Design and implementation of enterprise systems in fine-grained concurrent computation

Kenji Ohmori *

Hosei University, Faculty of Computer and Information Sciences
3-7-5 Kajino-cho, Koganei-shi, Tokyo, Japan

Abstract

The computational power required to an enterprise system changes dynamically depending on services and the number of users. Massive concurrent computing is excellent in scalability, so that the paper proposes utilizing it for designing and implementing enterprise systems. The proposed accounting system is developed using Agile software development method to study the scalability of massive concurrent systems. When combining two subsystems in Agile, an attaching function is introduced. The system is designed using Model-View-Control software architecture and communicating sequential processes and then implemented with JCSP. Before implementation, the validity of the model is verified using verification tools. It reduces development cost and time.

© 2012 Published by Elsevier Ltd. Selection and/or peer review under responsibility of CENTERIS/SCIKA - Association for Promotion and Dissemination of Scientific Knowledge Open access under CC BY-NC-ND license.

Keywords: Communicating Sequential Processes; Model-View-Control Software Architecture; Agile Software Development.

E-mail address: ohmori@hosei.ac.jp.
1. Introduction

Massive concurrent computing is the next technology for solving scalability, where required computational power changes dynamically, which is observed in Web database systems including enterprise systems. A Web database system consists of Web component, application and database servers where each server is conventionally composed by single or multiple processors. An application program for the Web database system is usually designed by the software architecture of Model-View-Controller (MVC). The model part manages the behavior and data of the services supplied by the application program. The view part renders the model part into a form suitable for interaction. The controller part receives input and initiates a response by making calls on model objects. The model, view and controller parts are installed in database, Web component and application servers, respectively.

Application programs including enterprise systems that are designed for a Web database system are conventionally implemented by coarse-grained concurrency. On the other hand, fine-grained concurrency makes the best use of scalability, so that an experimental system of an enterprise system is developed for studying technologies required for accommodating enterprise systems in a massive concurrent computing environment such as a virtual machine in Cloud computing. When using MVC software architecture, the model, view and controller parts have to be designed and implemented by fine-grained concurrency by [1]. In addition to that, these parts have to be installed in a parallel computing environment that a virtual machine technology of Cloud computing can realize.

This paper describes design and implementation of an accounting system with fine-grained concurrency. The accounting system is designed by an agile software development method where the most important functions of the accounting system are firstly designed, verified, implemented and tested by [2]. Each function is designed using the concept of communicating sequential processes (CSP) by Hoare [3]. CSP is a model for concurrent processing and equipped by verification tools such as FDR2 and PAT3 by Roscoe [4]. JCSP [5] is a Java-like language with enhancement of CSP concepts, is used to implement the model. Functional programming language Clojure [6] is an alternative to CSP, but the lack of verification tools is its serious disadvantage.

After the most important functions have been successfully implemented, the second ones are added to the first ones. This process is repeated until the whole accounting system is completed. When adding functions, new mathematical concepts called an attaching function is introduced. Joining of two entities is an important operation for constructing a reliable and secure system. An attaching function combines two entities holding an equivalence relation [7]. If attaching spaces of two programs combined together are not identical, the combined program causes serious faults, which is avoided using an attaching function.

Conventionally, enterprise systems are designed based on Java 2 Enterprise Edition (J2EE) and Enterprise Java Beans (EJB) technologies by Johnson [8]. For avoiding deadlocks in J2EE and EJB, the development of an application program that guarantees reliability and security to transaction processing requires a huge amount of time and cost since no verification tools are provided to detect livelocks and deadlocks. On the other hand, when designing a system using CSP, Failures-Divergences Refinement 2 (FDR2) or Process Analysis Toolkit (PAT) can detect these hazards. As verification is carried out before implementation, it reduces development time and cost.

2. Fine-grained concurrency

To realize a large scale fine-grained concurrency, the number of processes has to be maximized. When MVC software architecture that has been used in the conventional enterprise system is adapted in the large scale fine-grained concurrency, it is important to realize each part of MVC with huge number of processes. In
this paper, a process stands for a process of CSP. Processes of CSP are executed each other concurrently. Communications among processes are carried out by message sending and receiving through a channel.

2.1. The granularity of model part

MVC consists of three parts. Among them, the model part handles persistent data. In the conventional system, persistent data are processed in the database system where the unit of an access is defined by the database system itself. Usually, a table or multiple tables are an access unit. In this system, all persistent data are kept in the main memory. The consistency between the database system and the main memory is kept by another system, so that the database system is not considered in this paper.

Each persistent data is processed by a single process. When it is implemented in an object oriented method, one class is provided for each persistent data and only one instance is created for each class and handles accesses to the persistent data, so that deadlocks are easily avoided.

In the accounting system, account items are typical persistent data. In this system, an account item is represented as a process. Cash, deposit, receivable and payable are examples of account items. Some account item has subordinate accounts. For example, a deposit account consists of several types. Each type is also represented as a process. For example, a checking account is a process. As a big company uses several banks, the account item of the checking account has subordinate account items corresponding to each bank.

As CSP is a model for describing concurrent processing, a programming language has to be selected when implementing its model as a software system. Communicating Sequential Processes for Java (JCSP) is used in this paper. JCSP provides a process template as an interface of Java, which is called CSProcess. A Java class for realizing a process is realized by implementing the interface. Each process of CSP is realized by an instance of its class. Therefore, each class has one or multiple instances when implementing the model represented by CSP. The relation of deposit account, checking account and bank is depicted in Figure 1, which construct a class hierarchy by inheriting the properties of the upper classes to a lower class.

The process of a deposit account is created in a JCSP program by the following procedure. By implementing CSProcess, a class Account is created as a general account item. As a subclass of Account, a class Deposit is provided, which has common properties of deposit and withdrawal operations. A variable amount is provided as a class variable to show the current remaining amount in Deposit. To restrict the access to amount, only one instance is provided for the class, so that only this instance can read, write and update amount, which avoids conflicts of the resource amount among processes.
As a checking account inherits a deposit account, a class Checking is provided as a subclass of Deposit. New properties such as no interest rate are added to this class. Only one instance is also provided to Checking. In the same way, a class and instance are also provided to bank A.

A specific process example of CSP is now described using a cash account item. Increase or decrease of the amount of money in the cash account item is received by a channel cash?. The process of cash is represented here by \( CASH(y) \). \( y \) is the current amount of money. When \( x \), which shows the amount of increase or decrease, is received by the channel, the process expressing this behavior is described as follows,

\[
CASH(y) = cash? x \rightarrow CASH(y + x).
\]

The program corresponding to the process is depicted as follows. The first class that has to be defined is a class Account, which implements the interface CSProcess and an input channel configure is provided and configured.

```java
public class Account implements CSProcess {
    ChannelInput configure;
    public Account(ChannelInput configure)
        this.configure = configure;
    public void run() {}
}
```

The next class is Cash, which is defined as a subclass of Account. When receiving inc that is the amount of increase or decrease, the class variable accumulated is accumulated by inc.

```java
class Cash extends Account {
    static Integer accumulated = new Integer(0);
    public void run() {
        while (true) {
            Object inc = configure.read();
            if (inc == null) break;
            else if (inc instanceof Integer)
                accumulated = new Integer(accumulated.intValue() + ((Integer) inc).intValue());
            else System.error.println("Cash Operation Error");
        }
    }
}
```

### 2.2. The granularity of view part

The view part of MVC provides graphical user interfaces (GUI) of homepages. The granularity of the view part mainly depends on JCSP functions. The current version of JCSP supports Abstract Window Tools (AWT) in such a way that widgets such as buttons, scroll bars and pop-up menus are provided as independent concurrent processes so that JCSP gives fine-grained granularity to the view part.

A homepage of an accounting system is provided by a Java applet implemented by AWT. Java Swing that becomes available in JCSP in future gives more elegant and sophisticated GUI than the current system. However, there are no differences from the view point of concurrency. Therefore, AWT gives enough functions to examine fine-grained concurrency regardless of poor GUI.
2.3. The granularity of control part

The control part gets data from the view part, processes the data and sends the processed data to the model part. Reversely, it gets data from the model part, processes them and rendered them to the view part. Therefore, at least one process is given to each user at the control part.

The accounting system basically provides the view part for transaction services including sales and purchasing. The control part gets transaction data, sorts it to credit and debt accounts and generates a journal. According to the journal, the control part updates the corresponding credit and debt account items.

Suppose that a company is selling goods by cash. The journal for this transaction is provided with a cash account item with sales amount as the debt and a sales account item with the same amount as the credit. After asking for recording the journal, the control part requires the model part to update these accounts by increasing or decreasing the accumulated amount of these account items by the sales amount. A process $SELLING$ is describes as follows,

$$SELLING = submit \rightarrow content? y \rightarrow journal! y \rightarrow cash! x_3 \rightarrow sales! x_1 \rightarrow SELLING,$$

where $y \in \{x_1, x_2, x_3, x_4\}, x_1 \in date, x_2 \in goods, x_3 \in prise, x_4 \in customer.$ \hspace{1cm} (2)

3. Process integration

It is unlikely that the enterprise system equipped in a company will not change. The accounting system is exposed by the revision of tax laws. The concept of accounting system itself changes. The enterprise system is required to adapt to various changes such as modification, addition and elimination of functions or services. As modification can be carried out by elimination and addition, the system is required to be adaptable of addition and elimination. An attaching function gives mathematical foundation when designing and implementing addition and elimination.

3.1. Attaching functions

Let us start with a topological space $X$ and attach another topological space $Y$ to it. Then, $Y_f = Y \oplus f X = Y \oplus X / \sim$ is an attaching space obtained by attaching $Y$ to $X$ by an attaching map $f$ (or by identifying each point $y \in Y_0 | Y_0 \subseteq Y$ with its image $f(y) \in X$ by a continuous map $f$). $\oplus$ denotes a disjoint union, which is usually represented by a square cup symbol.

An attaching map $f$ is a continuous map such that $f: Y_0 \rightarrow X$, where $Y_0 \subseteq Y$. The attaching space $Y_f = Y \oplus X / \sim$ is a case of quotient spaces $Y \oplus X / \sim = Y \oplus_j X = Y \oplus X / \{(x \sim f(y)) | \forall y \in Y_0\}$.

A matchmaking party gives a good example to explain how separate groups are merged together using an attaching map. There are a group of girls, which is represented by a set $G = \{Betty, Michel, Alice, Chelsea, Jackie\}$, and another group of boys described by a set $B = \{Tom, Mike, Peter, Jack, Alex\}$. A member of the girl group has never met a member of the boy group. Suppose that these groups have a matching party. The situation before the meeting is described by the disjoint union of two groups. The situation after the meeting is described using an attaching map, which describes new partners that have been made at the party.

The disjoint union of these sets is represented by

$$G \oplus B = \{Betty, Michel, Alice, Chelsea, Jackie, Tom, Mike, Peter, Jack, Alex\}.$$

(3)
At the meeting, Betty and Mike like each other and become a partner. Also, Chelsea and Peter do another partner. If a partner is considered to be identical by an attaching map \( f \), then,

\[
G \oplus_f B = \{(B,M) = ((Betty,f),(Mike,m)),(Michel,f),(Alice,f),(C,P) = ((Chelsea,f),
(Peter,m)),(Jackie,f),(Tom,m),(Jack,m),(Alex,m)\},
\]

where \( f : G_0 \rightarrow B,G_0 = \{(Betty,f),(Chelsea,f)\} \).

The party caused the change of relation between girl and boy groups. The change is depicted using an identification function \( g \) as follows,

\[
g : G \oplus B \rightarrow G \oplus_f B
\]  

As \( g \) can be defined as a continuous function, \( g^{-1} \) gives the two original disjoint spaces \( G \) and \( B \) by disjoining the attaching space \( G \oplus_f B \).

3.2. A consolidated accounting system

The consolidated accounting system gives the financial situation of a group of companies tightly related by financial alliance. In general, an accounting system depends on a business type. If different business companies are combined into a consolidated accounting system, the same or similar account items of both companies should be merged into one account item. Also, the internal transactions that have been carried out between combined companies should be merged and then eliminated.

Suppose that a product company \( Y \) is a subsidiary of a trade company \( X \). Both companies have individual accounting systems and give sales and purchasing services. The account items related to these services are depicted in Fig. 2. \( X \) has suppliers \( Y \) and \( Z \) and customers \( A,B \) and \( C \). These suppliers and customers constitute subordinate account items of \( Purchase_X \) and \( Sales_X \), respectively. A set of subordinate account items of \( Purchase_X \) is \( \{SupY_X , SupZ_X \} \).

\( Y \) has also suppliers and customers. The sets of account items of these companies are represented as follows.

\[
X = \{Purchase_X = \{SupY_X , SupZ_X \}, Sales_X = \{CustA_X , CustB_X , CustC_X \} \},
Y = \{Purchase_Y = \{SupW_Y \}, Sales_Y = \{CustX_Y , CustA_Y , CustD_Y \} \}.
\]  

Therefore, the set of the accounting systems of two companies is represented by \( X \oplus Y \) as the disjoint union of \( X \) and \( Y \).

By introducing topology into the set such as the discrete topology and using an attaching function that attaches the same or similar account items of both companies together, the following set is obtained.
Suppose that $X$ purchases products from $Y$. The accounting system of $X$ records it in the purchase account item from $Y$ and one of $Y$ does it in the sales account item from $X$. As the products produced by $Y$ becomes only goods being sold by $X$, $SupY_X$ is equivalent to $CustX_Y$ in the consolidated accounting system.

$X \oplus_f Y = \{Purchase = \{null = \{SupY_X, CustX_Y\}, SupZ_X, SupW_Y\}, Sales = \{CustA = \{CustA_X, CustA_Y\}, CustB_X, CustC_Y, CustD_Y\}\}$. \hspace{1cm} (7)

$SupY_X$ is a debt account item and $CustX_Y$ a credit one. Both account items have the same amount of money. If one of them is selected as the representative, another account is moved to the other side (in the above expression, $CustX_Y$ is moved from the credit side to the debt) where the amount of money is negated, so that the amount of money of the representative item becomes zero. Therefore, both accounting items can be eliminated. In $X \oplus_f Y$, the representative is expressed as $null$. (In the same way, $PayableY_X$ meaning that $X$ has to pay the product cost to $Y$ and $ReceivableX_Y$ meaning that $Y$ can receive the product cost from $X$ are also equivalent and eliminated.)

$X$ and $Y$ have $A$ as a customer, which is represented as $CustA_X$ and $CustA_Y$ in the accounting systems, respectively. By $f$, both of them are attached together and is expressed by the representative $CustA$ in the consolidated accounting system.

The selling process of $Y$ in CSP is obtained by the following procedure. Selling date, product name, selling amount and customer name are entered at a homepage of the sales service when products have been sold. When pushing the enter bottom at the homepage, a celling process $SELLING$ is executed.
An identification map \( g : X \oplus Y \rightarrow X \oplus_f Y \) gives the transformation from the set of individual accounting systems to the consolidated accounting system. In this transformation, the attaching map \( f \) plays an important role.

A process example in the consolidated accounting system is now described. By receiving selling information from the homepage, the selling process sends selling information to the account item provided for each customer, which is different from the process in Section 2.1 where the sales account item is not divided into individual customers. \( SELLING \) as a CSP process is described as follows.

\[
SELLING = \text{submit} \rightarrow \text{content}?y \rightarrow \text{journal!y} \\
\rightarrow ((\text{receivable}_x \! \cdot \! x_3 \rightarrow \text{sales}_x \! \cdot \! x_3) \triangleleft x_4 = \text{CompanyX} \triangleright) \\
\rightarrow (\text{receivable}_i \! \cdot \! x_3 \rightarrow \text{sales}_i \! \cdot \! x_3 \rightarrow \text{receivable}_c \! \cdot \! x_3 \rightarrow \text{sales}_c \! \cdot \! x_3) \triangleleft x_4 = \text{Companyi} \triangleright) \\
\rightarrow SELLING, \\
\text{where } i \in \{A, D\}. \tag{8}
\]

In the above expression, the statement of the 2\textsuperscript{nd} line is related to an internal transaction between \( Y \) and \( X \). Therefore, the transaction is recorded not in the consolidated accounting system, but only in the accounting system of \( Y \). The 3\textsuperscript{rd} line statement is for an external transaction between \( Y \) and \( A \) or \( D \). The transition is recorded not only in the accounting system of \( Y \), but also in the accounting system of \( Y \). In each case, the debt account is \( \text{receivable} \) and the credit is \( \text{sales} \).

4. The implementation of an accounting system

Using agile software development, the accounting system has been constructed from a simple system to the final system. The simple system provides only one service for sales, which has one homepage as the view part, one process \( SELLING \) as the control part and two processes \( \text{CASH}(y) \) and \( \text{SALES}(y) \) as the model part. \( SELLING \) and \( \text{CASH}(y) \) are described by expressions 2 and 1, respectively.
Then, another important service of purchasing is independently provided with a homepage for purchasing, a process \textit{PURCHASING} for the control part and a process \textit{PURCHASE}(y) for the model part. Then, the two services are combined using an attaching function. When combining the two services, the channel \textit{CASH} of \textit{SELLING} is identical to the one of \textit{PURCHASING}, so that the two channels are attached together by the attaching function. The channel is implemented by JCSP as shown in Fig 3. The channel has multiple inputs and single output where an amount of money is received from the \textit{SELLING} instance or the \textit{PURCHASING} instance and sent to the \textit{CASH} instance.

Based on the bottom-up method of agile software development, two accounting systems dedicated for trade and product companies are independently provided by adapting the sales and purchasing services to more realistic ones by replacing \textit{CASH}(y) of \textit{SELLING} and \textit{PURCHASING} with \textit{RECEIVABLE}(y) and \textit{PAYABLE}(y), respectively and adding more services. Then, the two accounting system is combined so as to be part of the consolidated accounting system as shown in the previous chapter.

5. Evaluations

The number of processes of each homepage is as many as the number of widgets used in the homepage. As multiple users utilize a service, total number of threads of the view part is \((\text{the average number of widgets per service} \times \text{the number of services} \times \text{the average number of users per service})\). If about 1000 users for each service are concurrently using the system and several tens of services are provided with about 10 widgets, the number of threads becomes \(10^4-10^5\).

The control part receives a request from the view part. For each service, one or multiple processes are provided. As the number of threads required in the control depends on the number of active uses, it becomes \(\text{(the number of users} \times \text{the number of processes per service})\).

The model part manages the persistent data of the accounting system. Generally, the number of account items is several thousands, which is the number of threads in the model part.

Consequently, the required number of the accounting system depends on the number of users when the system is used by a large number of people. As the system can increase the number of threads flexibly according to the number of users, the system works effectively when it is installed in a massive concurrent system.

6. Conclusion

A new mathematical model using an attaching function is very useful in agile software development where a new developing subsystem is added to the developed subsystem so that the validity of combining two subsystems is guaranteed by the equivalence relation of an attaching function, which reduces development cost and time.

CSP is useful in designing concurrency. Serious hazards in concurrency are avoided using channels since resource conflicts are controlled by channels. Furthermore, validation tools for CSP are effective to detect hazards, which allow guaranteeing valid design before implementation.

JCSP is a very experimental tool for CSP. It has enough capability to verify or demonstrate an expected system. In particular, as JCSP provides a network environment, it is useful to test an experimental system in a network environment where many personal computers are connected. However, JCSP needs to be enhanced when using it for developing a practical system.
References