CS441 – Distributed Object Programming Using Middleware

Course Project

Total: 25% of your final grade

As part of taking CS441, you will work on a course project that allows you to get good hands-on experience with programming distributed objects using middleware. A goal of this project is to create and use distributed objects using a variety of different technologies, and these objects should interact with backend relational and so-called NoSQL databases as well as publically available services that can be accessed using the Internet. This project is different from the others that you worked on, since it has no clearly defined specification for an application that you will implement.

With explosion of social networking and data mining applications that operate on "big data," a marketing term for is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools[1], a main engineering effort is centered on achieving good performance of these applications. Recent industry trend is to use NoSQL databases, which is a buzzword and a misnomer for non-relational data management systems like map-reduce implementations, GemFire, column stores, etc. Essentially, the idea is that the ACID principle is too strong for many applications that use back-end databases, and in many cases, this principle can be relaxed and applications can tolerate occasional inconsistencies. Relational databases are by large fully ACID-compliant, and when many transactions from different applications are sent to a back-end relational database, this database becomes a bottleneck, since it enables only limited concurrency. Therefore, many organizations combine relational databases with NoSQL storages to increase the throughput (measured as the number of transactions executed per some unit of time, where a transaction is a collected of requests for data) at the expense of consistency of data.

Your task is to design an application that processes "big data" using a distributed rule-based architecture. Your application should be written in Java and I ask you to use WebRatio IDE. You should create and run a number of tests using JUnit, collect results and compare them with expected values (when possible), so that you make sure that your

[1] http://en.wikipedia.org/wiki/Big_data

application behaves as expected. In addition, you should develop performance tests using Apache JMeter[2] that load-test your application with different numbers of virtual users who request different services at the same time.

An intention for this course project is to boost your creativity in exploring various distributed object middleware platforms. It does not matter what type of application you create, but what matters is your sophistication in using different technologies and dealing with the complexity of different solutions. Since most nontrivial software applications use databases, you will design large database schemas for your application and populate your databases using randomly generated or some publically available data. You can obtain ideas for their projects from different sources including open-source software repositories; however, you cannot blindly reuse existing source code and documentation – it will be treated as academic dishonesty. If you reuse some parts of existing projects, please clearly specify in your submission what you reused and for what purpose.

Independent of what application you will choose to develop, this course project has a sequence of general well-defined steps. First, you come up with an idea of a domain for which you will develop your application. Suppose that you want to develop a comprehensive insurance risk assessment application. One function in this application is to compute an insurance risk quote for a user, and this quote consists of medical, rental, and auto insurance risk quotes. Each of these risk quote computations will be implemented in separate distributed objects. For example, one object may represent a 45 year old male Texan with incomes over $100,000 who is a waiter and lives in Dallas and rides a bike to work. A formula for computing a quote can be any of your choosing (you should avoid obvious flaws like division by zero), since it is not result that matters but the process. A result of this exercise should be a requirements document that may loosely confirm to IEEE/ANSI 830-1998 – you will find many samples of software requirements specification (SRS) documents on the Internet. Please check with me about the chosen format of your SRS.

The second step is you define data that your application uses and relationships among these data elements. The output of this step includes different database schemata that describe this data and relationships. When designing your databases, you should

[2] http://jmeter.apache.org/

figure out how your applications use this data, that is you will revisit step 1, possibly multiple times. There is no need to keep data in your database if your application has no use for this data – if you decide to put some data attributes in your schemata, you must have some rules in your application that use this data. In addition, you should decide what data you put in a relational database and what data goes to a NoSQL database. For NoSQL databases, I suggest that you consider Neo4J[3], MongoDB[4], and Riak[5], although I am open to other open source NoSQL databases.

Of course, the majority of enterprise-level applications use relational databases rather than their NoSQL counterparts. A primary way for these applications to communicate with relational databases is to use call-level interfaces, which allow applications to access database engines through standardized application programming interfaces (APIs), e.g., Java DataBase Connectivity (JDBC). There are many examples and tutorials on programming with JDBC[6], and you should be able to use them to learn this API quickly. As I said, all attributes in databases should be accessed and manipulated by the application as part of some functional requirements.

Once you design your databases, the next step is to populate them with data, which may come from a variety of sources including but not limited to the Web and different public repositories mentioned above in addition to the UCI KDD Archive (http://datamob.org/datasets/show/uci-kdd-archive). You may also consider populating databases with randomly generated data. An average relational database size should be around 300MB, it should contain around 20 tables that contain on average five attributes. These databases should contain primary and foreign keys. You should be cognizant of the fact that you will submit their projects via the Internet, and they should not create databases that are excessively large – databases over 2GB are large for this project.

Not all data comes from your local databases; some data will come from publically available data that are accessible using standard Application Programming Interfaces (APIs), such as Census API[7], Freebase API[8], Facebook API[9], Twitter REST

[3] http://neo4j.org/
[4] http://www.mongodb.org/
[5] http://wiki.basho.com/Riak.html
[6] http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm
[7] http://www.census.gov/developers/data/
[8] http://wiki.freebase.com/wiki/Main_Page

API[10], and Netflix[11]. You should set up your access to the public data of your choosing and get keys to access this data. Make sure that these keys work by testing them using your web browsers to obtain data from these public repositories.

The next step involves learning middleware that you will need to create your application – it includes but not limited to JDBC or Hibernate, and different APIs for publically available data sources. Different tutorials are available on the Internet and you should utilize them.

Finally, you should put together a simple skeleton of your application with only a dozen of classes that accesses and manipulate data from different sources. Once you can run this skeleton application, you can continue to build the rest of the application on top of this skeleton.

Your goal is not just to build an application, but also to investigate how different technologies are used together to interoperate. You should run your system with different clients that retrieve data and insert/update data in the database, and interleavings of requests from these clients would eventually lead to different deadlocks and inconsistencies. Your job is to observe and document few deadlock situations and determine how you could avoid them.

Each project submission should include at least one back-end relational and one NoSQL databases, together with their logical and physical design. That is, the database schema and SQL statements that realize it should be submitted along with instructions on how to create and populate the database. You should use open-source relational database engines, for example, MySQL or Apache Derby. Your application should retrieve data from and insert and update data in your databases using Structured Query Language (SQL) statements. For NoSQL databases, you should use appropriate API calls that are provided by a given NoSQL database engine of your choice.

Once requirements are defined, team members should proceed with creating traceability matrices as Excel spreadsheets. First three columns (i.e., A, B, and C) contain information about requirements and functionalities from the document that describes a given application. The first column contains the title/name of a requirement, the second

[9] https://developers.facebook.com/docs/reference/api/
[10] https://dev.twitter.com/docs/api
[11] http://developer.netflix.com/

column contains the number of the paragraph in the document where this requirement is located, and the third column contains the requirements location in this paragraph -- a path to a document plus the offset to the requirement in bytes. This is a part of the project where team members can be creative in terms of defining the location.

The next columns specify elements of design artifacts (e.g., an actor from use case), elements of the source code (e.g., classes, methods), and test cases by their names that uniquely identify them, that is every column corresponds to a design artifact. Cells contain numbers zero and one. Zero corresponds to the situation where an artifact is not related to a requirement and one specifies an existing relation, for example, that running a test case executes application's code which is linked to some requirement. Please note that each test case should have one or more ``1" in some of its cells as well as each requirement. That is, each test case should be related to at least one requirement and each requirement should be tested by at least one test case.

You will be evaluated based on comprehensiveness of your own design, the use of middleware features, comprehensiveness of your tests, the number of rules and their sophistication in your application (an application should contain at least 100 rules) and your documentation. The course project submission date is Sunday, December 8, 2013 at 9:00PM via dropbox with confirmation email to me drmark@uic.edu, CC to the TA. You shall receive a confirmation receipt from me. Failure to submit your project by the deadline shall result in losing your project grade.

For an additional 5% bonus, please install and configure an open-source software-defined network, deploy your application on top of this network, and experiment with provisioning of resources using this network.

Below is a table that contains deadlines for various project milestones. Noncompliance with these deadlines will result in losing 0.5% of the total grade for each day of submission delay up to the total grade of the course project, i.e., 25%.

Milestone	Deliverable	Deadline	Grade
	Intermediate Milestones		
Group formation	Email with your group project mates, one email from group, CCed to all group members and the TA	09/19/2012	0.5%

Requirements specification	A (WORD) document with a description of the functionality for your application	09/26/2012	1.5%
Design documents	UML diagrams and (WORD) documents that describe the architecture and design of your application and databases.	10/07/2012	2%
Physical databases populated with data	Schemata and statements that populate databases	10/18/2012	2%
Demo of the app skeleton	A “happy path” demo	11/05/2012	4%
	Final Delivery		
Documentation	Requirements, design, database design documents and schemata, and traceability matrices. Installation and configuration docs.	12/09/2012	10%
Tests	JUnit and JMeter tests	12/09/2012	5%
Application and databases	Source code, jar and zip files	12/09/2012	