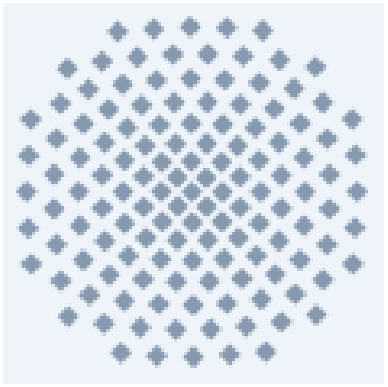

Studienarbeit Nr. 2268

Wait Activity for SWoM

Changsheng Qian

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Phys. Dieter H. Roller
Begonnen am:	01. März 2010
Beendet am:	31. August 2010
CR-Klassifikation:	C.3, C.4, H.3.5, H.4.1, J.1, J.7

UNIVERSITÄT STUTTGART
INSTITUT FÜR ARCHITEKTUR VON ANWENDUNGSSYSTEMEN

Inhaltsverzeichnis

1 Einführung

Auf service-orientiertem Architekturentwurf basierte Web-Services dienen als Grundlagen für die modernen verteilten heterogenen Anwendungen. Die Web-Services sind perfekt geeignet als Funktionsschicht des Zwei-Ebene-Programmierungsmodells, das eine wichtige Eigenschaft für Workflow-basierte Anwendungen ist.

Workflow-basierte Anwendungen bestehen aus zwei verschiedenen Teilen:
- Ein Prozessmodell. Es beschreibt, in welcher Reihenfolge die verschiedenen Aktivitäten von dem Prozessmodell durchgeführt werden (Programmieren in Großen).
- Die einzelnen Komponenten: Die die verschiedenen Aktivitäten durchführen. (Programmieren in Klein).

In der Web-Services Umgebung werden die Prozessmodelle durch „Web Services Business Process Execution Language“ (WS-BPEL) geschrieben.

Das Ziel eines Workflow-Management-System (WFMS) zur Umsetzung der WS-BPEL Spezifikationen ist es, das Lebenszyklus der Geschäftsprozesse zu verwalten, durch die verbundenen Prozessmodelle zu navigieren und zum Aufruf der entsprechenden Web-Services. Die Stuttgarter Workflow Maschine (SWoM), die am Institut für Archtektur von Anwendungssystem der Universität Stuttgart entwickelt wurde, implementiert teilweise die entsprechenden WS-BPEL Standard.

1.1 Ziel und Ablauf der Studienarbeit Wait Activity

Das Ziel dieser Studienarbeit ist es, eine Warten-Aktivität zu implementieren. Es besteht aus zwei Teilen:

- Die Erzeugung der entsprechenden Anfragen für IBM® WebSphere®
- Die Ausführung der Aufruf wird durch IBM® WebSphere® signalisiert.

Eine sehr kurze Wartezeit, die insbesondere für die Leistungsmessungen verwendet, soll inline behandelt werden.

2 Laufzeitumgebung

In diesem Kapital werden die erforderliche Laufzeitumgebung und das eingesetzte Betriebssystem vorgestellt.

2.1 Eingesetzte Software und Hardware

Seit 2009 stand die SWoM Version 3. Um SWoM 3 zum Laufen zu bringen, wurde IBM® WebSphere® Application Server V7.0.0 (1.0.0.20080911_1339) als Laufzeitumgebung installiert. Die Installation wurde sehr einfach durch IBM® Installation Manager Version 1.3.2 (1.3.2000.20090917_2216) erfolgreich ausgeführt, hat aber mehrere Stunden gedauert.

Als Datenbank wurde IBM® DB2 Express-C Version 9.7.1 verwendet und IBM® Rational® Application Developer™ for WebSphere® Software Version 7.5.4 (7.5.4.20090916_1552) wurde als Programmierplattform benutzt.

Das Testsystem wird wie folgt eingesetzt:

Hardwareumgebung:

Prozessor: Intel® Atom™ N280 1.66GHz

Arbeitsspeicher: 1GB RAM

Festplatte: 160GB

Softwareumgebung:

Betribssystem: Microsoft Windows XP Home Edition mit Service Pack 3

2.2 Stuttgarter Workflow Maschine 3

Stuttgarter Workflow Maschine, „SWoM“ ist ein Workflow Management System (WfMS). WfMS können zentral, verteilt, mobil und in ersten Forschungsprototypen auch Peer-to-Peer (P2P) realisiert sein. Die SWoM realisiert, welches wesentliche Vorzüge der unterschiedlichen Architekturen aufweist. Hierbei sind die wesentlichen Konzepte aus BPEL, als auch die BPEL-Spezifikation selber, die Grundlage für die Implementierung.[2]

Die SWoM stellt eine Umgebung zur Verfügung, die anhand aktueller Forschungsergebnisse eine BPEL Maschine umsetzt, die auf dem neuesten Stand der Technik ist. Neben aktuellen Erkenntnissen fließen auch langjährige Erfahrungen in den Bereich Datenbanken, Workflow und Web Services in die Umsetzung der Maschine ein.[2]

Die SWoM besteht aus folgenden Modulen (Abbildung 2-1):

• Prozess Ausführungsmodul

• Verwaltungsmodul
• Aufruf-Handler Modul
• Service Anbieter

Jedes Modul enthält verschiedene Komponenten. Die Module sind miteinander über Nachrichten-Middleware verbunden. Die detaillierte Abhängigkeit verschiedener Module ist in folgender Abbildung 2-1 dargestellt.

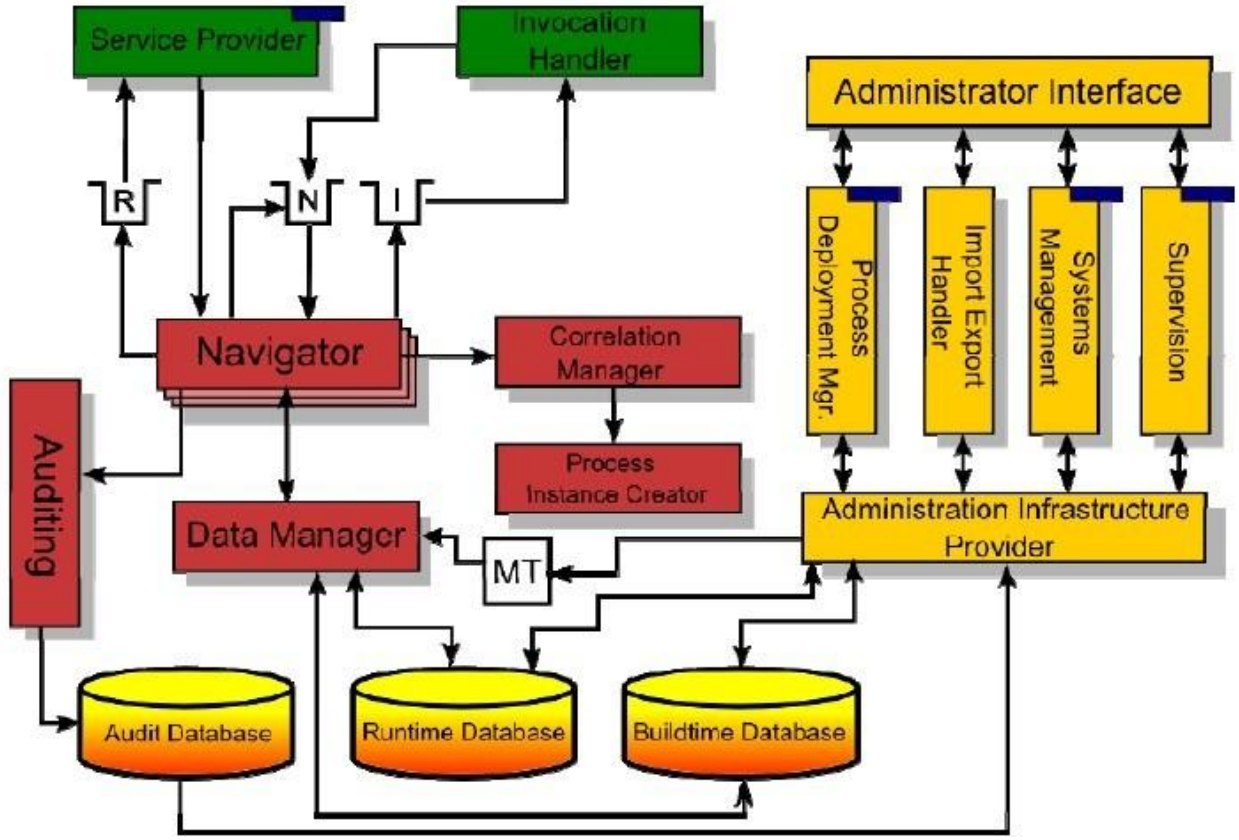

Abbildung 2-1: SWoM Architektur[1]

Der grüne Bereich zeigt Aufruf-Handler Modul und Service Anbieter. Der gelbe Bereich zeigt die Komponenten des Verwaltungsmoduls und der rote Bereich ist die Komponente des Prozess Ausführungsmodel. Die kleinen blauen Kästchen zeigen die Web-Service Schnittstellen. „I“ zeigt die Aufrufqueue, „N“ die Navigationsqueue und „R“ die Antwortqueue. Das Symbol „MT“ zeigt den Managerbereich. Die Zylinder unten sind die Datenbanken.[1]

SWoM ist im Prinzip eine SOA Anwendung, die auf WebSphere Application Server aufgebaut und eingerichtet wird. Die bei WAS verfügte Komponenten können direkt bei SWoM angewendet werden.

Die Installation der SWoM3 wurde zwar erfolgreich durchgeführt und das Einloggin der Administrationsschnittstelle war auch reibungslos, jedoch erscheinten bei Rekonstruktion der Codestruktur knapp Hundert Fehlermeldungen. Deshalb versuchte das IBM® WebSphere® Application Server V7 neu zu installieren. Dabei

wurde eine Sicherheitsfunktion ausgeblendet. Die Installation der SWoM3 wurde noch einmal durchgeführt und anschließend wurden nur noch 2 Fehlermeldungen gezeigt „Beim Lauf des WAS V7.0 sind die Zielobjekte nicht definiert.“ und mehrere Warnungen.

Trotz mehreren Versuch die beide Fehlermeldungen zu beseitigen ist dies nicht gelungen. Da die SWoM3 nicht bei WAS laufen konnte, war es somit nicht möglich weitere Tests durchzuführen.

2.3 IBM® WebSphere® Application Server

WebSphere® ist eine Produktlinie der IBM, die unter schiedliche Software für Anwendungsintegration, Infrastruktur und eine integrierte Entwicklungsumgebung umfasst. WebSphere Application Server(WAS) ist ein bekanntes WebSphere-Produkt. WAS wird sehr oft auch einfach mit WebSphere bezeichnet.[3] Bei diesem Application Server handelt es sich um eine Laufzeitumgebung und bietet eine leistungsfähige und in der Praxis bewährte Umgebung für JavaEE-Anwendungen. Er ist führende auf J2EE- und Web-Services-basierte Anwendungsserver, der für die meisten Plattformen verfügbar ist.[4]

WebSphere Application Server V 7.0 bietet folgende Funktionalitäten[5]:

- Application Server Zweck
- Entwicklung der Java Application Development Standard
- Verbesserte Verwaltung
- Leichtere Integration
- Extra Werkzeuge und Erweiterungen

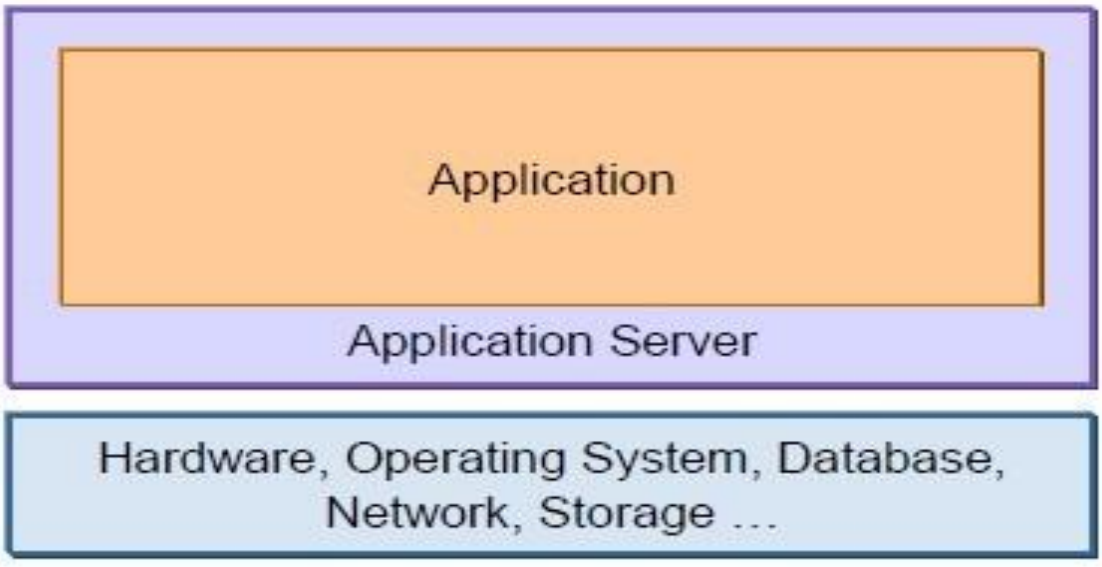

Abbildung 2-2: Darstellung eines Applicationsservers und seiner Umgebung[5]

Der Application Server fungiert als Middleware und bietet ein Programmiermodell, ein Infrastrukturrahmen und viele Stardards für eine konsenquent entworfene Verbindung zwischen Back-End Systemen und Clients.

WebSphere Application Server bietet die Umgebung für die Ausführung der Solution und integriert die Solution mit jeder Plattform und System als die Services der Geschäftsanwendungen nach SOA-Struktur.

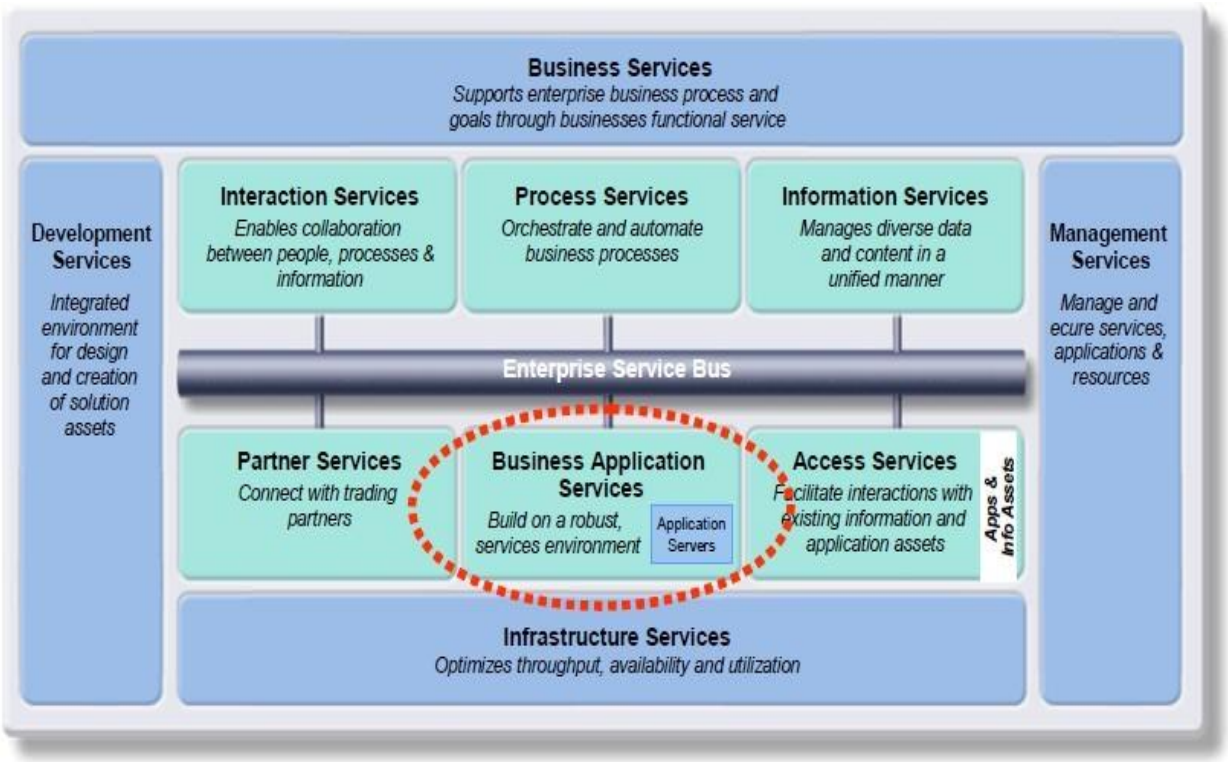

Abbildung 2-3: Die Position des Business Application Services in SOA-Struktur[5]

WebSphere Application Server ist ein zentraler SOA-Baustein. Von einer SOA-Aussicht kann man mit WAS die folgenden Funktionen ausführen:

• Schnell und einfach die wiederverwendbare Anwendungsservices erstellen und bereitstellen.

• Ausführung der Services in einer sicheren, skalierbaren, hochverfügbaren Umgebung

• Verbindet Softwarewert und erweitert dessen Reichweit

• Einfache Anwendungsverwaltung

• Erfüllung hoher Anforderungen und steigerung der Kernwiederverwendbarkeit

Mit zunehmenden geschäftlichen Anforderungen, stehen immer mehr neue Technologie- Standards zur Verfügung. Seit 1998 ist WebSphere stark gewachsen

und passt sich mit neuen Technologien und neuen Standards für eine innovative spitze Umgebungen, um den Entwurf der voll integrierten Solution zu ermöglichen und die Geschäftsanwendungen zu realisieren.[5]

2.4 IBM DB2 Express

DB2 ist ein kommerzielles relationales Datenbank Management System der Firma IBM.[6] DB2 bietet branchenführende Leistung, Skalierbarkeit und Zuverlässigkeit.

SWoM verfügt über 3 Arten von Datenbanken[7]:

• Die Prüfungsdatenbank(Audit Database). Die Prüfungsdatenbank speichert die Informationen über die Lebensdauer der Prozess-Instanz, die von Prozessmodellen instanziiert wurden. Die Informationen enthalten auch die Aktivitäten und Ereignisdaten der Prozess-Instanz.

• Die Erstellungsdatenbank(Buildtime Database). Die Erstellungsdatenbank speichert alle Informationen über Prozessmodelle, die in die SWoM importiert oder eingesetzt werden. Die Informationen enthalten BPEL- und WSDL-Daten sowie Verwaltungsdaten.

• Die Laufzeitdatenbank(Runtime Database). Zur Laufzeit werden alle Informationen über eine Prozess-Instanz eines bestimmten Prozessmodells in der Laufzeitdatenbank gespeichert. Darüber hinaus enthält die Laufzeitdatenbank die Informationen über Fehlerprotokoll, Laufzeitfehler sowie bei Nachrichtenübertragung auftretende Fehler.

3 Entwurf und Ausführung

Dieses Kapitel beschreibt die Grundidee der Studienarbeit, die Methode des Entwurfs und die Implementierung der Wartenaktivität.

3.1 Zweck der Wait Activity

Die Geschäftsprozesse können einen Punkt erreichen, wo sie notwendig auf ein Signal oder eine bestimmte Zeit warten müssen, bevor sie fortfahren. Ein Beispiel wäre ein Prozess, der braucht ein Teil von einen dritten geliefert werden, bevor er weitere Maßnahmen ergriffen kann. In solchem Szenario verwendet man eine Wait Activity, um den Prozess für eine bestimmte Zeit oder bis zu einem bestimmten Datum und Uhrzeit zu warten.[8]

3.2 Konfiguration der Wait Activity

Die Wartenaktivität hält den Prozess für eine gewisse Zeit, die von der Bedingung angegeben wird. Die Bedingungen können entweder eine bestimmte Dauer sein oder ein konkretes Datum und Uhrzeit.

Bei der Bedingung für eine bestimmte Dauer handelt es sich in dem WS-BPEL um ein For-Befehl während es sich bei einem konkretes Datum und Uhrzeit um ein Until-Befehl handelt, der beschreibt, wann der Prozess neu gestartet werden muss.

Option	Description
Date	Choose this when you want the expiration to occur when a specific time and date has been reached.
Duration	Choose this when you want expiration to occur after a certain period of time has elapsed.

Abbildung 3-1: Bedingungen für For und Until[8]

3.3 Entwurf der Wait Activity

Die Wait Activity wird verwendet, um für eine Periode oder bis zu einem bestimmten Zeitpunkt zu warten. Sie verändert kein Prozess, sondern verzögert nur die Ausführung des Prozesses mit Hilfe der WAS Scheduler für eine bestimmte Zeit. Es muss nur exakt eine von den zwei Zeitbedingungen erfüllt werden.

Der WAS Scheduler ist eine Erweiterung von WebSphere Programming und verwaltet die Tasks, um sie zu bestimmten Zeitpunkten bzw. Intervallen zu starten. Der WAS Scheduler muss richtig konfiguriert werden, bevor er aufgerufen werden kann.

Der Navigator in SWoM3 benötigt eine Pause, bevor der Prozess weiter ausgeführt werden kann. Der Navigator gibt eine Nachricht an Wait Activity Instance. Eine Wait Activity ruft den WAS Scheduler weiter, um eine Task vom Typ MessageTaskInfo zu generieren. Die Task muss an einem bestimmten Zeitpunkt rechtzeitig definiert werden und hängt mit relevanten Informationsdaten der Aktivität zusammen. Dann erzeugt der WAS Scheduler den Task und generiert eine Task ID. Die Task ID muss in der Laufzeitdatenbank gespeichert werden. Der WAS Scheduler wird zum entsprechenden Zeitpunkt eine Nachricht rechtzeitig an den Navigator senden und informiert den Navigator, die Wait Activity zu beenden und dessen Prozess weiter auszuführen. Bevor der entsprechende Zeitpunkt erreicht ist, bleibt die Wait Acitvity im Lauf. Der relevante Task von WAS Scheduler muss nach allen Aktionen weggeworfen und gelöscht werden.

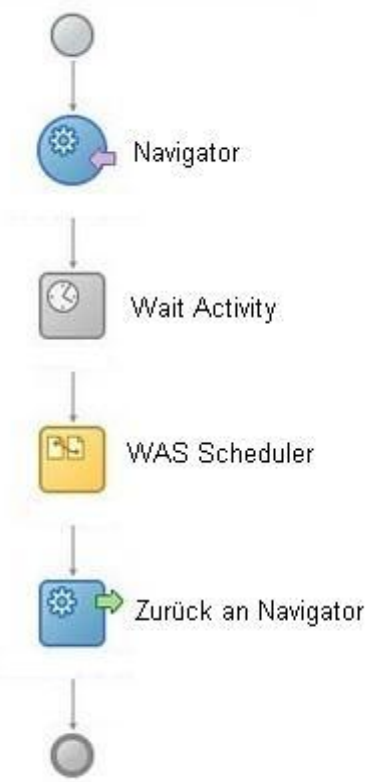

Abbildung 3-3-1: Verlauf der Wait Activity

3.3.1 Navigator

Der Navigator realisiert die Ausführungen gesamter Prozessen. Es ist eine generische Komponente, die unabhängig von einem bestimmten Prozessmodell ist. Ein Webdienst, welcher sich aus einem spezifischen Prozessmodell zusammensetzt, befindet sich in der Service-Provider Komponente. Ein speziales Beispiel der Webdient-Implementierung, d.h. die Fassade Bean wird aufgerufen und eine Startmeldung erzeugt. Dann wird die Nachricht in die von Navigator lesbare

Navigationsqueue gelegt. Der Navigator bearbeitet unter vordefiniertem Kontrollfluss alle Aktivitäten von einem gegebenen Prozessmodell.

Der Navigator benutzt Aufruf-Handler, um Webdienste remote aufzurufen. Dazu sendet der Navigator die Nachrichten an die Aufrufqueue und empfängt die Nachrichten von Navigationqueue. Der Navigator greift die Prozessdaten indirekt über ein Cache.

Der Navigator ruft auch die Operationen von der Auditing-Komponente auf, falls sie für ein Prozessmodell verfügbar sind.

Anschließend werden die Operationen des Navigators definiert, die die von SWoM unterstützten Elemente der BPEL Sprache decken.[1]

3.3.2 Durationausdrücke in Wait Activity

Die Durationausdrücke(FOR) entsprechen der XPath 1.0 Expr. Produktion und die Auswertungsergebnisse sind vom Typ unsignedInt von XML Schema.[1]

Die Methode FOR verifiziert zuerst die Gültigkeit von Eingabe des Durationausdrucks. Bei Ungültigkeit wird zur Fehlerbehandlung aufgerufen.

Als Ergebnis wird die aktuelle Zeit mit Verzögerungsdauer addiert zurückgegeben.

```
//parse a duration-expression, if not valid throw a fault
try {
    //get current time
    aDate = new GDate(Calendar.getInstance());
    this.log.debug("current: " + aDate.getDate().toString());

    //parse expression
    GDuration duration = new GDuration(waitExpression);
    if (!duraion.isValid()) {
      BPELFault invalidExpressionValue = new BPELFault(waitExpression);
      invalidExpressionValue.setPiid(this.piid);
      invalidExpressionValue.setSourceAIID(this.aiid);
      invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVALUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calculateFinishWai
tMode");
      throw invalidExpressionValue;
    }
    this.log.debug("duration-expression: " + duration.toString());


    // aDate = current time + duration
    aDate = aDate.add(duration);
    this.log.debug("current + duration: " + aDate.getDate().toString());
} catch (Exception e) {
    BPELFault invalidExpressionValue = new BPELFault(waitExpression,e);
```

```
    invalidExpressionValue.setPiid(this.piid);
    invalidExpressionValue.setSourceAIID(this.aiid);
    invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVALUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calculateFinishWai
tMode");
    throw invalidExpressionValue;
}
break;
```

Abbildung 3-3-2: Programmcode für Methode FOR

3.3.3 Ablaufausdrücke in Wait Activity

Die Ablaufausdrücke(UNTIL) entsprechen auch der XPath 1.0 Expr. Produktion und die Auswertungsergebnisse sind vom Typ DateTime oder Date von XML Schema.[1]

Die Methode UNTIL verifiziert zuerst die Gültigkeit und die Vollständigkeit der Eingabe des Durationausdrucks. Falls es das Datum nicht eingegeben hat, wird das aktuelle Datum geliefert. Bei fehlerhafter Eingabe wird zur es Fehlerbehandlung aufgerufen.

Als Ergebnis wird die eingegebene Bedingung auf den Typ XML Daten umgewandet und zurückgegeben.

```
//parse a deadline-expression, if not valid throw a fault
try {
     //parse a wait expression
     XmlCalendar xCal = new XmlCalendar(waitExpression);
     this.log.debug("deadline-expression: " + waitExpression);

     //check the waitExpression, if not valid throw a fault
     if (!xCal.isSet(Calendar.HOUR_OF_DAY)|| !xCal.isSet(Calendar.MINUTE)
        ||!xCal.isSet(Calendar.SECOND)) {
        BPELFault invalidExpressionValue = new BPELFault(waitExpression);
        invalidExpressionValue.setPiid(this.piid);
        invalidExpressionValue.setSourceAIID(this.aiid);
        invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVALUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calculateFinishWai
tMode");
        throw invalidExpressionValue;
     }

     //for the case, by expression has only time no date
     Calendar now  = Calendar.getInstance();
     if (!xCal.isSet(Calendar.YEAR)) {
        xCal.set(Calendar.YEAR, now.get(Calendar.YEAR));
     }
     if(!xCal.isSet(Calendar.MOUTH)||!xCal.isSet(Calendar.DAY_OF_MONTH)){
        xCal.set(Calendar.MOUTH, now.get(Calendar.MOUTH));
        xCal.set(Calendar.DAY_OF_MOUTH, now.get(Calendar.DAY_OF_MOUTH));
        }

        aDate = new GDate(xCal.getTime());
       this.log.debug("evaluated deadline: " + aDate.getDate().toString());
```

```
  } catch (Exception e) {
        BPELFault invalidExpressionValue = new BPELFault(waitExpression,e);
        invalidExpressionValue.setPiid(this.piid);
        invalidExpressionValue.setSourceAIID(this.aiid);
        invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVALUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calculateFinishWai
tMode");
        throw invalidExpressionValue;
}
break;
```

Abbildung 3-3-3: Programmcode für Methode UNTIL

3.3.4 Websphere Application Server Scheduler

Bis jetzt bietet Websphere noch kein verfügbare zeitsteuerte Aktionen. Wie immer gab es eine Option, einen proprietären Mechanismus zu bauen. Aber am Ende wäre es nicht gut in der Laufzeit des Webspheres integriert.

Mit dem Schedulerdienst sind die Aktionen nur noch einmal auszuführen, manchmal sogart erst an einem späteren Zeitpunkt oder in regelmäßigen Abständen. Der Scheduler kann auch die Nachrichten über Task Aktivitäten empfangen.

Es gibt keine Werkzeuge, die beim Erstellen und Planen der Tasks helfen. Alle Aktionen müssen in der J2EE-Applikation erzeugt werden. Um die Dienste zu nutzen, müssen folgende Schritte folgen:

• Konfiguration des Schedulerdienstes

• Entwicklung des Scheduler-Client-Code
  Definieren der Aufgaben
  Definieren der Mitteilungen
  Definieren des Kalendars
  Scheduler und deren Aufgaben finden
  Verwaltung der geplanten Aufgaben

• Den Code in die Applikation hinzufügen

• Ausführen

Konfiguration des Schedulerdienstes

Der erste Schritt besteht darin, die Schedulerdienste und deren Voraussetzungen als Aufgabenverwaltung und Schedulerdatenbank zu konfigurieren. Dies kann mittels WebSphere Administration gemacht werden.

Definieren der Aufgaben

Eine Aufgabe wird durch die Konstruktion eines Aufgabeninformationsobjektes definiert. Dieses Objekt enthält alle Aufgabeninformationen wie Startzeiten, wiederholtes Intervall, um den Kalendar zu verwenden, Startintervall, Zustände, Dauer.

Generische Aufgabeninformationen werden durch Schnittstellen der Aufgabeninfo definiert. Zusätzlich werden zwei Schnittstellen von Aufgabeninfo erweitert:

• BeanTaskInfo

Wird verwendet, wenn das Zielobjekt eine EJB Session Beans ist.

• MessageTaskInfo

Um geplante Aufgaben zu erzeugen, die eine JMS-Nachricht zu einer Queue sendet.

Definieren des Kalendars

Schedulerdienst verwenden immer Kalendar. WebSphere Application Server hat auch einen Kalendar implementiert. Wenn kein Kalendertyp bei der Definierung der Aufgabeninfo Objekt angegeben ist, dann wird der Stardardkalendar verwendet.

Eine andere Möglichkeit ist, für den Standardkalendar einen JNDI Namen zu finden, sodass der kalendar dann als eigener Kalendar wiederverwendet werden kann.

Scheduler und deren Aufgaben finden

Der Schedulerdienst liegt offenbar in JNDI. Man kann JNDI verwenden, um den Schedulerdienst zu finden und dann die Aufgaben planen.

Verwaltung der geplanten Aufgaben

Sobald Aufgaben geplant sind, kann man sie im Schedulerdienst verwalten. Die Aufgaben können gepaust, fortgesetzt, abgebrochen und gelöscht werden.[9]

3.4 Schema der Wait Activity Datenbank

Die folgende Abbildung zeigt alle zusätzlichen Informationen über BPEL Wait Activity. Die Waitmode, in der die Wait Activity steht, stellt auch die Frist dar.

Name	Data	Description
AID	Char(8) for bit data	It contains the unique identifier which represents the <wait> activity in the database. This is the primary and foreign key.
WaitMode	SMALLINT	It contains the wait mode, represents FOR or UNTIL.
WaitExpression	VarChar(255)	It contains the wait expression, represents duration expression or deadline expression.
WaitExpressionLanguage	VarChar(255)	expressionLanguage attribute for duration expression or deadline expression.

Abbildung 3-4: Schema der WaitActivity Datenbank[7]

4 Ausführung

Bei der Ausführung werden zwei BPEL-Prozesse erzeugt:

TTProcRIR.BPEL und TTProcRAI.BPEL. TTProcRIR wird gestartet und anschließend wird TTProcRAI aufgerufen. TTProcRAI wartet eine Wait Activity für eine Minute und ruft dann TTProcRIR über eine entsprechende Korrelation auf.

Hier wird TTProcRIR.BPEL gestartet:

TTProcRIR.BPEL:

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="TTProcRIR"
         expressionLanguage="http://www.w3.org/TR/1999/REC-
xpath-19991116"
         suppressJoinFailure="yes"
         targetNamespace="http://iaas.perfTest.org/TTProcRIR"
         xmlns="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
         xmlns:bpws="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
         xmlns:tns="http://iaas.perfTest.org/wsdl/TTProcRIR/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <partnerLinks>
    <partnerLink name="TTProcRIRPL"
                 myRole="TTProcRIR"
                 partnerLinkType="tns:TTProcRIRLT"/>
    <partnerLink name="TTProcRAIPL"
                 myRole="CallbackForTTProcRAI"
                 partnerRole="TTProcRAI"
                 partnerLinkType="tns:TTProcRAILT"/>
  </partnerLinks>

  <variables>
    <variable messageType="tns:Message" name="Request"/>
  </variables>

  <correlationSets>
    <correlationSet name="correlation"
properties="tns:correlationProperty"/>
  </correlationSets>

  <flow>

    <links>
```

```
      <link name="LinkRI"/>
      <link name="LinkIR"/>
    </links>

    <receive createInstance="yes"
             name="R"
             operation="start"
             partnerLink="TTProcRIRPL"
             portType="tns:TTProcRIRPT"
             variable="Request">
      <sources>
        <source linkName="LinkRI"/>
      </sources>
    </receive>

    <invoke name="I"
            operation="startTTProcRAI"
            partnerLink="TTProcRAIPL"
            portType="tns:TTProcRAIPT"
            inputVariable="Request">
      <targets>
        <target linkName="LinkRI"/>
      </targets>
      <sources>
        <source linkName="LinkIR"/>
      </sources>
      <correlations>
        <correlation initiate ="yes" set="correlation"/>
      </correlations>
    </invoke>

    <receive name="R"
             operation="receiveResponseFromTTProcRAI"
             partnerLink="TTProcRAIPL"
             portType="tns:TTProcRAICallbackPT"
             variable="Request">
      <targets>
        <target linkName="LinkIR"/>
      </targets>
       <correlations>
        <correlation initiate ="no" set="correlation"/>
      </correlations>
    </receive>

  </flow>
</process>
```

Abbildung 4-1: Programmcode TTProcRIR.BPEL

TTProcRIR.BPEL ruft jetzt TTProcRAI.BPEL:

TTProcRAI.BPEL:

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="TTProcRAI"
         expressionLanguage="http://www.w3.org/TR/1999/REC-
xpath-19991116"
         suppressJoinFailure="yes"
         targetNamespace="http://iaas.perfTest.org/TTProcRAI"
         xmlns="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
         xmlns:bpws="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
         xmlns:tns="http://iaas.perfTest.org/wsdl/TTProcRAI/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <partnerLinks>
    <partnerLink name="TTProcRAIPL"
                 myRole="TTProcRAI"
                 partnerRole="CallbackForTTProcRAI"
                 partnerLinkType="tns:TTProcRAILT"/>
  </partnerLinks>

  <variables>
    <variable messageType="tns:Message" name="Request"/>
  </variables>

  <sequence>
    <receive createInstance="yes"
             name="receive"
             operation="startTTProcRAI"
             partnerLink="TTProcRAIPL"
             portType="tns:TTProcRAIPT"
             variable="Request">
    </receive>

    <wait name="wait">
       <for>PT1M</for>
    </wait>

    <invoke name="CallProcB"
             operation="receiveResponseFromTTProcRAI"
             partnerLink="TTProcRAIPL"
             portType="tns:TTProcRIRPT"
             inputVariable="Request">

    </invoke>
  </sequence>
</process>
```

Abbildung 4-2:Programmcode TTProcRAI.BPEL

TTProcRAI.BPEL wartet eine Wait Activity für eine Minute und dann wird TTProcRIR.BPEL wieder aufgerufen.

5 Zusammenfassung

In dieser Studienarbeit wird eine Anwendung entwickelt, um die Ausführung der Prozesse bei Bedarf für eine Periode oder bis zu einer bestimmten Uhrzeit zu halten. Dabei wurden die Methoden für beide Bedingungen verworfen und insbesondere der Code für die Methode UNTIL der alten Version neu geschrieben.

Bei Ausführung wurden zwei BPEL-Prozesse erzeugt. TTProcRIR wird gestartet und anschließend wird TTProcRAI aufgerufen. TTProcRAI wartet eine Wait Activity für eine Minute und ruft dann TTProcRIR über eine entsprechende Korrelation auf.

6 Anhang: Wait Activity Programm

```
/*
 * All Rights Reserved.
 *
 * Project: SWoM 3
 *
 * Filename: WaitActivityInstance.java
 * Package: iaas.sowm.pe.pmodel.basic
 * Created: 29.07.2010
 */

package iaas.swom.processExecution.instances.BPELConstructs;

import iaas.swom.shared.exception.BPELFault;
import iaas.swom.shared.exception.InternalErrorException;
import iaas.swom.shared.exception.UserException;
import iaas.swom.shared.exception.nav.ExitingException;
import iaas.swom.shared.exception.nav.TerminatingException;
import
iaas.swom.sharedejbs.ModelCache.activityTypes.WaitActivity;

import org.apache.xmlbeans.GDate;

import java.io.Serializable;
import java.util.Date;

import com.ibm.websphere.scheduler.Scheduler;

public class WaitActivityInstance extends ActivityInstance
implements erializable {

     /**
      * WAS Scheduler
      */

     private Scheduler scheduler;

     /**
      * Wait mode as stored in the wait activity
      */

     private short waitMode;

     /**
      * Wait expression as specified in the wait activity
      */
```

```
    private String waitExpression;

    /**
     * Expression language specified in the wait activity
     */

    private String waitExpressionLanguage;

    /**
     * Associated wait activity
     */

    private WaitActivity waitActivity;

    /**
     * Calculated date when the WAS scheduler should send back
     * the message
     */

    private Date finishedWaitDate = null;

    /**
     * Destination queue to which the WAS scheduler should
     * send the message
     */

    private String schedulerMessageDestination;

    /**
     * Connection Factory to which the WAS scheduler should
     * send the message
     */

    private String schedulerJMSConnectionFactory;

   /**
    * Calculate finishWaitDate from duration or deadline
    * expression
    *
    * @param waitMode   wait mode
    * @param waitExpression   wait expression
    * @param waitExpressionLanguage   wait expression
    * language
    * @return A Date to finish wait activity, if the Data is
    * evaluated in the past time, then return null.
    * @throws InvalidExpressionValue
    * @see InvalidExpressionValue
    * @throws ConfigurationExpression   unpredictable failure
    * @throws BPELFault
    */
```

```
     protected Date calculateFinishWaitDate (short waitMode,
String waitExpression, String waitExpressionLanguage ) throws
ConfigurationException, BPELFault {
       // only XPATH in default supported
       if ((waitExpressionLanguage != null) &&
(waitExpressionLanguage != "")) {
          throw new ConfigurationException
(waitExpressionLanguage + " is not
                        implemented yet" );
       }

       this.log.debug("calculate Finish WaitDate");
       GDate aDate = null;

       switch (waitMode) {
       case WaitModes.FOR:
            //parse a duration-expression, if not valid throw a
fault
            try {
                 //get current time
                 aDate = new GDate(Calendar.getInstance());
                 this.log.debug("current: " +
aDate.getDate().toString());

                 //parse expression
                 GDuration duration = new
GDuration(waitExpression);
                 if (!duraion.isValid()) {
                      BPELFault invalidExpressionValue = new
                     BPELFault(waitExpression);
                      invalidExpressionValue.setPiid(this.piid);
            invalidExpressionValue.setSourceAIID(this.aiid);
invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVA
LUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calcu
lateFinishWaitMode");
                 throw invalidExpressionValue;
            }
            this.log.debug("duration-expression: " +
duration.toString());

            // aDate = current time + duration
            aDate = aDate.add(duration);
            this.log.debug("current + duration: " +
aDate.getDate().toString());
} catch (Exception e) {
  BPELFault invalidExpressionValue = new
  BPELFault(waitExpression,e);
  invalidExpressionValue.setPiid(this.piid);
```

```
  invalidExpressionValue.setSourceAIID(this.aiid);
invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVA
LUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calcu
lateFinishWaitMode");
       throw invalidExpressionValue;
}
break;
case WaitModes.UNTIL:
  //parse a deadline-expression, if not valid throw a fault
  try {
       //parse a wait expression
       XmlCalendar xCal = new XmlCalendar(waitExpression);
      this.log.debug("deadline-expression: " + waitExpression);

       //check the waitExpression, if not valid throw a fault
       if (!xCal.isSet(Calendar.HOUR_OF_DAY) ||
        !xCal.isSet(Calendar.MINUTE)
        || !xCal.isSet(Calendar.SECOND)) {
            BPELFault invalidExpressionValue = new
            BPELFault(waitExpression);
            invalidExpressionValue.setPiid(this.piid);
            invalidExpressionValue.setSourceAIID(this.aiid);
invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVA
LUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calcu
lateFinishWaitMode");
       throw invalidExpressionValue;
    }

    //for the case, by expression has only time no date
    Calendar now  = Calendar.getInstance();
    if (!xCal.isSet(Calendar.YEAR)) {
       xCal.set(Calendar.YEAR, now.get(Calendar.YEAR));
    }
    if (!xCal.isSet(Calendar.MOUTH)
      | !xCal.isSet(Calendar.DAY_OF_MONTH)){
       xCal.set(Calendar.MOUTH, now.get(Calendar.MOUTH));
       xCal.set(Calendar.DAY_OF_MOUTH,
       now.get(Calendar.DAY_OF_MOUTH));
    }

    aDate = new GDate(xCal.getTime());
    this.log.debug("evaluated deadline: " +
      aDate.getDate().toString());
  } catch (Exception e) {
       BPELFault invalidExpressionValue = new
         BPELFault(waitExpression,e);
       invalidExpressionValue.setPiid(this.piid);
```

```
        invalidExpressionValue.setSourceAIID(this.aiid);
invalidExpressionValue.setQName(FaultNames.INVALIDEXPRESSIONVA
LUE);
invalidExpressionValue.setSourceIdentifier("Waitactivity.calcu
lateFinishWaitMode");
  throw invalidExpressionValue;
}
break;
default:
  String msg =
iaas.swom.shared.prop.pe.Messages.getString("unknownWaitMode");
  this.log.error(msg);
  throw new ConfigurationException(msg + " " + waitMode + " "
+ waitExpression);
} //end switch case

/*
 * if calculated aDate is earlier than now the time throw a
 * exception to inform this unpredictable behaviour
 */
if (aDate.compareToGDate(new GDate(Calendar.getInstance())) !=
1) {
  // for a past time
this.log.debug(iaas.swom.shared.prop.pe.Messages.getString("un
reasonableWaitDateTime"));
   return null;
}
return aDate.getDate();
}

/**
 * Interprets the information provided for the wait activity
 * and sends the calculated time when the activity should
 * complete to the WAS scheduler
 */

@Override
public void execute() throws ExitingException,
TerminatingException, UserException, BPELFault,
InternalErrorException {

/*
 * Obtain the basic information from the wait activity
 */
waitActivity = (WaitActivity) this.getActivity();
waitMode = waitActivity.getWaitMode();
waitExpression = waitActivity.getWaitExpression();
waitExpressionLanguage =
waitActivity.getWaitExpressionLanguage();
```

```
/*
 * Get the scheduler from the process execution context
 */
scheduler = processExecutionContext.getBaseScheduler();

/*
 * Set the navigator queue as the destination to which the WAS
 * Scheduler should send the message
 */
schedulerMessageDestination = processExecutionContext
.getSystemDeploymentDescriptor().getWebSphereOptions()
.getInvocationQueue();

/*
 * Set the connection factory which the WAS scheduler should
 * use
 */
schedulerJMSConnectionFactory = processExecutionContext
.getSystemDeploymentDescriptor().getWebSphereOptions()
.getJmsConnectionFactory();
}

/**
 * Handles the completion of the wait. There s basically
 * nothing to do, just setting the state of the activity
 * instance to state FINISHED
 */
public void completeExecution() {
     this.finished();
  }
}


/*
 * All Rights Reserved.
 *
 * Project: SWoM 3
 *
 * Filename: WaitActivity.java
 * Package: iaas.sowm.pe.pmodel.basic
 * Created: 29.07.2010
 */

package iaas.swom.sharedejbs.ModelCache.activityTypes;


import  iaas.swom.sharedejbs.ModelCache.BPELConstructs.Activity;


public class WaitActivity extends Activity {
```

```
    /**
     *
     */
    private static final long serialVersionUID = -
9073389003717567958L;

    /**
     * This attribute represents which mode the Wait Activity
     * uses. The only two possible values are FOR and UNTIL,
     * which are defined in class WaitModes.
     *
     * @see WaitModes
     */
    private short waitMode;

    /**
     * This attribute represents the expression owned by the
     * Wait Activity. If the attribute waitMode is
     * FOR the attribute waitExpression represents the
     * duration expression, otherwise waitExpression
     * represents the deadline expression. The maximal length
     * is 255 which is defined in database.
     */
    private String waitExpression;

    /**
     * This attribute represents the expressionLanguage
     * attribute of the waitExpression, which could be
     * a duration or a deadline expression as described
     * above. The maximal length is 255 which is defined in
     * database.
     */
    private String waitExpressionLanguage;

    /**
     * Set/get waitMode
     */

    public short getWaitMode() {
        return this.waitMode ;
    }

    public void setWaitMode (short waitMode) {
        this.waitMode = waitMode ;
    }


    /**
     * Set/get waitExpression
```

```
     */

    public String getWaitExpression () {
         return this.waitExpression ;
    }

    public void setWaitExpression (String waitExpression) {
         this.waitExpression = waitExpression ;
    }


    /**
     * Set/get waitExpressionLanguage
     */

    public String getWaitExpressionLanguage () {
         return this.waitExpressionLanguage ;
    }

    public void setWaitExpressionLanguage (String
waitExpressionLanguage){
        this.waitExpressionLanguage = waitExpressionLanguage ;
    }

    /**
      * Calculate finishWaitDate from duration or deadline
      * expression
      */

    protected Date calculateFinishWaitDate (short waitMode,
String waitExpression, String waitExpressionLanguage ) throws
ConfigurationException, BPELFault {
          return aDate.getDate();
    }

    /**
     * Returns the type of the activity
     */

    @Override
    public short getActivityType() {
         return ActivityTypes.WAIT;
    }

}
```

Literaturverzeichnis

[1] SWoM 3 Team

[2] SWoM Projekt
http://www.iaas.uni-stuttgart.de/forschung/projects/swom/

[3] Wikipedia WebSphere
http://de.wikipedia.org/wiki/WebSphere

[4] WebSphere Anwendungsinfrastruktur
http://www-01.ibm.com/software/de/websphere/application/app-transaction.html

[5] WebSphere V7.0 Redbook

[6] Wikipedia DB2
http://de.wikipedia.org/wiki/DB2

[7] Database Design – SWoM3 Team

[8] IBM WebSphere Version 7.0 Information Center

[9] WebSphere Application Server Enterprise V5 and Programming Model Extensions
http://www.redbooks.ibm.com/redbooks/pdfs/sg246932.pdf

Abbildungsverzeichnis

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Unterschrift: (Changsheng Qian)

Stuttgart, 31.08.2010