

Coordination Management with two Types of Databases in a Web-based Cooperative System for Teamwork

Yun Yang
School of Comp & Maths
Deakin University
Geelong, VIC Australia 3217
yun@deakin.edu.au

Dengsheng Zhang
Department of Mathematics
Anqing Normal Institute
Anqing, Anhui P.R. China
dszhang@deakin.edu.au

Paul Wojcieszak
School of Comp & Maths
Deakin University
Geelong, VIC Australia 3217
wojciesz@deakin.edu.au

Abstract

Coordination is the fundamental issue in computer-mediated process-centered teamwork. This paper focuses on how to manage coordination in our Web-based cooperative environment implemented in Java. In particular, attention is drawn to the coordination information storage with using the relational and objected-oriented databases. Comparison of using two types of databases is addressed as the result of experiments which is in favour of the object-oriented database.

Keywords: cooperative systems, teamwork, Web, relational databases, object-oriented databases, Java

1. Introduction

Teamwork is a key feature in any workplace organization. In this computing era, a process/project is usually carried out by a cooperating team who may physically dispersed by using various (software) tools. Systems for computer-mediated teamwork [Galegher90], groupware [Ellis91], workflow [Sheth97] or computer-supported cooperative work (CSCW) [Rodden91] offer various automatic support for team cooperation to improve the productivity. Generally speaking, a process/project is normally composed of tasks which are partially ordered [Feiler93]. How to manage tasks is the key issue for completion of the entire project. Hence, task-oriented technology is management-centered to facilitate project management focusing on coordination. With software support, team members are coordinated automatically by a system which is clearly more effective than managed and controlled manually by a human being. Team members may, for example, reside in Asia-Pacific, Europe and North America. With the 8-hour time differences among locations, 24 hours a day working mode can be facilitated potentially [Gorton96].

Nowadays, there is a growing interest to support cooperative work over the Internet (or Intranet) and the Web. The emergence and wide-spread adoption of the Web offers a great deal of potential for the development of collaborative technologies as an enabling infrastructure [Oreizy97]. In addition, the Java programming language, which has the capabilities of delivering applets over the Web as well as the slogan of "write once and run anywhere" - platform independence, has encouraged us to prototype our work in Java based on the Web environment. In addition, no particular software needs to be installed for team members regarding teamwork coordination since Java applets can be downloaded on the fly and then run directly. Meanwhile, it has become popular recently with the component-based software solutions. Java matches the solutions quite smoothly. Furthermore, using combination of Web/Java seems better than using Web/CGI (common gateway interface) [Evans97] in terms of performance and control/data granularity. Therefore, we have treated the Web and Java as an excellent, if not ideal, vehicle to prototype our software process support mechanisms in a global distributed environment.

The primary objective of our entire project is to research into effective Web-based teamwork support. In this paper, we focus on using and comparing the relational and object-oriented databases as data repository for teamwork coordination. In the real world, we have found some general comparisons between the two types of databases. However, to the best of our knowledge, we have not seen any Web-based cooperative teamwork systems using and comparing these two most important types of databases.

This paper is organised as follows. First of all, we introduce the background of our environment in Section 2. We then focus on the data repository for teamwork coordination in both relational and object-

oriented databases in Sections 3 and 4 respectively, followed by a comparison in Section 5. Finally, we conclude our work and point out future directions.

2. Background of process-centred cooperative system support

The variation of the semi-centralised multi-tiered client-server architecture of Web-based teamwork support is depicted in Figure 1 [Yang98a]. It includes (1) clients as front-ends using local Web servers and tools, (2) centralised servers with tools, and (3) supporting tools such as databases and file systems as back-ends.

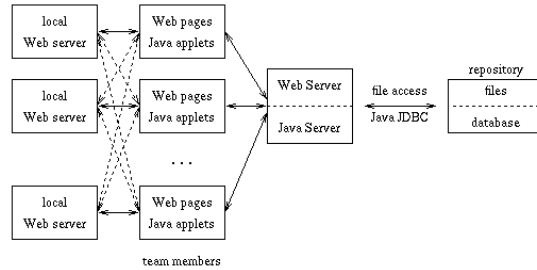


Figure 1. Architecture for supporting teamwork

In Figure 1, the centralised server site plays the key role for management of a task. The process information related to coordination is stored in the database repository. Please note that the database repository is a general concept which can include various databases such as relational and object-oriented databases. During enactment, information such as documents can be stored locally at the client sites or at the server site and accessed by team members based on the Web support which implies that information can be distributed rather than only centralised.

To illustrate what kinds of information need to be stored in the data repository, a simplified cooperative project for developing a course booklet is used to demonstrate the process of teamwork coordination. This course development process is only composed of five tasks: two writing tasks, two reviewing tasks and one integration task as shown in Figure 2. There are also two dummy tasks for start and finish.



Figure 2. User interface for process coordination

For the writing tasks, it relates to write up Part 1 (Ch1.v1) and Part 2 (Ch2.v1) by team members A and B respectively who may be at two different places using different tools such as Notepad (on PC) and vi (on Unix) for text editing. After that, for the review tasks, A and B exchange their drafts in order to review and then produce the second version of those two parts (Ch1.v2 and Ch2.v2). Tools can be specified in the project in order to be invoked automatically as indicated in the confirmation box in Figure

2. Finally, for the integration task, the two parts (Ch1.v2 and Ch2.v2) are integrated into a course material (Booklet) by both members using, say, the REDUCE tool [Yang98b] for synchronous cooperative editing. Therefore, it can be summarised that the following essential kinds of information for such as project ID, task ID, user ID, tool, input and output need to be stored. These are the key data items which need to be stored in the database for teamwork coordination.

3. Relational database repository

In our cooperative system, the Oracle relational database (RDBMS) is employed first as the repository for teamwork information. During the initialisation stage, the server of our system is connected to Oracle. The connection has two steps. Firstly, the server calls the Java class loader to load the Oracle JDBC driver class. Secondly, the Java driver manager uses the Oracle JDBC driver to establish a connection to the database by the given URL. After the connection, we can use the Java SQL package to access the database with the standard SQL query language.

In order to store task information into the database, a task has to be defined first. Tasks are the basic element in a project. A task of a project consists of attributes. Each task is uniquely identified by some of its attributes. Based on the previous section, some common attributes identified in a task include *project ID*, *task ID*, *user ID*, *tool*, *input* and *output*. In addition, from the example in Figure 2, it also implies that some tasks are *dependent* of other tasks such as that the integration task cannot start unless both the reviewing tasks have finished which is reflected by *status*. Furthermore, normally a project is driven by some deadlines which means that each task is normally associated with a *start date* and a *finish date* from the management point of view. A task may also have such attributes as *user name*, *user email address* and so forth to facilitate effective communication for coordination.

In order to carry out a project in an automatic fashion, the project should be able to flow forward. The *status* attribute in a task is specially designed to keep the project progressing. A task can have one of the three statuses: *unenacted*, *enacting*, *enacted* represented as 0, 1, 2 respectively. When a task has not been launched yet, its status is *unenacted*. The cooperative system itself changes a task's status from *unenacted* to *enacting* automatically when that task is ready to launch and at the same time notifies the appropriate team member(s) for coordination. The *enacting* status means that the task is currently active and ongoing whilst the *enacted* status indicates that the task has already finished. A task can be launched when all its dependent tasks have all been enacted, and in this manner, a project can keep flowing forward. At run time, the team member can only change his/her own task's status from *enacting* to *enacted* when the task is completed to notify the cooperative system to proceed.

Data repository in our system consists of tables and Java objects. Eight tables are designed to store persistent project information. The tables are defined below which play an important role in the coordination process. Any change or update during the execution of the project is recorded into the corresponding elementary tables in the database. The server is then requested to check each subsequent task information in tables to decide if any task should be launched and any input (documents) should be sent to the related members. All the necessary information are derived from those elementary tables such as that status and dependants are from tables TASKS and DEPENDANTS respectively. Similarly, information such as user email address and user ID are retrieved from table USERS.

Persistent Elementary Tables in RDBMS:

1 TASKS

Attributes: task_id project_id task_name start_date end_date status
 Data type : char char char date date number

2 TOOLS

Attributes: task_id tool_name
 Data type: char char

3 DEPENDANTS

Attributes: task_id dependant_id
 Data type: char char

4 INPUTS

Attributes: task_id input_doc_name
Data type: char char

6 PROJECTS

Attributes: project_id project_name
Data type: char char

8 USERS

Attributes: user_id user_name password user_email
Data type: char char char char

5 OUTPUTS

Attributes: task_id output_doc_name
Data type: char char

7 PEOPLE

Attributes: task_id user_name
Data type: char char

In effect, JDBC is a low level middleware tool that provides the basic features to interface a Java application with tables in a relational database. Hence, we decided to deploy a suitable object-oriented database so that we only need to handle objects in the system. Since the ObjectStore object-oriented database system has a Java interface which meets our needs, we chose ObjectStore's PSE Pro [ObjectDesign98] for experiment.

4. Object-oriented database repository

PSE (persistent storage engine) provides persistent storage for Java objects. With using ObjectStore's PSE Pro (ODBMS), persistent data is available to programmers in such a way that it appears as normal Java objects. Persistent Java objects and regular Java objects are manipulated and behave in the same way. Since ObjectStore's PSE Pro is written in Java, it runs entirely within the application process. Hence, no ODBC bridge/driver is needed for the connection to database like the JDBC driver for RDBMS. PSE Pro supports transactions and provides access to objects without reading the entire database. In this section, we focus on differences with using RDBMS and ODBMS first.

With JDBC/RDBMS, making an object persistent requires more development work. For example, we have to design a relational schema to which they will map Java objects. Then, to write a Java object to the database, we have to write code to map the Java object to the corresponding rows of the corresponding relations. The same process has to be done in the other direction to read a Java object from the database. The most difficult step is to map Java objects onto tables in RDBMS using available SQL data types. This process becomes quite difficult if a class includes inheritance and complex Java objects.

Meanwhile, retrieval of same information with JDBC/RDBMS can be more time consuming than that with ODBMS. For instance, with RDBMS, we use table PEOPLE to store team member information which contains just task ID and user name for every member. Other information such as task name, start date, status etc. are stored in table TASKS which stores task information for each task. In this case, two steps are needed to retrieve task information of a team member. The first step is to find the member in PEOPLE and the second step is to retrieve the task information for this team member from TASKS. It is possible to have just one-step retrieval to derive all information. However, in this case, tables have to be designed to contain redundant data which is at the cost of difficult and time consuming maintenance. With ODBMS, these problems can easily be solved. Object PEOPLE is designed to inherit from TASKS, so each team member in PEOPLE inherits the whole information for each task in TASKS, when we want to derive the task information of a team member in PEOPLE, only one step is needed, that is, to just find the team member in PEOPLE, and there is no redundant data in the database.

With ODBMS, it is clear that those eight tables in RDBMS in the preceding section are now replaced by corresponding Java objects which are persistent-capable. The attribute types of objects are changed when necessary. The behaviors for these objects are common `gets` and `sets` methods. These objects are put into ObjectStore collections which are stored persistently in the database. Objects here play similar roles in the coordination to tables in relational database.

Persistent Elementary Objects in ODBMS:

1. TASKS

Attributes: task_id project_id task_name start_date end_date status
Data type: String String String Date Date int

2. TOOLS

Attributes: task_id tool_name
Data type: String Vector

3. DEPENDANTS

Attributes: task_id dependant_id
Data type: String Vector

4. INPUTS

Attributes: task_id input_doc_name
Data type: String Vector

5. OUTPUTS

Attributes: task_id output_doc_name
Data type: String Vector

6. PROJECTS

Attributes: project_id project_name
Data type: String String

7. PEOPLE

Attributes: task_id user_name
Data type: String Vector

8. USERS

Attributes: user_id user_name password user_email
Data type: String String String String

With the ObjectStore object-oriented database, there is no need for JDBC or SQL in the code. The code is more integrated and neat. Using the assembling of runtime task list for the initialisation of user's interface as an example, the assembling process needs to retrieve tools information from the database. In the JDBC/RDBMS case, it's derived from table TOOLS. Since tools are stored in a Java Vector at run time, a separate method is needed here to first assemble the tools in the table TOOLS into a Java Vector. While in the ObjectStore/ODBMS case, tools information is derived from object Tools which is directly stored in the database. Data in the object is directly sent which means that no assembling to the Java object is needed. The same difference of this kind of operation also happens in update, refresh and data manipulation. In fact, the code of database interface is reduced more than half after changing from JDBC/RDBMS to ObjectStore/ODBMS.

5. Comparison of implementation with using two types of databases

By experimenting with both the Oracle relational database (RDBMS) and ObjectStore object-oriented database (ODBMS), we have reached the following comparison:

(i) Database connection

- RDBMS: two steps — loading JDBC driver and connecting to database using Java driver manager
- ODBMS: simple database opening

(ii) Data storage

- RDBMS: Oracle tables, extra query or data redundancy
- ODBMS: ObjectStore collections (objects), using inheritance, no extra query and data redundancy

(iii) Data accessing

- RDBMS: bi-directional mapping between (Java) objects and tables in the database
- ODBMS: objects to objects, no mapping

(iv) Query

- RDBMS: SQL through JDBC bridge
- ODBMS: Java collection navigation

(v) Data operation

- RDBMS: copying the table for each query
- ODBMS: creating a hollow reference to an object in the database, no copying of the object

(vi) Code

- RDBMS: extra code for mapping and assembling data in tables into objects and vice versa
- ODBMS: pure Java, no SQL, lines of code reduced dramatically

In summary, experiment results are in favour of deploying an object-oriented database in our Web-based cooperative system to support process-centred teamwork. With a Java interface, such as that in ObjectStore, we only need to handle objects directly which not only can improve the productivity but also increase the efficiency. In addition, use of an object-oriented database matches our needs better since Java itself is object-oriented. In fact, it is more natural to carry out teamwork in the object-oriented manner which is another important reason that why we are in favour of using an object-oriented database as data repository.

6. Conclusions and future work

In this paper we have mainly focused investigation of data repository for supporting automatic coordination for Web-based teamwork. In particular, two types of different databases, namely (Oracle) relational database and (ObjectStore) object-oriented database, have been experimented for the persistent data repository purpose for teamwork coordination. The general conclusion drawn is in favour of the object-oriented database. Since teamwork is more naturally represented in an object-oriented manner, with using an object-oriented database, there is no mapping needed which is required with using a relational database. In addition, with using an object-oriented database, data storage, accessing, query, operation, coding and so forth can be dramatically simplified.

In the future, we need to further experiment with the persistent data repository with object-oriented databases. We also need to work on more general issues like visual programming support for Web-based teamwork modelling and enactment, with the reactive system paradigm as the underlying technology. In addition, many other things can be further investigated such as better process evolution, mobility, interoperability and tool integration.

Acknowledgement

Work reported in the paper has been supported partially by a seeding grant from School of Computing and Mathematics, Deakin University in 1998 and an ARC (Australian Research Council) grant in 1999. We are grateful for some implementation support from Damien Brain.

Reference

- [Ellis91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: some issues and experiences. *CACM*, 34(1):39-58, 1991.
- [Evans97] E. Evans and D. Rogers. Using Java applets and CORBA for multi-user distributed applications. *IEEE Internet Computing*, 1(3):43-55, 1997.
- [Feiler93] P. H. Feiler and W. S. Humphrey. Software development and enactment: concepts and definitions. In *Proc. of 2nd Int. Conf. on Software Process*, pages 28-40, Berlin, Feb. 1993.
- [Galegher90] J. Galegher, R. E. Kraut, and C. Egido, editors. *Intellectual Teamwork*. Lawrence Erlbaum Associates, Publishers, 1990.
- [Gorton96] I. Gorton and S. S. Motwani. Issues in cooperative software engineering using globally distributed teams. *Information and software technology Journal*, 38(10): 647-655, 1996.
- [ObjectDesign98] ObjectStore PSE/PSE Pro Release 3.0 for Java. <http://www.odi.com>
- [Oreizy97] P. Oreizy and G. Kaiser, The Web as enabling technology for software development and distribution, *IEEE Internet Computing*, 1(6):84—87, 1997.
- [Rodden91] T. Rodden. A survey of CSCW systems. *Interacting with computers*, 3(3):319-353, 1991
- [Sheth97] A. Sheth. Workflow and process automation in information systems: state-of-the-art and future directions. *ACM SIGGROUP Bulletin*, 18(1):23-24, 1997.
- [Yang98a] Y. Yang. Issues on supporting distributed software processes. *Software Process Technology, Lecture Notes in Computer Science*, Vol. 1487, pages 243-247, 1998.
- [Yang98b] Y. Yang, C. Sun, Y. Zhang and X. Jia, A Web-based real-time cooperative editor in Java. *Proc. of WebNet98 (World Conf. Of the Web, Internet and Intranet)*, pages 975-980, Orlando, USA, Nov. 1998.