generix Collaborative Entreprise

GMO Nouveaux timers

Type diffusion		**Auteur**	*Roa*
Révision *27*		**Date de création**	*09/03/2012*

Ce GMO a pour but d'expliquer le fonctionnement des timers dans le socle technique 8.0.

Cette refonte a été réalisée avec l'objectif de faciliter la mise en place de mesures objectives sur l'applicatif. On retiendra principalement que cela autorise à écrire certains timers en base de données, ce qui améliore grandement l'exploitation et le recoupement des informations, puisque de simples requêtes à la base de données permettent de connaitre les temps de réponse des pages HTML (ou autres).

Société	:	***GENERIX Group***	Titre	:	*Projet : socle technique 8.0*
Dernière modification	:	*Administrateur*	Sujet	:	*GMO Nouveaux timers*
Temps de modification	:	*6784*	Résumé	:	
Nom du fichier		*GMO Nouveaux timers TF80.doc*			

Liste de diffusion

Siège Social et Agence Nord
Immeuble le Verdi
6 rue du Moulin de Lezennes
BP 10215
59654 VILLENEUVE D'ASCQ Cedex
Tél. : +33 (0)3 20 41 48 00
Fax. : +33 (0)3 20 41 48 09
Service Technique :
Tél. : +33 (0)3 20 41 48 07
Fax. : +33 (0)3 20 41 48 08

Agence Région Parisienne

69/71 rue Beaubourg
75003 PARIS

Tél. : +33 (0)1 47 49 36 66
Fax. : +33 (0)1 47 51 22 17

Table des matières

1 Principe de base

Historiquement les timers sont tracés sur la base du framework Apache LOG4J, les explications sur ce sujet sont décrites précisément dans le Release Note Technique de la GCE155.

Il s'avère que l'exploitation des timers n'est pas très complexe, mais nécessite quelques connaissances de base. On propose donc dorénavant une agrégation des timers principaux afin de disposer le plus rapidement et le plus opérationnellement possible des informations relatives à la performance de chaque page GCE générée.

Une possibilité équivalente existe sur les WebServices (indiqués **WS** dans le reste du document).

L'exploitation du MDC (basé sur une ThreadLocalMap) reste d'actualité, c'est le plus simple d'utilisation que l'on puisse trouver et le plus performant aussi. Il est important de noter que déclencher des timers ne doit rien coûter aux performances applicatives, et donc ne doivent pas grever les temps utilisateurs.

La grande nouveauté réside dans la possibilité d'écrire directement les timers dans 2 nouvelles tables dédiées :

- TIMER_REQUEST : requête http, logger requestTimer en « mode LOG4J »
- TIMER_WS : exécution d'un WS unitaire

Important : l'utilisation de la publication en base de données n'est pas une obligation mais ce sera la nouvelle valeur par défaut. Il est conseillé de laisser actifs ces timers en production afin d'avoir un suivi de l'activité de ses utilisateurs. Cela autorise donc une « auto-surveillance » de son environnement de production.

Il conviendra de mettre en place un système de purge périodique des tables concernées, la volumétrie peut rapidement devenir conséquente. Qui plus est ça ne sert pas forcément à grand-chose de conserver des données dont nous n'avons – théoriquement – qu'un usage de surveillance journalier. Ce système d'auto-purge est paramétrable depuis gce.properties en fréquence de passage, ainsi que durée de conservation des temps. Par défaut, il est actif avec le TF 8.0.

1.1 Description TIMER_REQUEST

Tableau 1		
Nom de champ	**Type**	
CODSOC	NUMBER(38)	Toujours 0
DATMOD	VARCHAR2(8)	Date actuelle
UTIMOD	VARCHAR2(8)	Toujours MONITOR
INDEX_SEQ	NUMBER(38)	Numéro unique de transaction dans la JVM
TIMESTAMP	VARCHAR2(9)	Heure précise de la prise du chrono
UUID	VARCHAR2(100)	UUID utilisateur
JVM_ID	VARCHAR2(128)	ID unique de la JVM
UTICOD	VARCHAR2(8)	Utilisateur de la transaction
ENTITY	NUMBER(38)	Entité de la transaction
REQUEST_URI	VARCHAR2(256)	URI
REQUEST_URL	VARCHAR2(4000)	URL correspondant au chrono
REQUEST_CINEMATIC	VARCHAR2(256)	Cinématique correspondante
HTTP_METHOD	VARCHAR2(10)	Méthode http (GET ou POST)
BV_NAME	VARCHAR2(256)	Nom de la BusinessView
CODPEV	VARCHAR2(8)	Cible de paramétrage
ELAPSE_TIME	NUMBER	Temps global de la requête http
UNIT	VARCHAR2(2)	Unité de mesure (ms en standard)
PRECISION	NUMBER	Précision de la mesure
ELAPSE_TIME_PROCESS	NUMBER	Temps d'exécution de Processus
ELAPSE_TIME_BUILD	NUMBER	Temps de construction de la BV
ELAPSE_TIME_FLUSH_OUT	NUMBER	Temps de flush de la servlet
ELAPSE_TIME_DISCONNECT	NUMBER	Temps de déconnexion de la BDD (suivant paramétrage de deconnectFrequency)
ELAPSE_TIME_RENDER_XMLPRES	NUMBER	Temps pour générer le flux de présentation
ELAPSE_TIME_RENDER	NUMBER	Temps global du rendu
ELAPSE_TIME_TRANSFORMATION	NUMBER	Temps de la transformation XSLT
ELAPSE_TIME_EXECUTEQUERY	NUMBER	Temps passé à réaliser des requêtes SQL
ELAPSE_TIME_DML	NUMBER	Temps passé à réaliser des requêtes SQL « lockantes »
ELAPSE_TIME_XMLRPC	NUMBER	Temps passé à exécuter des API C via XMLRPC
HTMLPRES_SIZE	NUMBER	Taille du flux de présentation en nombre de caractères
COUNT_VO_EXECUTEQUERY	NUMBER	Nombre de requêtes SQL exécutées

1.2 Description TIMER_WS

Tableau 2		
Nom de champ	**Type**	
CODSOC	NUMBER(38)	Toujours 0
DATMOD	VARCHAR2(8)	Date actuelle
UTIMOD	VARCHAR2(8)	Toujours MONITOR
INDEX_SEQ	NUMBER(38)	Numéro unique de transaction dans la JVM
TIMESTAMP	VARCHAR2(9)	Heure précise de la prise du chrono
UUID	VARCHAR2(100)	UUID utilisateur
JVM_ID	VARCHAR2(128)	ID unique de la JVM
UTICOD	VARCHAR2(8)	Utilisateur de la transaction
ENTITY	NUMBER(38)	Entité de la transaction
CODPEV	VARCHAR2(8)	Cible de paramétrage
WS_SERVICE	VARCHAR2(64)	Nom du WS
WS_OPERATION	VARCHAR2(64)	Méthode du WS
ELAPSE_TIME	NUMBER	Temps global de la requête http
UNIT	VARCHAR2(2)	Unité de mesure (ms en standard)
PRECISION	NUMBER	Précision de la mesure
ELAPSE_TIME_EXECUTEQUERY	NUMBER	Temps passé à réaliser des requêtes SQL
ELAPSE_TIME_DML	NUMBER	Temps passé à réaliser des requêtes SQL « lockantes »

2 Utilisation de « butterfly »

Il s'agit d'un projet initié en parallèle du développement du socle technique. Il vise à proposer un certain nombre d'APIs facilitatrices et communes à nombre de projets java.

Le cœur des nouveaux timers est donc réalisé dans ce projet, le socle technique n'en faisant qu'une implémentation concrète liée à son propre mode de fonctionnement.

Butterfly propose en particulier :

- Une couche basée sur des « generics » Java afin de déclarer ses définitions de timer comme on le souhaite (énumération Java dans notre cas)
- La gestion des unités de temps
- La création de chronos pour mesurer
- Une couche abstraite – donc à implémenter – afin de publier les résultats de timers. Cette notion de publication permet de s'affranchir du support de présentation des timers. On pourrait aussi utiliser ce principe pour publier des chiffres agrégés dans JMX par exemple.

On notera que Butterfly dispose de son propre releasing, le socle technique se base sur une version donnée. Extrait du MANIFEST :

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 22.0-b10 (Oracle Corporation)
Implementation-Title: Generix ) Collaborative Entreprise
Implementation-Vendor: GENERIX Group
Implementation-Version: 32
GxCommon-Version: 1.0
J2EE-Version: J2EE1.4
JDK-Compliance: 1.5
Created-Date-Time: 2012-02-16 13:42:45
Main-Class: com.generixgroup.PackageInfo
Class-Path: log4j-1.2.15.jar
```

3 Définition des timers

3.1 gce.properties

La déclaration des nouveaux timers a été complètement revue et simplifiée :

```
<timer_def>
	<timer name="ASSOCIATION_LINE_FIELD" active="false" unit="ms" comment="Mesure
le temps de génération du flux de pres pour chaque attribut de type ViewLink
BC4J."/>
	<timer name="ACTION" active="true" unit="ms" comment="Temps d'exécution de
l'action"/>
	<timer name="API" active="true" unit="ms" comment="Temps d'exécution de
l'API."/>
	<timer name="AUTHENTICATION" active="true" unit="ms" comment="Temps
d'exécution du processus d'authentification."/>
	<timer name="BETWEEN" active="true" unit="ms" comment="Chrono intermédiaire,
mesure le temps entre chaque chrono."/>
	<timer name="BUILD_AM" active="true" unit="ms" comment="Construction d'un
ApplicationModule"/>
	<timer name="BUILD_BV" active="true" unit="ms" comment="Construction de la
BusinessView"/>
	<timer name="BUILD_VIEW" active="true" unit="ms" comment="Construction d'une
View dans la ViewStruct."/>
	<timer name="BUILD_VO" active="true" unit="ms" comment="Construction d'une
ViewObject"/>
	<timer name="BUSINESSFLOW" active="true" unit="ms" precision="0.5"
comment="Mesure le temps d'exécution des BusinessFlow. On ne prend rien en compte
en deça de 0.5ms"/>
	<timer name="DATA" active="true" unit="ms" precision="0.0" comment="Mesure le
temps d'exécution des requêtes SQL de tous types DML"/>
	<timer name="DISCONNECT" active="true" unit="ms" precision="0.0"
comment="Mesure le temps de déconnexion à la base. Comprend le reset sur les
RowSet. Ce timer peut-être influent suivant les environnements (deconnectFrequency
en particulier)."/>
	<timer name="EXECUTE_QUERY" active="true" unit="ms" precision="0.0"
comment="Execute query"/>
	<timer name="FLUSH_OUT" active="true" unit="ms" comment="Temps de mise à jour
du flux de sortie de la Servlet."/>
	<timer name="METHOD_XMLRPC" active="true" unit="ms" comment="Temps
d'exécution des invocations XMLRpc"/>
	<timer name="PROCESSUS_LISTENER" active="false" unit="ms" comment="Temps
d'exécution des différents listener de Processus."/>
	<timer name="RENDER" active="true" unit="ms" comment="Lecture + temps
d'application de la feuille de style pour le rendu HTML."/>
	<timer name="RENDER_XMLPRES" active="true" unit="ms" comment="Temps complet
de génération du flux XML."/>
	<timer name="REQUEST" active="true" unit="ms" publisher="rdbms"
comment="Temps d'éxécution de la requête pur, sans les traitements annexes de
Servlet."/>
	<timer name="ROW_DOCUMENTLINE_FIELD" active="false" unit="ms" comment="Mesure
le temps de génération du flux de pres pour chaque attribut simple d'un Row
BC4J."/>
	<timer name="TRANSACTION" active="true" unit="ms" comment="Temps d'exécution
des différentes étapes sur le transaction."/>
	<timer name="VIEW" active="true" unit="ms" comment="Temps de construction du
flux de présentation d'une View dans la ViewStruct. Provoque aussi le déclenchement
des RowfieldTimer"/>
	<timer name="WS_SERVER" active="false" unit="ms" comment="Temps d'exécution
d'un webservice."/>
	<timer name="XSLT" active="true" unit="ms" comment="Temps d'application de la
feuille de style pour le rendu HTML."/>
</timer_def>
```

Dorénavant, le nom du timer est en majuscule et c'est un nom logique qui correspond directement au nom traité par le socle technique (cf la définition même de l'énumération TIMERDEF).

4 attributs sont importants (en plus du nom évidemment) :

- **@active** : spécifie si le timer est actif au global de la JVM concernée, ceci indépendamment de l'utilisateur
- **@unit** : l'unité de temps, la liste est finie (s, ms, us et ns)
- **@precision** : spécifie le temps à partir duquel le chrono sera considéré comme significatif et publié
- **@publisher** : par défaut, c'est log4j si rien n'est précisé, mais pour REQUEST on peut spécifier la valeur « *rdbms* » si on souhaite une publication dans la table TIMER_REQUEST. Il est possible à ce sujet de créer ses propres *publisher* (donc il faut faire un peu de java ☺). Dans le standard, une classe spécifique au traçage des performances des WS est définie.

A titre d'exemple, du code java « exemple » est fourni dans ce document afin de pouvoir créer ses propres *publisher* sur site.

3.1.1 Paramétrage dans gce.properties

L'activation de la publication en base de données est pilotée dans gce.properties. Elle est globale, quel que soit le nombre de timers publiés en base de données : cela active simplement la fonctionnalité.

```
<RDBMSPublishing active="true">
      <delay value="10"/>
</RDBMSPublishing>
```

L'alimentation des tables concernées se fait en mode différé, dans un Thread à part afin de ne pas polluer les performances de l'utilisateur. Le délai correspond à la fréquence (en seconde) à laquelle le Thread se déclenche. A ce moment, une connexion base est créée le temps du traitement, puis directement relâchée. Suivant le paramétrage de *deconnectFrequency* (usage des Data-Sources poolées ou pas), la connexion peut être récupérée du pool de connexions. C'est le mode conseillé à ce jour car il permet grandement de limiter le nombre de connexions à la base de données.

Ce mode de fonctionnement va de pair avec le monitoring dans gce.properties :

```
<monitoring active="true">
      <delay value="10"/>
</monitoring>
```

C'est le scheduler (scrutateur de monitoring) qui va déclencher le travail de mise à jour des timers en base de données. En conséquence de quoi, le délai fourni dans cette dernière section de gce.properties doit être <= à cette de RDBMSPublishing.

Il est donc fortement conseillé de conserver les valeurs paramétrées en standard.

3.1.2 Connexion à la base de données

Comme évoqué, la connexion base (suivant le paramétrage : DataSource ou connexion directe JDBC) peut être créée au besoin, ceci de manière systématique. Exemple : il y 20 JVM qui travaillent sur la même base de données, donc si le traitement de mise à jour des timers en base se déclenchent au même moment, on peut avoir ponctuellement besoin de 20 connections BDD supplémentaires ! C'est la raison pour laquelle il est fortement conseillé d'avoir au préalable définit des DataSources poolées !
On y gagne sur tous les tableaux :

- limitation du nombre de connexions en base de données (donc gain mémoire sur le serveur BDD)
- temps de création (et coût) de cette connexion car on un très fort taux de réutilisation des connexions d'une JVM.

3.1.3 Système d'auto-purge

Il est défini en standard dans gce.properties de la manière suivante :

```
<autoPurge active="true">
      <delay value="14400" comment="purge toutes les 4h"/>
      <day value="2" comment="suppression de tous les éléments datant de plus de 2
jours"/>
</autoPurge>
```

Un job se déclenche en fonction de la fréquence défini et supprime tous les enregistrements des tables TIMER_REQUEST et WS_REQUEST en conséquence. Ce traitement est multi-JVM, il n'y a pas à s'en préoccuper.

Ces tables devraient donc se trouver à « volume » constant sur une production, il conviendra de décider sur site s'il est nécessaire d'ailleurs de calculer des statistiques ou pas (table + indexes).

Remarque : ce traitement est aussi utilisé pour purger certaines autres tables, notamment UT_UUID, UT_DMA et UT_PRG.

3.2 Définition des timers dans le socle technique

Plutôt qu'un long discours, voici la définition (énumération Java) que l'on trouve au niveau du code source :

```
public enum TIMERDEF
{
	/**
	 * Timer d'actions, plus vraiment utile depuis GCE155 car tout est délégué à
la génération du flux de présentation
	 */
	ACTION("actionTimer"),
	/**
	 * Pas un timer, mais permet de tracer l'activité des utilisateurs
	 */
	ACTIVITE("activite"),
	/**
	 * Temps pour la phase d'authentification, surtout utile en cas d'authen
déléguée (LDAP/SSO)
	 */
	AUTHENTICATION("authenticationTimer"),
	/**
	 * Timer pour les APIs
	 */
	API("APITimer"),
	/**
	 * Timer pour les ViewLink Oracle (fichiers de conf et potentiellement
utilisés pas la couche métier)
	 */
	ASSOCIATION_LINE_FIELD("associationRowFieldTimer"),
	/**
	 * Timer entre les timers
	 */
	BETWEEN("betweenLog4JTimer"),
	/**
	 * Temps de construction d'un ApplicationModule (non significatif)
	 */
	BUILD_AM("buildApplicationModuleTimer"),
	/**
	 * Temps de construction d'une BusinessView
	 */
	BUILD_BV("buildBusinessViewTimer"),
	/**
	 * Temps de construction d'une View d'une BusinessView
	 */
	BUILD_VIEW("buildViewTimer"),
	/**
	 * Temps de construction d'une ViewObject (non significatif)
	 */
	BUILD_VO("buildViewObjectTimer"),
	/**
	 * Timer mesurant les BusinessFlow
	 */
	BUSINESSFLOW("businessFlowProviderTimer"),
	/**
	 * Timer mesurant les requêtes SQL, en et hors base de données
	 */
	DATA("data"),
	/**
	 * Temps de déconnection à la base de données, utile lorsqu'on est en mode
deconnectFrequency=request et Data-Source poolée
	 */
	DISCONNECT("disconnect"),
	/**
	 * Temps passer à exécuter executeQuery, s'approche fortement du timer data
```

```
         */
        EXECUTE_QUERY("executeQueryTimer"),
        /**
         * Temps de flush de la servlet pour envoyer le message au travers du
PrintWriter
         */
        FLUSH_OUT("servletFlushOutTimer"),
        /**
         * Temps mis pour exécuter XMLRpc
         */
        METHOD_XMLRPC("methodXmlRpcTimer"),
        /**
         * Marshalling des APIs
         */
        MARSHALL("MarshallTimer"),
        /**
         * (non significatif)
         */
        PROCESSUS_LISTENER("processusListenerTimer"),
        /**
         * Comprend la phase de transformation + lecture des feuilles de style
         */
        RENDER("renderOutputTimer"),
        /**
         * Phase de génération du flux de présentation
         */
        RENDER_XMLPRES("renderOutputTimer"),
        /**
         * Taille en caractère du rendu généré
         */
        RENDER_SIZE("renderSize"),
        /**
         * Le timer d'une requête HTTP. Elle englobe tous les traitements applicatifs
sur processus, c'est le timer le plus significatif pour mesurer un temps
         * de génération de page.
         */
        REQUEST("requestTimer"),
        /**
         * Temps de parsing des paramètres de la requête HTTP
         */
        REQUESTPARSER("parsingRequestTimer"),
        /**
         * Temps de génération d'un champ dans le flux de présentation (plus actif à
ce jour)
         */
        ROW_DOCUMENTLINE_FIELD("rowDocumentRowFieldTimer"),
        /**
         * Temps utilisé par les solveurs d'expressions (plus actif à ce jour)
         */
        SOLVER("solverTimer"),
        /**
         * Temps mis par les transactions commit/rollback/checkConnexion base
         */
        TRANSACTION("transactionTimer"),
        /**
         * Unmarshalling des APIs
         */
        UNMARSHALL("UnmarshallTimer"),
        /**
         * Le pendant de RENDER_XMLPRES mais pour une View, c'est donc un niveau plus
fin. Cela comprend le temps de génération du flux de présentation pour la
         * View, ainsi que les requêtes BDD associées.
         */
        VIEW("viewTimer"),
        /**
         * Temps global d'exécution d'un WS
         */
        WS_SERVER("webservice.server.Timer"),
```

```
        /**
         * Temps d'exécution d'un WS hors temps de traitements pre/post
         */
        WS_SERVER_EXECUTE("webservice.server.executeOperation.Timer"),
        /**
         * Temps post opération d'un WS
         */
        WS_SERVER_POST_EO("webservice.server.postExecuteOperation.Timer"),
        /**
         * Temps pré opération d'un WS
         */
        WS_SERVER_PRE_EO("webservice.server.preExecuteOperation.Timer"),
        /**
         * Phase de transformation XSLT. A noter que les autres moteurs de
transformations (CHARTT, PDFT, BIRTT et SVGT) n'ont pas de timers associés.
         */
        XSLT("XSLTTimer");
```

*Le nom du timer (ex : **REQUEST**) correspond à celui qu'on retrouve dans la définition de gce.properties (attribut **@name**).*

3.3 LOG4J

Les fichiers develop_log4j.xml et exploit_log4j.xml ont quelque peu évolués afin de mettre à disposition les nouveaux timers, en particulier REQUEST.

Ils annulent et remplacent les versions précédentes. A noter que le socle technique n'exploite pas de nouvelle version du framework LOG4J.

Exemple d'appender :

```
<appender name="requestTimerAppender"
class="org.apache.log4j.DailyRollingFileAppender">
	<param name="File" value="log/requestTimer.log"/>
	<param name="DatePattern" value="'.'yyyy-MM-dd"/>
	<layout class="org.apache.log4j.PatternLayout">
		<param name="ConversionPattern"
value="[%d]¤%X{TRX_COUNTER}¤%X{TIMER_NAME}¤%X{ELAPSE_TIME}¤%X{ELAPSE_TIME_UNIT}¤%X{
ELAPSE_TIME_LEVEL}¤%X{ELAPSE_TIME_PROCESS}¤%X{ELAPSE_TIME_BUILD}¤%X{ELAPSE_TIME_FLU
SH_OUT}¤%X{ELAPSE_TIME_DISCONNECT}¤%X{ELAPSE_TIME_RENDER_XMLPRES}¤%X{ELAPSE_TIME_RE
NDER}¤%X{ELAPSE_TIME_TRANSFORMATION}¤%X{HTMLPRES_SIZE}¤%X{COUNT_VO_EXECUTEQUERY}¤%X
{ELAPSE_TIME_EXECUTEQUERY}¤%X{ELAPSE_TIME_DML}¤%X{REQUEST_CURRENT_BUSINESSVIEW}¤%X{
UUID}¤%X{USER}¤%X{ENTITY}¤%X{TARGET}¤%X{REQUEST_URI}¤%X{QUERY_STRING}¤%X{REQUEST_CI
NEMATIC}%n"/>
	</layout>
</appender>
```

Le logger correspondant :

```
<logger name="requestTimer" additivity="false">
	<level value="info"/>
	<appender-ref ref="requestTimerAppender"/>
</logger>
```

Le paramètre de l'énumération Java correspond au nom du logger LOG4. Par exemple (en gros ci-dessous et ci-dessus) :

```
	/**
	 * Le timer d'une requête HTTP. Elle englobe tous les traitements applicatifs
sur processus, c'est le timer le plus significatif pour mesurer un temps
	 * de génération de page.
	 */
	REQUEST("requestTimer"),
```

3.4 Définition des clés du MDC dans le socle technique

Il s'agit ici encore d'une énumération Java :

```
public static enum MDC_KEY
{
	// Informations d'ordre générale
	DATE                                        (true),
	TIME                                        (true),
	REMOTE_ADDR                             (false),
	REMOTE_HOST                             (false),
	SERVER_HOST                             (false),
	/**
	 * Application Server identifier
	 */
	AS_ID                                       (true),
	/**
	 * JVM identifier. At runtime, the value is unique by classloader in case of
several WebModules contained in the same JVM. Should be WM_ID ... but not mind.
	 */
	JVM_ID                                      (true),
	// Informations liées à la connexion.
	ENTITY                                             (true),
	UUID                                        (true, MDC_DEFAUT_ZERO),
	TARGET                                             (true),
	USER                                        (true),
	LANGUAGE                                    (true),
	/**
	 * Le compteur de transaction
	 */
	TRX_COUNTER                                 (true),
	// Information liée au chrono de l'activité.
	// temps écoulé durant le chronométrage.
	ELAPSE_TIME                             (true),
	// Unité du ELAPSE_TIME
	ELAPSE_TIME_UNIT                        (true),
	// niveau de prise en compte du ELAPSE_TIME.
	ELAPSE_TIME_LEVEL                       (true),
	// Temps de construction de la BV
	ELAPSE_TIME_BUILD                       (true),
	// Temps de processus : méthode process, permet d'alimenter le MDC avec une
valeur supplémentaire de chrono
	ELAPSE_TIME_PROCESS                     (true),
	// Phase de transformation complète
	ELAPSE_TIME_RENDER                      (true),
	// Génération du flux de prés
	ELAPSE_TIME_RENDER_XMLPRES      (true),
	// XSLT ...
	ELAPSE_TIME_TRANSFORMATION      (true),
	// Flush du résultat à travers la servlet : mesure les perfs réseaux entre le
serveur et le poste client
	ELAPSE_TIME_FLUSH_OUT                   (true),
	/**
	 * Nombre de VO exécutées
	 */
	COUNT_VO_EXECUTEQUERY                   (true),
	/**
	 * Temps cumulé sur la transaction a exécuter des requêtes
	 */
	ELAPSE_TIME_EXECUTEQUERY        (true),
	/**
	 * Temps cumulé sur la transaction a exécuter des DML
	 */
	ELAPSE_TIME_DML                             (true),
	// Temps disconnect : suivant les cas de paramétrage cela peut être plus ou
```

```
moins long. Ca peut être intéressant de pister cette information
        ELAPSE_TIME_DISCONNECT                  (true),
        // Informations liées à l'invocation d'un service.
        APPMODULE                                       (false),
        SERVICE_NAME                            (false),
        // Informations liées aux ViewObject et au CRUD.
        ENTITYOBJECT                            (false),
        VIEWOBJECT                                      (false),
        VIEWOBJECT_INSTANCE                     (false),
        SQL_QUERY                                       (false),
        SQL_PARAM                                       (false),
        SQL_PARAM_PRIMARYKEY                    (false),
        SQL_OPTIMIZATION                        (false),
        REQUEST_BUSINESSVIEW_LIST       (false),
        // Informations liées à au contenu de la requête reçue par le socle.
        REQUEST_CURRENT_BUSINESSVIEW    (false, MDC_DEFAUT_EMPTY),
        REQUEST_CINEMATIC                       (false, MDC_DEFAUT_EMPTY),
        REQUEST_NAMESPACE_CHP                   (false),
        REQUEST_NAMESPACE_SEL                   (false),
        QUERY_STRING                            (false),
        HTTP_METHOD                                     (true),
        REQUEST_TRACE_ID                        (false),
        REQUEST_URI                             (false),
        REQUEST_URL                             (false),
        SERVLET_UUID                            (false),
        STYLESHEET_REF                                  (false),
        ACTION_NAME                             (false),
        VIEW_NAME                                       (false),
        BC4J_VIEWLINK_NAME                      (false),
        BUSINESSFLOW_NAME                       (false),
        LINEFIELD_NAME                                  (false),
        // Informations liées à la session HTTP.
        REQUEST_SESSION_ID                      (false),
        REQUEST_SESSION_ID_VALID        (false),
        HTTP_SESSION_ID                                 (false),
        // Génération de la pile d'appel.
        WST                                             (false),
        // Informations liées à la taille des flux.
        XMLPRES_SIZE                            (false),
        XMLPRES_NODECOUNT                       (false),
        HTMLPRES_SIZE                                   (true),
        // Information liées à la transaction
        TRANSACTION_MODE                        (false),
        // Informations liées aux webServices
        WS_SERVICE                                      (false, MDC_DEFAUT_EMPTY),
        WS_OPERATION                            (false, MDC_DEFAUT_EMPTY),
        // Nom du timer, ça peut servir en cas de logging de temps multiples sur la
même ligne de log
        TIMER_NAME                                      (true),
        // Méthode XMLRpc
        METHOD_XMLRPC                           (true),
        ELAPSE_TIME_XMLRPC                      (true),
        // Libellé open bar pour des infos non publiable par le .log classique.
        LABEL                                           (false);
```

3.5 Liens entre les timers

Voir la documentation déjà réalisée sur ce point dans le Release Note du socle technique de la version GCE140.

4 Exemples de publisher

Il suffit pour cela d'implémenter l'interface suivante :
com.generixgroup.timer.common.GXTimerPublisher<TIMERDEF>

Voici le code source permettant de publier des temps à destination de LOG4J :

```
package fr.generix.technicalframework.timer;

import com.generixgroup.timer.common.GXChronoDef.UNIT;
import com.generixgroup.timer.common.GXTimerPublisher;

import fr.generix.technicalframework.log4j.LOGGER;
import fr.generix.technicalframework.timer.GXTimerManager.TIMERDEF;

/**
 * The LOG4J timer publisher.
 *
 * @author roa / Technical Framework team - GENERIX Group
 * @version ET 7.1
 * @since 13 janv. 2012
 */
public class GXTimerLOG4JPublisher implements GXTimerPublisher<TIMERDEF>
{
	public void publish(TIMERDEF timerdef, long elapsed, UNIT unit, double
precision)
	{
		if (timerdef != null)
		{
			double d = UNIT.convert(elapsed, timerdef.getUnit());
			if (d > timerdef.getPrecision())
			{
				LOGGER.putMDC(LOGGER.MDC_KEY.TIMER_NAME,
timerdef.name());
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME,
Double.valueOf(d));
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME_UNIT,
timerdef.getUnit());
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME_LEVEL,
Double.valueOf(timerdef.getPrecision()));
				timerdef.getLogger().info("timer");
			}
		}
	}
}
```

L'unité et la précision proviennent directement de ce qui est paramétré dans *gce.properties*, il convient donc de faire les conversions idoines. On note ici un filtrage qui permet de ne publier les temps qu'à compter d'un certain niveau de précision.

Voici le code source permettant de publier des temps à destination de la base de données :

```
package fr.generix.technicalframework.timer;

import com.generixgroup.timer.common.GXChronoDef.UNIT;

import fr.generix.technicalframework.log4j.LOGGER;
import fr.generix.technicalframework.log4j.LOGGER.MDC_KEY;
import fr.generix.technicalframework.timer.GXTimerManager.TIMERDEF;

public class GXTimerRDBMSRequestPublisher extends AbstractGXTimerRDBMSPublisher
{
	private static String SQL_INSERT = "insert into timer_request "
		+ "(codsoc, datmod, utimod, index_seq, timestamp, uuid, jvm_id,
uticod, "
		+ "entity, request_uri, request_url, request_cinematic, http_method,
bv_name, "
		+ "codpev, elapse_time, unit, precision, elapse_time_process,
elapse_time_build, "
		+ "elapse_time_flush_out, elapse_time_disconnect,
elapse_time_render_xmlpres, "
		+ "elapse_time_render, elapse_time_transformation,
elapse_time_executequery, "
		+ "elapse_time_dml, elapse_time_xmlrpc, htmlpres_size,
count_vo_executequery) "
		+ "values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";


	public void publish(TIMERDEF timerdef, long elapsed, UNIT unit, double
precision)
	{
		if (timerdef != null)
		{
			double d = UNIT.convert(elapsed, timerdef.getUnit());
			if (d > timerdef.getPrecision())
			{
				LOGGER.putMDC(LOGGER.MDC_KEY.TIMER_NAME,
timerdef.name());
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME,
Double.valueOf(d));
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME_UNIT,
timerdef.getUnit());
				LOGGER.putMDC(LOGGER.MDC_KEY.ELAPSE_TIME_LEVEL,
Double.valueOf(timerdef.getPrecision()));

				Double disconnect = (Double)
LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_DISCONNECT);
				Object[] params = new Object[]
				{
					0,
					LOGGER.getMDC(MDC_KEY.DATE),
					"MONITOR", // utimod en dur, au moins on peut
estimer d'où vient l'enregistrement :-)
					LOGGER.getMDC(MDC_KEY.TRX_COUNTER),
					LOGGER.getMDC(MDC_KEY.TIME),
					LOGGER.getMDC(MDC_KEY.UUID),
					LOGGER.getMDC(MDC_KEY.JVM_ID),
					LOGGER.getMDC(MDC_KEY.USER),
					LOGGER.getMDC(MDC_KEY.ENTITY),
					LOGGER.getMDC(MDC_KEY.REQUEST_URI),
					formatString((String)
LOGGER.getMDC(MDC_KEY.QUERY_STRING), 4000),
					LOGGER.getMDC(MDC_KEY.REQUEST_CINEMATIC),
					LOGGER.getMDC(MDC_KEY.HTTP_METHOD),

	LOGGER.getMDC(MDC_KEY.REQUEST_CURRENT_BUSINESSVIEW),
					LOGGER.getMDC(MDC_KEY.TARGET),
```

```
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_UNIT).toString(),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_LEVEL),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_PROCESS),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_BUILD),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_FLUSH_OUT),
                                // On passe dans disconnect uniquement si on est en
DS poolées ou en WS.
                                // Dans les autres cas, la valeur peut être null
est c'est normal.
                                (disconnect == null) ? 0.0D : disconnect,
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_RENDER_XMLPRES),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_RENDER),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_TRANSFORMATION),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_EXECUTEQUERY),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_DML),
                                LOGGER.getMDC(MDC_KEY.ELAPSE_TIME_XMLRPC),
                                LOGGER.getMDC(MDC_KEY.HTMLPRES_SIZE),
                                LOGGER.getMDC(MDC_KEY.COUNT_VO_EXECUTEQUERY)
                        };

                        // On construit une nouvelle séquence pour
l'orchestrateur SQL
                        PublisherRDBMS p = new PublisherRDBMS(SQL_INSERT,
params);
                        add(p);
                    }
                }
        }

        private String formatString(String s, int maxLength)
        {
                if (s == null)
                {
                        return "";
                }
                else if (s.length() < maxLength)
                {
                        return s;
                }

                return s.substring(0, maxLength - 1);
        }
}
```

On dérive ici d'une couche abstraite qui permet d'utiliser des méthodes communes à l'ensemble des publicateurs vers la base de données. On fournit juste un objet (*PublisherRDBMS*) qui emballe la requête SQL bindée (de préférence, bien que ce ne soit pas une obligation) et ses paramètres. Cette la méthode *add* qui est chargée d'ajouter cette publication à la liste : on crée donc une pile d'ordres SQL qui seront exécutés en lot par le moniteur périodique (piloté par *gce.properties* comme évoqué dans un chapitre précédent).