Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. G. Alonso *C. Hagen, B. Kemme*

AR1: Transaktionsverwaltung SS 1997

Praktische Übung 5 Besprechung: 3.7.1997

Praktische Arbeit mit einem TP-Monitor

In dieser praktischen Übung werden Sie mit Encina, einem TP-Monitor-System, arbeiten. Wir haben dazu eine Beispielumgebung mit mehreren *Application Servern* vorbereitet. Ihre Aufgabe ist es, einen Client zu programmieren, der die von den Servern angebotene Funktionalität nutzt und mit Hilfe des TP-Monitors verteilte Transaktionen realisiert.

1 Organisatorisches

Sie haben in der Übung einen UNIX-Account bekommen. Loggen sie sich mit diesem Account auf dblab1 oder dblab2 ein. Initialisieren Sie dann DCE und Encina mit "module add dce encina". Ausserdem müssen Sie noch den Compiler mit "module add lang" initialisieren. Nun müssen Sie sich gegenüber DCE authentisieren. Geben Sie "dce_login" ein, und nehmen Sie als login "encina_ar" und als Passwort "encAr". Alle Programmfiles befinden sich in "/home/arv0/encina". Kopieren Sie diese Files in Ihr Verzeichnis.

Dokumentation zu Encina und seinen Komponenten findet sich im Rechnerraum D35. Hilfreich sind vor allem "Writing Encina Applications" (für den Überblick) und "Transactional C Programmer's Guide and Reference". Bei Fragen und Problemen können Sie natürlich auch jederzeit die AssistentInnen bemühen.

2 Das Szenario

Zur Verwaltung eines Weinkellers werden drei Datenbanken verwendet, die jeweils durch einen eigenen Application Server gekapselt werden. Die Produkt-Datenbank enthält Informationen über die verfügbaren Weine, Jahrgänge, etc. Sie wird durch den "product" - Server gekapselt. Die Bestands-Datenbank enthält Informationen über die momentan im Keller vorhandenen Bestände. Sie wird durch den "cellar" - Server gekapselt. Die Bestellungs-Datenbank schließlich speichert Bestellungen, die aufgegeben wurden. Sie wird durch den "order"-Server gekapselt. Die genaue Funktionalität der Server wird später in dieser Anleitung erklärt.

Das Client-Programm erlaubt es Anwendern, mit den Datenbanken zu kommunizieren. Es soll eine Reihe von Operationen zur Verwaltung des Weinkellers anbieten. Diese Operationen greifen in der Regel auf mehrere Datenbanken zu.

3 Die Konfiguration

Entsprechend dem vorgestellten Szenario besteht unsere Anwendung aus drei Servern. Die Server heissen "product", "cellar" und "order". Die Funktionalität, die sie anbieten, ist jeweils durch ein Interface-Definitions-File (Endung .tidl) definiert. Um festzustellen, welche Operationen der product-Server anbietet, schauen Sie sich z.B. das file "product.tidl" an. Die einzelnen Server exportieren die folgenden Services:

product:

Die Produkt-Datenbank speichert zu jedem Wein eine eindeutige ID, seinen Namen, Jahrgang, und den Winzer bzw. die Kellerei. Der Server hat die folgenden Services:

- lookup (name,year,key) : Erwartet einen Namen und einen Jahrgang und liefert die eindeutige ID zurück. Wenn kein Wein mit dem angegebenen Namen und Jahrgang existiert, wird die Transaktion abgebrochen.
- plist (entries) : Liefert den Inhalt der Datenbank (als String-Array) zurück.

cellar:

Diese Datenbank verwaltet den Weinbestand im Weinkeller. Zu jeder ID wird die Anzahl an Flaschen gespeichert. Diese Anzahl darf nicht kleiner als 0 sein. Ein Constraint sorgt dafür, dass andernfalls die ändernde Transaktion abgebrochen wird. Die folgenden Services werden exportiert:

- consume (id, count) : Erwartet eine ID und die Flaschenanzahl und zieht sie vom gespeicherten Bestand ab. Würde der bestand kleiner als 0, so wird die Transaktion abgebrochen.
- clist (entry) : Liefert den Inhalt der Datenbank (als String-Array) zurück.

order:

Hier werden Bestellungen gespeichert. Zu jeder Bestellung wird ihr Zustand (ordered / delivered) und Bestell- sowie Lieferungsdatum gespeichert. Die folgenden Services werden vom order-Server exportiert:

- newOrder (id, count, orderDate): Speichert eine neue Bestellung. Parameter sind die eindeutige ID des Weins, die Anzahl an Flaschen, und das Bestelldatum.
- deliver (id, deliverDate, count) : Ändert den Zustand der Bestellung auf "delivered" und liefert (im Parameter "count") die Anzahl bestellter Flaschen zurück.
- olist (entries) : Liefert den Inhalt der Datenbank (als String-Array) zurück. zurück.

4 Die Aufgaben

Aufgabe 1: Erweiterung des Clients

Machen Sie sich mit dem Source-Code vertraut. Sie müssen lediglich mit dem "client.c" - File arbeiten, die anderen Dateien können aber für das Verständnis der Anwendung hilfreich sein. Von Interesse sind insbesondere die Interface-Beschreibungen (z.B. cellar.tidl), die daraus automatisch generierten Header-Files (z.B. cellar.h), und die Dateien für die Server. Jeder Application Server besteht aus zwei Dateien. Fuer den cellar-Server z.B. existiert die Datei cserver.c, die die Initialisierungsroutinen für den TP-Monitor enthält, und die Datei clogic.pc, die die Services enthält und per embedded SQL auf die Datenbank zugreift.

Compilieren Sie den client (make client) und starten Sie ihn. Einige Funktionen sind bereits implementiert. Probieren Sie diese aus.

Erweitern Sie nun den Client um die folgenden Funktionen:

a) Weinbestellung: Der Benutzer gibt die gewünschte Weinsorte (Name, Jahrgang) und die Anzahl an Flaschen ein. Die Bestellung wird in die *order*-Datenbank eingefügt.

b) Weinlieferung: Der Benutzer gibt die Weinsorte und Jahrgang des gelieferten Weines ein. Daraufhin wird in der Bestellungsdatenbank der Unfang der Lieferung ermittelt. Die cellar-Datenbank und die order-Datenbank werden upgedatet.

c) Weinprobe: Es werden 5 Weine zufällig ausgewählt. (Sie können davon ausgehen, daß genau 30 Weine in der Produkt-Datenbank gespeichert sind.) Von jedem dieser Weine werden 2 Flaschen aus dem Keller entfernt.

Aufgabe 2: Geschachtelte Transaktionen

a) Bei der Weinprobe-Funktion aus der vorigen Aufgabe kann das Problem auftreten, dass eine Datenbankoperation abbricht, weil nicht mehr genügend Flaschen vorhanden sind. Wenn Sie die gesamte Weinprobe in *einer* Transaktion ablaufen lassen, so wird in diesem Fall alles abgebrochen. Modifizieren Sie Ihr Programm so, dass es geschachtelte Transaktionen verwendet, um Fehler abfangen zu können. (Siehe Handbuch "Transactional C Programmer's Guide and Reference, Kapitel 3 und Kapitel 8.1.1 für Informationen über geschachtelte Transaktionen). Ihr Programm soll im Fehlerfall autommatisch 36 neue Flaschen des ausgegangenen Weises bestellen.

b) Modifizieren Sie Ihr Programm nun so, dass statt 5 verschiedenen Weinsorten 5 mal dieselbe Sorte verbraucht wird, jeweils in einer Subtransaktion. Welches Verhalten stellen Sie fest? Können Sie es erklären? Informationen finden Sie im Handbuch "Encina Toolkit Server Core Programmer's Reference", Kapitel 28.5

Aufgabe 3: Parallele Ausführung

Ändern Sie erneut Ihr Programm, indem sie das "Verbrauchen" der 5 Weinsorten parallel ausführen. Verwenden Sie dazu das "concurrent"-Statement, das in Kapitel 8.1.2 des "Transactional C Programmer's Guide and Reference" beschrieben wird.

Welches Verhalten beobachten Sie hier, wenn zwei Subtransaktionen denselben Wein verbrauchen wollen? Was passiert, wenn eine Subtransaktion abbricht? Kontrollieren Sie insbesondere den Zustand der Datenbank nach dem Abbruch einer Subtransaktion.

5 Programmiertips

Die Programmierung verteilter Transaktionen ist durch die Verwendung von "Transactional C", das zu Encina gehört, sehr einfach. Durch das "transaction - onCommit - onAbort"- Konstrukt lassen sich verteilte Transaktionen , auch geschachtelte, sehr elegant beschreiben. Nehmen sie das existierende Client-Programm als Beispiel für die Verwendung von Transactional C. Genauere Informationen finden Sie im Transactional-C Reference Manual.

Der Aufruf von Services wird durch den RPC-Mechanismus sehr erleichtert. Der Aufruf eines entfernten Service im Client-Programm unterscheidet sich nicht von einem lokalen Prozeduraufruf. Das Client-Programm muss lediglich spezielle, für den Server generierte Header-Files (in unserem Beispiel product.h, cellar.h, und order.h) includieren. Ausserdem müssen beim Linken eine Reihe von automatisch generierten Modulen integriert werden. Eine genauere Beschreibung des Mechanismus findet sich in Handbuch "Writing Encina Applications", Kapitel 1.2 und 2.2.