# Moteur de Recherche Corese
# Rapport d'Activité

17/10/2005 – 20/06/2006

Virginie BOTTOLLIER – Virginie.Bottollier@sophia.inria.fr
Equipe ACACIA, INRIA Sophia-Antipolis
Contact: Olivier Corby – Olivier.Corby@sophia.inria.fr (04 92 38 78 71)

## Résumé

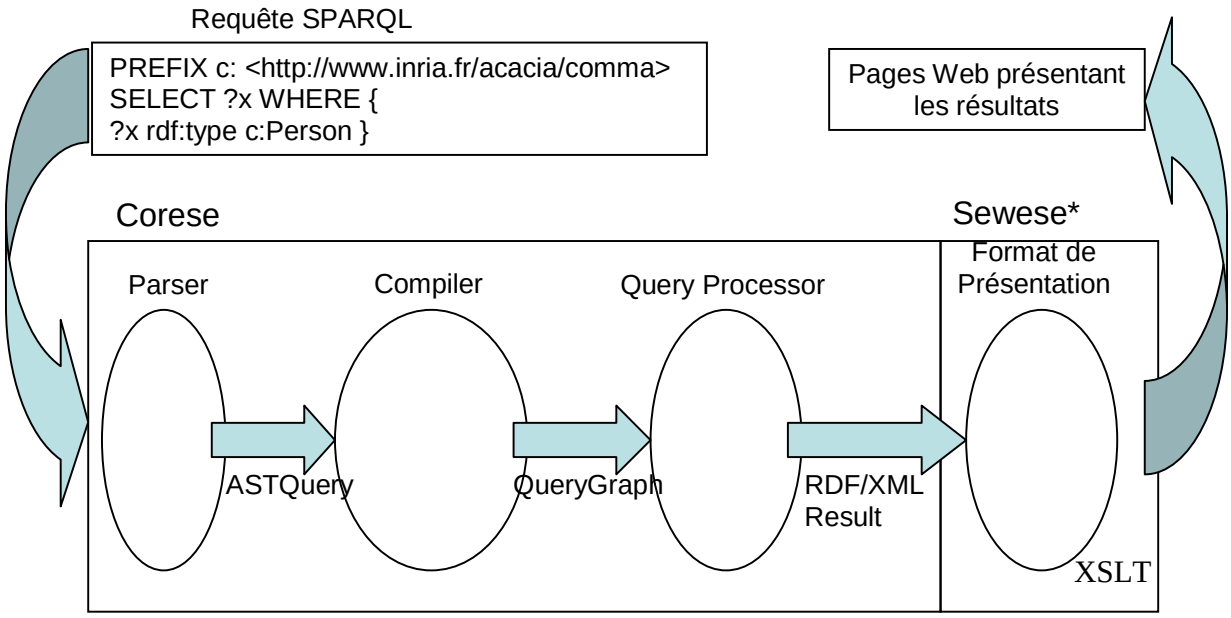
Ce document présente les travaux réalisés pendant mes huit premiers mois en tant qu'ingénieur associée dans l'équipe ACACIA (depuis le 17/10/2005). Le travail a consisté à intégrer le langage de requête standard SPARQL au moteur de recherche Corese. Il se découpe en trois parties : la réalisation d'un Parser (JavaCC), l'intégration de la syntaxe de Corese et l'ajout de nouvelles fonctionnalités SPARQL dans Corese. En parallèle, j'ai augmenté la base de test et participé à la gestion du projet de développement de Corese (GForge, Subversion).
Ce rapport décrit également les travaux prévus dans le cadre de la prolongation de cette mission.

## Table des Matières

## Présentation de Corese et Rappel des Objectifs

Corese [1] (Conceptual Resource Search Engine) est un moteur de recherche pour le web sémantique. C'est une plate-forme de recherche et d'expérimentation pour celui-ci, qui implémente le formalisme RDF/S du W3C en graphes conceptuels, avec une partie de OWL Lite. Corese offre un langage de règles et un moteur d'inférences ; il est utilisé dans des applications de mémoire d'entreprise, de mémoire de projet, de gestion des connaissances.

Le schéma ci-dessous présente le processus de traitement d'une requête, cœur de mon travail durant cette mission.




* Sewese = SErveur WEb SEmantique développé par l'équipe ACACIA

Ma mission peut se décomposer en deux points :

- Intégrer dans Corese le langage de requête SPARQL (futur standard du W3C, actuellement en « Candidate Recommandation ») [2]
- Participer à la gestion de projet du développement logiciel Corese

Plus précisément, le planning sur un an se décomposait de la façon suivante :

- Formation : (1 Mois)
  - Etude de RDF/S, SPARQL : (0.5 Mois)
  - Etude de l'existant, Expérimentation avec Corese : (0.5 Mois)

- Intégration de SPARQL : (8 Mois)
    - Parser JavaCC : (0.5 Mois)
    - Arbre Abstrait : (0.5 Mois)
    - Datatypes : (1 Mois)
    - Implémentation des nouveaux énoncés SPARQL, intégration dans Corese, tests, documentation : (4.5 Mois)
    - Implémentation du protocole SPARQL sur HTTP, tests, intégration : (1.5 Mois)
- Projet Logiciel : (3 Mois)
    - Passage de Corese en Java 1.5, Réécriture de parties de code critique : (1.5 Mois)
    - Profiling et optimisation de la mémoire et temps : (1 Mois)
    - Bug tracking, mailing list utilisateurs, versions de distribution : (0.5 Mois)

# Etat d'avancement des travaux

a. Bilan des tâches effectuées

- Formation

*Durée prévue : 1 Mois*
*Durée réelle: 0.5 Mois*

J'ai étudié les spécifications de SPARQL, RDF/S ; j'ai effectué un TP donné à l'ESSI permettant de prendre en main Corese et assisté à des cours sur le Web sémantique et les langages RDF/S et SPARQL.
J'ai récupéré les sources de Corese, installé le projet sous Eclipse et fait marcher les tests existants.

- Parser : 1ère partie

*Durée prévue : 0.5 Mois*
*Durée réelle : 1 Mois*

Le but de cette partie était de créer un parser à partir d'une grammaire, compatible avec Corese (capable de faire tout ce que le précédent parser de Corese pouvait faire). Pour cela, on a utilisé la technologie JavaCC [3] (Java Compiler Compiler) qui transforme une grammaire écrite sous la forme d'un .jj (mélange de grammaire et de java) en classes Java.

J'ai tout d'abord récupéré la grammaire sparql.jj sur le projet Jena [4] de HP. J'ai ensuite simplifié cette grammaire afin de ne faire que de l'analyse syntaxique. Avec la 1ère version du parser, généré à partir de cette grammaire (grâce à la technologie JavaCC), on pouvait seulement savoir si la requête entrée était correcte syntaxiquement.

J'ai complété cette grammaire afin d'accepter, les énoncés de Corese en plus de ceux de SPARQL (hormis certains peu utilisés). Pour cela, il a fallu faire plusieurs réunions afin de se mettre d'accord sur la syntaxe de la nouvelle grammaire : quels énoncés on gardait/supprimait/modifiait. Dans cette partie on a un peu modifié le langage de Corese afin qu'il s'accorde plus avec SPARQL. Un document listant toutes les modifications/suppressions a été rédigé. Il a également fallu réécrire toutes les requêtes de la base de tests afin de les faire correspondre à la nouvelle grammaire.

*Quelques Exemples :*

- option ( ?x ?p ?y ) => optional { ?x ?p ?y }
- accolades obligatoires après le mot clé « WHERE »
- Ajout du mot clé « FILTER » pour tester une (in)égalité
- Suppression de la syntaxe « not:: »

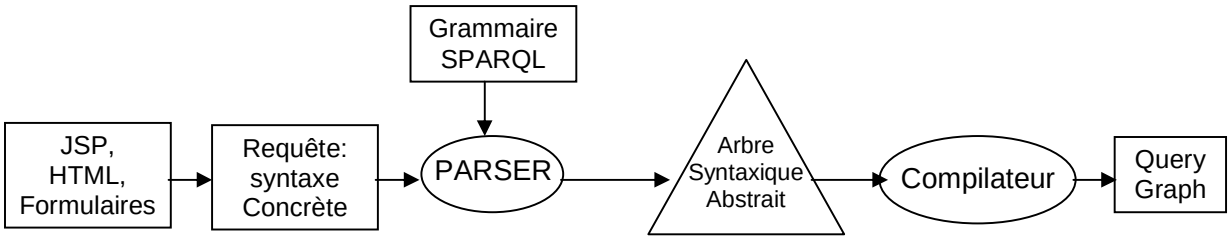
Apres avoir établi la grammaire du nouveau parser, j'ai réutilisé les classes existantes dans Corese (Variable, Term, Expression, Exp, Atom, Triple…) afin d'avoir comme résultat du parser un arbre de syntaxe abstraite, correspondant à la classe ASTQuery, que l'on peut transformer en QueryGraph, comme avec le précédent parser.

*Résultat* : le parser accepte la syntaxe du langage de Corese et d'environ 80% du langage de SPARQL.

- Arbre Syntaxique Abstrait

*Durée prévue : 0.5 Mois*
*Durée réelle : 0.5 Mois*

On a voulu bien séparer la phase de parsing et celle de compilation ; pour cela on a créé un arbre syntaxique abstrait, qui est le résultat du parser. Le compilateur le transforme ensuite en QueryGraph.




Cette modification a fait l'objet d'une réunion avec deux ingénieurs d'Acacia afin d'évaluer la future tâche d'intégration.

- Datatypes

*Durée prévue : 1 Mois*
*Durée réelle : 0.5 Mois*

J'ai suivi les évolutions de SPARQL en ce qui concerne les datatypes (qui se calque souvent sur la définition de XPath [5]).
J'ai testé les opérateurs avec tous les types (notamment avec les types Integer et Double), en vérifiant bien -lorsque les résultats étaient NaN, Infinity ou Error- qu'ils correspondaient à la spécification.

- Parser : 2ème partie

*Durée prévue : 4.5 Mois*
*Durée réelle : 4 Mois*

Après avoir développé un parser pouvant exécuter les requêtes de Corese, on a voulu ajouter les fonctionnalités de SPARQL qui n'étaient pas dans Corese :

- OFFSET : Ne donner les résultats qu'à partir d'un certain rang
  Exemple : La requête suivante donnera les résultats de 10 à 40
  ```
  PREFIX c: <http://inria.sophia.fr>
  SELECT * WHERE { ?x c:FamilyName ?name }
  LIMIT 30 OFFSET 10
  ```
- BASE : Permet de définir une IRI servant de base aux autres IRI utilisées.
  Exemple :
  ```
  BASE <http://inria.sophia.fr>
  SELECT * WHERE { ?x <FamilyName> ?name }
  ```
- CONSTRUCT : Permet de construire un graphe RDF à partir des données trouvées lors de la projection
  Exemple :
  ```
  PREFIX c: <http://inria.sophia.fr>
  CONSTRUCT { <http://inria.sophia.fr#John.Doe> ?p ?y }
  WHERE { ?x ?p ?y . ?x c:FamilyName 'Toto' }
  ```
- DESCRIBE : Pour décrire une ressource. (Construit un graphe RDF contenant tous les triplets ayant comme sujet la ressource)
  Exemple :
  ```
  DESCRIBE <http://inria.sophia.fr#John.Doe>
  ```
- ASK : Permet de savoir s'il existe une réponse à la requête (true ou false)
  Exemple :
  ```
  ASK { ?x ?p ?y }
  ```
- les Blank Nodes (plus ou moins équivalant à la notion de variable existentielle)
  Exemple :
  ```
  PREFIX c: <http://inria.sophia.fr>
  SELECT ?x WHERE {
  ?x c:FamilyName [] . _:a ?p ?x }
  ```

- le tri des résultats par une expression évaluable
  Exemple :
  ```
  PREFIX c: <http://inria.sophia.fr>
  SELECT ?x WHERE { ?x c:age ?y }
  ORDER BY self(?y)
  ```
- la possibilité pour l'utilisateur de définir une nouvelle fonction
  Exemple :
  ```
  PREFIX myurl: <http://appli/test/myclass#>
  SELECT ?x WHERE {
  ?x c:age ?y .
  FILTER (myurl:myfunc(?age)) }
  ```

Cela a été plus compliqué que la première partie car il ne s'agissait pas seulement de toucher à la grammaire mais aussi au noyau de Corese, car ces fonctionnalités n'existaient pas au départ.
La mise en place, dans un premier temps, de OFFSET et BASE a été assez rapide.
La gestion des blank nodes s'est faite en plusieurs fois.
ASK s'est mis en place relativement bien; mais CONSTRUCT et le tri des résultats par une expression évaluable ont nécessité beaucoup plus de temps, ainsi qu'une aide extérieure. (DESCRIBE découle de CONSTRUCT et a donc été facile à mettre en œuvre, une fois CONSTRUCT achevé).
La fonctionnalité permettant l'ajout de fonction dynamiquement m'a permis d'étudier la classe abstraite ClassLoader en java.

En plus de ce qui était prévu, un « Pretty Printer » permettant de générer la version texte de l'ASTQuery a été mis en place ; afin par exemple, de tracer/sauvegarder la requête.

- Tests

*Durée : En Parallèle*

J'ai essayé d'écrire systématiquement des tests ; pour tester la grammaire dans un premier temps, les nouvelles fonctionnalités mises en place dans un second temps.
J'ai finalement tenté de récupérer les tests du Working Group du W3C [6] qui étaient approuvés afin de les faire tourner avec Corese. Le problème qui s'est posé alors était que les données qui devaient être interrogées par les requêtes n'étaient pas écrites en RDF comme on l'aurait voulu mais en Turtle. Pour contourner cette difficulté, j'ai séparé les requêtes en 3 parties :

- Les requêtes sur la syntaxe, ne nécessitant pas de données au départ ; cela m'a permis de détecter et de signaler une erreur dans les tests du Working Group [7] (on ne peut pas avoir un blank node à la place d'une propriété)
- Les requêtes sur les booléens, pour lesquelles j'ai réussi à récupérer le graphe RDF sur lequel effectuer les requêtes.

- Les autres requêtes que j'ai transformées afin d'utiliser les données de notre base de tests.

- Aide au développement : Passage d'un serveur CVS à Subversion et GForge

*Durée prévue : 0.5 Mois*
*Durée réelle : 0.5 Mois*

J'ai installé Corese sur GForge [8], la forge de l'INRIA, permettant de gérer un projet, et j'en ai profité pour passer le projet à Subversion [9] (grâce à l'aide de David Rey, ingénieur Dream). Subversion est un système de gestion de version, plus facile d'utilisation et plus complet que CVS. J'utilise notamment TortoiseSVN [10] sous Windows. TortoiseSVN s'intègre complètement dans Windows et permet d'accéder au lancement des opérations courantes (checkout, commit, update, check for modifications, revert, merge…) par un simple clic droit sur un fichier ou un répertoire.
J'ai écrit un document expliquant comment récupérer Corese depuis la forge. D'autres notes indiquent comment paramétrer Corese et un expliquent comment utiliser subversion.

En réalité, cette tâche n'était pas vraiment prévue initialement, mais installer le projet dans GForge m'a permis de mettre en place un « Bug Tracking » et une Mailing List pour les utilisateurs.

Un autre membre de l'équipe (Fabien Gandon) a mis en place une Toolkit pour gérer les versions de distribution, donc je n'ai finalement pas eu à le faire.

- Intégration dans une nouvelle version de Corese

Durée prévue : Non Prévu
Durée réelle : 1 Mois

Grace à Subversion, j'ai pu développer le nouveau parser tout en gardant le 1er fonctionnel. Il a fallu intégrer mon travail à la version standard de Corese (en langage de subversion, faire un merge de ma branche vers le tronc).
J'ai récupéré l'ancienne base de tests afin de réécrire les requêtes pour le 2ème parser (syntaxe modifiée afin de concorder avec celle de SPARQL). Une fois la base de tests modifiée, j'ai effectué la fusion des 2 versions afin de créer une nouvelle version de Corese.

b. Planning pour la fin de l'année

La fin de la 1ère année de ma mission se termine le 16 Octobre 2006. Les quatre mois restants seront utilisés pour:

- Finaliser le parser : Extraire le parser SPARQL en tant que composant ; il faudrait pouvoir compiler le parser et l'ASTQuery indépendamment du reste, en ayant une interface entre ce composant et Corese. (0.5 Mois)
- Profiling de Corese : détecter les portions de code non utilisées et analyser la gestion de la mémoire (1 Mois)
- Java 1.5 : Basculer Corese de java 1.4 à java 1.5 en gérant les éventuels problèmes liés aux classes « deprecated », en réécrivant les parties de code critiques et en utilisant les optimisations de cette nouvelle version. (1 Mois)
- Langage de règles : l'adapter au nouveau parser et l'optimiser. Pour l'instant le langage de règles utilise l'ancien parser ; il faut intégrer le nouveau. (1.5 Mois)

# Proposition pour la seconde année

Dans le cadre de la prolongation éventuelle de ce contrat (17/10/2006 - 16/10/2007), nous souhaiterions réaliser les points suivants :

- **Qualité du logiciel** : (6 Mois)
  - Gérer les exceptions : lister les exceptions lancées dans Corese et homogénéifier leurs traitements, les messages d'erreurs (1 Mois)
  - Modulariser le reste du code : la projection, le langage de règles, les graphes conceptuels (1 Mois)
  - Ecrire une documentation technique et une documentation utilisateur (2 Mois)
  - Concevoir 2 APIs : une orientée Graphes Conceptuels et une autre RDF et modifier la JavaDoc (2 Mois)
- **Débuggeur de requête** : Créer un mode « débuggeur », qui pourrait donner à l'utilisateur une trace de l'exécution de la requête et donner la cause de l'échec. (1 Mois)
- **Sewese** : (Serveur Web Sémantique basé sur Corese) (3 Mois)
  - Aider au portage de Corese dans Sewese, et peut être dans les autres applications (En parallèle, 0.5 Mois)
  - Intégrer le protocole WSDL SPARQL [11] dans Sewese en collaboration avec d'autres membres de l'équipe (1 Mois)
  - Architecture Distribuée : Pour répondre à la question du passage à l'échelle, il serait souhaitable de pouvoir avoir plusieurs Corese et de distribuer les requêtes sur le Corese approprié ; pour cela on peut utiliser les Web Services (1.5 Mois)
- **RIF** (Rule Interchange Format [12]) : Suivant l'avancement du Working Group du W3C, intégrer ce nouveau (futur) standard au langage de règles de Corese. Remarque : dans le cas où cette intégration ne serait pas possible, on passerait plus de temps sur les autres tâches. (2 Mois)

## Bénéfice Personnel

Au cours de cette mission, j'ai pu acquérir de nouvelles connaissances et en perfectionner d'autres :

- D'un point de vue humain :
  - Travail au sein d'un Institut National de Recherche
  - Intégration au sein d'une équipe compétente et agréable
  - Participation à des séminaires, des Colloquium, quelques cours à l'ESSI
  - Présentation de mon travail aux IA/ODL, participation aux réunions mensuelles (DREAM)
  - Réunions hebdomadaires avec un membre de l'équipe de développement de l'INRIA Sophia (David REY)
- A un niveau plus technique :
  - Découverte du web sémantique avec RDF/S, SPARQL
  - Suivi du processus de standardisation du W3C : les différentes étapes (Working Draft, Last Call, Candidate Recommandation), les forums, les documents relatifs au sujet ; Veille Technologique
  - Apprentissage de JavaCC
  - Perfectionnement de la programmation en Java
  - Pratique de Ant, JUnit
  - Utilisation d'outils dédiés au développement : Eclipse, Subversion, GForge…
  - Réalisation complète d'un parser

Au cours de la seconde année, je compte me former au développement des web services ainsi qu'aux technologies liées aux applications web : tomcat, jsp, servlets, XSLT…

## Références


[1] Corese :
http://www-sop.inria.fr/acacia/corese/

[2] SPARQL :
http://www.w3.org/TR/rdf-sparql-query/

[3] JavaCC :
https://javacc.dev.java.net/

[4] Jena, sparql.jj :
http://cvs.sourceforge.net/viewcvs.py/jena/ARQ/Grammar/sparql.jj

[5] XPath :
http://www.w3.org/TR/xpath

[6] Tests du Working Group de SPARQL :
http://www.w3.org/2001/sw/DataAccess/tests/

[7] Forum sur les tests SPARQL :
http://lists.w3.org/Archives/Public/public-rdf-dawg/2006AprJun/0027.html

[8] GForge :
http://gforge.inria.fr/
[9] Subversion :
http://subversion.tigris.org/
[10] TortoiseSVN :
https://tortoisesvn.tigris.org/
[11] Sparql Protocol :
http://www.w3.org/TR/rdf-sparql-protocol/
[12] RIF (Rule Interchange Format) :
http://www.w3.org/2005/rules/