Der Einsatz von Oracle Job-Chains im ETL-Prozeß

Norbert Klamann

Klamann Software Ltd.

Erftstadt

Schlüsselworte:

DBMS_SCHEDULER, Chain, Job, Step, Rule, ETL, Batch

Einleitung

Dies ist ein Erfahrungsbericht über den Einsatz von Oracle-Job-Chains mit dem DBMS_SCHEDULER-Package von Oracle. Es sollen praktisch bewährte Problemlösungen dargestellt werden, es ist nicht das Ziel, alle Möglichkeiten dieser technischen Methode darzustellen. Der Vortrag wendet sich an Entwickler und setzt Kenntnisse in der Entwicklung mit PL/SQL voraus.

Ausgangslage und Problemstellung

Beim Kunden werden nächtlich Daten aus den operativen Systemen in die Tabellen eines Datawarehouse geladen. Dieses DWH ist eine Oracle-Instanz in Version 11.2.

Zur freien Auswertung werden die Daten später in eine EXASOL-Datenbank kopiert, auf die mit SAP BO zugegriffen wird. Im Zuge des Ladens der DWH-Tabellen werden auch Berichte erstellt, die direkt an die operativen Abteilungen geleitet werden und dort aktuell benötigt werden.

Der Prozeß vom Eintreffen der Daten in der DWH-Datenbank an bis zum Befüllen der EXASOL und dem Erstellen und Versenden der Berichte wird im folgenden ETL-Prozeß genannt.

Für den ETL-Prozeß steht ein Zeitfenster von ca. 01:00 Uhr morgens bis 8:00 Uhr zur Verfügung und beim Beginn des Projektes war abzusehen, daß die Datenmenge in Zukunft deutlich wachsen würde und andererseits war der Zeitplan nur zu halten, wenn der Prozeß reibungslos durchlief.

Die verschiedenen Verarbeitungsschritte (etwa 30) waren als einzelne Prozeduren realisiert und diese wurden von einer übergeordneten Stored Procedure sequentiell aufgerufen. Dies führte zu folgenden Problemen:

- Durch die feste zeitliche Reihenfolge wurde die Maschinenkapazität nicht ausgenutzt, da immer nur ein Schritt zu einer Zeit lief, während rein fachlich mehrere hätten parallel laufen können.
- Es war schwierig, den Prozeß im Fehlerfalle korrekt wieder aufzusetzen. Es mußte manuell ermittelt werden, in welchem Schritt das Problem aufgetreten war. Danach mußten in der aufrufenden Prozedur die richtigen Zeilen auskommentiert und diese dann neu gestartet werden.

Diese Arbeiten zu ungünstigen Tageszeiten auszuführen, ist äußerst unangenehm und fehlerträchtig.

Zur Lösung dieser Probleme haben wir die Steuerung der einzelnen Schritte des ETL-Prozesses auf Oracle Job-Chains umgestellt und damit folgendes erreicht:

- Verringerung der Laufzeit um etwa 2 Stunden ohne Tuning der einzelnen Prozeduren.
- Größere Sicherheit durch klares Erkennen von Problemen und einfachen Restart nach Korrektur.
- Weniger Verzögerungen im Fehlerfall, weil die vom Fehler nicht betroffenen Verarbeitungsschritte in der Jobkette weiter laufen und ihre Arbeit verrichten.

Auch zur gezielten Weiterentwicklung des Systems ist das Job-Chain-System nützlich, da die individuellen Laufzeiten der einzelnen Schritte von Oracle bereits in der normalen Job-Protokollierung erfaßt werden. Es ist relativ einfach, diese Daten zu sichern und damit langfristige Trends in der Laufzeit zu erkennen. Damit können sich langsam entwickelnde Probleme frühzeitig erkannt werden.

Ein konstruiertes Beispielsystem

Die folgende Abbildung zeigt ein einfaches Batch-System mit einigen Programmen und Tabellen. Es werden von außen kommende Informationen (Input1 etc.) über mehrere Programme (P1 etc.) und Tabellen (T1 etc.) in die Ergebnisdaten (Result1 etc.) umgewandelt. Das Oracle Job-Chain-System ist für die Steuerung solcher Jobketten ausgelegt.

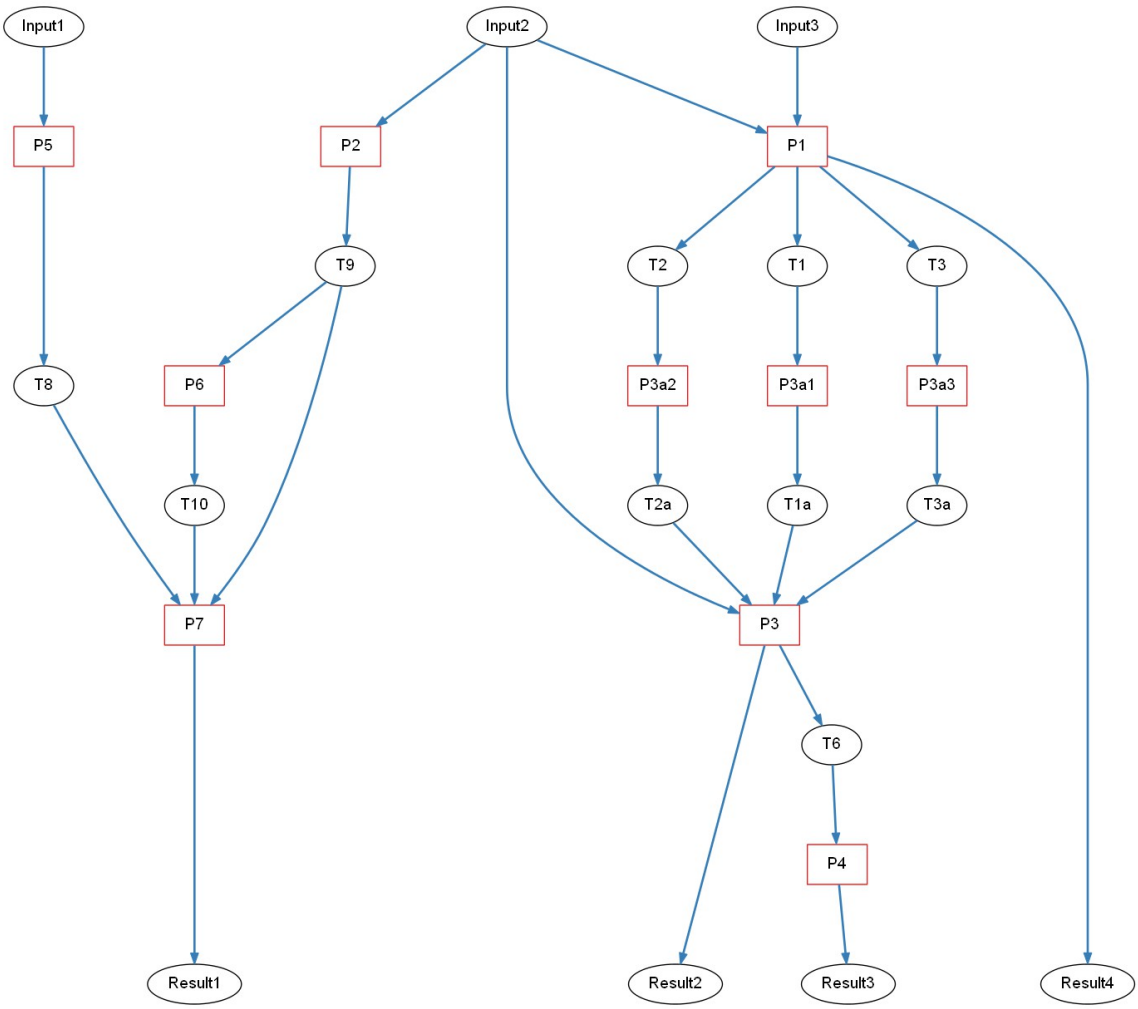

Abbildung 1: Beispielsystem mit Programmen und Tabellen

An diesem Beispiel wird später gezeigt, wie es in die Steuerung einer Chain übersetzt wird.

Das Oracle Job-Chain-System

Seit der Version 10 liefert Oracle das Package DBMS_SCHEDULER mit aus und dieses bietet zahlreiche Möglichkeiten, innerhalb der Datenbank Hintergrundprozesse zu starten und zu koordinieren.

Die verschiedenen Aspekte des Systems sind über Systemviews und die einzelnen Einstiegspunkte des

Packages DBMS_SCHEDULER aus PL/SQL heraus gut kontrollierbar und damit bietet sich DBMS_SCHEDULER zur Steuerung komplexer Batch-Prozesse geradezu an. Man kann damit auch direkt Programme auf Betriebssystemebene aufrufen und hat damit eine aus Oracle heraus besser kontrollierbare Alternative zu cron-Jobs des Betriebssystems.

Für die Steuerung komplexer Batch-Prozesse sind die folgenden Bestandteile des Systems zentral:

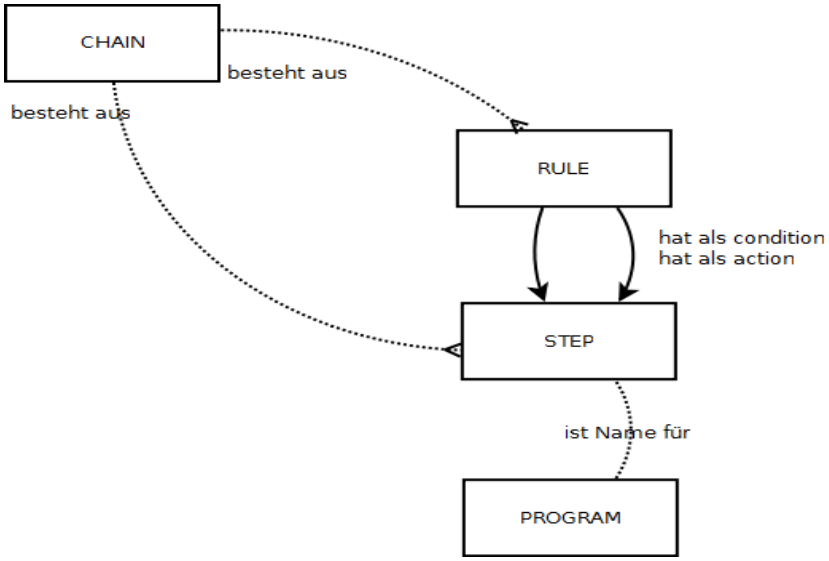

Abbildung 2: Wichtige Objekte des Chain-Systems

Die CHAIN ist die logische Struktur der Abhängigkeiten zwischen einzelnen STEPS. Wenn man eine CHAIN definiert, übersetzt man im Grunde eine Abbildung wie die Abbildung 1 in eine formale Darstellung in Form einer Reihe von Aufrufen von DBMS_SCHEDULER-Prozeduren . Eine Chain hat keinen Sinn, ohne daß man die einzelnen Verarbeitungsschritte und ihre Verbindungen definiert und dieses geschieht durch STEPS und RULES.

Die RULE bildet die Logik der Chain ab, der Hauptaufwand beim definieren von Chains steckt im Festlegen der RULEs. Eine RULE verbindet immer eine Vorbedingung mit einer Aktion. Wenn die Vorbedingung erfüllt ist, wird die definierte Aktion ausgeführt.

Es gibt folgende Vorbedingungen:

- TRUE ist immer wahr und führt dazu, daß die Aktion der RULE beim Start der Chain sofort ausgeführt wird. Mindestens eine RULE der Chain muß diese Vorbedingung haben, sonst geschieht nichts
- Xxx SUCCEDED gilt bei fehlerfreier Ausführung von Xxx
- Xxx COMPLETED gilt bei Ausführung von Xxx , mit oder ohne Fehler
- Xxx FAILED gilt bei fehlerhafter Ausführung von Xxx

Vorbedingungen können mit AND oder OR verknüpft werden.

Aktionen können u.a. sein

- END beendet die CHAIN. Ein STEP muß diese Aktion haben, sonst bleibt die CHAIN immer im Status RUNNING, obwohl alle STEPs beendet sind.
- START Xxx startet den entsprechenden STEP der gleichen CHAIN, es können auch mehrere STEPS gleichzeitig gestartet werden.

Ein STEP ist im Grunde nur ein Name für ein PROGRAM (oder eine andere Chain) im Kontext einer bestimmten CHAIN. Dieser Name wird in den RULES verwendet, um die logischen Zusammenhänge auszudrücken.

Ein PROGRAM wiederum ist 'irgend etwas lauffähiges', also eine anonymer PL/SQL-Block, eine Stored Procedure (auch mit Package) oder ein Programmaufruf auf Betriebssystem-Ebene.

Zur Laufzeit generiert Oracle aus dem STEP einen JOB, der die im PROGRAM beschriebene Aktion in einer eigenen Session und im Hintergrund ausführt. PROGRAMs existieren unabhängig von CHAINs. Sie müssen vorhanden sein, damit STEPS angelegt werden können

Zur Laufzeit wird die CHAIN durch einen JOB gestartet und Oracle überprüft nun regelmäßig, ob für irgendeine der definierten RULEs deren Vorbedingungen erfüllt sind. Sobald das für eine RULE zutrifft, wird deren Aktion ausgeführt. Falls diese Aktion der Start eines STEPs ist, wird – wie oben beschrieben – ein entsprechender JOB gestartet.

> (Es stellt sich die Frage, warum die auszuführende Aktion nicht direkt im STEP, sondern in einem PROGRAM definiert sein muß. Vermutlich steht dahinter die Idee, PROGRAMs in mehreren CHAINs wieder verwenden zu können. Ob es wirklich die richtige Entscheidung war, dafür ein eigenes Laufzeitobjekt einzuführen, soll dahin gestellt bleiben.)

Kurzbeschreibung der wichtigsten Schnittstellen

Hier sollen nur die wichtigsten Prozeduren und Systemviews aufgezählt und kurz erläutert werden, Details sollten der offiziellen Oracle-Dokumentation zum Thema DBMS_SCHEDULER entnommen werden.

Die Konstrukte des Oracle Job-Chain-Systems werden alle mit Unterprozeduren des Packages DBMS_SCHEDULER angelegt und gelöscht. Diese Prozeduren haben fast alle einen Parameter für comments und dieser sollte benutzt werden, um den Zweck des Konstrukts zu beschreiben. Realistische Chains können durchaus 200 Steps haben und dann ist jedes Stück Beschreibung im System wertvoll.

```
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name              IN VARCHAR2,
   rule_set_name           IN VARCHAR2 DEFAULT NULL,
   evaluation_interval     IN INTERVAL DAY TO SECOND DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Der rule_set_name wird normalerweise nicht gebraucht, man kann damit eine bereits vorher definierte Menge an RULEs referenzieren. Das evaluation_interval sollte auf mindestens eine Minute gesetzt werden. Der Defaultwert führt dazu, daß nach jedem abgeschlossenen Schritt überprüft wird, ob die Vorbedingungen von RULEs erfüllt sind und das ist theoretisch auch sinnvoll. Wenn aber die CHAIN schwerwiegende Logik-Fehler hat und dadurch ein Schritt immer sofort endet, ohne daß etwas sinnvolles folgen kann, führt dieses Verhalten zu deutlicher Maschinenlast und sehr umfangreichen Jobprotokollen.

Vor dem Anlegen von Steps müssen die PROGRAMs existieren

```
DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name             IN VARCHAR2,
   program_type             IN VARCHAR2,
   program_action           IN VARCHAR2,
   number_of_arguments      IN PLS_INTEGER DEFAULT 0,
   enabled                  IN BOOLEAN DEFAULT FALSE,
   comments                 IN VARCHAR2 DEFAULT NULL);
```

number_of_arguments dient dazu, Laufzeitargumente zu definieren, die an das PROGRAM zur Laufzeit übergeben werden. Dies können nur Eingabeparameter sein und wir haben dieses Feature nicht benutzt.

Die Steps werden angelegt mit

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name              IN VARCHAR2,
   step_name               IN VARCHAR2,
   program_name            IN VARCHAR2);
```

leider kann man hier keinen Kommentar eingeben.

Die Rules werden definiert mit

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name              IN VARCHAR2,
   condition               IN VARCHAR2,
   action                  IN VARCHAR2,
   rule_name               IN VARCHAR2 DEFAULT NULL,
   comments                IN VARCHAR2 DEFAULT NULL);
```

Der rule_name ist normalerweise unnötig.

Zu beachten ist, daß die Eingaben für condition und action beim Aufruf dieser Prozedur nicht streng überprüft werden. Schreibfehler an dieser Stelle, z.B. das Referenzieren von nicht vorhandenen STEP-Namen sind **kritisch** und führen u.U. dazu, daß auf Betriebssystemebene des Oracle-Servers der entsprechende Kontrollprozeß abgeschossen werden muß, weil das System sehr viele Jobs in kurzer Zeit startet.

Das Scheduling-System kann über die Statischen Systemviews ALL/DBA/USER_SCHEDULER_* kontrolliert werden.

Zur Analyse der statischen Struktur können benutzt werden:

- DBA_SCHEDULER_CHAIN_RULES
- DBA_SCHEDULER_CHAIN_STEPS
- DBA_SCHEDULER_JOBS
- DBA_SCHEDULER_PROGRAMS

Zur Laufzeitkontrolle dienen:

- DBA_SCHEDULER_RUNNING_CHAINS
- DBA_SCHEDULER_RUNNING_JOBS
- DBA_SCHEDULER_JOB_LOG
- DBA_SCHEDULER_JOB_RUN_DETAILS
  (in dieser Tabelle ist die Log-Id der gesamten Chain in der Spalte ADDITIONAL_INFO zu finden.)

Benutzung von Chains zur Laufzeit

Die Chain wird mit einem DBMS_SCHEDULER-Job gestartet, dessen TYPE CHAIN und dessen ACTION der Chain-Name ist. Wann dieser Job startet, wird genauso festgelegt, wie bei anderen DBMS_SCHEDULER-Jobs: Entweder durch durch enablen, wenn kein Schedule gegeben ist, durch einen im Job selber definierten SCHEDULE oder durch Bezug auf einen benanntes SCHEDULE-Objekt.

Mit einem Statement ähnlich dem folgenden kann man kontrollieren, welche Chains gerade laufen und wie deren Zustand ist

```
select JOB_NAME, CHAIN_NAME, STEP_NAME, STATE, ERROR_CODE,
START_DATE,END_DATE, DURATION
  from DBA_SCHEDULER_RUNNING_CHAINS
 order by OWNER, JOB_NAME, START_DATE
```

ergibt z.B. :

JOB_NAME	CHAIN_NAME	STEP_NAME	STATE	ERROR_CODE	START_DATE	END_DATE	DURATION
RUN_DB2	C_DB2	S_0000_START	SUCCEEDED	0	31.03.12 14:25:04,539470 +02:00	31.03.12 14:25:10,601074 +02:00	+000000000
RUN_DB2	C_DB2	S_0102_SERVICE	SUCCEEDED	0	31.03.12 14:25:10,722009 +02:00	31.03.12 16:08:15,205578 +02:00	+000000000
RUN_DB2	C_DB2	S_0103_RETOURE	SUCCEEDED	0	31.03.12 14:25:10,743131 +02:00	31.03.12 16:22:12,860838 +02:00	+000000000
RUN_DB2	C_DB2	S_0104_LOGISTIK	FAILED	4063	31.03.12 14:25:10,827835 +02:00	31.03.12 14:25:33,762727 +02:00	+000000000
RUN_DB2	C_DB2	S_0401_PAYMENT_START	SUCCEEDED	0	31.03.12 16:32:21,919985 +02:00	31.03.12 16:32:21,920017 +02:00	+000000000
RUN_DB2	C_DB2	S_0411_PAYM_FILL_BI	SUCCEEDED	0	31.03.12 16:32:23,430245 +02:00	31.03.12 17:01:34,108099 +02:00	+000000000
RUN_DB2	C_DB2	S_0412_PAYM_FILL_ORDER	SUCCEEDED	0	31.03.12 16:32:23,529822 +02:00	31.03.12 16:56:04,032304 +02:00	+000000000
RUN_DB2	C_DB2	S_0413_PAYM_FILL_NOTI	SUCCEEDED	0	31.03.12 16:32:23,530043 +02:00	31.03.12 17:12:16,675682 +02:00	+000000000
RUN_DB2	C_DB2	S_0414_PAYM_FILL_REMD	SUCCEEDED	0	31.03.12 16:32:23,597860 +02:00	31.03.12 16:35:48,366374 +02:00	+000000000
RUN_DB2	C_DB2	S_0420_PAYMENT_PHASE2	SUCCEEDED	0	31.03.12 17:12:16,690766 +02:00	31.03.12 17:44:19,272098 +02:00	+000000000
RUN_DB2	C_DB2	S_0302_MERGE_SERVICE	NOT_STARTED				
RUN_DB2	C_DB2	S_0202_AGG_LOGI_UNIV	NOT_STARTED				
RUN_DB2	C_DB2	S_0201_AGG_ORDER	NOT_STARTED				

Abbildung 3: Laufende Chain mit Fehler

Diese Ausgabe zeigt, daß der Schritt S_0104_LOGISTIK gescheitert ist und daß daher die letzten drei Schritte noch nicht gestartet sind. Die Schritte der S_4%_PAYMENT- Reihe dagegen sind weiter gelaufen, da sie nicht von diesem Schritt abhängig waren.

Zur Korrektur wird das PROGRAM des Schrittes S_0104_LOGISTIK separat nachgefahren und danach wird der Schritt mit folgendem Statement auf SUCCEEDED gesetzt:

```
begin
dbms_scheduler.alter_running_chain
(job_name=>'RUN_DB2',step_name=>'S_0104_LOGISTIK',attribute=>'STATE'
,value=>'SUCCEEDED');
end;
```

Danach wird die Kette weiterlaufen und die letzten 3 Schritte abarbeiten.

Man sollte in solchen Fällen die Kette selber unbedingt weiterlaufen lassen. Oracle hält selber nach, welche Schritte getan werden können.

Die Entwicklung von Chains

Bei der Entwicklung von Chains müssen zahlreiche Einzelobjekte zueinander passen und einige Regeln beachtet werden.

Damit das Chain-System sinnvoll eingesetzt werden kann, müssen zunächst die aufgerufenen Programme eine Reihe von Voraussetzungen erfüllen.

Programme dürfen keine OUT-Parameter benutzen

OUT-Parameter können nicht verwendet werden. Da die PROGRAMs zur Laufzeit als eigene JOBs in eigener SESSION aufgerufen werden, gibt es keinen Ort, wo ein solcher OUT-Parameter sinnvoll hin geschrieben werden könnte.

IN-Parameter können verwendet werden, müssen aber als zusätzliche JOB-Argumente im Chain-System deklariert werden. Wir haben diese nicht verwendet.

Empfehlung : Programme sollten ihre Steuerinformationen aus gemeinsamen Steuertabellen beziehen oder selbständig arbeiten können.

Exceptions müssen eskaliert werden

Ein Programm darf auf gar keinen Fall EXCEPTIONs 'verschlucken'. Die Steuerungslogik der Chain muß sich darauf verlassen, daß im Fehlerfall vom Programm eine EXCEPTION geworfen wird. Deren Nummer kann sogar in RULEs ausgewertet werden.

Jedes EXCEPTION WHEN OTHER muß also ein RAISE oder ein RAISE_APPLICATION_ERROR enthalten.

Wenn diese Regel nicht beachtet wird, kann die Chain im Fehlerfall nicht genau an der richtigen Stelle anhalten und ein wesentlicher Sinn des Verfahrens wird nicht erreicht.

Die Abhängigkeiten zwischen Programmen und Tabellen müssen dokumentiert sein

Um die Regeln und Abhängigkeiten definieren zu können, die ja den eigentlichen Inhalt einer Chain bilden, muß klar sein, welches Programm welche Daten verwendet und erzeugt.

Für jede beteiligte Funktion muß dokumentiert werden, welche Tabellen sie als Eingabe und Ausgabe benötigt bzw. erzeugt. Wenn in der Funktion ein Update einer Tabelle erfolgt, ist die Tabelle sowohl Eingabe als auch Ausgabe und es gilt festzuhalten, welchen Status die Eingabe haben muß, damit die Verarbeitung erfolgen kann

Es liegt auf der Hand, daß relativ kleine Funktionseinheiten mit wenigen , klar erfaßten

Abhängigkeiten zu Tabellen leichter parallelisierbar sind, als monolithische Programme. Hier muß mit Augenmaß entschieden werden, denn mehr kleine Einheiten bedeuten auch höheren Aufwand bei der Deklaration und Verwaltung der Chain. Wenn der Prozeß der Chain-Entwicklung allerdings beherrscht wird (siehe unten), wirken sich kleinere Einheiten sehr positiv auf die Laufzeit aus und nutzen die Maschine besser.

Die Strukturierung der Chain

Aus den gesammelten Informationen über die Abhängigkeiten zwischen Programmen und Tabellen läßt sich ein Bild ähnlich der Abbildung 1 oben erzeugen:

Aus der Abhängigkeit zwischen Programmen und Tabellen kann man die RULEs und STEPs der Kette formulieren. Die folgende Abbildung zeigt die Übersetzung der oben dargestellten Zusammenhänge:

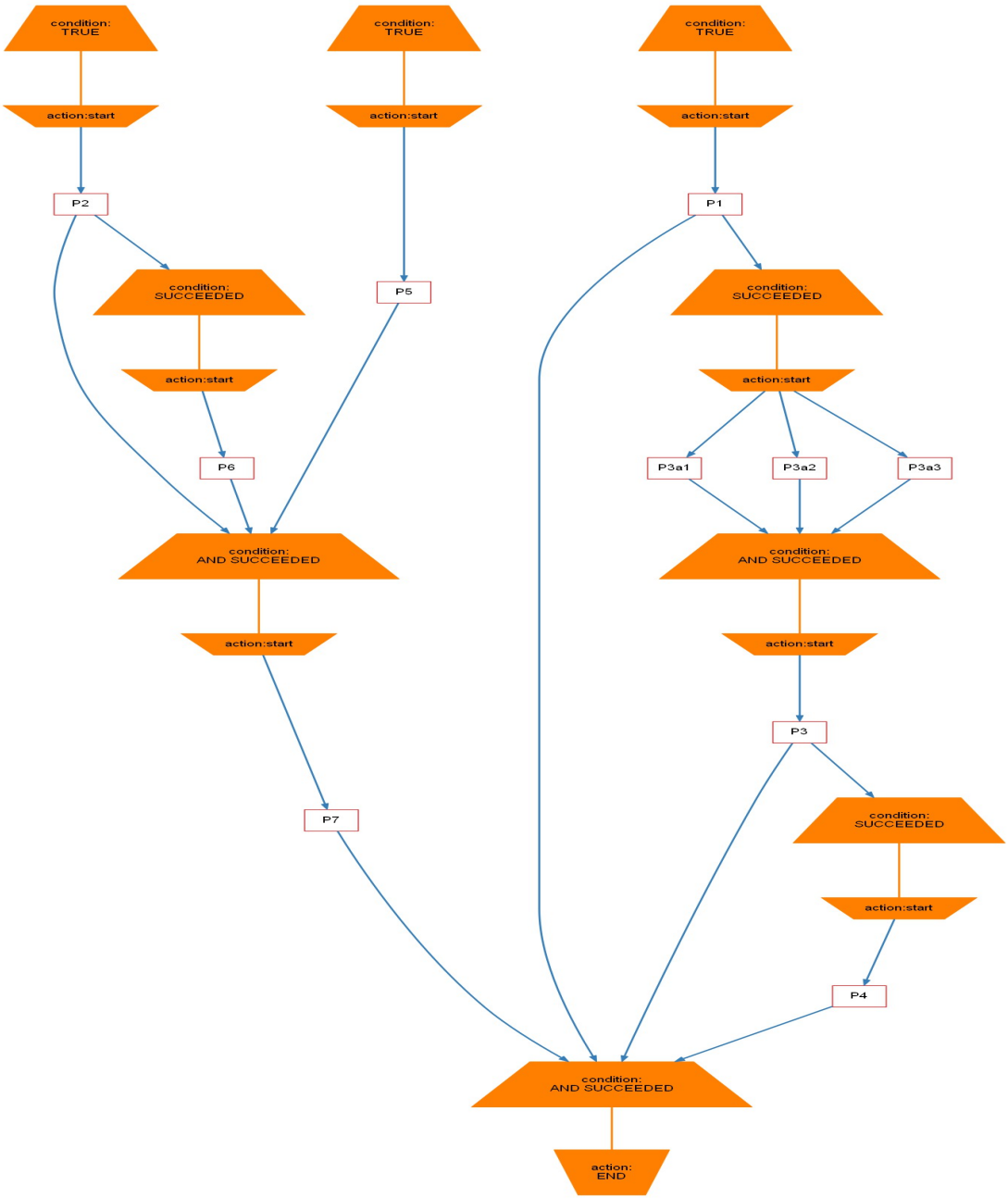

Abbildung 4: Beispielsystem mit RULEs und STEPs

Die orange unterlegten Symbole zeigen die RULEs, jeweils getrennt nach condition und action. Die Kästchen P1 etc. sind jetzt als STEPs aufzufassen, die jeweils das entsprechende Programm starten.

Diese Umsetzung ist der wichtigste Schritt bei der Entwicklung von Chains. Es ist empfehlenswert, vom einzelnen Schritt des Prozeßablaufes rückwärts zu schauen und zu fragen:

- Welche Tabellen werden gelesen und bearbeitet ?
- Welche Programme bringen die Tabellen in den Zustand, der gebraucht wird ?
- Alle diese Programme müssen erfolgreich abgeschlossen sein , bevor das betrachtete Programm starten kann.

Zusätzlich sind noch einige Rahmenbedingungen zu berücksichtigen:

Die Chain braucht mindestens eine RULE mit Vorbedingung TRUE

Wie oben bereits angesprochen, muß mindestens eine RULE die condition TRUE haben, damit in der Chain nach dem Start etwas geschieht. Wenn mehrere RULES diese Vorbedingung haben, starten sie alle gleichzeitig.

Die Chain braucht genau eine RULE mit der Aktion END

Wenn nicht eine Rule die action END hat, endet die Chain nicht, auch wenn alle STEPS beendet sind. Es ist gefährlich, wenn mehrere RULEs die action END haben, denn die erste, die zieht beendet dann die gesamte Kette ohne Rücksicht auf noch laufende STEPs, diese werden hart abgeschossen, wenn die Chain endet.

Es ist daher sinnvoll, einen letzten Schritt einzusetzen, der selber keine fachliche Arbeit leistet, sondern nur die condition einer RULE ist, deren action END ist.

Die Chain sollte keine losen Enden haben

Es sollten alle Verarbeitungswege in der Chain direkt oder indirekt mit dem oben genannten letzten Step verbunden werden. Andernfalls wäre die Chain vielleicht beendet, während diese Steps ohne Nachfolger noch weiter laufen.

Ein mehrfacher Start der Chain sollte abgefangen werden

Je nachdem , wie häufig die Chain gestartet wird und wie lange sie läuft, ist es u.U. erforderlich, beim Start der Chain zu überprüfen, ob sie bereits läuft. In diesem Fall sollte die Chain wahrscheinlich beendet werden, da die meisten Prozeduren nicht darauf ausgelegt sein werden, mehrfach gestartet zu werden. Ein Überprüfungsstep am Beginn der Chain sollte ein RAISE_APPLICATION_ERROR durchführen, wenn die Chain bereits läuft, andernfalls sollte er die Tatsache des Laufs in einer Datenbanktabelle protokollieren.

In der Chain muß dann eine RULE existieren, die diesen STEP auf FAILED anfragt und zum Ende der Verarbeitung springt.

Andererseits muß vor dem Ende der Verarbeitung die am Anfang gesetzte Markierung in der Datenbanktabelle zurückgesetzt werden, damit die Chain überhaupt ein weiteres Mal aufgerufen werden kann.

Verfahren zur Entwicklung von Chain-Sourcecode

Während der Entwicklungsphase müssen Chains immer wieder 'überschrieben' werden, weil Änderungen erforderlich sind. Die Definition einer Chain mit ihren PROGRAMs, STEPs und RULEs ist ein komplexes Gebilde, in dem jedes Element genauestens zu den anderen passen muß, damit das Ganze funktioniert.

Es ist zweckmäßig, die Definition einer Chain als ganze wie PL/SQL-Code zu behandeln und zu managen. Damit man Änderungen auf der Basis dessen machen kann, was wirklich in Betrieb ist, muß man diesen Code auch aus dem Stand in der Datenbank wiederherstellen können.

Wenn man Chains ersetzten will, muß man außerdem den Fall behandeln können, daß manche Elemente der Chain schon existieren und andere nicht. Die existierenden müssen gelöscht werden, damit sie neu angelegt werden können, die nicht existierenden wiederum dürfen nicht gelöscht werden.

Es ist daher praktisch, die oben genannten Prozeduren des DBMS_SCHEDULER-Packages in Wrapper zu packen, welche die Problematik des manchmal erforderlichen Löschens kapseln. Damit wird dann eine CREATE OR REPLACE-Semantik zur Verfügung gestellt.

In unserem Projekt haben wir zu diesem Zweck ein Package geschrieben, daß alle Hilfsfunktionen zum Umgang mit dem Chain-System bündelt.

Daher sieht der Sourcecode einer sehr einfachen Chain z.B. so aus :

```
-- Anlegen aller Steps und Rules  der Chain  C_DB2, generiert am 29.09.11
-- Diesen Text als "C_DB2.sql" im Unterverzeichnis "scripts" speichern und mit svn
versionieren
-- $HeadURL: xxxx $
-- $Id: c_db2.sql 1086 2011-10-19 10:00:19Z norbert.klamann $
DECLARE
L_CHAIN_NAME varchar2(30) := 'C_DB2';
  L_COMMENT       varchar2(200) := 'Berechnung der Deckungsbeiträge II';-- Anzahlen (vor
diesen Änderungen) : steps 14 rules 15
BEGIN
PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_CHAIN(L_CHAIN_NAME, L_COMMENT, P_FORCE => TRUE);
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0000_START','P_AGG_PREPARE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0102_SERVICE','P_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0103_RETOURE','P_RETOURE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0104_LOGISTIK','P_LOGISTIK');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0201_AGG_ORDER','P_AGG_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0202_AGG_LOGI_UNIV','P_AGG_LOGI_UNIV');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE','P_AGG_SERVICE');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0401_PAYMENT_START','P_PAYMENT_START');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME,
'S_0411_PAYM_FILL_BI','P_PAYMENT_FILL_BI_PCTRL_LOG');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME,
'S_0412_PAYM_FILL_ORDER','P_PAYMENT_FILL_ORDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME,
'S_0413_PAYM_FILL_NOTI','P_PAYMENT_FILL_PAYMENT_NOTI');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME,
'S_0414_PAYM_FILL_REMD','P_PAYMENT_FILL_REMINDER');
DBMS_SCHEDULER.DEFINE_CHAIN_STEP(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2','P_PAYMENT_PHASE2');
--

-- Rules

--
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'TRUE','START "S_0000_START"', comments=>
'Vorbereitung');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START
"S_0401_PAYMENT_START"', comments=> 'kann ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START
"S_0102_SERVICE"', comments=> 'kann ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START
"S_0103_RETOURE"', comments=> 'kann ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0000_START SUCCEEDED','START
"S_0104_LOGISTIK"', comments=> 'kann ohne Vorbedingungen starten');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START
"S_0411_PAYM_FILL_BI"', comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START
"S_0412_PAYM_FILL_ORDER"', comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START
"S_0413_PAYM_FILL_NOTI"', comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0401_PAYMENT_START SUCCEEDED','START
"S_0414_PAYM_FILL_REMD"', comments=> 'Payment Phase I , parallel');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0411_PAYM_FILL_BI SUCCEEDED AND
S_0412_PAYM_FILL_ORDER SUCCEEDED AND S_0413_PAYM_FILL_NOTI SUCCEEDED AND S_0414_PAYM_FILL_REMD
SUCCEEDED ','START "S_0420_PAYMENT_PHASE2"', comments=> 'Nach den Berechnungen aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0420_PAYMENT_PHASE2 SUCCEEDED AND
S_0102_SERVICE SUCCEEDED AND S_0103_RETOURE SUCCEEDED AND S_0104_LOGISTIK SUCCEEDED ','START
"S_0201_AGG_ORDER"', comments=> 'Nach den Berechnungen aggregieren');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0104_LOGISTIK SUCCEEDED','START
"S_0202_AGG_LOGI_UNIV"', comments=> 'Logistik-Universum kann parallel laufen');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0201_AGG_ORDER SUCCEEDED','START
"S_0302_MERGE_SERVICE"', comments=> 'weitere parallele Merges');
DBMS_SCHEDULER.DEFINE_CHAIN_RULE(L_CHAIN_NAME, 'S_0302_MERGE_SERVICE COMPLETED AND
S_0202_AGG_LOGI_UNIV COMPLETED ','END ', comments=> 'Chain fertig.');
DBMS_SCHEDULER.ENABLE(name => L_CHAIN_NAME);
 PKG_UTIL_SCHEDULING.CREATE_OR_REPLACE_SC_JOB(P_JOB_NAME =>'RUN_DB2',P_CHAIN_NAME=>
l_chain_name, P_SCHED_NAME => '', P_COMMENTS=> 'DB2-Prozess',P_FORCE => TRUE);
END;
```

Dieser Code ist ein SQL-Skript, das so aus der Datenbank generiert worden ist. Er kann nun geändert

werden und anschließend wird er als Skript wieder aufgerufen und die CHAIN hat die geänderte Struktur.

Rules dürfen keine nicht vorhandenen Steps refernzieren

Das wird beim Anlegen der Chain nicht kontrolliert und führt zu zahllosen Job-Starts in kürzester Zeit. Hier ist große Sorgfalt beim Editieren der Source nötig.

Die Logik der RULEs visualisieren

Chains können durchaus 200 Steps umfassen und diese werden bekanntlich durch die RULEs miteinander verbunden. Es ist sehr schwer, nur anhand des Textes der condition in der RULE nachzuvollziehen, ob die logische Verknüpfung wirklich dem entspricht, was geplant ist.

Man muß also die logische Verknüpfung zwischen den einzelnen STEPS in der CHAIN graphisch darstellen. Wichtiger als Schönheit der Darstellung ist dabei die Korrektheit: Alle Verknüpfungen müssen genau so abgebildet werden, wie sie definiert sind, damit man Fehler und Unklarheiten sehen kann.

> Der Oracle Enterprise-Manager ist grundsätzlich dazu in der Lage, die Logik von Chains darzustellen, nach unseren Erfahrungen funktioniert das aber nur in bestimmten Kombinationen von Internet-Explorer und diversen Plugins.

Man kann die wirklich existierenden Beziehungen per SQL aus der Systemview DBA_SCHEDULER_CHAIN_STEP ableiten und daraus den Sourcecode für Graphviz (http://www.graphviz.org) generieren. Mit diesem Tool erzeugt man aus textlichen Beschreibungen von Abhängigkeiten Graphen, die diese verständlich darstellen.

Für die oben genannte Chain C_DB2 ergibt sich folgender Graphviz-Sourcecode :

```
/* Graphviz/dot- Dokumentation der Chain xxx.C_DB2, generiert am 31.03.12
 speichern als Datei C_DB2.gv, dann mit Graphviz starten
*/
digraph  C_DB2  {
    ratio=auto; charset=latin1; center=true; fontname=Helvetica; pad=0.5;
    concentrate=true;ranksep="1.0 equally"; nodesep=1.0;colorscheme="set19";
    node [fontname=Helvetica, colorscheme="set19", color="black"];
    edge [fontname=Helvetica, colorscheme="set19", penwidth=2, color=2];
  label="Jobchain C_DB2, generiert am : 31.03.12";
TRUE [style=filled,color=3];
END  [style=filled,color=3];
S_0302_MERGE_SERVICE [shape="record",label="{S_0302_MERGE_SERVICE|P_AGG_SERVICE|
PKG_DB2_AGG.REFRESH_COST_SERVICE_DATE|Aggregiert\ auf\ Service-Ebene}"] ;
S_0412_PAYM_FILL_ORDER [shape="record",label="{S_0412_PAYM_FILL_ORDER|
P_PAYMENT_FILL_ORDER|PKG_PAYMENT_COSTS.FILL_ORDER|Payment\ Costs,,\ Fill}"] ;
S_0420_PAYMENT_PHASE2 [shape="record",label="{S_0420_PAYMENT_PHASE2|
P_PAYMENT_PHASE2|PKG_PAYMENT_COSTS.PHASE2|Payment\ Costs,\ Phase\ 2}"] ;
S_0201_AGG_ORDER [shape="record",label="{S_0201_AGG_ORDER|P_AGG_ORDER|
PKG_DB2_AGG.REFRESH_COST_ORDER|Aggregiert\ auf\ Order-Ebene}"] ;
S_0202_AGG_LOGI_UNIV [shape="record",label="{S_0202_AGG_LOGI_UNIV|P_AGG_LOGI_UNIV|
PKG_DB2_AGG.PEREPARE_LOGISTIC_UNIVERSE|DB2\ Logistik-Universum\ hinstellen}"] ;
S_0104_LOGISTIK [shape="record",label="{S_0104_LOGISTIK|P_LOGISTIK|
PKG_LOGISTIC_COSTS.PROC_L_ETL_ORDER|Logistics\ Costs}"] ;
S_0401_PAYMENT_START [shape="record",label="{S_0401_PAYMENT_START|P_PAYMENT_START|
PKG_PAYMENT_COSTS.START_PROCESS|Payment\ Costs,\ Protokollierung\ zum\ Start}"] ;
S_0411_PAYM_FILL_BI [shape="record",label="{S_0411_PAYM_FILL_BI|
P_PAYMENT_FILL_BI_PCTRL_LOG|PKG_PAYMENT_COSTS.FILL_BI_PCTRL_LOG|Payment\ Costs,\
Fill}"] ;
S_0414_PAYM_FILL_REMD [shape="record",label="{S_0414_PAYM_FILL_REMD|
P_PAYMENT_FILL_REMINDER|PKG_PAYMENT_COSTS.FILL_REMINDER|Payment\ Costs,\ Fill}"] ;
S_0102_SERVICE [shape="record",label="{S_0102_SERVICE|P_SERVICE|
PKG_SERVICE_COSTS.CALCULATE_SERVICE_COSTS|Service\ Costs}"] ;
S_0413_PAYM_FILL_NOTI [shape="record",label="{S_0413_PAYM_FILL_NOTI|
P_PAYMENT_FILL_PAYMENT_NOTI|PKG_PAYMENT_COSTS.FILL_PAYMENT_NOTI|Payment\ Costs,\
Fill}"] ;
S_0103_RETOURE [shape="record",label="{S_0103_RETOURE|P_RETOURE|
PKG_RETOURENRUECKSTELLUNG.KOMPLETT|Rückstellungen\ für\ Retouren}"] ;
S_0000_START [shape="record",label="{S_0000_START|P_AGG_PREPARE|
PKG_DB2_AGG.PREPARE_RUN|System\ vorbereiten\ (Rechte\ etc)}"] ;
TRUE -> S_0000_START;
S_0411_PAYM_FILL_BI -> S_0420_PAYMENT_PHASE2[color=3];
S_0412_PAYM_FILL_ORDER -> S_0420_PAYMENT_PHASE2[color=3];
S_0413_PAYM_FILL_NOTI -> S_0420_PAYMENT_PHASE2[color=3];
S_0414_PAYM_FILL_REMD -> S_0420_PAYMENT_PHASE2[color=3];
S_0420_PAYMENT_PHASE2 -> S_0201_AGG_ORDER[color=3];
S_0102_SERVICE -> S_0201_AGG_ORDER[color=3];
S_0103_RETOURE -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0201_AGG_ORDER[color=3];
S_0104_LOGISTIK -> S_0202_AGG_LOGI_UNIV[color=3];
S_0201_AGG_ORDER -> S_0302_MERGE_SERVICE[color=3];
S_0302_MERGE_SERVICE -> END;
S_0202_AGG_LOGI_UNIV -> END;
S_0000_START -> S_0401_PAYMENT_START[color=3];
S_0000_START -> S_0102_SERVICE[color=3];
S_0000_START -> S_0103_RETOURE[color=3];
S_0000_START -> S_0104_LOGISTIK[color=3];
S_0401_PAYMENT_START -> S_0411_PAYM_FILL_BI[color=3];
S_0401_PAYMENT_START -> S_0412_PAYM_FILL_ORDER[color=3];
S_0401_PAYMENT_START -> S_0413_PAYM_FILL_NOTI[color=3];
S_0401_PAYMENT_START -> S_0414_PAYM_FILL_REMD[color=3];
}
```

Der Code wurde durch eine Routine unseres Hilfspackages aus der Datenbank generiert, seine Inhalte basieren auf den Systemtabellen und sind daher wahr. In der Routine wurden sie nur um Stücke der Graphviz-Syntax ergänzt.

Wenn dieser Code mit Graphviz verarbeitet wird, entsteht folgende Graphik

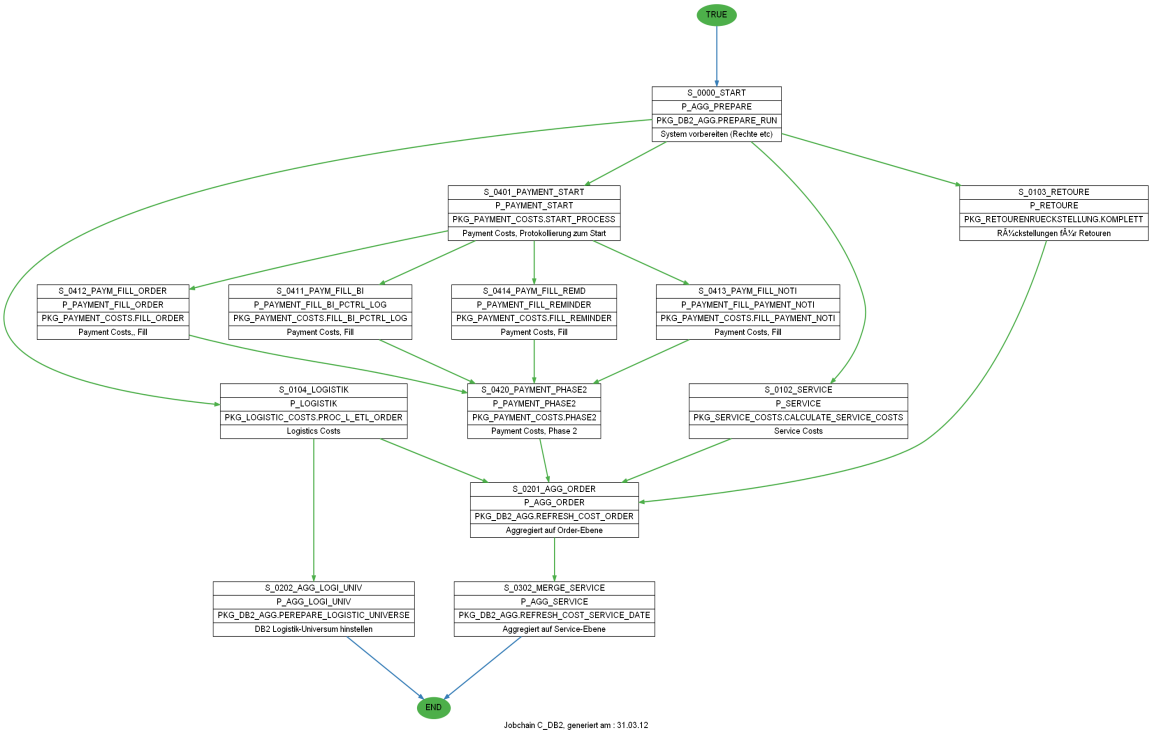

Abbildung 5: Graphviz-Darstellung einer Chain

In dieser Abbildung kann man genau verfolgen, ob der Ablauf fachlich sinnvoll ist. In dieser Variante wurden noch die Programmnamen und die Programm-Aktion sowie ein Kommentar mit ausgegeben, um die Benutzung angenehmer zu machen.

In der Chain C_DB2 wurde - wie man hier sieht - vor dem END kein Dummy-Schritt eingeführt, aber eine Sichtkontrolle des Sourcecodes oben zeigt, daß in der Tat nur eine RULE die Action END hat.

Namenskonventionen

> "There are only two hard things in Computer Science: cache invalidation, naming things and off by 1 errors"

Wenn man Chains benutzt, müssen zahlreiche neue Laufzeitobjekte eingerichtet werden und damit stellt sich das Problem der Benennung dieser Objekte.

Wir haben zu Beginn der Projekte versucht, in den Step-Namen eine sinnvolle Gruppierung durch die Namenskonvention S_Nr_Name auszudrücken, wobei die Nummer 4-stellig ist und Nummernkreise die Strukturierung in die Hauptpfade der Verarbeitung ausdrücken sollte. Die oben abgebildete Chain C_DB2 zeigt diese Konvention.

Insgesamt hat sich dieses Vorgehen **nicht** bewährt, da bei komplexeren Chains keine Pfade mehr klar zu erkennen sind, die Nummernkreise nicht ausreichen und durch das spätere Einfügen zusätzlicher Steps die Numerierung durcheinander kommt. Zu allem Überfluß können Step-Namen nicht länger als

27 Zeichen sein.

Es ist sinnvoller, den Steps einfach Namen zu geben, die ausdrücken, was in ihnen getan wird. Übersicht über die Struktur von Chains kann nur graphisch gewonnen werden.

Die Namen der PROGRAMs sind ein Problem, weil in ihnen Aufrufe von Prozeduren in Packages enthalten sein können. Ein solcher Aufruf kann bis zu 60 Zeichen + '.' lang sein, für den Namen des PROGRAM stehen aber nur 30 Zeichen zur Verfügung. Man kann also nicht einfach den Namen des aufgerufenen Objekts wiederverwenden.

Ausblick und Bewertung

In unserem konkreten Projekt haben wir nur einen Teil der Funktionalität benutzt, die das Chain-System bietet. Wir haben keine JOB-ARGUMENTs verwendet und in den condition-Deklarationen der RULEs nur AND-Verknüpfungen benutzt. An einer Stelle haben wir CHAINs geschachtelt, also eine CHAIN aus einer anderen gestartet.

Das System bietet die Möglichkeit, laufende Chains zu beeinflussen. Dies nutzt man z.B. , um nach einer Fehlerkorrektur einen gescheiterten STEP auf SUCCEEDED zu setzen. Man kann auf diese Weise auch STEPS auf PAUSED setzten, so daß die Chain dort wartet oder auf SKIPPED, so daß der Schritt sofort ohne Ausführung als erfolgreich betrachtet wird.

Theoretisch könnte man wahrscheinlich eine Chain bauen, die zur Laufzeit so umgesteuert wird, daß sie ganz unterschiedliche Prozesse durchführt. Das scheint mir nicht ratsam.

Insgesamt reicht der von uns benutzte Teil der Funktionen aus, um die angestrebten Ziele (bessere Parallelisierung und damit kürzere Laufzeit und bessere Kontrollierbarkeit der Abläufe) zu erreichen.

Das Chain-System hat Schwächen und die Entwicklung von Chains könnte komfortabler sein, der kritischste Punkt ist, daß nicht alle theoretisch abfangbaren Fehler zur 'Kompilierzeit' abgefangen werden, sondern zu unangenehmen Laufzeitfehlern führen können.

Angesichts der Tatsache, daß das System von Oracle mitgeliefert wird und sehr gut in Oracle integrierbar ist, erscheint mir sein Einsatz lohnend für alle Betriebe, die komplexe Batchabläufe in der Datenbank durchführen müssen.

Danksagungen

Ich danke dem BI-Team der Zalando GmbH in Berlin für die sehr schöne Zusammenarbeit an dieser interessanten Aufgabe. Insbesondere danke ich Eric v. Czapiewski und Jan Strassenburg für die kritische Durchsicht dieses Manuskripts.

Kontaktadresse:

Norbert Klamann

Klamann Software Ltd.

Franz-Marc-Str.174

D-50374 Erftstadt

Telefon: +49 (0) 172 2797723

Fax: +49 (0) 2235 9918299
E-Mail Norbert.Klamann@klamann-software.de
Internet: www.klamann-software.de