal operator involvement.
- Demonstrate the use of DisableCommit.
- Do a little performance measurement.
- Develop a resource dispenser and/or integrate a simple resource manager with the Distributed Transaction Coordinator (2-phase commit). These are particularly challenging features — probably more than is doable in the available time. So if you're interested in this challenge, read the COM+ documentation carefully before deciding to dive in.

Feel free to suggest features other than those in the above list.

It's important to proceed incrementally. *Make sure you have a solid working solution of your base features before immersing yourself in the challenging feature*. Keep it simple enough to get an end-to-end system working well enough to run a demo. In particular, keep the user interface very simple.

Deliverables - A write-up is required to summarize what you have done. Especially, explain what product features you have exercised and where they appear in your code. The write-up can be short, as long as the code is readable and well commented. A final demonstration is required.