Generalitat de Catalunya
Centre de Telecomunicacions
i Tecnologies de la Informació

Framework Corporatiu J2EE

Servei de Planificació de Tasques

Versió 1.2

Barcelona, 13 / febrer / 2007

Històric de modificacions

Data	Autor	Comentaris	Versió
16/01/2006	Atos Origin, sae openTrends	Versió inicial del document	1.0
13/02/2007	Atos Origin, SAE	Versió 1.2 d'OpenFrame	1.2

Llegenda de Marcadors

Índex

1. Introducció

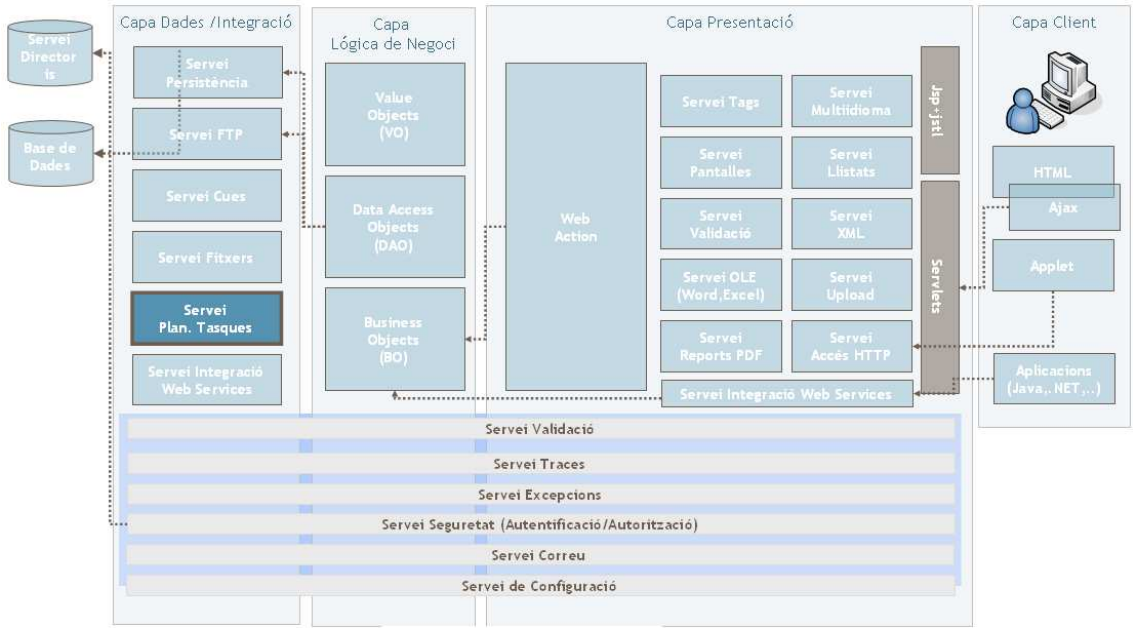
1.1. Propósit

El Servei de Planificació de Tasques d'**openFrame** permet configurar l'execució de tasques de forma diferida en els moments en els que es determini:

- en qualsevol moment del dia (precisió de milisegons)
- en alguns dies de la setmana
- en alguns dies del mes
- en alguns dies de l'any
- un número específic de repeticions
- repetidament fins a una data/instant determinat
- repetida e indefinidament
- repetidament en un determinat interval de temps

1.2. Context i Escenaris d'Ús

El Servei de Planificació de Tasques es troba dins la Capa de Dades/Integració en el context dels serveis proporcionats per openFrame.



1.3. Versions i Dependències

En el present apartat es mostren quines són les versions i dependències necessàries per fer ús del Servei.

Dins la llista de dependències es mostren diferenciades:

- Dependències bàsiques. Llibreries necessàries per fer ús del servei
- Dependències adicionals. Aquestes dependències són necessàries per poder fer ús de característiques concretes del servei o l'ús dels tests unitaris proporcionats amb el servei

1.3.1. Versions

No s'han produït canvis respecte la versió 1.0.

1.3.2. Dependències Bàsiques

Nom	Tipus	Versió	Descripció
commons-beanutils-core	jar	1.7.0	
commons-logging	jar	1.0.4	http://jakarta.apache.org/commons
log4j	jar	1.2.12	
openFrame-core	jar	1.0	
openFrame-services-exceptions	jar	1.0	
openFrame-services-logging	jar	1.0	
quartz	jar	1.4.0	
spring	jar	1.2.5	

1.3.3. Dependències Adicionals

- Proves Unitàries del Servei

Nom	Tipus	Versió	Descripció
easymock	jar	1.1	
junit	jar	3.8.1	

1.4. A qui va dirigit

Aquest document va dirigit als següents perfils:

- Programador. Per conéixer l'ús del servei
- Arquitecte. Per conéixer quins són els components i la configuració del servei

- Administrador. Per conéixer cóm configurar el servei en cadascun dels entorns en cas de necessitat

1.5. Documents i Fonts de Referència

[1] Quartz http://www.opensymphony.com/quartz.

1.6. Glossari

2. Descripció Detallada

2.1. Arquitectura i Components

El Servei de Planificació de Tasques de openFrame ofereix interfícies d'ús que s'abstreuen de la implementació escollida per millor mantenibilitat en futures versions i futures implementacions alternatives.

En l'actualitat, openFrame proporciona una implementació basada en Quartz (projecte open source desenvolupat per PartNET).

Els components podem classificar-los en:

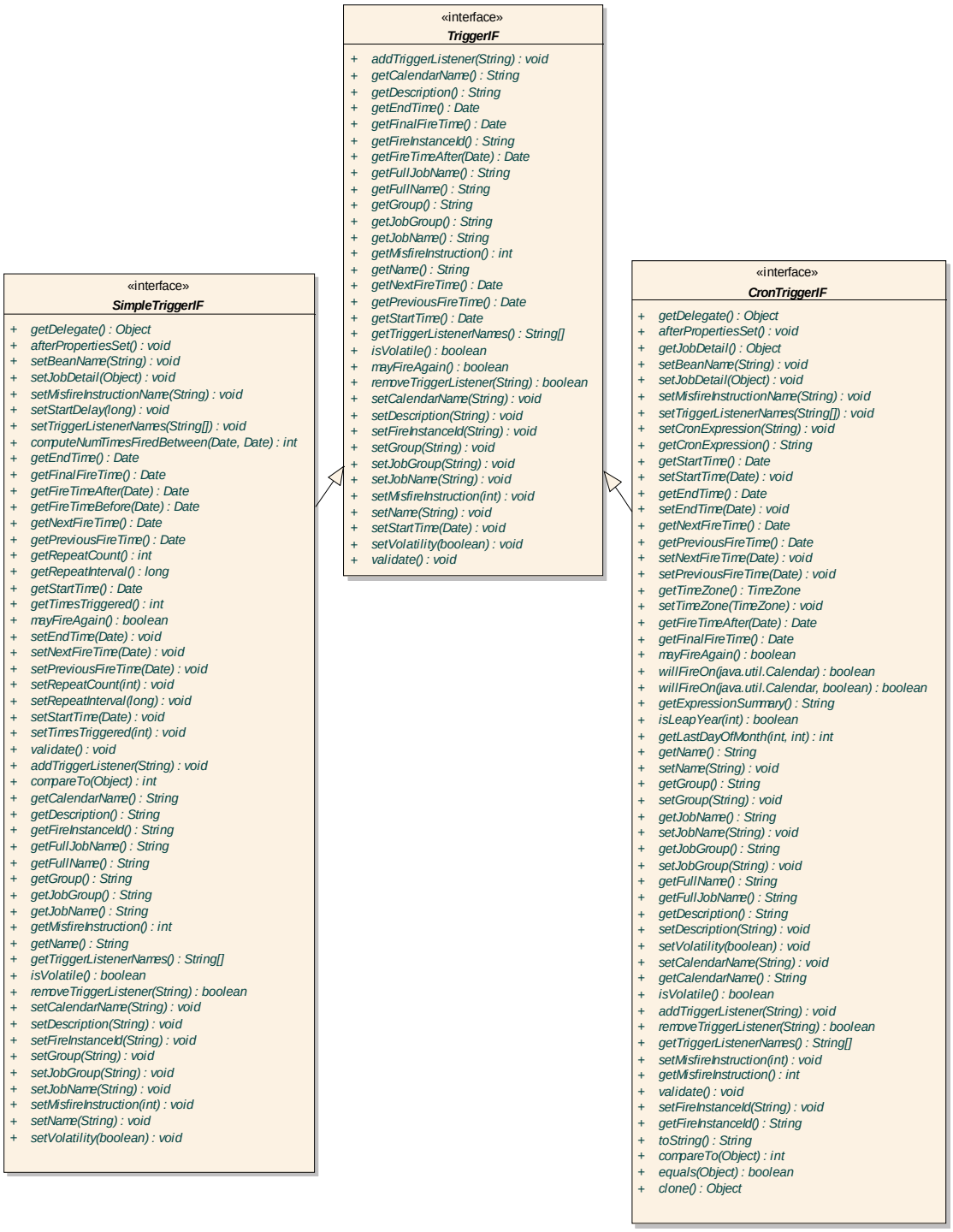
- Interfícies i Components Genérics. Interfícies del servei i components d'ús general amb independència de la implementació escollida.
- Implementació de les interfícies basada en Quartz

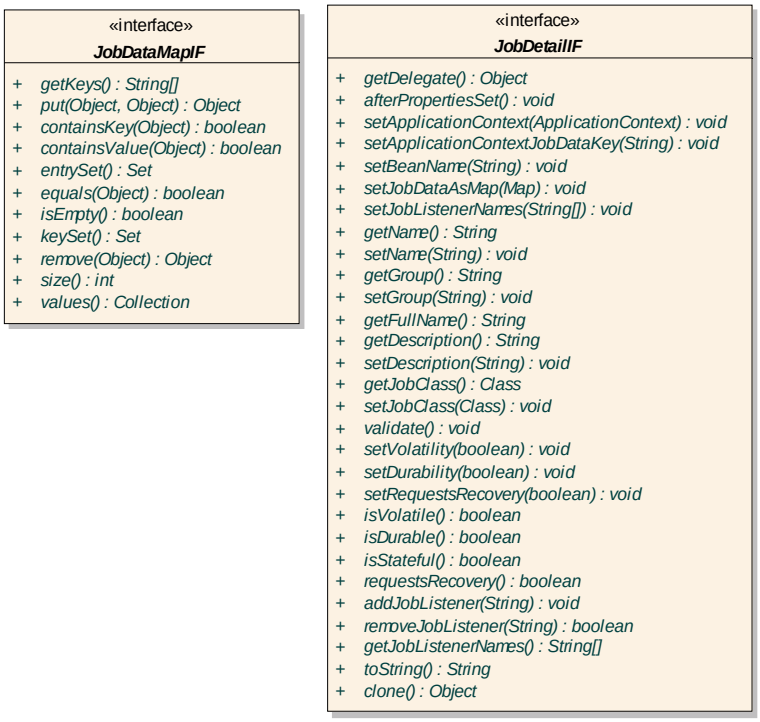
2.1.1. Interfícies i Components Genérics

El servei defineix les següents interfícies:

Component	Package	Descripció
TriggerIF	net.opentrends.openframe.services.scheduler	Interfície de definició del trigger que llençarà la tasca.
SimpleTriggerIF	net.opentrends.openframe.services.scheduler	Interfície per definir triggers que executaran la tasca en un moment determinat i opcionalment especificant l'interval de repetició
CronTriggerIF	net.opentrends.openframe.services.scheduler	Interfície per definir triggers amb execució de la tasca segons expressió de cron
JobDetailIF	net.opentrends.openframe.services.scheduler	Interfície per definir la informació de la tasca diferida (classe, grup en el que es troba, etc.)
JobDataMapIF	net.opentrends.openframe.services.scheduler	Interfície que permet definir quines dades s'han de passar a la tasca en el moment de la seva execució
SchedulerFactoryBeanIF	net.opentrends.openframe.services.scheduler	Interfície factoria que permet registrar els llençaments de tasques (triggers) i la informació que s'ha de passar a les tasques en el moment d'iniciar el Servei de Planificació.
MethodInvokingJobDetailFactoryBeanIF	net.opentrends.openframe.services.scheduler	Interfície per exposar un JobDetailIF que permeti delegar l'execució a un mètode concret i així evitar crear un Job únicament per invocar un mètode d'un

Component	Package	Descripció
		servei

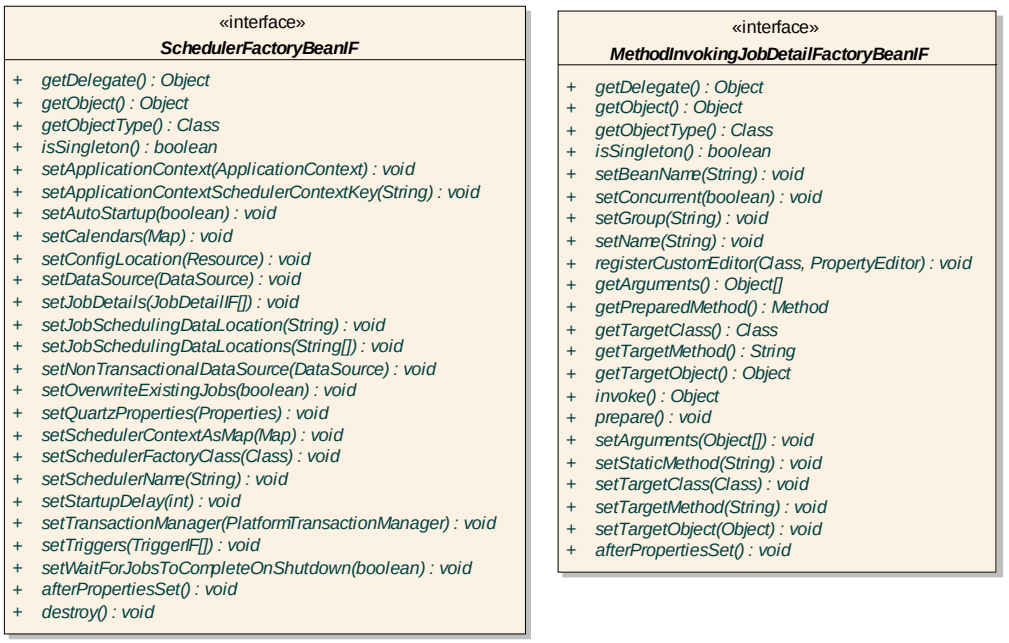




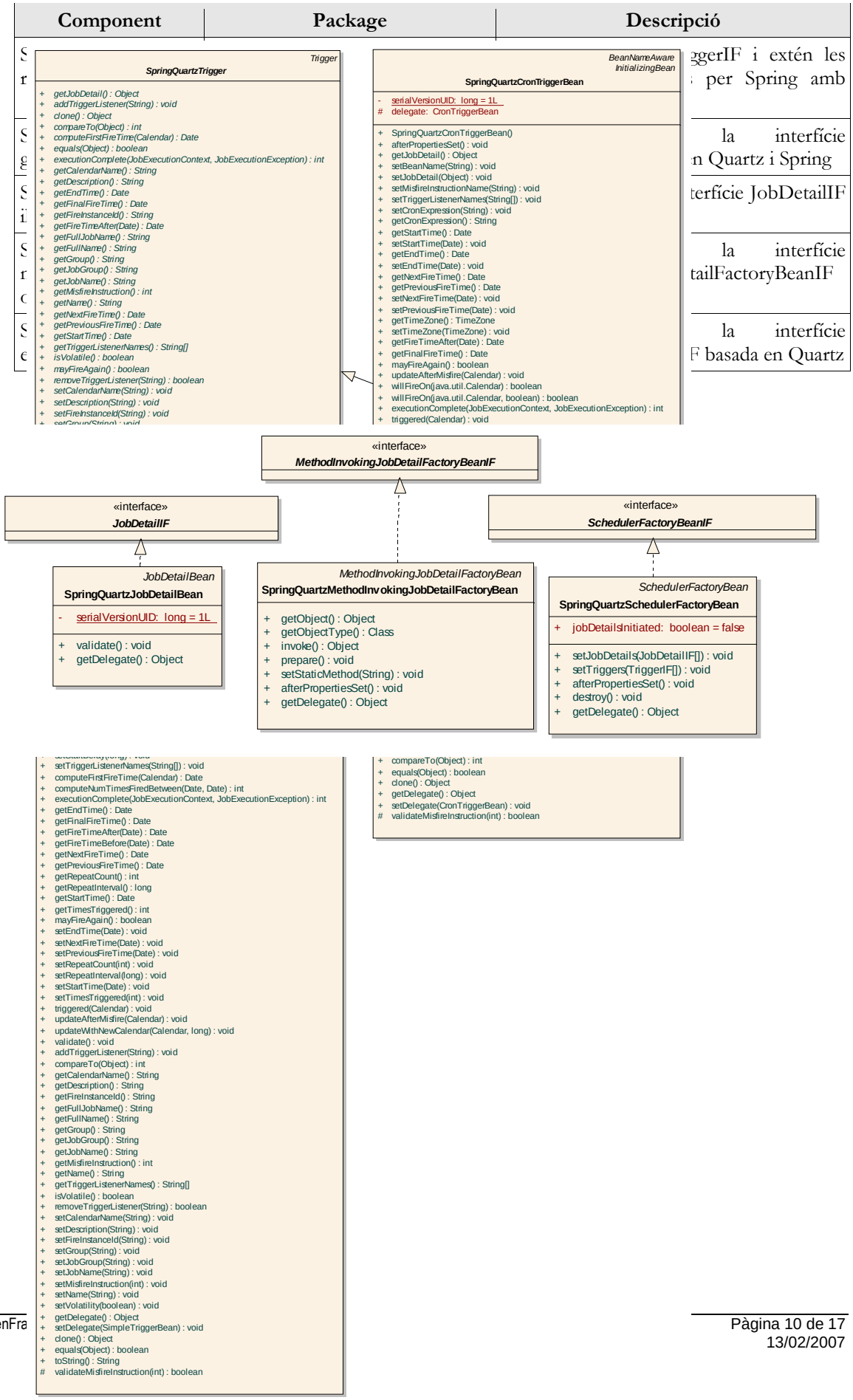
2.1.2. Implementació basada en Quartz

El servei defineix les següents interfícies:

Component	Package	Descripció
SpringQuartzTrigger	net.opentrends.openframe.services.scheduler.impl.quartz	Implementació de trigger basada en Quartz



Component	Package	Descripció
S r		ggerIF i extén les per Spring amb
S g		la interfície n Quartz i Spring
S i		terfície JobDetailIF
S r c		la interfície tailFactoryBeanIF
S e		la interfície F basada en Quartz



2.2. Instal·lació i Configuració

2.2.1. Instal·lació

La instal·lació del servei requereix de la utilització de la llibreria 'openFrame-services-scheduler' i les dependències indicades a l'apartat 'Introducció-Versions i Dependències'.

2.2.2. Configuració

Fitxer de configuració: openFrame-services-scheduler.xml

Ubicació proposada: <PROJECT_ROOT>/src/main/resources/spring

2.2.3. Configuració

La configuració del Servei de Planificació de tasques implica 3 pasos:

1) Definir les tasques que volem executar de forma diferida
2) Definir els triggers que defineixen en quin moment s'executaran les tasques
3) Definir la factoria per executar les tasques amb els triggers

Definició de les tasques

Per a definir una tasca cal definir 2 parts:

- Tasca

  La tasca és simplement la referència a la classe que conté el mètode que volem executar de forma diferida.

  Exemple:

```
<!-- TASKS DEFINITIONS -->
<bean id="taskWriteLog"
class="net.opentrends.openframe.samples.jpetstore.scheduler.TaskWriteLog"/>
```

  La classe tasca és un POJO que no ha d'implementar cap interfície en concret ni extendre cap classe.

- Detall de la tasca

  Per definir els detalls de la tasca per tal que pugui ser executada de forma diferida podem fer ús de 2 classes:

a) 'SpringQuartzMethodInvokingJobDetailFactoryBean'. Amb aquesta podrem fer referència al mètode concret a executar. Es permet definir les següents propietats:

Propietat	Requerit	Descripció
targetObject	Sí	Referència a la classe que conté el mètode a executar de forma diferida
targetMethod	Sí	Referència al mètode de la classe que s'executarà

arguments	No	Llista d'arguments a passar al mètode. Podem fer ús dins la llista de valors directes (amb <value>) o referències (amb <ref bean>). Aquesta llista ha de ser creada en el mateix ordre que la llista d'arguments definida al mètode.
		
		Exemple:
		
		`<property name="arguments">`
		`  <list>`
		`  <value>5</value>`
		`  <ref bean="logService"/>`
		`  </list>`
		`</property>`

concurrent	No	Indicar si es poden executar de forma concurrent vàries instàncies de la tasca Recomanació: Usar 'false' Per defecte: true

Exemple:

```
<bean id="taskWriteDetail"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuart
zMethodInvokingJobDetailFactoryBean">
<property name="targetObject" ref="taskWriteLog"/>
<property name="targetMethod" value="writeLog"/>
<property name="concurrent" value="false"/>
</bean>
```

b) 'SpringQuartzJobDetailBean'. En aquest cas la classe haurà d'extendre la classe 'SpringQuartzJobBean'.

Propietat	Requerit	Descripció
jobClass	Sí	Referència a la classe que extén la classe 'SpringQuartzJobBean'
jobDataAsMap	Sí	Map de dades que passarem a la tasca. Cadascun dels keys ha de tenir un 'set' corresponent a la classe

Exemple:

```
<bean name="exampleJob"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzJobDetailBean">
<property name="jobClass" value="example.ExampleJob"/>
<property name="jobDataAsMap">
        <map>
                <entry key="timeout" value="5"/>
        </map>
</property>
</bean>
```

En aquest cas, la tasca seria definida així:

```
package example;

// import section
...

public class ExampleJob extends SpringQuartzJobBean {

 private int timeout;

 /**
  * Setter called after the ExampleJob is instantiated
  * with the value from the JobDetailBean (5)
  */
 public void setTimeout(int timeout) {
   this.timeout = timeout;
 }

 protected void executeInternal(JobExecutionContext ctx)
 throws SchedulerServiceException {
    // do the actual work
 }
}
```

Com veiem, la classe defineix el mètode 'executeInternal' on realitzarà el procediment de la tasca.

Definició dels triggers

Mitjançant els triggers configurarem en quin moment s'ha d'executar la tasca.

openFrame permet la utilització de 2 classes:

a) 'SpringQuartzSimpleTriggerBean'

Propietat	Requerit	Descripció
jobDetail	Sí	Referència a la tasca definida en el pas anterior
endTime	No	Hora de finalització del trigger
startDelay	No	Temps (en mil·lisegons) que ha de transcórrer abans d'executar-se el primer trigger
startTime	Sí	Hora d'inicia de la tasca
repeatInterval	No	Temps (en mil·lisegons) que ha de transcórrer abans d'executar-se el següent trigger

Exemple:

```
<!-- TRIGGERS DEFINITIONS -->
<bean id="taskWriteTrigger"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzSimpleTriggerBean">
    <!-- see the example of method invoking job above -->
    <property name="jobDetail" ref="taskWriteDetail"/>
    <!-- 10 seconds -->
    <property name="startDelay" value="10000"/>
    <!-- repeat every 20 seconds -->
    <property name="repeatInterval" value="20000"/>
    <property name="concurrent" value="false"/>
</bean>
```

b) 'SpringQuartzCronTriggerBean'.

Propietat	Requerit	Descripció
jobDetail	Sí	Referència a la tasca definida en el pas anterior
cronExpression	Sí	Expressió de tipus cron de Unix. Per a més informació consultar http://wiki.opensymphony.com/display/QRTZ1/CronTriggers+Tutorial

Exemple:

```
    <bean id="taskWriteTrigger2"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzCronTriggerBean">
            <!-- see the example of method invoking job above -->
            <property name="jobDetail" ref="taskWriteDetail"/>
            <!-- run every morning at 6 AM -->
            <property name="cronExpression" value="0 0 6 * * ?"/>
            <property name="concurrent" value="false"/>
    </bean>

</beans>
```

Definició de la factoria dels triggers

Per últim, definirem un bean de la classe 'net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzSchedulerFactoryBean' on definirem les propietats:

Propietat	Requerit	Descripció
triggers	Sí	Llista de referències als triggers anteriorment definits

Exemple:

```
<!-- SCHEDULER FACTORY BEAN DEFINITION -->
<bean class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzSchedulerFactoryBean">
        <property name="triggers">
                <list>
                        <ref bean="taskWriteTrigger"/>
                        <ref bean="taskWriteTrigger2"/>
                </list>
        </property>
</bean>
```

2.3. Utilització del Servei

La instanciació, la preparació i la petició del servei es fa de manera transparent, de tal manera que el servei s'activa en el moment en que el "*Scheduler Factory Bean*" conté algun *Trigger* amb alguna tasca associada (tal i com he definit a la configuració).

En el servei, es defineixen a la configuració les tasques (*Jobs)* i els disparadors (Triggers) que llençaran les tasques. Les tasques, en general no han d'extendre o implementar cap interfície, però sí serà necessari en cas de definir 'SpringQuartzJobDetailBean'.

Per tant, la utilització del servei es realitzar pràcticament en la seva totalitat mitjançant la seva configuració.

2.4. Eines de Suport

2.5. Integració amb Altres Serveis

2.6. Preguntes Freqüènts

3. Exemples

Com exemple d'utilització del servei de Planificació de Tasques s'inclou un exemple en el que 2 Triggers invoquen un mètode d'una classe Java que genera un log (mitjançant el servei de traces):

En el codi mostrat a baix, la tasca definida no implementa cap classe. Defineix únicament el mètode 'writeLog' on realitza la traça

```
package net.opentrends.tutorial.web.temp;

import org.apache.log4j.Logger;

public class TaskWriteLog {


     public TaskWriteLog() {
   }

   public void writeLog(LoggingService logService) {
                if (logService!=null) {
     logService.getLog(this.getClass()).debug("Doing log at:" +                    System.currentTimeMillis());
                }


   }
}
```

Un exemple de la configuració de la seva execució diferida és la següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

    <!-- SCHEDULER FACTORY BEAN DEFINITION -->
    <bean class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzSchedulerFactoryBean">
                <property name="triggers">
                        <list>
                                <ref bean="taskWriteTrigger"/>
                        </list>
                </property>
    </bean>

    <!-- TASKS DEFINITIONS -->

    <bean id="taskWriteLog" class="net.opentrends.tutorial.web.temp.TaskWriteLog"/>
    <bean id="taskWriteDetail"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzMethodInvokingJobDetailFactoryBean">
                <property name="targetObject" ref="taskWriteLog"/>
                <property name="targetMethod" value="writeLog"/>
                <property  name="arguments">
                        <list>
                                <ref  bean="logService"/>
                        </list>
```

```
                <property name="concurrent" value="false"/>
    </bean>

    <!-- TRIGGERS DEFINITIONS -->
    <bean id="taskWriteTrigger"
class="net.opentrends.openframe.services.scheduler.impl.quartz.SpringQuartzSimpleTriggerBean">
                <!-- see the example of method invoking job above -->
                <property name="jobDetail" ref="taskWriteDetail"/>
                <!-- 10 seconds -->
                <property name="startDelay" value="10000"/>
                <!-- repeat every 20 seconds -->
                <property name="repeatInterval" value="20000"/>
    </bean>

</beans>
```

Nota

En el cas que 2 *Triggers* utilitzin la mateixa tasca, es podria donar el cas de que abans de que el primer *Trigger* finalitzi, ja es comenci a executar el segon. Per evitar aquesta situació s'afegirà en la definició de la tasca la propietat *concurrent* amb el valor "false"