Projet ROSES
Programme MDCO – Edition 2007

Livrable no D1.2
Architecture d'un Système ROSES centralisé

Identification

Acronyme du projet	ROSES
Numéro d'identification de l'acte attributif	ANR-07-MDCO-011-01
Coordonnateur	Paris 6
Rédacteur (nom, téléphone, email)	Bernd Amann (LIP6) – 01 44 27 70 09 Bernd.Amann@lip6.fr Jordi Creus (LIP6) Nicolas Travers (CNAM)
No. et titre	D1.2 Architecture d'un Système ROSES centralisé
Version	
Date de livraison prévue	31 décembre 2008 / t0+12
Date de livraison	Mars 2010 / t0+12

Résumé

Ce document décrit l'architecture d'un système ROSES centralisé. Ce système doit être capable de gérer et de traiter des grandes quantités de *publications* et de *souscriptions ROSES [voir livrable D2.2 Model and Algebra]*.

Sommaire

1. Architecture générale

Les principaux services offerts par un système ROSES sont :

- La gestion (création, modification et suppression) de publications. Une **publication** est un flux personnalisé définie par la composition de sources externes (RSS, Atom…) et d'autres publications ROSES [livrable D5.1]).
- La gestion (enregistrement et suppression) des sources externes utilisées par les publications
- La gestion (création, modification et suppression) des souscriptions aux publications. Une **souscription** est un abonnement utilisateur à des publications ROSES.

Ces services ROSES sont accessibles à travers une *interface de gestion*.

Les *traitements internes* effectués par le système sont :

- L'acquisition du contenu des sources externes au système (flux RSS, données)
- Le traitement de l'ensemble des publications ROSES définies par les utilisateurs (graphe de publication)
- Enfin, la diffusion des items produits par les publications aux utilisateurs abonnés à ces publications en fonction des types de souscription qu'ils ont définis

Le Système ROSES est découpé en trois modules : un module d'**Acquisition**, un module d'**Evaluation** et un module de **Diffusion**. Le module d'Acquisition est chargé principalement de rafraichir les données des sources externes ; le module d'Evaluation s'occupe de la gestion des publications et du traitement et de l'optimisation du graphe de publications ; enfin, le module de Diffusion est chargé notamment de la diffusion des flux de publication.

La **Figure 1 : Architecture générale** montre une vue globale du fonctionnement du système. Les arcs verticaux montrent les modes de communication entre les modules (pull ou push) : le module d'acquisition transforme des données (document XML-RSS) en flux d'items qui sont traités *en continu* par les modules suivants (évaluation, optimisation et diffusion). Les métadonnées (définitions des sources, publications et souscriptions) sont stockées dans des bases de données indiquées à droite du schéma. L'interface de gestion (à gauche) fournit l'ensemble des services nécessaires à la gestion et l'utilisation du système.

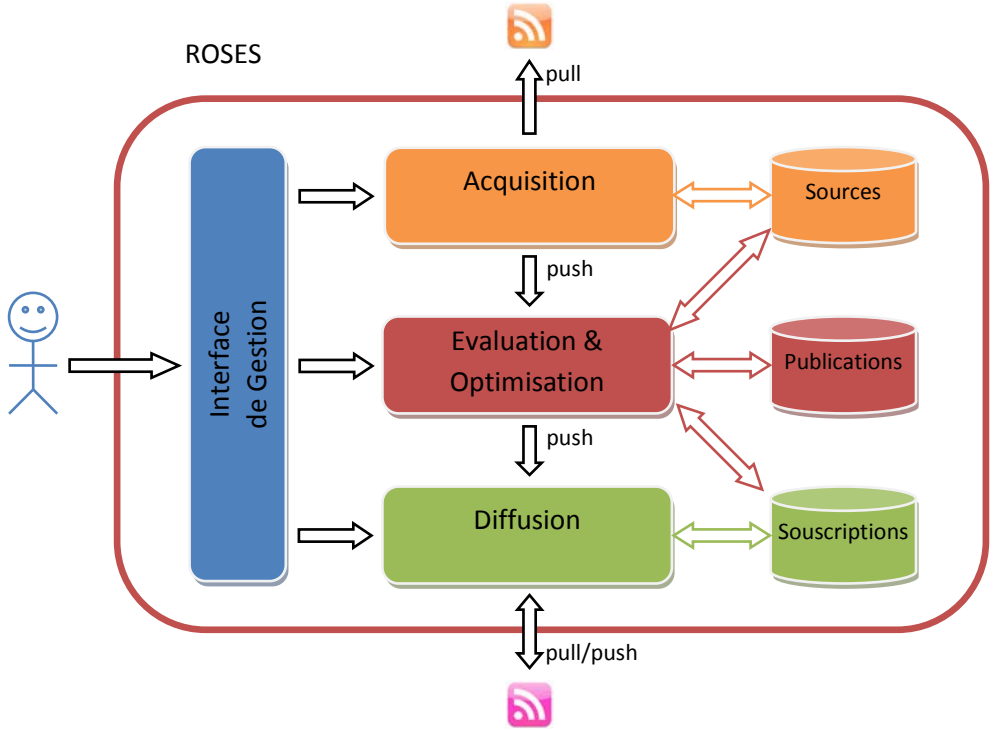


Figure 1 : Architecture générale

Les sections suivantes décrivent plus en détail le fonctionnement les différents modules qui composent un système ROSES. Un aspect important lors de la définition et de l'implémentation de chaque module concerne le *passage à l'échelle*. Le système doit être capable de passer à l'échelle par rapport au nombre de sources, par rapport au nombre de publications et par rapport au nombre de souscriptions.

2. Module d'Acquisition

Le module d'Acquisition est en charge de la lecture des sources externes utilisées par les publications ROSES. Il remplit deux fonctionnalités principales :

1. Transformation de données externes en item ROSES (voir livrable D2.2 Modèle et Algèbre) : les sources utilisées par des publications ROSES peuvent être variées (flux RSS ou Atom, bases de données, services web). Le module d'acquisition applique la solution traditionnelle à ce problème par l'intégration d'un adaptateur (wrapper) pour chaque type de source qui

est en charge de la transformation des données obtenues lors de l'acquisition dans un format homogène d'item qu'on appelle item ROSES.

2. Rafraîchissement des données : L'accès aux sources nécessite un protocole de rafraîchissement efficace qui prend en compte les limitations du réseau et des serveurs (bande passante, nombre de demandes / minutes, …) tout en maximisant le volume de données à traiter. Dans la version actuelle du module d'acquisition, nous considérons uniquement le mode pull qui est le mode standard pour accéder à des flux RSS.

L'architecture du module d'acquisition est montrée dans la **Figure 2 : Module d'acquisition** :

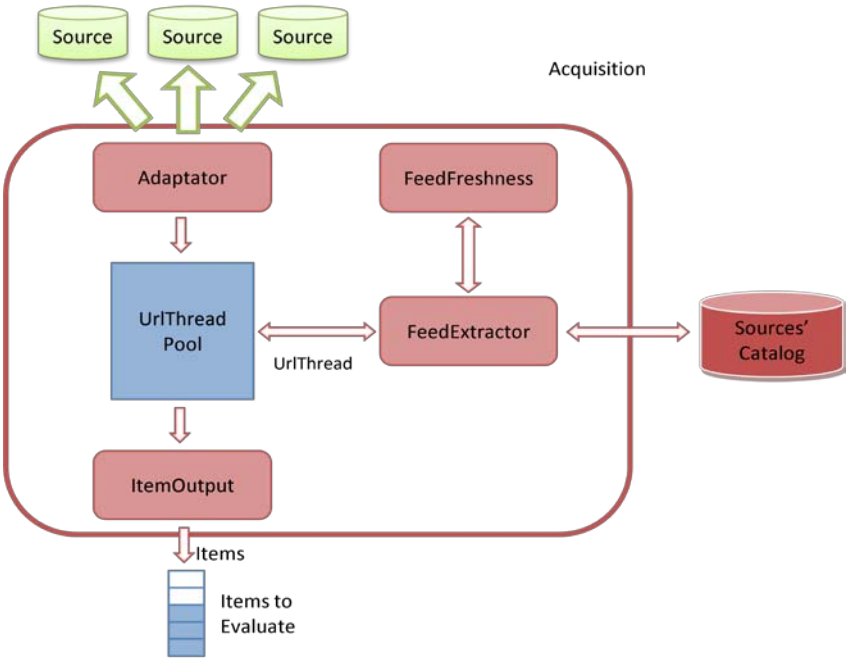


Figure 2 : Module d'acquisition

Le processus d'extraction des flux est déclenché par le composant **FeedExtractor** qui choisit les sources à rafraîchir dans un catalogue de sources. Le choix des sources est fondé sur des dates de rafraîchissement estimées et mises-à-jour continuellement (cf. *FeedFreshness*). Chaque source est traitée par un thread d'extraction séparé (UrlThread) faisant partie d'un « pool de threads » à exécuter. Le thread UrlThread contient toutes les informations nécessaires à l'extraction :

- Adaptateur en fonction du type de source
- Dernier item détecté dans le flux
- Statistiques de téléchargement

Chaque **Adaptateur** transforme les nouveaux éléments détectés dans la source en *Items*.

Lors de l'extraction d'une source, il est nécessaire de différencier les anciens des nouveaux éléments. Pour cela l'identifiant du dernier item détecté de cette source est stocké. Lorsque celui-ci est retrouvé lors de la nouvelle acquisition de la source, nous pouvons alors cesser l'acquisition puisqu'aucun nouvel item ne pourra être trouvé. Le temps d'extraction est ainsi diminué.

Une fois que l'ensemble des nouveaux items sont extraits par l'adaptateur, un *paquet d'items* (ensemble d'items provenant de la même source) est envoyé à l'interface de sortie **ItemOutput**. Cette interface peut être spécialisée par rapport aux différentes utilisations des items Roses générés. Plusieurs implémentations ont déjà été réalisées :

- *ItemOutputImpl* : transmet les items à évaluer au module d'évaluation à l'aide du Dispatcher (cf. Module d'évaluation).
- *ItemOutputStorage* : stocke les items provenant des sources dans une base de données.
- *ItemOutputStatistic* : calcule des statistiques à la volée sur les sources.
- *ItemOutputEmulator* : interface qui régénère le flux de paquets d'items produit par une acquisition antérieure dont les items ont été stockés par ItemOutputStorage.

Une fois que les items de la source ont été notifiés, il est nécessaire de mettre à jour les statistiques de téléchargement de celle-ci, ainsi que d'estimer sa prochaine date de mise à jour. Cette estimation est effectuée par le composant **FeedFreshness** qui calcule à chaque rafraîchissement d'une source, la durée (delta) jusqu'au prochain rafraîchissement de la même source. La fonction de mise à jour pour le nouveau delta après chaque rafraîchissement est la suivante :

$$\Delta_{new} = \begin{cases} \Delta_{old} * 1.5 & \text{si erreur} \\ \Delta_{old} * 1.35 & \text{si pas de nouveaux items} \\ \Delta_{old} & \text{si entre 1 et 5 nouveaux items} \\ \Delta_{old} * 0.33 & \text{si tous les items sont nouveaux} \\ \Delta_{old} * \alpha & \text{sinon } (\alpha < 1) \end{cases}$$

- La source a produit une erreur (serveur introuvable, timeout, analyse du fichier…), le delta est augmenté de 50%. Ce qui permet au serveur de redevenir disponible.
- La source n'a pas produit de nouveaux items, le delta est augmenté de 35%. Ce qui laisse du temps à la source pour être modifiée, et de s'approcher de sa vitesse de croisière
- La source a produit entre 1 et 5 items, le delta n'est pas modifié
- La source n'a produit que des nouveaux items (aucun ancien item), le delta est alors diminué à 33%
- Sinon, le delta est diminué proportionnellement :
    - au nombre de nouveaux items par rapport aux anciens. Plus un flux produit de nouveaux items, plus il faut le mettre à jour
    - A l'ancien delta. Plus l'ancien delta est grand, plus il faut le réduire (et inversement)
    - Au nombre de sources dont le delta est plus petit. Au vu du nombre de flux à mettre à jour, il faut tenir compte de ce nombre pour déterminer si cette source est plus urgente que les autres. Ainsi, plus il y a de sources avec un delta inférieur, plus on peut se permettre de la faire avancer, et inversement.

La nouvelle valeur doit être comprise entre 1 minute et 5 jours. La borne inférieure évite que les sources soient sollicitées trop souvent (risque de refus de connexion). La borne supérieure évite que les sources erronées ne deviennent pas inaccessibles.

3. Module d'Evaluation

Les tâches principales réalisées par le module d'Evaluation sont :

- L'insertion et suppression de publications dans le graphe de publications
- Le traitement du graphe de publications
- L'optimisation dynamique ou « *at runtime* » du graphe

La **Figure 3 : Module évaluation** montre le schéma de fonctionnement de ce module et ses différents composants :

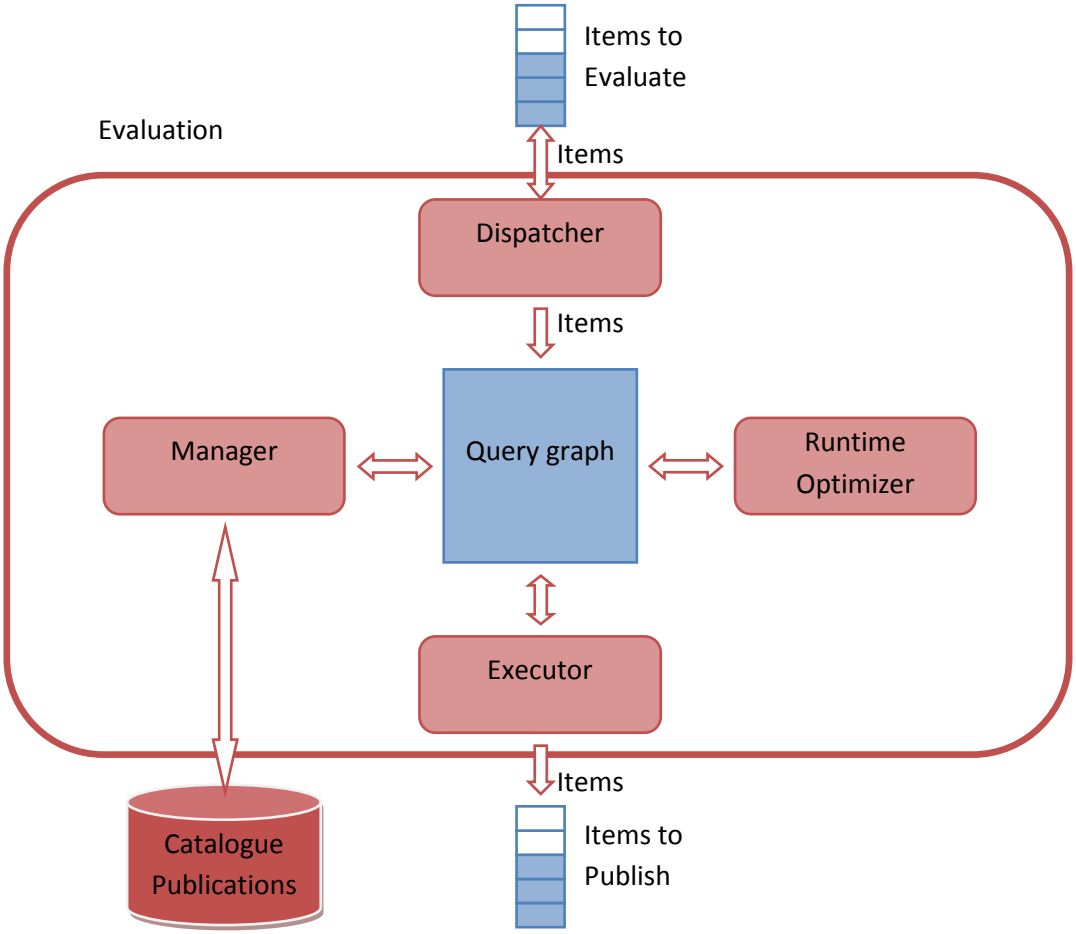


Figure 3 : Module évaluation

3.1. Graphe de publication

Le module d'Evaluation est en charge de la gestion et de l'évaluation du **graphe de publications (query graph)**. Ce graphe est construit à partir des publications définies par les utilisateurs avec le langage ROSES (voir livrable D5.1) à travers le module Manager. Il représente un plan d'exécution dynamique qui évolue avec l'ensemble des publications stockées dans le catalogue des publications. On peut distinguer deux niveaux de représentations. Au niveau logique, le graphe de publications correspond à une composition d'expressions algébriques (voir livrable D2.2 Modèle et Algèbre) avec des opérateurs d'union, filtrage, fenêtrage et de jointure. Ces expressions sont évaluées *en continu* par un ensemble de threads qui communiquent à travers des *files d'attente*.

Prenons, par exemple, les trois publications et leurs expressions algébriques[1] suivantes :

P_1	return (S_1 | S_2 | S_3) where $pred_1$	$\sigma_{pred1}(S_1 \cup S_2 \cup S_3)$
P_2	return (S_4 | S_5 | S_6 | S_7) as f | (S_8 | S_9) as g where pred2(f) and pred3(g)	$\sigma_{pred2}(S_4 \cup S_5 \cup S_6 \cup S_7) \cup \sigma_{pred3}(S_8 \cup S_9)$
P_3	return P_2 join window win_pred on (S_{10} | S_{11} | S_{12}) as w where pred4(w)	$P_2 \bowtie_{join\_pred} (\omega_{win\_pred}(\sigma_{pred4}(S_{10} \cup S_{11} \cup S_{12})))$

La publication P_1 retourne tous les items des sources S1, S2 et S3 qui satisfont le prédicat pred1. Ce type de requêtes simple permet de surveiller les items publiés par un ensemble de sources concernant par exemple un thème particulier (union filtrée). La publication P_2 exprime une union de deux unions filtrées. La publication P_3 exprime une jointure de la publication déjà existante P_2 avec des informations publiées par les sources S10, S11 et S12 (voir livrable D5.1).

La **Figure 4 : Graphe de publication** montre un parmi différents graphes possibles pour traiter ces publications :

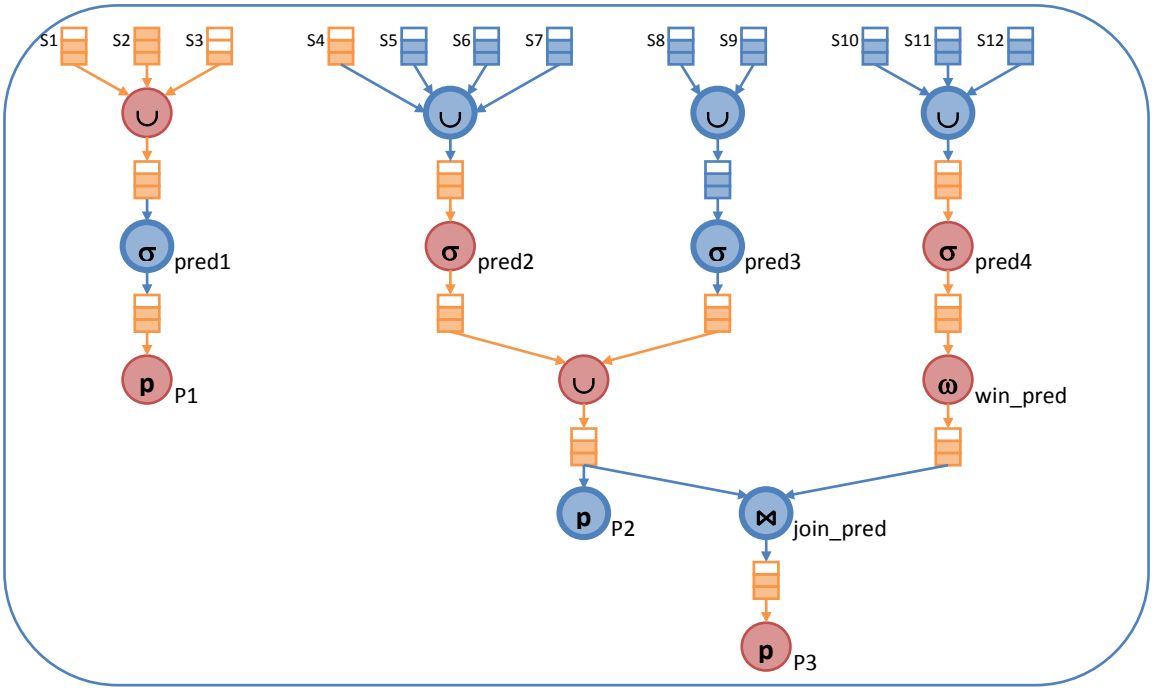


Figure 4 : Graphe de publication

On considère que chaque opérateur est évalué périodiquement par un thread séparé (évaluation asynchrone). Le module d'évaluation dispose d'un pool de threads (le nombre de threads peut être spécifié dans un fichier de configuration) qui exécute en parallèle un ensemble de threads (les

[1] L'algèbre ROSES est détaillée dans le livrable D2.2

opérateurs en activité sont indiqués en gras/rouge). Le graphe de publication se situe au centre du module d'Evaluation et il est utilisé pour synchroniser les autres composants du module (voir plus loin).

3.2. Dispatcher

Le composant **Dispatcher** récupère les items générés par le module d'Acquisition et les distribue sur les files d'attente correspondantes aux sources du graphe de publications.

3.3. Executor

La **Figure 5 : Composant Executor** décrit l'architecture interne du module *Executor* :

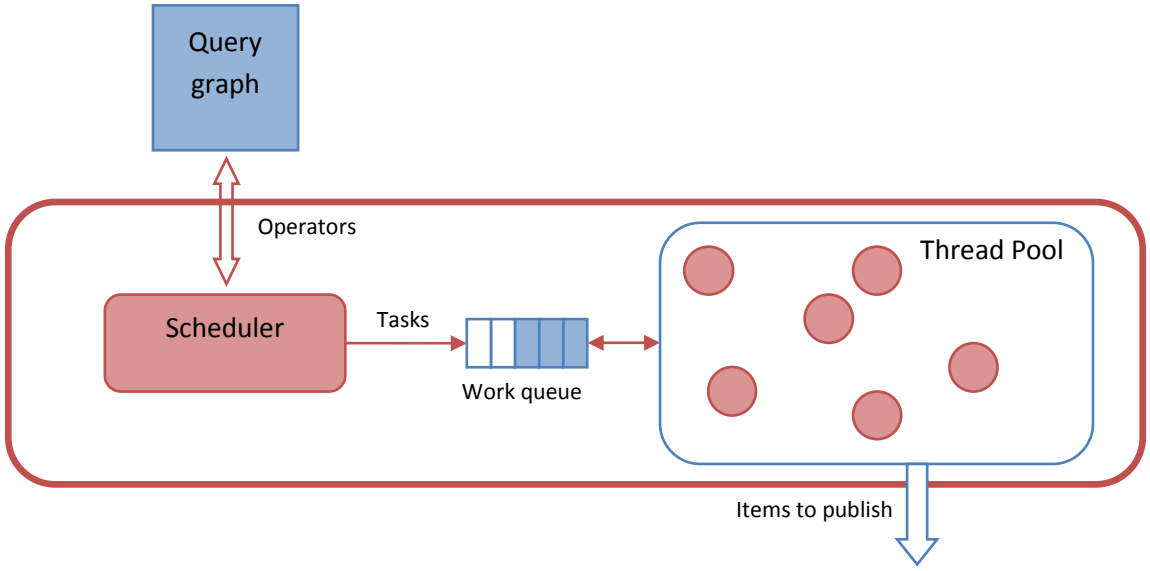


Figure 5 : Composant Executor

Le composant **Executor** observe le graphe de publications et choisit les opérateurs à exécuter. Ce choix est fait par le sous-composant *Scheduler* qui transmet les tâches à exécuter dans un *Pool de Threads* et permet ainsi l'exécution parallèle de plusieurs opérateurs. Les items générés par les opérateurs de publication (sortie du graphe) sont insérés dans une file d'attente d'items à publier, qui est ensuite traitée par le module de *Diffusion*.

Plusieurs stratégies peuvent être appliquées par le *Scheduler* pour décider quels opérateurs exécuter:

- rendre prioritaire l'exécution des opérateurs consommateurs dont les files d'attente sont les plus remplies
- rendre prioritaire les consommateurs de files d'attente ayant les items les plus anciens
- rendre prioritaire les opérateurs utilisés par les publications ayant le plus grand nombre de souscriptions

3.4. Manager

Le composant **Manager** gère l'insertion et la suppression de publications dans le catalogue et dans le graphe de publications. Il inclut un compilateur qui traduit les requêtes de publications en graphe d'opérations. L'insertion et la suppression des publications est faite « en-ligne », c'est-à-dire sans arrêter l'évaluation du graphe de publications par le composant Executor. La synchronisation est réalisée par la possibilité de suspendre temporairement l'évaluation d'opérateurs et de sous-graphes.

3.5. Runtime optimizer

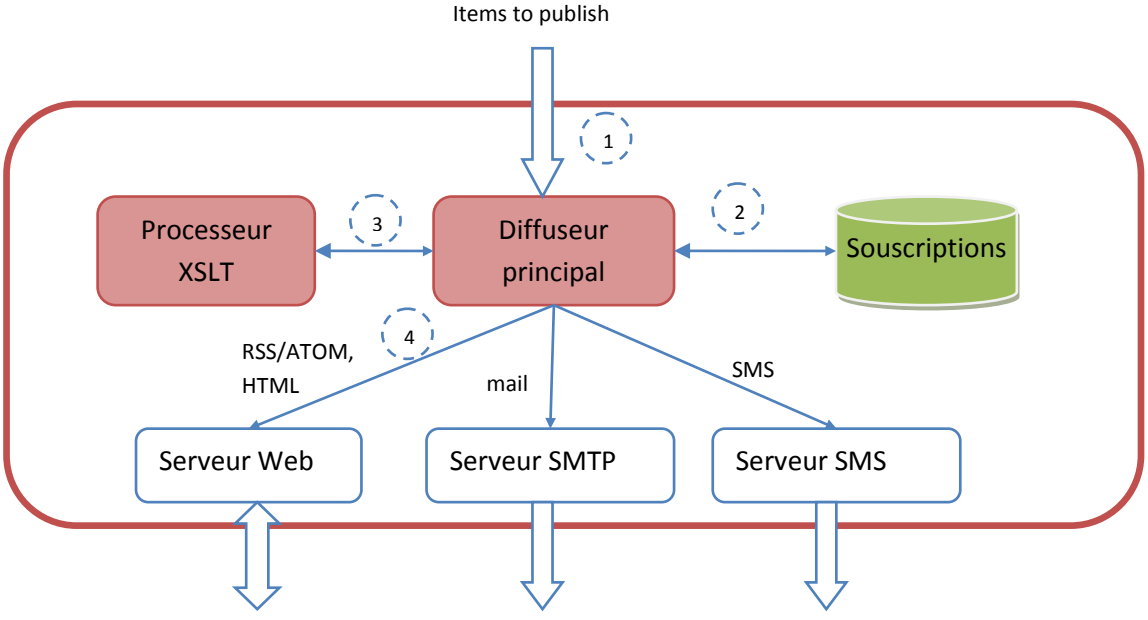
Le composant **Runtime Optimizer** est en charge de l'optimisation du graphe de publications, dans le but d'améliorer les performances et d'augmenter le nombre de publications traités. Deux techniques d'optimisation principales sont appliquées :

1. Optimisation par réécriture logique en utilisant un ensemble de règles de réécriture et des heuristiques (par exemple, « sélection avant jointure ») pour réduire le coût d'évaluation du graphe.
2. Optimisation par multi-requêtes physique en introduisant des multi-opérateurs et des multi-flux (canaux) qui permettent de diminuer les couts de traitements et de communication par factorisation.

Cette optimisation du graphe de publications est également effectué « en-ligne » pour s'adapter aux changements qui se produisent sur les entrées et sur les sorties du graphe (par exemple, les changements de débit des sources des publications ou le changement du nombre de souscriptions aux publications), mais aussi pour s'adapter au changement du graphe de publications dû à l'addition et la suppression de publications.

4. Module de Diffusion

Le module de Diffusion est chargé de la diffusion des items générés par les publications. Le processus de diffusion est contrôlé par les souscriptions soumises au système. Ainsi, une publication peut être sujette à zéro ou plusieurs souscriptions qui spécifient les *formats* et les *modes de diffusion* (page web, fichier RSS/ATOM, mail, SMS) souhaités. Les transformations vers un format particulier (RSS, plein texte, HTML, …) sont définies par des règles XSLT qui permettent également de supprimer ou d'agréger les informations contenues dans les items (voir D5.1 Langage de publication/souscription).



Le *diffuseur principal* récupère les items envoyés par le module d'Evaluation (1), identifie les souscriptions concernées (2), applique les transformations nécessaires (3) et répartit les items sur les différents serveurs de diffusion (4).