Université de Technologie de Compiègne
Septembre 2013 - février 2014

Mise en place de moteurs d'inférences dans la plateforme CubicWeb

---

Léa Capgen GI03 ICSI

Sous la direction de :

M. Vincent Michel Logilab (tuteur)
M. Gérard Govaert UTC (enseignant suiveur)

Logilab, 104 boulevard Auguste Blanqui, Paris 13ème
`http://logilab.fr`

Remerciements

Je tiens à remercier Monsieur Vincent Michel, chef de projet au sein du département Web sémantique et tuteur de ce stage, pour la confiance et l'autonomie qu'il m'a accordées dès mon arrivée dans l'entreprise, pour les conseils qu'il m'a apportés lors des différents suivis, et pour le temps qu'il m'a consacré. Je remercie également Monsieur Gérard Govaert, enseignant suiveur de l'UTC, pour sa visite sur le lieu du stage.

Je tiens à témoigner ma reconnaissance aux personnes suivantes : Monsieur Olivier Cayrol, directeur adjoint de Logilab, pour son intégration au sein de l'entreprise et ses conseils sur la communication du projet au sein de la communauté CubicWeb ; Messieurs Nicolas Chauvat, PDG de l'entreprise Logilab, Adrien Di Mascio, directeur du Département Web sémantique, Sylvain Thénault, Directeur de l'Agence de Toulouse et Anthony Truchet, chef de projet au sein du département Informatique Scientifique, pour l'intérêt qu'ils ont porté au sujet de mon stage ; Madame Katia Saurfelt, chef de projet au sein du Département Web Sémantique, pour sa gentillesse, l'accueil au sein de son bureau et son aide tout long de ce stage ; Monsieur Alain Leufroy, chef de projet au sein du Département Informatique scientifique, pour sa formation sur le Python scientifique et sa bonne humeur quotidienne ; ainsi que l'ensemble du personnel de Logilab pour sa sympathie et son accueil.

Résumé technique

Le stage que j'ai effectué s'est déroulé au sein du département "Web Sémantique" de Logilab, entreprise spécialisée dans l'informatique numérique et la gestion des connaissances. Logilab propose des outils de gestion de connaissances basés sur le logiciel libre CubicWeb. Le sujet de ce stage porte sur la mise en place d'inférences sur la plateforme CubicWeb, à la base de la création des applications produites par Logilab. Il s'agit d'offrir la possibilité de traiter l'ensemble des données d'une application avec des règles, en vue d'en extraire des nouvelles informations pertinentes. Afin de mettre en place un mécanisme de règles, mon stage s'est principalement articulé autour de deux actions : la mise en place d'un système expert et la mise en place de règles pour permettre une réécriture de requêtes RQL, langage d'interrogation et de modification de données.

Dans un premier temps, la réalisation d'un système expert avec Pyke a permis d'inférer sur les données de l'application. Pyke est un moteur d'inférences écrit en Python qui introduit de la programmation logique inspirée de Prolog. La base de faits est composée des données de la base de données de l'application. Les données inférées selon les règles mises en place peuvent alors être intégrées à la base de données ou stockées séparément. Dans le cadre de la plateforme CubicWeb, les règles suivent un algorithme de chaînage-avant par saturation. Afin de réaliser ce système expert, mes compétences en matière d'algorithmique et de programmation logique ont été sollicitées.

Dans un deuxième temps, la mise en place de règles au niveau de la modélisation des données a pour objectif de faciliter l'interrogation de la base de données et d'en extraire de nouvelles données déduites de celles existantes. Ce mécanisme, apportant de la flexibilité à la modélisation de données, a nécessité des ajouts de fonctionnalités à la plateforme CubicWeb et de mettre en œuvre un mécanisme de réécriture de requêtes RQL. En effet, le RQL utilise des règles définies dans le modèle de données afin d'accéder aux données déduites de cette règle. Pour cela, il m'a fallu comprendre et manipuler le graphe de requêtes RQL.

Les méthodologies et outils, utilisés tout au long de ses vingt-quatre semaines, sont une méthode de développement logiciel agile, le *test driven development*, Mercurial, Python, Javascript, CubicWeb, Debian.

Table des matières

Introduction

Les stages en entreprise ou laboratoire font partie intégrante de la formation d'ingénieur de l'Université de Technologie de Compiègne (UTC). La finalité essentielle est d'acquérir une expérience dans le milieu professionnel, afin de s'initier à la pratique du métier d'ingénieur et d'en saisir les enjeux. Les travaux confiés sont l'occasion de développer mes connaissances et de mettre en application les enseignements théoriques et pratiques reçus ainsi que de développer mes facultés d'adaptation, d'analyse et de synthèse. Ce premier stage de six mois a eu lieu au sein de l'entreprise Logilab du 2 septembre 2013 au 14 février 2014, dans le département Web sémantique. Logilab est une entreprise d'une vingtaine de personnes qui développe des logiciels, propose du conseil et des formations de haut niveau dans les domaines de l'informatique scientifique et du Web sémantique. Ce rapport vise à présenter l'entreprise, la méthodologie employée, les travaux effectués, ainsi que mon retour d'expérience.

Première partie

Présentation de l'entreprise

Chapitre 1

Logilab : une entreprise acteur du logiciel libre

1.1 Activité et domaines de compétences

Logilab est une société anonyme (S.A) d'une vingtaine de personnes créée en septembre 2000 dont l'activité consiste à développer des logiciels, les intégrer dans les systèmes d'information clients et à proposer du conseil et des formations de haut niveau dans les domaines de l'informatique scientifique et du Web sémantique [1]. Les principales formations proposées par Logilab concernent la programmation Python, la programmation scientifique en Python (e.g analyse de données, statistiques, fouilles de données, calcul numérique), programmation d'interfaces graphiques, programmation Web, le développement logiciel (e.g gestion des sources, gestion d'infrastructure) et le système d'exploitation Debian GNU / Linux.

Dans le domaine de l'informatique scientifique, elle propose à ses clients diverses prestations comme la simulation numérique, le calcul hautes performances ou l'analyse de données. Dans le domaine du Web sémantique, elle offre des services comme la gestion de connaissances, l'agrégation de bases de données et des outils de recherche et de veille. Les experts de Logilab ont tous une formation d'ingénieur ce qui leur permet de maîtriser les domaines de l'informatique scientifique afin d'établir des solutions adaptées aux problématiques de chacun de ses clients.

1.2 Le choix de l'*open source* et du libre

Un acteur du logiciel libre Logilab se place comme un acteur du Logiciel Libre en mettant à disposition gratuitement certains de ses développements sur ses sites `logilab.org` et `cubicweb.org`. L'entreprise développe un nombre important de logiciels, bibliothèques et modules, dont beaucoup sont publiés gratuitement sur le site `logilab.org` en tant que logiciels libres. La plupart de ces logiciels sont volontairement de taille réduite, l'objectif étant de disposer de composants réutilisables et interopérables qui offrent des fonctionnalités précisément circonscrites. Parmi ces logiciels, on peut compter :

- des modules pour améliorer la conception et la qualité de code Python : *pylint* et *apycot*
- le système de gestion d'informations : *CubicWeb*
- la plateforme d'assistants intelligents : *Narval*
- le gestionnaire de version décentralisée : *Mercurial*

Des publications *open source* La désignation *open source* s'applique aux logiciels dont la licence respecte les possibilités de libre redistribution, d'accès au code source et de créer des travaux dérivés. Pour Logilab, l'*open source* est une vraie stratégie. En effet, l'accès au code permet une visibilité des contributions de l'entreprise aux projets *open source* ce qui permet d'attirer et de recruter des développeurs talentueux, de

1. Le Web sémantique est un mouvement Mouvement qui vise à permettre aux machines de comprendre la sémantique, c'est-à-dire la signification, de l'information sur le Web.

faire connaître l'entreprise et d'obtenir de nouveaux clients. De plus, la maintenance d'un programme est partagée entre tous les utilisateurs du logiciel ; la fiabilité et la robustesse du code sont donc accrues.

Ce choix de l'*open source* est un engagement, une conviction car l'entreprise, à l'instar des nombreux éditeurs et prestataires informatiques, pourrait maintenir une dépendance des clients en ne leur permettant pas le contrôle de leurs programmes informatiques. Or dans la mesure du possible, elle développe, utilise, adapte et contribue aux logiciels *open source*.

1.3 Contexte économique et situation sur le marché

1.3.1 Bilan économique de l'entreprise

Chiffre d'affaires Le chiffre d'affaires d'une entreprise est le total des ventes d'une entreprise sur un exercice comptable.

Bénéfices Le bénéfice d'une entreprise est la différence positive entre : d'une part la somme des recettes réelles (produits comptabilisés) et produits calculés (factures à émettre, charges comptabilisées d'avance) et d'autre part la somme des dépenses réelles (charges comptabilisées) et des coûts calculés (amortissements, provisions, factures à recevoir...)

1.3.2 Position de Logilab sur le marché

De nos jours, l'informatique est au cœur de toutes les entreprises. De nombreuses sociétés de services informatiques ont vu le jour ces dernières années. Les principaux concurrents de Logilab, sont de grandes SSII comme Capgemini, Accenture, Logica, Atos ou Steria. Afin de se démarquer sur un marché saturé, Logilab a mis l'accent sur sa spécialisation dans les domaines scientifiques et sémantiques ainsi que dans le développement du langage Python. Toutefois, sa spécialisation dans le domaine de l'informatique scientifique et le Web sémantique ne leur permet d'apparaître que sur ce marché.

Logilab étant spécialisée dans le domaine scientifique et la gestion de données, l'entreprise possède de nombreux clients dans des domaines très divers comme : l'énergie, la télécommunication, les établissements publics, l'enseignement, l'aéronautique et l'espace. Ces principaux clients sont EDF, la BnF [1], l'IRSN [2], la SNCF, la RATP, France Télécom R&D, Bouygues Telecom, mais aussi des établissement publics comme le Ministère de l'Intérieur.

Logilab participe ainsi à des projets d'envergure comme par exemple :

- `http://data.bnf.fr/`
- `http://brainomics.net/`
- `http://collections.musees-haute-normandie.fr/collections/`

1.3.3 Analyse des risques encourus par l'entreprise

Comme la majorité des petites entreprises, il existe des risques inhérents à l'activité de Logilab, détaillés ci-après, liés aux produits, aux équipes et aux clients. La connaissance et la compréhension de ces risques sont essentielles afin de les maîtriser impérativement. Le risque majeur, concernant les produits de Logilab, serait l'incapacité à financer leur développement. Par ailleurs, le produit phare de Logilab CubicWeb ne possède pas une communauté propre importante. Or dans le monde des logiciels *open source*, une forte communauté est un atout important. Logilab est composée d'une vingtaine de personnes, le départ d'une personne clé est donc un risque important pour le fonctionnement de l'entreprise. De plus, l'évolution de carrière étant limitée dans une entreprise de cette taille, les départs se renouvellent fréquemment. Il existe également un autre risque lié au pilotage des développeurs : les entreprises de logiciels *open source* attirent des personnes qui adhèrent à une vision libertaire et peuvent avoir des difficultés à accepter le principe hiérarchique d'une entreprise, même si dans le contexte de Logilab la pression hiérarchique est faible. De part sa petite taille,

1. Bibliothèque nationale de France
2. Institut de Radioprotection et de Sûreté Nucléaire

l'existence de Logilab repose sur le financement de ses clients ; la perte de plusieurs clients ou encore une baisse de réputation qualité serait un risque majeur pour l'entreprise.

Chapitre 2

Organisation générale

2.1 Organigramme et postes

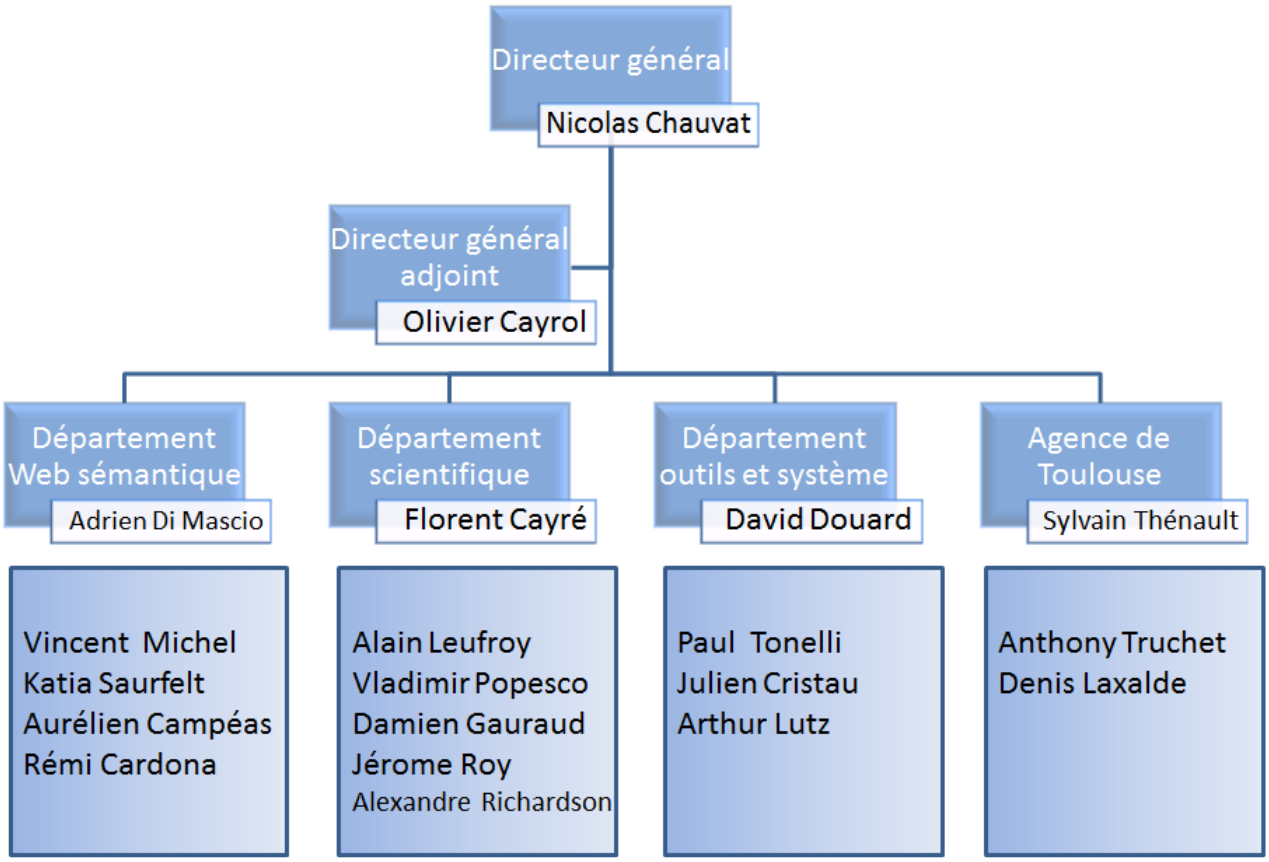


FIGURE 2.1 – Organigramme de l'entreprise Logilab

Organigramme et flexibilité Logilab est divisée en quatre départements, comme le montre la figure ci-dessous : le département Web sémantique (SEM), le département informatique scientifique (DIS), le département outils et système (DOS) et l'Agence de Toulouse, département transversal. Les départements ne sont pas hermétiques : les affections sur les projets priment sur la division en département afin d'acquérir une flexibilité maximum. Logilab a récemment ouvert un établissement à Toulouse qui comptent aujourd"hui trois personnes. Cette expansion de Logilab à Toulouse permet un rapprochement avec les clients de cette région.

Profil des employés Le profil des personnes recrutées est varié. En effet, les personnes travaillant dans l'entreprise sont toutes ingénieurs mais ont des formations scientifiques très diverses. On retrouve des personnes spécialisées dans les mathématiques, l'électronique, la mécanique, la physique des plasmas, le traitement du langage naturel, etc. Cette diversité des profils est capitale pour Logilab qui, étant spécialisée dans la vente de compétences et d'expertises, doit disposer d'interlocuteurs utilisant le même vocabulaire et maitrisant les mêmes concepts scientifiques que ses clients.

2.2 Le département Web sémantique

La recherche est au cœur de Logilab afin de rester compétitive sur un marché en expansion. Chaque département intègre une section recherche. Ce stage, effectué au sein du département Web sémantique, s'inscrit dans cette démarche de recherche et d'amélioration interne. Les principaux projets réalisés par le département au cours de ce stage sont les projets data.bnf.fr, Fevis, Brainomics.

2.2.1 La Bibliothèque nationale de France

Le projet data.bnf.fr s'inscrit dans une démarche d'ouverture des données sur le Web. Afin d'encourager la réutilisation des données brutes, cette ouverture a un aspect technique, avec le respect des standards du « Web sémantique » et ouverture sur le « Web de données » définis par le W3C, et un aspect juridique. En effet, les données sont placées sous Licence ouverte de l'État autorisant la réutilisation libre, y compris commerciale, avec mention de la source. Les outils du « Web de données » répondent à des problèmes anciens des bibliothèques, en particulier autour de la gestion de formats divers et de l'échange de données. Ils offrent une présence nouvelle de ces ressources sur le Web en les rendant plus facilement accessibles, réutilisables et en les liant à des ressources complémentaires.

2.2.2 Fevis

Le projet Fevis est un projet pour la Fédération des Ensembles Vocaux et Instrumentaux Spécialisés. La particularité du projet repose sur une aide à la saisie et la création d'un moteur de recherche sur les compositeurs.

2.2.3 Brainomics

Brainomics est un projet de recherche collaboratif financé en partie par l'ANR (Agence Nationale pour le Recherche) qui apporte des solutions méthodologiques et logicielles pour l'intégration de la neuro-imagerie et des données génomiques. La génomique est une discipline de la biologie moderne qui étudie le fonctionnement d'un organisme à l'échelle du génome. Les ensemble de données de la neuro-imagerie et génomiques contiennent un très grand nombre de données cliniques et génétiques.

Ce projet a deux objectifs principaux :
- la création d'un outil de gestion de données issues de la neuro-imagerie (imagerie fonctionnelle et anatomique), de la génomique, et de la clinique (questionnaires comportementaux) ;
- sa mise en œuvre sur des architectures d'algorithmes de fouille de données et d'apprentissage statistique pour ces données.

Le point fort de ce projet est l'accès aux informations pertinentes dans un grand nombres de données. En effet, le modèle de données est optimisé pour des grand volumes. Brainomics apporte donc une solution *open source* pour gérer les bases de données d'imagerie cérébrale et méta-données associées.

2.3 Composition de l'équipe projet

Chaque membre de Logilab prend part à un projet soit en tant que développeurs soit en tant que responsable soit les deux. La composition type d'une équipe de développement associée à un projet est la suivante :
- un chef de projet, interlocuteur principal du client, responsable des livrables et du respect des délais.

- un responsable qualité de Logilab, s'assure de la qualité des livrables
- l'équipe en charge de la réalisation, composée de développeurs et d'experts, selon le profil du projet.

Les postes de développeur et de chef de projets ne sont pas exclusifs. Un chef de projet est également un développeur, qui peut travailler en parallèle sur un projet où il n'est pas chef de projet. Cela lui permet ainsi de mieux comprendre et appréhender les difficultés rencontrées par l'équipe de développement.

Deuxième partie

Méthodologie de l'entreprise et organisation du travail

Chapitre 3

Gestion de projet

3.1 Méthode agile

Une méthode agile fonctionne sur la base de l'itératif et de l'incrémental. Les tâches sont réalisées progressivement, par ordre de priorité, avec des phases de contrôle et d'échange avec le client. Les méthodes agiles, plus pragmatiques que les méthodes traditionnelles, impliquent au maximum le client ce qui permet une grande réactivité à ses demandes. Elles visent la satisfaction réelle du client, même si ce dernier n'a pas défini précisément ses attentes lors de la rédaction du contrat. En effet, ses attentes vont se préciser au cours de la réalisation du projet. Logilab privilégie les méthodes agiles pour ses clients et son développement interne. Toutefois, bien qu'elle soit spécialisée dans les méthodes agiles et offre même des formations à ce sujet, elle laisse le choix à ses clients entre une méthode de gestion de projet traditionnelle et une méthode agile.

Les méthodes agiles présentent l'avantage de travailler sur des sous-projets, moins complexes que le système complet, de les intégrer au fur et à mesure et de valider régulièrement leur fonctionnement correct par rapport aux besoins et exigences des utilisateurs finaux. Le client suit donc l'évolution du projet et peut réorienter le projet à tout moment ce qui diminue le risque de non-satisfaction. La limite des méthodes agiles réside dans la disponibilité et l'engagement du client. Ce dernier doit être impliqué ; il doit effectuer des tests, rediriger les spécifications si nécessaire, tout au long du projet.

Dans le cadre de ce stage, les développements effectués visent à l'amélioration des outils de Logilab et donc appartiennent au développement interne. La méthode agile a donc été appliquée tout au long de ce stage. Cette méthode fut particulièrement efficace car elle représentait une sécurité pour garantir la satisfaction de mon tuteur et de la direction ainsi que pour garantir la réalisation du projet.

3.2 Procédures de développement du projet

Le projet est découpé en itérations. Une itération dure environ deux semaines. Une itération comprend typiquement les étapes suivantes :

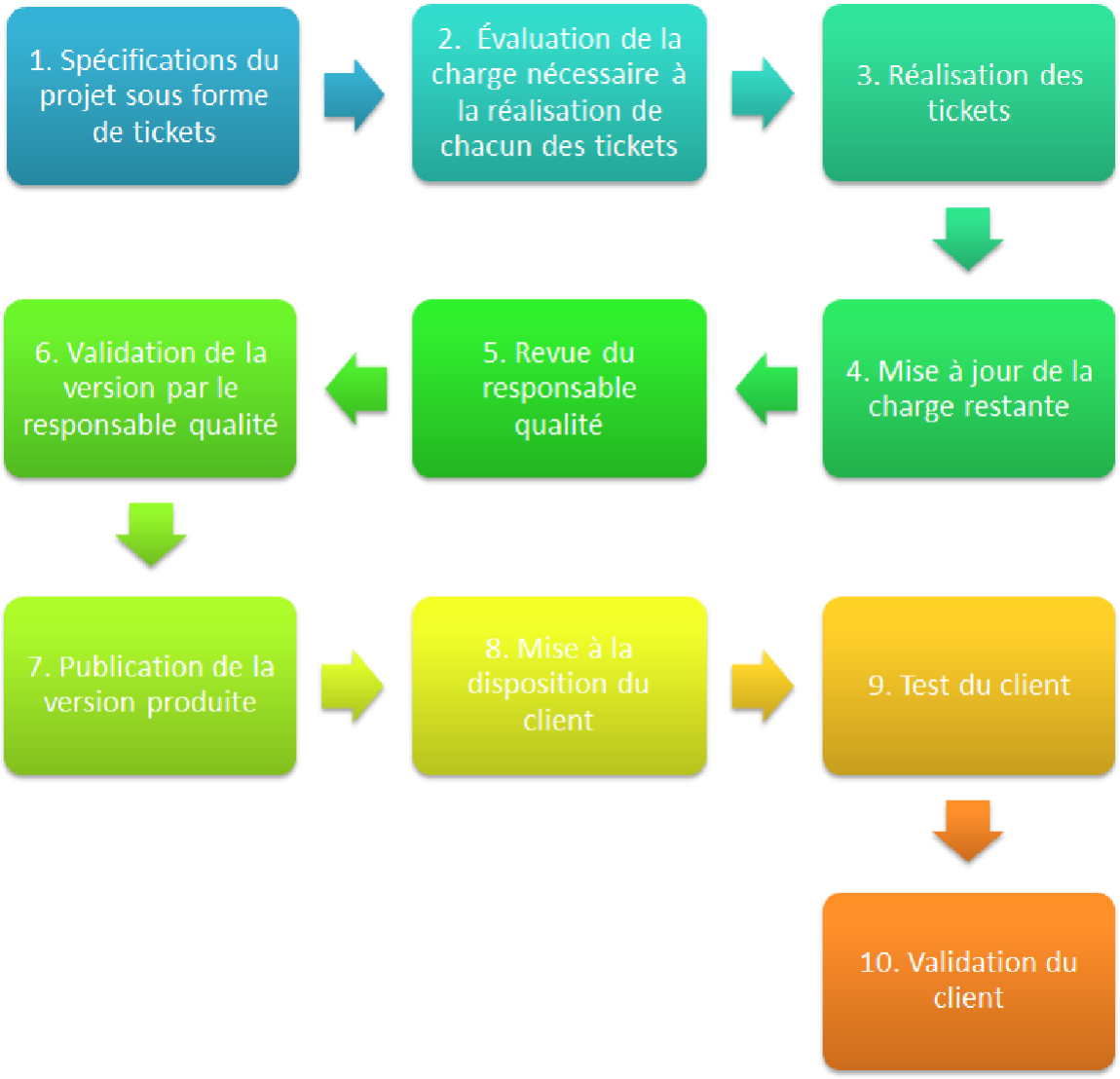


FIGURE 3.1 – Exemple d'itération d'un projet

Le processus de développement permet de disposer d'une version fonctionnelle du projet à chaque fin d'itération. L'ordonnancement des tâches, le versionnement et la planification du développement sont explicités ci-après.

Ordonnancement des tâches Dans le contexte d'un projet client, l'itération des tâches est déduite du cahier des charges et se fait par l'élaboration de tickets, précisant la durée de la tâche, son importance et son statut. Un exemple de ticket se trouve ci-dessous. Un ticket est caractérisé par le nom de la personne qui traite le ticket, l'état du ticket, une description et des commentaires. Il peut avoir quatre états : *en cours*, *ouvert*, *fait* ou *en attente de retour*. La description du ticket reflète le besoin du client. Il est mis à jour si le besoin évolue ou s'il est précisé. Sur chaque ticket, un mécanisme de commentaires permet un dialogue entre Logilab et le client (demandes de précisions, alternatives techniques, compléments d'information, etc.) De plus, la charge de réalisation de chaque ticket (en jours.hommes) est évaluée par les personnes susceptibles de le réaliser afin d'être la plus réelle possible. Travaillant dans le cadre d'un projet interne, l'élaboration des tickets décrivant les travaux que je devais réaliser a été directement faite par mon tuteur. La version du projet est construite à partir de ces tickets. L'ordonnancement des tâches par des tickets apporte dynamisme et flexibilité.

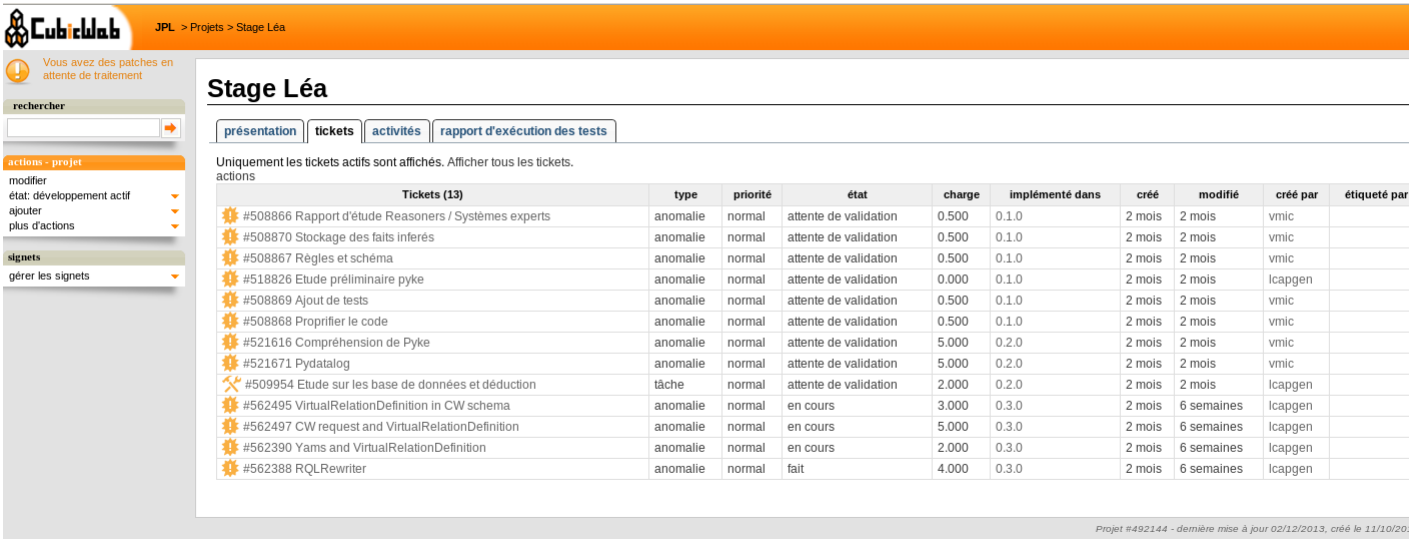


FIGURE 3.2 – Exemple de ticket

Version et livraison Dans le cadre d'une méthode agile, chaque projet est découpé en versions, livrées régulièrement au cours du projet. Les versions du projet peuvent être *"prévues"*, *"en cours de réalisation"*, *"livrées"*, ou *"recettées"*. La procédure de développement d'une version est explicitée par des tickets qui indiquent les différentes tâches à effectuer. Avant chaque livraison, la version est testée de manière approfondie. La version qui est livrée est clairement identifiée comme telle dans le gestionnaire de version Mercurial, contenant le code source. Les caractéristiques de cet outil essentiel seront détaillées ultérieurement (Cf paragraphe 5.2.1). De plus, chaque livrable est accompagné de rapports qui identifient les tests unitaires qui échouent. Un test unitaire vérifie, sur un morceau de code précis, qu'un exemple particulier de la problématique, couvert par le test, a le comportement attendu. La validation d'une version d'un projet aboutit à sa publication. Il existe un entrepôt public pour les logiciels libres publiés par Logilab, un entrepôt sécurisé pour les logiciels développés en interne, et des entrepôts sécurisés pour chaque client. L'ensemble de mon code source a donc été déposé dans l'entrepôt sécurisé consacré aux projets internes.

3.3 Contrôle et échange avec le client

Dans un projet suivant une méthode agile, l'implication du client conditionne la réussite du projet. Les clients doivent donc être disponibles pour définir les versions et les tickets, pour tester les versions livrées et effectuer un travail de spécification tout au long du projet. Dans le contexte du stage, le projet étant un projet interne, la direction était très impliquée et disponible.

Dans le cadre d'un projet client, toutes les deux semaines environ une version préliminaire du projet est livrée afin d'obtenir le retour du client. Il peut alors préciser son besoin et échanger sur le travail rendu. Cela permet de garantir qu'il est satisfait du travail rendu. Dans le cadre de ce stage, chaque ticket était validé par le tuteur et chaque livraison était suivie d'une réunion avec la direction pour établir la suite du projet en fonction des attentes et de la réalisation. Cela a permis de visionner l'avancement du travail et de rediriger le projet au moment voulu.

La direction, comme les clients de Logilab, a eu la possibilité de suivre de façon précise l'avancée des travaux à travers le gestionnaire de sources contenant le code source et à travers l'extranet. L'extranet contient la description de l'ensemble des versions du logiciel et de leur état d'avancement. Il constitue une photographie précise et toujours d'actualité du projet et de son avancement. Les anomalies éventuelles peuvent être rapportées sur l'extranet. Une nouvelle version qui contient la correction d'une ou plusieurs des anomalies signalées peut alors être définie.

Chapitre 4

Plan Qualité

4.1 Objectifs qualité

Les objectifs qualité concernent tous les aspects du service fourni au client et visent à satisfaire ses besoins sur les plans de l'exploitation, de la maintenance et de l'assistance à l'utilisation. À chaque besoin correspond des exigences nécessaires à l'assurance de la qualité. Ainsi concernant l'exploitation, Logilab s'engage à fournir une adéquation fonctionnelle, une stabilité de la version, une facilité de maintenance et une disponibilité pour l'assistance à l'utilisation.

4.2 Mise en œuvre du plan Qualité

Pour respecter ces exigences, Logilab assure la traçabilité et conserve un historique des événements et des actions, conçoit des systèmes modulaires réutilisables, teste tous les livrables et développe les compétences des équipes, à travers des formations, des conférences et du travail en groupe.

4.2.1 Traçabilité et conception modulaire

Pour assurer la traçabilité, Logilab utilise des outils et systèmes qui permettront de conserver un historique des actions de développement (gestion de versions), des actions de déploiement (gestion de configuration) et de documentations et de dialogues entre partenaires du projet (gestion des connaissances). Cette traçabilité permet de résoudre plus rapidement une anomalie et de savoir si elle a déjà été rencontrée. La conception modulaire facilite le remplacement et l'évolution des composants, ainsi que la répartition des responsabilités, que ce soit entre les sous-systèmes ou au sein de l'architecture globale.

4.2.2 Tests

Tous les éléments et livrables produits sont testés avec des jeux d'essai validés par le client et passés en revue pour garantir qu'ils correspondent à leurs spécifications. Ces validations seront appliquées tant à la documentation, qu'au déploiement des systèmes ou aux logiciels. Dans le cas des logiciels, Logilab utilise des outils d'analyse statique de code, des outils de couverture de test, et des plateformes de tests automatiques effectuant des tests unitaires, des tests d'intégration et des tests de non-régression. Toutes les nuits, un outil de test automatique récupère, dans l'outil de gestion des sources, la version du projet en cours de développement puis va effectuer une série de tests permettant de s'assurer de la qualité :

- lancement automatique des tests disponibles dans le projet (tests unitaires et tests d'intégration),
- calcul de la couverture de test,
- production automatique de la documentation,
- production automatique des éléments déployables (par exemple les paquets binaires installables).

Le résultat de cette série de tests est disponible dans l'extranet.

4.2.3 Accroissement des compétences

Formations Le maintien ou l'accroissement des compétences des équipes impliquées dans le projet permet d'assurer un suivi de la qualité dans le temps. Les membres du personnel peuvent ainsi s'inscrire à des formations comme celles dédiées au management ou à Hadoop [1].

Vsprint Le vsprint correspond à une plage horaire hebdomadaire (les vendredi après-midi) où chaque membre du personnel est affecté par son directeur de département à un binôme et doit travailler sur un ensemble de tickets, généralement lié au développement d'outils internes. L'objectif est de partager les bonnes pratiques de développement, de faire connaître les projets, de mutualiser le code et les idées et d'améliorer de façon continue les outils internes à l'entreprise. Cette pratique est pertinente car elle permet de diffuser la connaissance entre les développeurs et d'imposer une plage horaire pour le développement des outils de l'entreprise afin qu'elle reste compétitive.

4.3 Bilan et retours d'expérience d'un projet

Chaque fin de projet est l'occasion d'effectuer un bilan et de dégager des pistes d'amélioration de la méthode de travail et du plan qualité. La pertinence de ces pistes d'amélioration est jugée par le responsable qualité de Logilab et un groupe de travail dédié. Les pistes retenues sont immédiatement mises en application. Le plan qualité et la méthode de travail font donc tous deux l'objet d'améliorations constantes directement issues des retours d'expérience.

1. Hadoop est un framework Java libre destiné à faciliter la création d'applications distribuées et échelonnables. Il permet aux applications de travailler avec des milliers de nœuds et des pétaoctets de données.

Chapitre 5

Outils utilisés

5.1 Cubicweb

5.1.1 Les valeurs ajoutés de CubicWeb

CubicWeb est une plateforme logicielle libre implémentée par l'entreprise Logilab et à la base de la majorité de leurs applications ; il m'a donc été amené à utiliser ce framework [1] quotidiennement.

Des composants assemblables et adaptables La plateforme CubicWeb est composée de fonctionnalités préprogrammées, développées en Python, appelées "cubes". Ces "cubes" adaptables sont assemblables et permettent de construire des applications facilement et rapidement. De cette manière, elle permet aux développeurs de se concentrer sur les spécificités de leur application plutôt que d'avoir à réinventer les briques essentielles de l'import, la fusion, la publication, l'interrogation et la visualisation de données.

Prototypage rapide La plateforme permet de développer rapidement des applications de gestion de données. En effet, le développeur dispose de fonctionnalités automatisant, à partir du modèle des informations, les tâches habituelles comme la gestion de la base relationnelle sous-jacente, la génération d'une interface Web par défaut et la migration entre versions successives. Cette technique de réutilisation de code améliore considérablement le temps de production d'application Web et permet un prototypage rapide permettant de répondre aux critères des méthodes agiles.

Une plateforme centrée sur les données L'avènement du Web sémantique et de l'Open Data [2] nécessite de disposer d'outils adaptés pour développer des applications centrées sur les données. *CubicWeb* place les données au cœur du système en offrant la possibilité d'importer des données facilement, de les mettre en relation lorsqu'elles proviennent de sources disjointes, de les republier et de faciliter leur interrogation et leur visualisation.

Une liberté pour le client CubicWeb s'appuie sur le modèle métier des informations gérées qui laisse à l'utilisateur la possibilité de naviguer dans le graphe des informations, de sélectionner des données et de choisir la vue qui lui permettra de les visualiser.

Respect des standards ouverts d'Internet CubicWeb respecte les standards ouverts d'Internet afin de simplifier les communications et les échanges. Elle suit les standards du W3C [1] (RDF [2], SPARQL [3], HTML5 [4], CSS3 [5], Responsive Design [6]) et peut gérer nativement plusieurs modèles de données faisant office de standards

1. framework : ensemble de composants logiciels permettant de créer l'architecture d'un logiciel
2. Open data : données diffusées de manière structurée selon une méthodologie et une licence ouverte garantissant son libre accès et sa réutilisation par tous, sans restriction technique, juridique ou financière.

de faits (FOAF [7], SIOC [8], DOAP [9]).

5.1.2 Architecture générale de CubicWeb

L'architecture de CubicWeb sépare clairement la partie dédiée au client (*Web engine*) et l'entrepôt des données. Elle est représenté par la figure suivante :

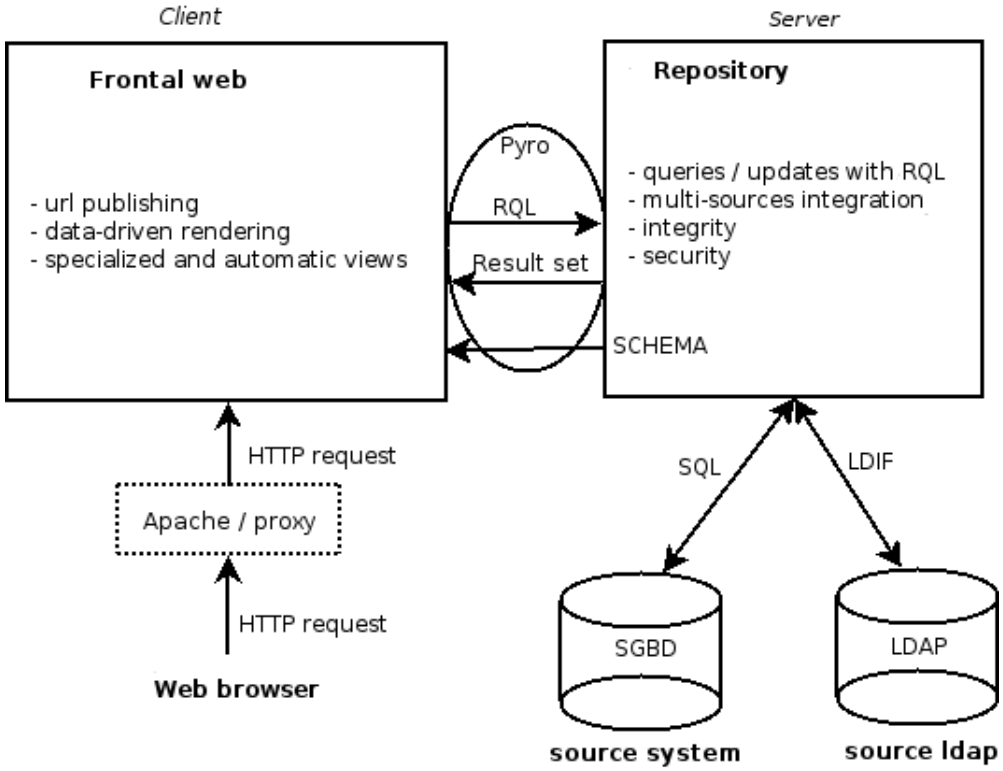


FIGURE 5.1 – Schéma de l'architecture de CubicWeb

L'entrepôt de données L'entrepôt de données encapsule et assemble un accès à une ou plusieurs sources de données : données SQL, annuaires LDAP, autres applications de CubicWeb, systèmes de fichiers etc. Toutes les interactions avec l'entrepôt de données sont effectuées en utilisant le langage *Relation Query Language*, que nous développerons ultérieurement. Il fédère les sources de données. L'émetteur de la requête ignore si une requête s'étend sur plusieurs sources de données ou nécessite l'exécution des sous-requêtes. Cette séparation entre l'entrepôt de données et l'interface utilisateur simplifie l'écriture de la requête pour l'utilisateur et permet de cacher la complexité du multisource.

1. W3C (World Wide Web Consortium) est un organisme de normalisation à but non lucratif chargé de promouvoir la compatibilité des technologies
2. RDF (Resource Description Framework) est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de ces descriptions.
3. SPARQL est un langage de requêtes et un protocole qui permet de rechercher, d'ajouter, de modifier ou de supprimer des données RDF disponibles à travers Internet.
4. HTML5 est la cinquième version du format de données HTML conçu pour représenter les pages Web
5. CSS3 est un langage servant à décrire la présentation des documents HTML et XML.
6. Responsive Design est une notion de conception de sites Web qui permet à l'utilisateur de consulter le même site Web à travers une large gamme d'appareils (moniteurs d'ordinateur, smartphones, tablettes, TV, etc.) avec le même confort visuel.
7. FOAF est un vocabulaire RDF permettant de décrire des personnes et les relations qu'elles entretiennent entre elles. C'est une application du Web sémantique.
8. SIOC est un vocabulaire utilisant RDF permettant de décrire des objets couramment utilisés sur les sites communautaires et leurs relations. SIOC est une application du Web sémantique pour décrire des blogs, des forums, des wikis...
9. DOAP est un vocabulaire RDF décrivant les projets logiciels

Modélisation des données Les données des applications de CubicWeb sont modélisées suivant un schéma. Le module python *Yams*, qui signifie *Yet Another Magic Schema*, définit le schéma générique des relations et des entités de manière simple et générique. Une entité est un objet qui continet une séquence d'attributs, un identifiant (*eid*) et des relations. Un type de relation est utilisé pour définir une relation binaire orientée entre les types d'entité . La partie gauche d'une relation est nommée le sujet et la partie droite est nommée l'objet. Une définition de relation est un triplet (type de sujet de l'entité, type de relation, type d'entité de l'objet) associé à un ensemble de propriétés telles que la cardinalité, les contraintes, etc.

Les données de l'application sont ensuite modélisées suivant un schéma entités-relations, où chaque entité ou relation hérite du modèle correspondant dans *Yams*. Cette modélisation des données se trouve dans un composant de CubicWeb appelé "cube" (Cf paragraphe 5.1.3) L'exemple ci-dessous modélise l'organisation de projet :

Exemple 5.1 – Schéma de l'organisation d'un projet

```
class Projet(EntityDefinition):
    nom : String()
    domaine : String()
    duree : int()

class Personne(EntityDefinition):
    nom : String()
    personne : String()

class responsable_de(RelationDefinition)
    subject : Personne
    object : Projet
```

Requêtes RQL RQL est le langage informatique d'exploitation de bases de données, utilisé par *CubicWeb*. Grâce à une syntaxe de haut niveau, il permet d'accéder aux données stockées dans les bases de données et d'explorer les entités, attributs, relations du schéma très facilement. RQL utilise un solveur pour automatiquement déterminer les types possibles pour une variable en fonction du modèle de données et des contraintes exprimées dans la requête. Il se base sur le concept de relation binaire ordonnée : <subject> <relation> <object> . Voici ci-dessous différents exemples de requêtes retournant respectivement : les 10 derniers projets créés, les projets commençant par C, et le nombre de personne responsable d'un projet.

Exemple 5.2 – Requêtes RQL

```
Any X ORDERBY XD DESC LIMIT 10 WHERE X creation_date XD, X is Projet
Any X WHERE X is Projet, X name LIKE 'C%'
Any X,COUNT(P) GROUPBY X WHERE P responsable_de X
```

Une requête retourne un *ResultSet* qui est un tableau à deux dimensions (liste de listes) où le nombre de lignes correspond au nombre de résultats et le nombre de colonnes correspond au nombre d'éléments sélectionnés dans la requête. La classe *ResultSet* expose, en plus d'une interface de tuple, les principaux attributs et fonctions suivants :

- description : description du résultat sous forme de liste de listes,
- entities() : itère sur les entités pour une colonne donnée,
- get_entity() : renvoie l'entité pour une ligne et une colonne données.

Dans CubicWeb, les requêtes RQL sont définies sous forme d'un graphe où chaque nœud peut être une variable, une constante, une condition d'existence ou une relation. Il existe quatre types d'arbre de requêtes

RQL répondant aux fonctionnalités classiques d'un langage de requête : sélection, modification, supression, insertion. Soit la requête suivante, composée de trois unions, une condition d'existence et d'une sous-requête :

Exemple 5.3 – Requête RQL

```
(Any X WHERE X relation Y)
UNION (Any X WHERE X relation Y, EXISTS(X name A))
UNION Any X WHERE X relation Y WITH X,Y BEING (Any A,B WHERE A name B))
```

Sous forme d'arbre elle s'écrit ainsi :

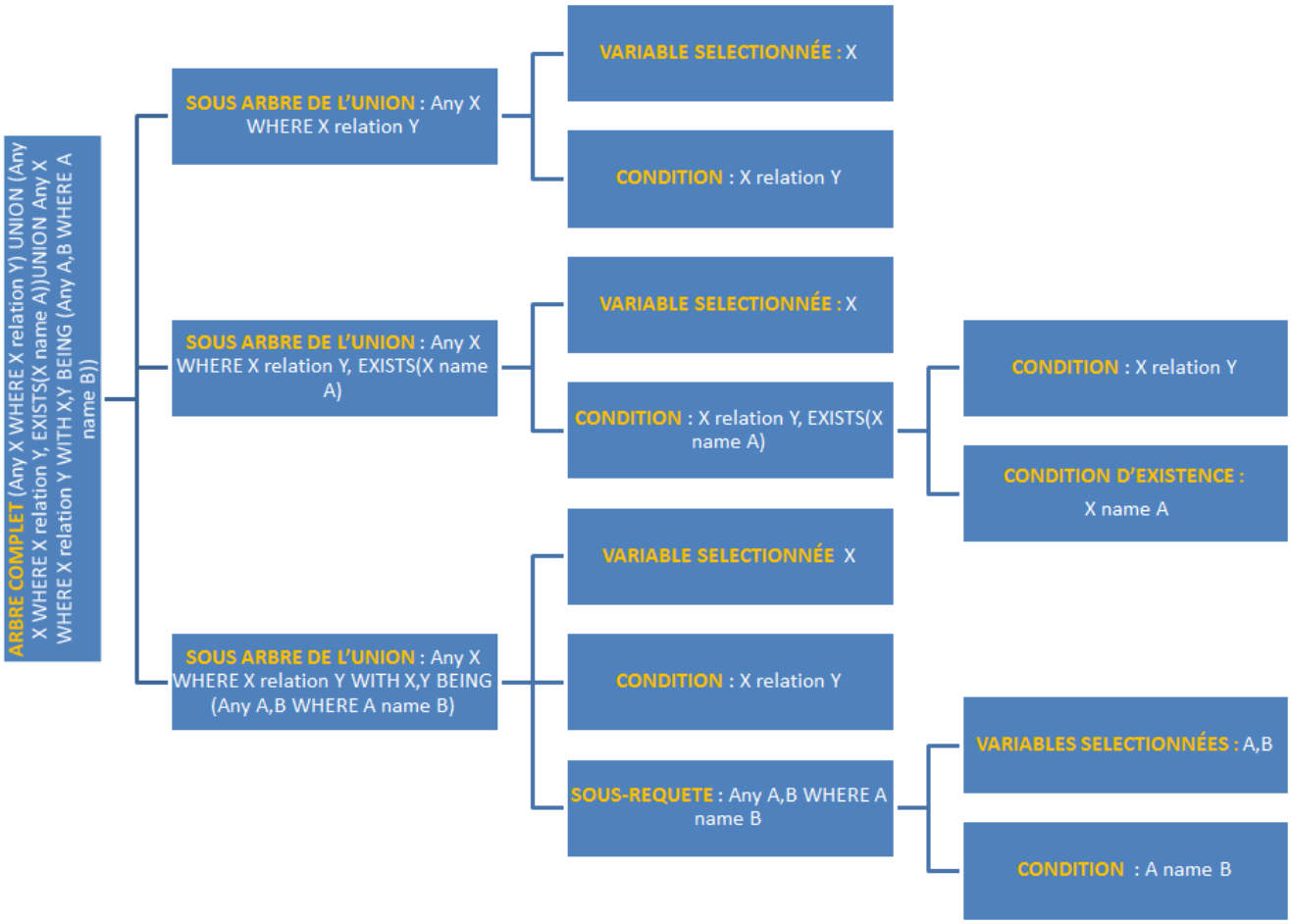


FIGURE 5.2 – Exemple de graphe d'une requête RQL

Le module python `rql` contient un analyseur et des fonctions utilitaires pour manipuler l'arbre de syntaxe de la requête. On a alors accès à différentes méthodes qui permettent par exemple d'ajouter des nœuds, des relations, des sous-requêtes etc.

Génération de tables SQL Les tables SQL sont générées automatiquement par Yams à partir du modèle de données décrit dans CubicWeb.

Web engine Le moteur Web répond à des requêtes HTTP et construit l'interface utilisateur. Par défaut, le moteur Web fournit une interface utilisateur sur la base du modèle de données de l'instance. Les entités peuvent être créées, affichées, mises à jour et supprimées.

5.1.3 Le cube

Un **cube** est le composant logiciel de base de CubicWeb, comportant trois parties principales : le modèle de données *(schema)*, la logique métier *(entities)*, l'interface utilisateur *(views)*, ainsi que des dépendances vers d'autres cubes. On peut distinguer deux types de cube : un cube métier qui définit un schéma et une logique propre à un concept métier particulier (cube *file* pour la gestion des fichiers, *person* pour la modélisation du concept de personne, etc) et un cube applicatif qui définit la logique nécessaire à une application complète, en s'appuyant sur un certain nombres d'autres cubes (métiers).

L'infrastructure de base d'un cube est :

data/	stocke les données Web (CSS, JS, images)
debian/	comprend les informations pour la construction de paquets Debian
entities.py	
i18n/	comprend les fichiers .po pour l'internationalisation du cube (français, anglais, espagnol...)
migration/	contient les fichiers de migration et le code qui sera exécuté après l'initialisation de la base de données
___pkginfo___.py	contient les méta-données du cube et les dépendances vers les autres cubes
hooks.py	fichier permettant la gestion d'évènements (comme la suppression d'une entité par exemple)
setup.py	fichier pour la distribution et l'installation du cube
schema.py	définit le modèle de données
test/	comprend les tests du cube
views.py	définit les vues de l'interface propre au cube
uiprops.py	indique les fichiers css et javascript à prendre en compte pour la mise en forme

Création et lancement d'une instance Dans la plateforme CubicWeb, pour créer une instance, la commande `cubicweb-ctl create <nom-cube> <nom-instance>` permet de créer une instance. La commande `cubicweb-ctl start <nom-cube> <nom-instance>` permet de lancer une instance. La création de l'instance du cube crée donc une application avec l'architecture par défaut.

Gestion d'évènements Les *hooks* sont des traitements à effectuer lorsque certains événements se produisent dans une application CubicWeb. Ils servent en particulier à ajouter un ensemble de contraintes ne pouvant être exprimées dans le schéma, ou permettre le calcul automatique d'un attribut ou d'une relation (e.g. à l'insertion ou la mise à jour d'une entité). Les évènement sur les données sont créées par des *hooks* mais ne sont exécutées qu'au `commit` ou au rollback d'une `transaction`. Ils se sélectionnent sur un ensemble des évènements suivants :

- **Évènements du serveur** :
  - démarrage ou arrêt du serveur,
  - ouverture ou fermeture de session.
- **Évènements sur les données** :
  - avant ou après la création d'une entité
  - avant ou après la mise à jour d'une entité
  - avant ou après la suppression d'une entité
  - avant ou après l'ajout d'une relation
  - avant ou après la suppression d'une relation

5.2 Autres outils techniques

5.2.1 Mercurial : gestionnaire de version

Mercurial est un logiciel de gestion de versions décentralisé qui administre les différentes versions d'un programme écrites par un ou plusieurs développeurs. Il conserve l'historique des modifications, permet de paralléliser des réalisations puis de les intègrer, et met en place des drapeaux marquant les versions publiées en fin d'itération. Dans un contexte de développement de logiciel en équipe et dynamique dans le temps, maintenir un ensemble de versions est indispensable pour l'aboutissement du projet. Mercurial permet la gestion avancée de l'historique des modifications dans un environnement distribué (plusieurs contributeurs).

Il gère aussi bien le code source des éléments logiciels que la documentation du projet. La gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. De fait, il existe plusieurs dépôts pour un même logiciel. Ce système est très utilisé par les logiciels libres. Cet outil, utilisé quotidiennement tout au long de ces vingt-quatre semaines, est indispensable pour le plan qualité du code. Voici ci-dessous un aperçu de l'interface de ce gestionnaire.

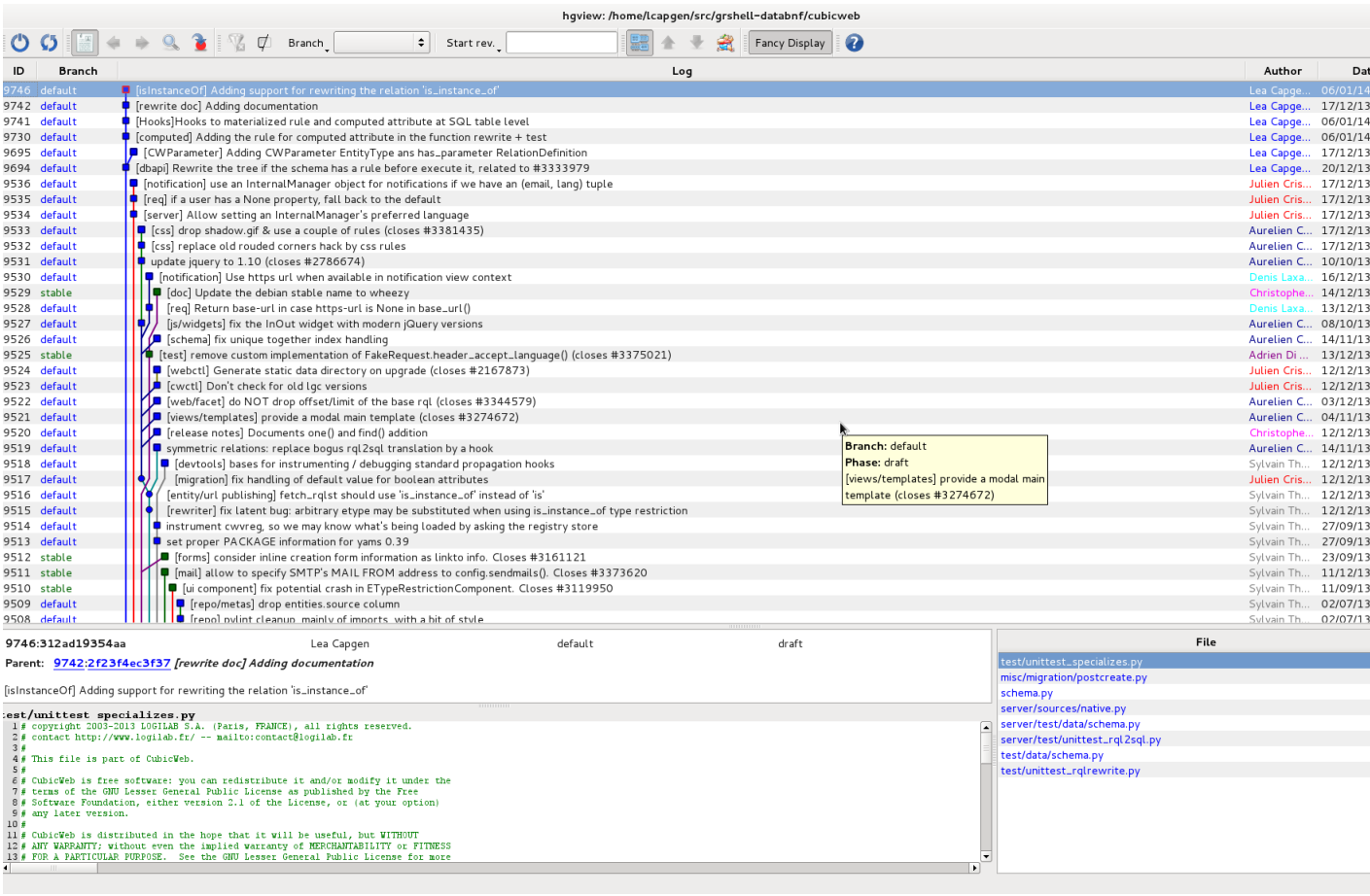


FIGURE 5.3 – Interface du gestionnaire de version Mercurial

5.2.2 Python : langage de programmation

Logilab a choisi de se spécialiser dans le langage de programmation Python car il présente de nombreux avantages comme la compacité, l'expressivité, la richesse de la bibliothèque standard et la portabilité de l'interpréteur. Tous ces avantages facilitent l'écriture d'un code logiciel maintenable et modulaire. Lors de ce stage, l'ensemble de mes implémentations a donc été écrites en Python.

5.2.3 Debian GNU Linux

Les solutions développées s'intègrent avec la distribution Debian GNU/Linux. De part son aspect non commercial et sa constitution uniquement en logiciel *open source*, le choix de cette distribution de Linux est en parfaite adéquation avec la vision de l'entreprise. De plus, elle est facile à administrer et à déployer sur de grands parcs informatiques.

5.2.4 Entrepôt de l'extranet

Le projet est accessible au client par un extranet sécurisé `http://hg.logilab.fr/master/clients/`. Dans un espace sécurisé, le site extranet contient chacun des projets menés avec le client. On y trouve les indicateurs et les informations nécessaires à la gestion du projet, comme l'avancement et la planification de la charge, les délais et le budget ; ainsi que les documents échangés comme les spécifications, les rapports d'anomalies et les demandes d'évolutions.

Le site extranet de gestion de projets mis en place par Logilab joue un rôle central dans le processus de travail et les relations avec le client. S'appuyant sur CubicWeb, il permet, outre les échanges d'informations, de conserver un historique complet du projet car il conserve toutes les dates de création et de changement d'états pour tous ces éléments. Le site extranet permet d'accéder à l'outil de gestion des sources et au répertoire de publication. Il est ainsi possible de récupérer le code en cours de développement, de naviguer dans ce code au travers d'une vue Web, de récupérer les différentes versions des documents publiés ou les différentes versions des éléments déployables (paquets binaires installables). Enfin, sur le site extranet, sont mis à disposition tous les documents échangés dans le cadre du projet : rapports, spécifications, comptes-rendus de réunions, etc. On y trouvera également le nom et les coordonnées des différents acteurs.

Ainsi l'extranet permet, d'une part, de juger des priorités et mieux maîtriser les risques, et d'autre part, de garantir la traçabilité des requêtes, la gestion des incidents et du suivi du projet dans son ensemble. Il propose une approche de la conduite de projet qui favorise la collaboration entre les parties et prône la transparence des travaux.

5.3 Outils de communication et planification interne

5.3.1 Forum

Outre les rencontres réelles au cours de réunions ou d'ateliers, Logilab met à disposition un ensemble de forums permettant aux développeurs de communiquer rapidement en cas de besoin et de tenir des réunions de travail, sans que leur localisation précise soit un obstacle (télétravail, résolution collaborative d'un problème rencontré sur un site client, support aux partenaires, etc.). Cinq forums constituent les canaux de communication internes et permettent une communication continue par thème (direction, développement, informatique scientifique, gestion de connaissances, gestion du système informatique).

5.3.2 Points d'avancement

Synthèse hebdomadaire Chaque lundi, le personnel envoie un compte-rendu par courriel à l'ensemble des développeurs. L'objectif est d'identifier rapidement les dysfonctionnements possibles. L'avantage de compte-rendu par courrier électronique est que le temps est plus optimisé qu'en planifiant des réunions. Les inconvénients sont que ces comptes-rendus ne sont pas lus de tous ni rédigés de façon systématique le lundi mais peuvent être reportés dans la semaine. L'information circule plus difficilement. Le contenu type d'un compte-rendu se compose du travail effectué la semaine précédente, celui prévu pour la semaine en cours, les difficultés rencontrées et nécessitant de l'aide immédiate et un retour sur le Vsprint (Cf paragraphe 4.2.3)

Réunion mensuelle par département Une fois par mois, chaque département se réunit pour analyser les résultats, communiquer sur les projets en cours et signaler tout dysfonctionnement éventuel.

5.3.3 Outil de planification du travail : JPL

JPL est une application *CubicWeb* qui permet de planifier le développement. L'équipe en charge du développement crée des versions non datées pour signaler les corrections et ajouts prévus. La date d'une version est déterminée par le client. En fonction du budget (en jours.hommes) attribué à Logilab par le client, des tickets peuvent être ajoutés ou retirés d'une version.

Troisième partie

Déroulement du stage

Chapitre 6

Le sujet

6.1 Problématique

Les applications du Web Sémantique, telles celles développées sur la plateforme *CubicWeb*, contiennent généralement d'importantes quantités de données hétérogènes, souvent issues de sources différentes. Une importante valeur ajoutée réside dans la capacité à traiter ces grands volumes de données pour en extraire des informations pertinentes. Une des solutions envisagée est de traiter les prédicats contenus dans les données avec des règles logiques visant à produire de nouveaux prédicats. Ce type de traitements fait appel à un moteur d'inférences. L'objectif des travaux est donc, à terme, d'offrir aux développeurs et aux utilisateurs d'une application *CubicWeb*, la possibilité de traiter l'ensemble des données avec des règles en vue d'en extraire des nouvelles informations pertinentes. Mon stage vise donc à améliorer la plateforme *CubicWeb*, utilisée dans tous les projets clients.

6.2 Un sujet innovant et prospectif

Le choix de ce stage fut motivé par la qualité innovante du sujet. Ce projet de mise en place de moteur d'inférences fait appel à la programmation logique, enseignée à l'Université de Technologie de Compiègne (UTC), mais encore très peu utilisée dans le monde industriel. Les enseignements de l'UTC IA01 [1] et IA02 [2] m'ont fait découvrir la représentation des connaissances, le Web sémantique et la programmation logique. Étant très intéressée par l'ingénierie des connaissances et système d'information, ce stage, sous la tutelle d'un ingénieur spécialiste de la gestion de données, et portant sur les moteurs d'inférence, s'inscrit parfaitement dans mon projet professionnel et ma spécialisation future en filière ICSI.

Cette volonté de mettre en place des mécanismes intelligents dans des applications Web reste néanmoins un sujet prospectif. La recherche de solutions d'implémentation dans un contexte précis faisant partie du rôle de l'ingénieur, ce sujet ouvre des perspectives dans tous les projets clients à venir car elle améliore la plateforme CubicWeb.

6.3 Rôle prédéfini et rôle réel

L'intitulé du stage défini avant mon arrivé était : "Intégrée à l'équipe de recherche et développement "Web Sémantique" de Logilab, sous la tutelle de Vincent Michel, ingénieur spécialiste de la gestion de données, le rôle est de l'assister dans son travail quotidien et effectuer tout ou partie des travaux suivants, en collaboration avec l'équipe : étudier différents moteurs d'inférences utilisés dans le monde du Web Sémantique, intégrer un de ces moteurs dans la plateforme CubicWeb, tester le moteur sur une ou plusieurs applications existantes. Le sujet de mon stage a bien été respecté puisque j'ai effectivement travaillé sur la mise en place de moteur d'inférences dans *CubicWeb*. Toutefois, la partie collaborative fut limitée. En effet, excepté lors des Vsprints (Cf paragraphe 4.2.3) , j'ai essentiellement cherché des solutions et des implémentations seule, présentées à

1. Intelligence artificielle - représentation des connaissances
2. Logique et résolution de problèmes par recherche

ma hiérarchie et modifiées suivant les différents retours. Ce contexte m'a permis m'a permis de prendre de l'autonomie.

Chapitre 7

Analyse du besoin réel et études

7.1 Les objectifs attendus

La première partie de ce stage fut de comprendre les enjeux et les problématiques de la mise en place d'un moteur d'inférences dans la plateforme CubicWeb ; puis d'étudier les options possibles pour mettre en place le moteur d'inférences. Cette analyse a conduit à l'étude de nombreuses thèses en anglais et à la publication en interne d'une série de dossiers en français et en anglais, l'anglais étant privilégié.

Les objectifs réels de la mise en place des moteurs d'inférence dans la plateforme *CubicWeb* sont de réduire le temps d'exécution des requêtes RQL et d'offrir aux utilisateurs la possibilité de traiter des ensemble de données via un comportement intelligent. Au cours de l'avancée de ce stage, de nouvelles applications possibles aux inférences se sont ajoutées :

- inférer les fermetures transitives
- inférer sur les relations ternaires
- apporter de la programmation logique aux attributs calculés pour améliorer le temps d'exécution des requêtes
- améliorer les mécanismes de propagation des permissions dans une application.
- simplifier à l'utilisateur l'écriture de requêtes complexes.

7.2 Les options évaluées

Concernant la simplification de l'écriture des requêtes, l'option envisagée est d'établir des règles simples au moyen de Python, dans un premier temps, et un mécanisme de réécriture, dans un second temps. Ce cas d'application de mise en place d'inférence est traité ultérieurement dans ce rapport. Pour mettre en place des inférences plus complexes que dans le cas d'application de la réécriture, plusieurs outils sont possibles : les *reasoner* (solution proposée par l'équipe à mon arrivée), les systèmes experts (solution que j'ai ensuite proposée), les bases de données déductives (dernière option évaluée).

7.2.1 Reasoners

Un *reasoner* est un programme capable d'inférer des conséquences logiques à partir de faits et d'axiomes et de vérifier qu'une ontologie [1] est cohérente. Pour vérifier la cohérence d'une ontologie, un *reasoner* vérifie :

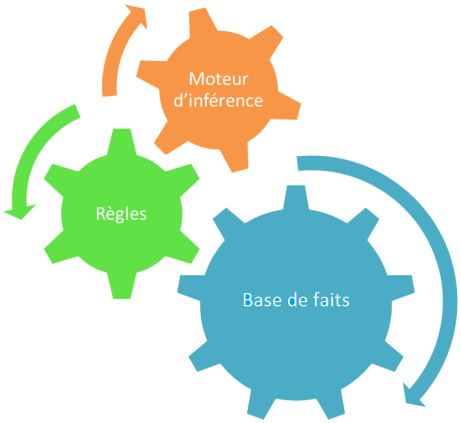
- la généralité d'un concept : un concept ne doit pas englober un autre concept
- si la définition d'un concept est instanciable
- s'il n'y a pas de contradiction entre les axiomes et les relations
- la cohérence du vocabulaire de l'ontologie.

1. Une ontologie est l'ensemble structuré des termes et concepts représentant le sens d'un champ d'informations. L'ontologie constitue en soi un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts.

Après avoir étudié différents *reasoner open source* comme FaCT++, Pellet, Jena, cette solution a été écartée car, de mon point de vue, elle ne correspondait pas à l'exigence des demandes. En effet, les *reasoners* actuels agissent comme des boites noires et le contrôle des inférences est limité. Par ailleurs, une autre demande du cahier des charges que l'on m'a confié, était la préférence de Logilab pour le langage Python. Or, pour le moment, aucun *reasoner* n'est codé en Python.

7.2.2 Les systèmes experts

Après avoir écarté l'option des *reasoner*, j'ai proposé les systèmes experts. Un système expert reproduit les mécanismes cognitifs d'un expert dans un domaine particulier. Un système expert est composé de trois parties. (Cf le schéma ci-dessous)



La base de faits La base de faits regroupe un ensemble de connaissances considérées comme vraies appelés "fait". Il existe deux types de faits : les faits spécifiques et les faits universels. Les faits spécifiques correspondent à des données propres au problème en cours. Les faits universels sont considérés comme toujours vrai indépendamment du problème.

Les règles Il existe deux types de règles :
- Les règles en **chaînage avant** : méthode de déduction qui part des prémisses pour déduire de nouvelles conclusions
- Les règles en **chaînage arrière** : méthode de déduction qui part des conclusions et tente de remonter aux axiomes.

Le moteur d'inférence Le moteur d'inférences peut déduire de nouveaux faits et répondre à une question en ajoutant de nouveaux faits déduits dans la base de fait (méthode par saturation) ou non.

Exemple d'un système expert

Base de faits
Pierre est le père de Jean
Jean est le frère de Simon
Nathalie est la mère de Simon

Base de règles
Si x est frère de y alors la mère de x est aussi la mère de y
Si x est frère de y alors le père de x est aussi le père de y

Le moteur d'inférences déduirait les faits suivants
Nathalie est la mère de Jean
Pierre est le père de Simon

Cette proposition a été étudiée par le département pour sa capacité à être appliquée à tous les cas particuliers, sa flexibilité, le contrôle des déductions et son implémentation possible en Python. Néanmoins, construire des règles pertinentes nécessite une expertis approndie. Or dans le cas d'une application CubicWeb, le modèle métier est définit par le client. Or ce dernier ne possède pas systématiquement les compétences pour construire des règles.

7.2.3 Les bases de données déductives

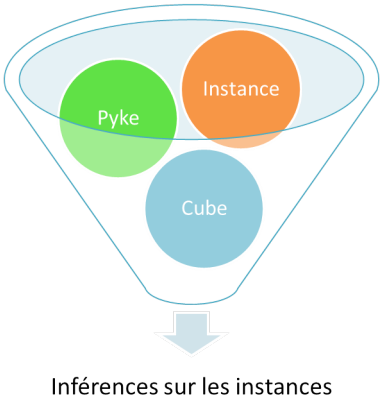
Une base données déductive est une base de données capable d'effectuer des déductions à la manière d'un système expert, c'est-à-dire en utilisant des faits et des règles stockés dans la base de données déductive. Elle combine le paradigme de la programmation logique et les bases de données relationnelles. Aujourd'hui, les nouvelles applications des bases de données déductives sont l'intégration de données, l'extraction d'information, les réseaux sociaux, l'analyse de programmes, la sécurité et le *cloud computing*[1]. Dans le cas du développement des applications *CubicWeb*, cette solution est extrêmement pertinente car elle offre les mêmes avantages que les systèmes experts, avec une possibilité d'ajout de fonctions logiques. Deplus, les bases de données déductives ont été pensées pour travailler avec des bases de données. Dans l'optique de la spécialisation en Python de Logilab, l'étude du module Python PyDatalog a été engagée.

1. *Cloud computing* : technologie qui permet de mettre sur des serveurs localisés à distance des données de stockage ou des logiciels.

Chapitre 8

Création du cube système expert

8.1 Objectifs et cas d'utilisation



8.1.1 Objectifs

Le cube *Expert System* est un cube métier qui a pour rôle de fournir à un autre cube applicatif la possibilité d'y introduire des systèmes experts. Ce mécanisme permettra d'apporter des raisonnements inductifs simples. Le cube *Expert System* utilise le module Python PyKE. Dans un premier temps, comprendre le fonctionnement de Pyke a été déterminant pour réaliser le cube *Expert System*. Pyke fournit un mécanisme d'inférences de connaissances, écrit en Python, basé sur le modèle du système expert. Il est inspiré de Prolog et apporte des possibilités de programmation logique. Néanmoins, contrairement à Prolog, Pyke s'intègre parfaitement à Python. Grâce à un ensemble de règles, le moteur d'inférences PyKE infère sur la base de faits fournie par la base de données du cube applicatif *CubicWeb* via l'instance du cube applicatif.

8.1.2 Cas d'utilisation

Dans un forum, un commentaire peut commenter une discussion ou un autre commentaire. Or dans une chaîne de commentaire, le commentaire n'est pas rattaché au sujet de la discussion mais à un autre commentaire, lui même pouvant petre rattaché à un commentaire et ainsi de suite jusqu'au commentaire initial qui est rattaché au sujet de la discussion (Cf schema ci-après). Le système expert permet de déduire à quel sujet n'importe quel commentaire est rattaché.

FIGURE 8.1 – Chaîne de commentaires

8.2 Base de faits

8.2.1 Les faits issus du Cube Forum

Les faits sont modélisés par le schéma du cube `Forum` ci-dessous :

Code 8.1 – Schéma du cube Forum

```
class ForumThread(EntityType):
    title = String(required=True, fulltextindexed=True, maxsize=256)
    content = RichString(required=True, fulltextindexed=True)
    in_forum = SubjectRelation('Forum', cardinality='1*', inlined=True,
                               composite='object')

class Comment(EntityType):
    """a comment is a reply about another entity"""
    content = RichString(required=True, fulltextindexed=True)
    comments = SubjectRelation('Comment', cardinality='1*',
                               composite='object')
    in_thread = SubjectRelation('Comment', cardinality='?*',
                                composite='object')

class comments(RelationDefinition):
    subject = 'Comment'
    object = ('ForumThread', 'Comment')
    cardinality = '1*'
```

Les faits ont alors la forme suivantes :
- L'entité X commentes l'entité Y
- X est un commentaire
- X est un sujet de discussion

Fonctionnement de Pyke Le fonctionnement de Pyke ne permet pas à un fait d'être dupliqué. Une tentative d'ajout d'un fait déjà existant sera ignoré. Un fait est considéré comme immuable : il ne peut pas être modifiés ou rétractés. Chaque fait à un nom et un tuple d'arguments. Ses arguments sont des données Python. Contrairement aux faits universels qui ne sont jamais supprimés, les faits spécifiques seront supprimés lors de la réinitialisation du moteur de Pyke. En effet, Pyke peut réinitialiser son moteur d'inférences et ainsi le dernier ensemble de faits est supprimé.

8.2.2 Constitution de la base de faits

On ajoute un fait spécifique ou un fait universel de la manière suivante :
- `some_engine.add_case_specific_fact(kb_name, fact_name, arguments)`
- `some_engine.assert_(kb_name, fact_name, arguments)`

La base de faits est construite en interrogeant la base de données existante par requêtes RQL.

Code 8.2 – Création de la base de faits

```
def push(engine, _rql,base_name, relation, entity):
    for r in _rql:
        if len(r) == 2:
            engine.assert_(base_name, relation, (r[0], r[1]))
        else:
            engine.assert_(base_name, entity, (r[0], 0))
```

Les faits issus de la requête RQL sont de deux types : relation (eg : X comment Y) ou entité (eg : X is Forum). Si la requête porte sur une relation alors le résultat renvoie les identifiants (`eid`) des deux membres de la relation et la longueur sera de 2. Dans le cas où la requête porte sur une entité, le résultat renvoie l'identifiant (`eid`) de l'élément et la longueur de r est 1. Suivant ce résultat, on ajoute les faits différemment dans la base.

8.2.3 Mise à jour des faits

Afin de garder une base de faits cohérentes, sa mise à jour est indispensable. En effet, l'ajout d'un commentaire ou d'un sujet de discussion (ou leurs suppressions) va modifier la base de faits. Ainsi, le *hook* du cube applicatif appelle le fichier `sePyke.py`, chargé du lancement du système expert lors de l'ajout ou de la suppression d'un fait de la base. Cela permet de mettre à jour la base de faits à la création ou à la suppression d'un fait. En voici un exemple :

Code 8.3 – Hook du cube forum gérant l'activation du système expert

```
class ExpertSystemAfterAddComment(hook.Hook):
    """automatically activate the expert system after adding a comment"""
    __regid__ = 'forum.expert_system_after_add_comment'
    events = ('after_add_entity',)
    __select__ = hook.Hook.__select__ & is_instance('Comment',)

    def __call__(self):
        driver_forum(self._cw)
```

Cette classe est utilisée après l'ajout de l'entité `Comment`. Elle appelle alors le pilote situé dans le fichier `sePyKE.py` et permet le lancement du système expert à chaque ajout d'entité.

8.3 Base de règles

8.3.1 Les règles

Les règles sont écrites dans un fichier.krb. Une règle est composée de prémisses et de conclusions. Pyke permet d'établir plusieurs conclusions. Le fichier .krb peut contenir des règles de type chaînage-avant et des règles de type chaînage-arrière. Pyke les différenciera car elles n'ont pas la même syntaxe. Les règles de type chaînage-avant possèdent les mots clefs `assert` et `foreach` qui correspondent au raisonnement "Si/Alors". Les règles de type chaînage-arrière sont écrites à partir des mots clef `use` et `when` qui correspond au raisonnement "utilise ce fait si".

Pour déterminer à quel sujet de discussion est rattaché un commentaire, deux règles sont nécessaires :

- une première règle qui définit la transitivité d'un commentaire : si X commente Y, Y comment Z alors X commente Z.
- une deuxième règle qui indique à quel sujet de discussion appartient un commentaire : Si Y est un sujet de discussion et X commente Y alors X appartient au sujet de discussion Y.

8.3.2 Constitution de la base de règles

Dans l'optique de conserver un maximum de code Python, la fonction `driverBuilder()` génère le fichier .krb, propre à Pyke, où sont stockés les règles. On obtient la base de règle suivante :

Code 8.4 – Base de règle du cube Forum

```
comments_chains
    foreach
        comment_base.comments($Y, $Z)
        comment_base.comments($X, $Y)
        check $Y != $Z
        check $X != $Y
    assert
        comment_base.comments($X, $Z)

in_thread
    foreach
        comment_base.ForumThread($Y, 0)
        comment_base.comments($X, $Y)
        check $X != $Y
    assert
        comment_base.in_thread($X, $Y)
```

La première règle `comments_chains()` définit la transitivité d'un commentaire : si X commente Y, Y comment Z alors X commente Z. La règle `in_thread()` indique à quel sujet de discussion appartient un commentaire : Si Y est un sujet de discussion et X commente Y alors X appartient au sujet de discussion Y.

8.4 Moteur d'inférences

La mise en place du système expert est effectuée dans un fichier Python (`sePyke.py`). Elle suit les étapes suivantes :

1. Constitution de la base de faits
2. Constitution de la base de règles
3. Activation du fichier contenant les règles
4. Inférence de Pyke selon l'algorithme de chaînage avant par saturation.

La constitution de la base de faits et de la base de règles ont été respectivement développées dans les paragraphes 8.2.2 et 8.3.2. Dans ce paragraphe, nous détaillerons l'activation des règles et le fonctionnement de Pyke qui lui permet d'inférer sur la base de faits.

8.4.1 Activation des règles

Un ensemble de règles est activé avec la commande `some_engine.activate('monfichier')`. L'activation des règles n'est pas automatique. On active un fichier.krb à la fois. En effet, il peut en exister plusieurs. Créer plusieurs ensembles de règles permet de choisir entre les différentes bases de règles selon les situations spécifiques.

8.4.2 Algorithme de chaînage-avant par saturation

Les règles définies pour ce cas d'applications sont des règles de type chaînage-avant : *"if A B C then D E"*. Afin de pouvoir déduire à quel sujet de discussion est rattaché un commentaire, Pyke suit l'algorithme de chaînage-avant par saturation, expliqué ci-après. Si les règles avaient été de type-chaînage arrière, Pyke aurait suivi un algorithme de chaînage-arrière. Ce dernier se trouve en Annexe A.1.

Une correspondance est recherchée entre la prémisse et les faits connus. Cette correspondance est déterminée par un mécanisme de *Pattern variables*, décrit dans le paragraphe suivant. Pyke essaye toutes les combinaisons possibles à travers le procédé de chaînage-arrière. Potentiellement, une règle peut réussir plusieurs fois. Si Pyke réussit à trouver une correspondance pour une prémisse, il va tenter de traiter la prémisse suivante dans la liste. Si la dernière prémisse a une correspondance alors la règle est appliquée. Si Pyke échoue sur une prémisse, il remonte à la prémisse précédente dans la liste et trouve une autre solution. S'il remonte jusqu'à la première prémisse et échoue alors la règle échoue.

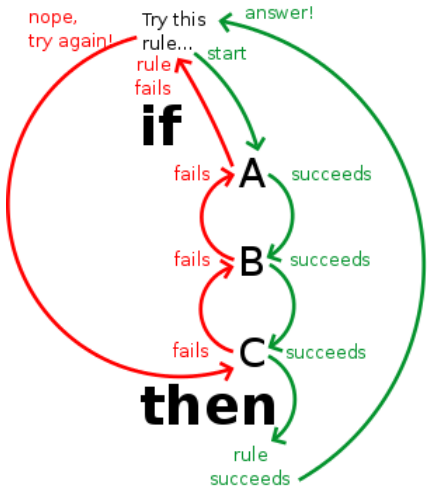


FIGURE 8.2 – Algorithme de chaînage-avant

L'algorithme de Pyke fonctionne par saturation. Un moteur en chaînage-avant fonctionnant par saturation boucle sur la base de règles tant que de nouveaux faits peuvent être rajoutés dans la base de faits. L'algorithme prend en entrée la liste des faits et la liste des règles et ajoute les faits intermédiaires déduits dans la base de faits.

8.4.3 Pattern Matching

Le *Pattern Matching* est un mécanisme qui vérifie si les données peuvent correspondre. Il est indispensable pour généraliser les faits. Il existe deux mécanismes de vérification : le *Literal Patterns* et le *Pattern variables*. Les *Literal patterns* sont des valeurs qui ne correspondent qu'à elles-mêmes. Par exemple, pour vérifier qu'un fait est vrai, tous les arguments du fait seront considérés comme des *Literal Pattern*. Pyke regardera alors si les arguments correspondent parfaitement aux arguments d'un fait contenu dans sa base. Les *Pattern variables* sont des valeurs de données qui peuvent correspondre avec n'importe quelle donnée ou variable. Contrairement aux *Literal Pattern*, les *Pattern variables* permettent de répondre à une requête ouverte et non à une requête booléenne. Par exemple, si on souhaite savoir *"Comment s'appelle le fils de Florence et Philippe ?"*, on écrira le fait ainsi : family.son_of($son, Florence, Philippe). Les variables sont obligatoirement précédées du symbole $ pour que Pyke puisse les identifier comme *Pattern variables* et puisse effectuer le *Pattern Matching*. Pour indiquer une variable dont on ne se préoccupe pas (variables anonymes), il faut précéder la variable du symbole "$_".

8.5 Schéma du cube *Expert System*

Dans un premier temps, les notions de règles et de bases de faits étaient définies dans le schéma d'un cube comme suit pour permettre de créer des règles depuis l'interface de l'application (Cf figure 8.3) :

Code 8.5 – Schéma d'un système expert

```
class ExpertSystem(EntityType):
    name = String(maxsize=50, required=True, unique=True)
    description = RichString()

class Base(EntityType):
   name = String(maxsize=50, required=True, unique=True)
    description = RichString()
    in_expert_system = SubjectRelation('ExpertSystem', cardinality='1*',
                                       composite='object')

class Rule(EntityType):
    name = String(maxsize=50, required=True, unique=True)
    description = RichString()
    in_base = SubjectRelation('Base', cardinality='1*',
                              composite='object')
    premisse = String(maxsize=1024, required=True)
    conclusion = String(maxsize=1024, required=True)
```

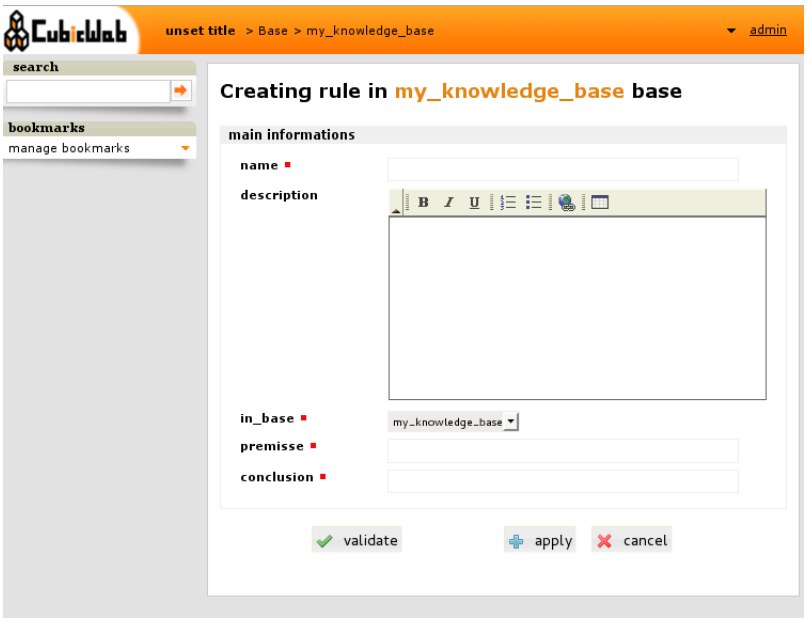


FIGURE 8.3 – Interface pour la création d'une règle

Dans une seconde version, les règles ne sont plus définies dans le schéma. Les règles nécessitent une expertise approfondie, et donc ne peuvent pas être créées à la volée. Elles ne seront donc pas définies via l'interface Web.

8.6 Résultats obtenus

Le cube métier *Expert System* apporte bien la fonctionnalité d'inférences attendue. Ainsi l'écriture de la requête RQL `Any X,Y WHERE Y in_thread X` dans la barre de recherche affiche l'ensemble des commentaires liés au sujet A (Cf figure 8.4).

FIGURE 8.4 – Résultat de la requête via l'interface de l'application

8.7 Apports personnels et pour Logilab

La création de mon propre cube au sein de la plateforme CubicWeb m'a permis d'en cerner le fonctionnement. De plus, ce projet a nécessité la mise en œuvre de mon expertise et des connaissances acquises à l'UTC. Ce cube ouvre la voie à une approche logique et vers les systèmes experts.

Chapitre 9

Réécriture de requêtes RQL

9.1 Les cas d'utilisations

Le RQL propose un langage explicite de haut niveau permettant de faciliter leur utilisation. Cependant, selon le modèle de données, une requête peut être longue et complexe. Établir une correspondance entre un ensemble de relations et une relation simple, par exemple, permet d'alléger l'écriture. Les cas d'utilisations sont multiples : réécriture des relations complexes mais aussi réécriture d'attributs calculés, génération de paramètres génériques.

9.1.1 La réécriture de relation

Certaines relations sont complexes ; le but de la réalisation est de les simplifier afin de faciliter l'expression syntaxique des requêtes. La requête ci-dessous permet, selon le modèle de données extrait du projet client pour la BnF, de sélectionner les illustrateurs A d'une œuvre B (ici appelé `Manifestation`).

Requête 9.1 – sélection des illustrateurs d'une œeuvre donnée

```
Any A,B WHERE C is Contribution, C contributor A, C manifestation B,
             C role R, R name "illustrator"
```

Le modèle de données établit qu'une personne contribue (`contributor`) à une `Manifestation` selon un `role` (illusteur, auteur etc.). Avec la réécriture, cette requête peut être écrite comme suit. On constate que la simplification n'est pas dérisoire.

Requête avec réécriture 9.2 – requête RQL réécrite

```
Any A,B WHERE A illustrator_of B
```

Cette réécriture s'intègre au modèle de données établi par CubicWeb. La définition de la règle de réécriture est intégrée au type de la relation `illustrator_of`. Les types d'objets et de sujets sont inférés depuis la règle et ajouter à la définition de la relation. On appelle définition d'une relation le triplet <sujet> <relation> <objet>. Le type d'une relation sont les paramètres spécifiques à la relation seule.

Schéma 9.3 – Relation utilisant une règle de réécriture

```
class illustrator_of(ComputedRelationType):
    rule  = ("C is Contribution, C contributor S, C manifestation O,
              C role R, R name 'illustrator'")
```

9.1.2 Les attributs calculés

Un attribut calculé est un attribut qui est déterminé par une requête RQL. Par exemple, la requête suivante sélectionne la somme des salaires versés aux salariés de chaque entreprise.

Requête 9.4 – Sélection de la somme des salaires

```
Any SUM(SA) GROUPBY S WHERE P works_for S, P salary SA
```

Avec le principe de la réécriture, cette requête est réécrite comme suit :

Requête avec réécriture 9.5 – Sélectionne de la somme des salaires

```
Any A WHERE S global_salary A
```

De même que pour la réécriture des relations, la réécriture des attributs calculés s'intègre au modèle de donnée CubicWeb :

Schéma 9.6 – Entité contenant un attribut calculé

```
class Societe(EntityType):
    name = String()
    value = SubjectRelation('Mark')
    global_salary = Int(formula=("Any SUM(SA) GROUPBY S WHERE
                                  P works S, P salary SA"))
```

9.1.3 Le paramètre générique

Il arrive que les instances d'une classe donnée n'aient pas de valeur à tous leurs attributs. Ue entité correspond à une tableau SQL où ses attributs sont des colonnes. Dans le cas où un attribut est peu rensigné, on obtient une colonne quasiment vide et ainsi une repésentation parcimonieuse. Définir un paramètre générique à la place d'utiliser un attribut classique évite d'obtenir un trop grand nombre de champs vides dans la table d'une entité. En effet, la colonne de l'attribut sera remplacé par des liens vers d'autres entités afin d'obtenir une meilleure optimisation. Ce paramètre générique possède un champ `name` et un champ `value`. La relation `has_parameter` et la relation entre une ou plusieurs entités quelconques vers un paramètre générique.

Schéma 9.7 – Paramètre générique

```
class CWParameter(EntityType):
    """Generic entity links an entity name to a value"""
    name = String()
    value = String()

class has_parameter(RelationDefinition):
    subject = '*'
    object = 'CWParameter'
    cardinality = '**'
```

Exemple Soit la classe `Person` avec l'attribute `gender`. Or seulement un quart des instances de `Person` ont une valeur pour cet attribut. Dans ce cas, on utilise le paramètre générique. Cette solution est plus efficace dans ce contexte.

Schéma 9.8 – L'entité Person

```
class Person(EntityType):
    name = String()
    gender = String(cw_parameter=True)
```

Paramètre générique et réécriture Dans le cadre du paramètre générique, les règles sont toutes identiques au modèle suivant : `S is A, S has_parameter C?, C name X, C value O`, où A est le type d'entité de S et X le nom du paramètre. La réécriture de la règle se fait donc automatiquement. La requête (1) pourra alors être remplacée par la requête (2)

Requête 9.9 – Paramètre générique et réécriture

```
(1) : Any S WHERE S is Person, S has_parameter C?, C name 'gender',
      C value 'Male'
(2) : Any A WHERE A is Person, A gender "Male"
```

L'implémentation d'un tel paramètre est née de discussions sur des projets clients. Pour le moment, ce cas d'utilisation reste en suspens car il dépend largement des besoins client.

9.2 Écriture de tests

Test driven development Lors de ce stage, l'intérêt de la mise en place de tests automatiques m'a été fortement démontrée. En effet, cela permet de maintenir le code lors de son évolution. Lors d'un changement d'implémentation d'un module, si les tests unitaires passent, il y a de grandes chances pour que le reste de l'application continue à fonctionner. Pour la réécriture de requêtes RQL, mon développement a été piloté par les tests, suivant le principe du *test driven development*. Cette technique de développement consiste à prendre en compte un aspect particulier du problème, d'écrire un test couvrant cet aspect, d'écrire le code correspondant permettant de faire passer ce test, puis de faire passer le test et d'ajouter un nouveau test. Cette méthode offre un rythme soutenu au codage. Dans le cadre de la réécriture de requêtes RQL, plusieurs tests ont été implémentés en utilisant la bibliothèque de tests unitaires `Unittest`. Cette bibliothèque fournit des fonctions qui testent le fonctionnement du code afin de vérifier si, par exemple, une fonction donnée lève une exception au moment voulu ou si deux objets sont égaux.

9.2.1 Réécriture de relation

Dans le langage RQL, une relation peut être située à différents emplacements sur le graphe des requêtes. Les tests correspondent à la division du problème en sous-parties gérant les différents cas possibles :

1. Remplacer une relation située dans la partie conditionnelle WHERE
2. Remplacer une relation située dans la sous-requête WITH BEING (exemple du test ci-dessous)
3. Remplacer une relation située dans une UNION
4. Remplacer une relation située dans une condition d'existence EXISTS
5. Remplacer une relation située dans la partie conditionnelle WHERE d'une requête de modification SET ou de supression DELETE ou d'insertion INSERT
6. Permettre le remplacement récursif (exemple du test-ci-dessous)

Test 9.10 – Relation située dans la sous-requête WITH BEING

```
def test_rewrite_subquery(self):
    rule = {'illustrator_of': "C is Contribution, C contributor S, \
            C manifestation O, C role R, R name 'illustrator'"}
    tree = parse("Any A,B WHERE T relation C
                  WITH A, B BEING(Any X, Y WHERE X illustrator_of Y)")
    r.rewrite(rule,tree)
    self.assertEqual(tree.as_string(),
                     "Any A,B WHERE T relation C
                      WITH A,B BEING (Any X,Y WHERE D is Contribution,
                      D contributor X, D manifestation Y, D role R,
                      R name 'illustrator')")
```

Test 9.11 – Remplacement récursif

```
def test_recursivity(self):
    rule = {'identify' : \verbatim{"S has C, C is Card, C value O"},
            'value' : "S is Number, S match O",
            'match' : "S match2 O"""}
    tree = parse("Any A,B WHERE A identify B")
    r.rewrite(rule,tree)
    self.assertEqual(tree.as_string(),"Any A,B WHERE A has C, C is Card,
                     C is Number, C match2 B")
```

L'écriture des tests m'a permis de développer rapidement le programme Python de réécriture des requêtes, dont le code et l'algorithme sont en Annexe A.

9.2.2 Réécriture d'attribut calculé

De même, la mise en œuvre de la réécriture d'attributs calculés a été soumise à de nombreux tests. Les exemples ci-dessous gèrent respectivement l'utilisation d'un attribut calculé dans une fonction de comparaison et l'utilisation d'attribut calculé et de relation réécrite.

Test 9.12 – Attribut calculé et fonction de comparaison

```
def test_computed_attribute(self):
        computed = {'salaire_total': 'Any SUM(SA) GROUPBY S WHERE
                                      P travaille S, P salaire SA'}
        tree = parse('Any S WHERE S salaire_total > 10000')
        r.rewrite(tree, None, computed)
        self.assertEqual(tree.as_string(), 'Any S GROUPBY S WHERE
                                            P travaille S, P salaire SA
                                            HAVING SUM(SA) > 10000')
```

Test 9.13 – Attribut calculé et réécriture de relation

```
    def test_computed_attribute_and__rule(self):
        computed = {'salaire_total': 'Any SUM(SA) GROUPBY S WHERE
                                      P travaille S, P salaire SA'}
        rule = {'travaille' : 'S travaille_pour O'}
        tree = parse('Any A WHERE S salaire_total A')
        r.rewrite(tree, rule, computed)
        self.assertEqual(tree.as_string(), 'Any SUM(SA) GROUPBY S WHERE
```

```
                                P travaille_pour S, P salaire SA')
```

9.3 Intégration à la structure en place

9.3.1 Schéma Yams et CubicWeb

Schéma Yams Pour construire et appliquer les règles aux requêtes RQL, le code écrit a dû être intégré à la structure de CubicWeb déjà en place. La règle pour être reconnue a dû être ajoutée au schéma Yams pour que les programmes reconnaissent les mots clés `rule`, `formula`, et `cw_parameter`. Ainsi, j'ai ajouté un attribut `rule` au type d'une relation, et un attribut `formula` au schéma d'une entité. Les modifications apportées apparaissent en annexe XXX.

CubicWeb Le mécanisme de réécriture doit être appliqué avant les mécanismes de permissions et de sécurité de la requête, il a donc été appelé par le schéma CubicWeb dans un point de branchement suffisamment haut.

9.3.2 Mises à jour des données

Cohérence des données Dans une application, les données peuvent êre mises à jour très fréquemment. Dans le cas où une relation est réécrite, la requête RQL utilisant cette relation virtuelle déduira sa réponse et ne stockera pas la relation en base. La mise à jour des données n'aura donc pas de conséquences sur la cohérence des données liés à une relation réécrite. Au contraire, dans le cadre d'un attribut calculé, la valeur de l'attribut est calculé grâce à la formule et stocké en base. La modification d'une relation contenue dans la formule peut alors éventuellement changer la valeur de l'attribut calculé. J'ai donc implémenté un *hook* qui met à jour la valeur d'un attribut calculé si une relation contenue dans sa formule est ajoutée ou modifiée.

Déclenchement de la mise à jour Le *hook* se déclenche à l'ajout d'une relation (`events=('after_add_relation')`) et si cette relation est contenue dans une formule d'un attribut calculé (`__select__=match_computed_attribute()`). Cette seconde contrainte est construite avec la fonction `match_computed_attribute()` qui se trouve en annexe XXX.

Implémentation du hook Le *hook* ci-dessous exécute la requête RQL de la formule de l'attribut calculé. La requête retourne alors la nouvelle valeur de l'attribut calculé ainsi que l'identifiant (*eid*) de l'attribut calculé. Il lance alors la requête `SET X attribut_calculé nouvelle_valeur WHERE X eid A` où A est l'identifiant de l'attribut calculé. Le code complet de ce hook se trouve en annexe (XXX).

Migration des applications Les applications sont régulièrement mises à jours. Afin d'optimiser le coût en terme de performance de la migration d'une version à une autre, les données ne sont mises à jour que si nécessaire. Si d'une version à l'autre, la formule d'un attribut calculé change, il faut alors recalculer l'ensemble des attributs et ajouter les nouvelles valeurs au moyens d'une requête RQL de modification (`SET`)

9.3.3 Génération de tables SQL

Dans le cadre de la réécriture, il était intéressant de ne pas générer la table d'une relation dont le seul but est de simplifier la réécriture. Après avoir compris le mécanisme, le programme de génération des tables SQL a été modifié en conséquence. Ces modifications se trouvent en annexe XXX.

9.4 Communication autour du projet

Le projet de réécriture de requêtes RQL modifie CubicWeb et *Yams* et concerne l'ensemble des utilisateurs de ce framework. Il a donc nécessité d'établir un large plan de communication. Ainsi, j'ai rédigé des blogs en anglais sur l'intranet et sur `cubicweb.org` afin de communiquer essentiellement sur la finalité du projet et l'interface de programmation applicative envisagée. CubicWeb étant une plateforme *open source*,

j'ai également transmis ces informations à la liste de diffusions des utilisateurs de CubicWeb. Les utilisateurs de CubicWeb sont principalement les membres de Logilab mais aussi Softtek, une multinationale d'origine mexicaine. L'ensemble de la communication se fait donc en anglais. Ce devoir de communication m'a permis d'étendre mes compétences de rédaction sur un sujet technique, ce qui fut très enrichissant.

9.5 Apports personnels et pour Logilab

La réécriture de requêtes RQL permet à l'utilisateur de simplifier ses requêtes puisque le programme déduira, en fonction des règles, quelles relations appliquer. Cette mise en place d'une inférence d'un concept assez simple m'a permis :

- de développer une fonctionnalité utile et appréciée pour toute l'entreprise
- de travailler sur le code de gestion de schéma (*Yams*) de CubicWeb
- et de travailler selon une nouvelle méthode : *test driven development*

De plus, la possibilité de ne pas générer des tables SQL pour des relations "virtuelles" ouvre une voie pour une meilleure gestion des performances lors de l'exécution de requêtes.

Chapitre 10

Réalisations secondaires

10.1 Vsprint

Liste des sujets affectés Voici la liste des sujets de Vsprint qui m'ont été attribués pendant les six mois de stage :

- Correction d'anomalies de mise en forme sur le cube Bootstrap
- Image search
- Tracé en d3js du graphe des projets du tableau de bord

Imagesearch L'objectif de ce projet est de réaliser un navigateur qui parcourt une base d'images et restreint les images affichées par thème. En sélectionnant une image, on sélectionne un thème et on se rapproche ainsi peu à peu vers les images souhaitées. Ce projet est très intéressant car il propose une fonctionnalité de navigation alternative et dynamique.

Graphe des projets Ce projet consiste à mettre en place un graphe dynamique résumant l'avancée des différents projets client. Il m'a permis de découvrir d3js, une bibliothèque JavaScript dédiée à la manipulation de documents qui utilisent HTML, SVG [1] and CSS.

Bilan des Vsprint Les Vsprint m'ont permis de travailler en binôme sur des sujets divers et donc d'accroitre mes connaissances notamment sur Bootstrap, *framework open source* de mise en forme de pages Web et les bibliothèques Javascript.

10.2 Création d'un cube d'import CSV

10.2.1 Objectifs

CSV est une repésentation des données tabulaires sous forme de valeurs séparées par des virgules ou par un quelconque séparateur comme un point-virgule. Il existait déjà un cube chargé de l'import CSV sous forme d'un tableau classique. L'objectif est de créer un nouveau cube métier et d'afficher les données tabulaires en utilisant SlickGrid. SlickGrid est une bibliothèque Javascript permettant d'obtenir des tableaux plus élaborés. Le type de tableau attendu est un tableau capable de trier et filtrer les données, dont les dimensions sont fixes selon la page et disposant de barres de défilement.

10.2.2 Implémentation

Fonctionnalités implémentées

- import du tableau
- barre de défillement

1. SVG (Scalable Vector Graphics) est un format d'image XML pour les graphiques à deux dimensions, qui supporte les interactivité et les animations.

- formulaire d'options
- filtres
- possibilités de déplacement des colonnes par glissement

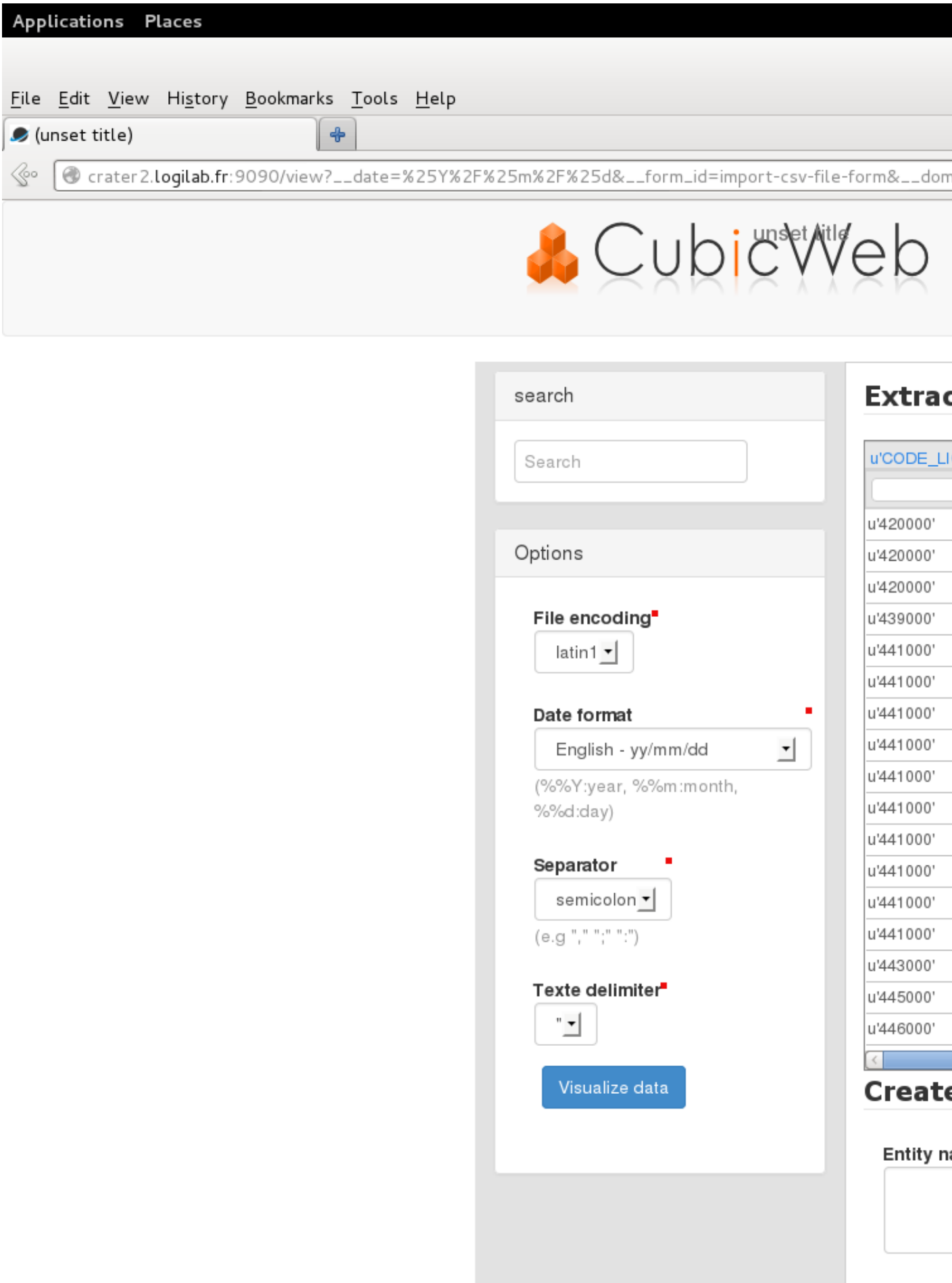

Applications Places
File Edit View History Bookmarks Tools Help
(unset title)
crater2.logilab.fr:9090/view?__date=%25Y%2F%25m%2F%25d&__form_id=import-csv-file-form&__dom
unset title
CubicWeb
search
Search
Options
File encoding
latin1
Date format
English - yy/mm/dd
(%%Y:year, %%m:month, %%d:day)
Separator
semicolon
(e.g "," ";" ":")
Texte delimiter
"
Visualize data
Extrac
u'CODE_LI
u'420000'
u'420000'
u'420000'
u'439000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'441000'
u'443000'
u'445000'
u'446000'
Create
Entity na

FIGURE 10.1 – Interface du cube *importcsv*

Fonctionnement L'implémentation des barres de défillement, du formulaire d'options, des filtres et de la possibilités de déplacement des colonnes ont été réalisées à partir de bibliothèques javascripts. Pour obtenir un fichier CSV sous forme de tableau présentable, il a fallu décoder le fichier afin de déterminer le type de séparateur (espace, virgule, point-virgule etc.). Puis, les données sont stockés dans une matrice.

Test 10.1 – code à insérer

Apports personnels et pour Logilab Ce projet m'a permis d'améliorer mes connaissances et ma pratique du javascript [1]. Il fut également enrichissant par ma découverte de la bibliothèque SlickGrid et m'incite à découvrir les nouveautés dans le milieu de l'*open source*. Pour Logilab, ce cube métier sera très utile. En effet, de nombreux projets client requièrent un import CSV, une mise une page compacte des tableaux, et des tableaux aux fonctionnalités évoluées. Une mise en forme compacte permet d'afficher le tableau sans faire défiler l'ensemble de l'écran, bien que ce dernier ait trois mille lignes.

1. javascript : langage de programmation de scripts utilisé dans les pages Web interactives

Chapitre 11

Conclusion

Ce stage au sein de Logilab a été une expérience très profitable. La richesse et l'originalité du sujet m'ont séduites. Mettre en place des inférences au sein d'un logiciel comme CubicWeb a été un véritable défi.

Ces vingt-quatre semaines ont été extrêment instructives au niveau de la méthodologie et des outils utilisés. J'ai pu appliquer la méthodologie agile, méthodologie très répandue dans les projets informatiques, et constater son efficacité pour s'assurer de la satisfaction du client. J'ai également pu en cerner les limites : sa réussite dépend de la disponibilité du client. De même, mon travail a été soumis à la méthodologie du *test driven development* où l'implémentation est guidée par des tests unitaires. Cette méthode m'a semblé très efficace pour générer du code rapidement et conserver une vérification des résultats en cas de changement d'implémentation. De plus, l'utilisation d'un gestionnaire de versions décentralisé comme Mercurial fut une remarquable découverte et je compte l'appliquer à mes projets futurs.

Le sujet de ce stage correspond à un travail de recherche pour le développement interne et consistait à apporter une plus-value à l'entreprise. Ma mission, pour un client interne, à savoir le comité de pilotage de CubicWeb, a été soumis aux mêmes problématiques que pour un projet client classique. En effet, il est fréquent que le client ne sache pas toujours ce dont il a réellement besoin. Confrontée à ce problème, j'ai appliqué les méthodes agiles et le projet a été réorienté sur les besoins réels. De même, la communication est au cœur de la relation client, afin d'expliquer les apports fonctionnels des changements proposés et éviter les incompréhensions. Ce devoir de communication et d'agilité, parti intégrante du métier d'ingénieur, fut très enrichissant. De plus, ayant très peu travaillé en équipe, ce contexte m'a permis de prendre de l'indépendance, de l'autonomie, et de réfléchir sur les besoins de l'entreprise.

Les difficultés rencontrées résident dans la mise en place du projet puisque le sujet de stage reste prospectif. Toutefois les spécifications se sont précisées au fil des études et démonstrations que je leur ai apporté. CubicWeb étant utilisé par l'ensemble des membres de Logilab, de nombreux avis, parfois divergents, ont réorienté le projet. Je pense néanmoins avoir réussi à faire avancer le projet et CubicWeb.

Travailler sur CubicWeb m'a permis de me plonger dans le domaine de la gestion de connaissances. Cela a confirmé mon désir de me spécialiser dans le domaine de l'Ingénieurie de Connaissances et Systèmes d'Information.

Annexe A

Algorithmes

A.1 Algorithme de chaînage-arrière

Struture et fonctionnement Les règles en chaînage-arrière sont traitées quand le programme demande à Pyke de prouver un but précis. Pour établir cette preuve, Pyke utilisera uniquement les règles déjà activées. Pour faire du chaînage-arrière, Pyke essaye récursivement de prouver tous les sous-buts. Quand tous les sous-buts sont prouvés, la règle aboutit et le but est prouvé.

Algorithme de chaînage-arrière L'algorithme de chaînage-arrière procède comme suit :
- Si la règle ne conduit pas à une correspondance avec la base de faits alors elle échoue et elle remonte au nœud précédent dans le graphe.
- Si la règle conduit à une correspondance avec la base de faits alors elle réussit et crée une nouveau nœud dans le graphe des solutions.

Cet algorithme inclut toute la chaîne d'inférence depuis le tout début pour prouver le but final. L'avantage d'un tel algorithme est sa capacité à retourner sur n'importe quel nœud du graphe pour trouver une solution alternative. Les règles sont appliquées dans l'ordre, il est donc préférable de mettre les règles qui ont le plus de chances de s'appliquer en premier.

L'algorithme de chaînage-arrière est en profondeur d'abord. En effet, il s'agit d'un parcours où l'on explore le graphe jusqu'à ce que l'on ne puisse plus appliquer une règle. Ce cas survient lorsque l'on a parcouru tout le graphe ou lorsque l'on est arrivé sur un nœud qui n'a pas de successeurs. Dans ce cas, l'algorithme retourne au nœud précédent et cherche un nœud qui n'a pas encore été visité. L'algorithme complet se trouve ci-dessous :

Fonction chaînage arriere
Paramètres : in BR, in BF, in listeButs.
```
    if est vide(listeButs) then
        res ← SUCCES
    else
        if demBut(premier(listeButs)) then
            res ← chaînageArriere(suite(listeButs))
        else
            res ← ECHEC
        end if
    end if
retourner res
```

Fonction demBut
Paramètres : in BR, in BF, in but.

```
if but ∈ BF then
    res ← SUCCES
else
    regles ← BR ; res ← ECHEC
    while regles = ∅ et res = SUCCES do
        r ← choix(regles) ; regles ← regles - r
        if conclusion(r) = fait then
            res ← chaînageArrière(BR, BF, premisse(r))
        end if
    end while
    retourner res
end if
```

Annexe B

Code

B.1 Hook de mise à jour des données d'un attribut calculé

Hook B.1 – Mise à jour des données d'un attribut calculé

```
class ComputedAttributeHook(SyncSchemaHook):
    """After adding relations which are in a computed attribute rule,
    the computed attribute is filled thanks to the rule."""
    __regid__ = 'schema.computed_attribute_after_add_entity'
    __select__ = SyncSchemaHook.__select__ & match_computed_attribute()
    events = ('after_add_relation',)

    def __call__(self):
        operation = SyncSchemaComputedAttributeOperation.get_instance(self._cw)
        operation.add_data((self.eidfrom, self.rtype, self.eidto))
        operation.postcommit_event()

class SyncSchemaComputedAttributeOperation(hook.DataOperationMixIn, hook.Operation):
    def postcommit_event(self):
        session = self.session
        schema = session.repo.vreg.schema
        for self.eidfrom, self.rtype, self.eidto in self.get_data():
            for entity in schema.entities():
                for attr in entity.attribute_definitions():
                    formula = attr[0].formula
                    if formula and self.rtype in formula:
                        rql = hook_rewrite_computed(formula, self.eidfrom, self.rtype, self.eidto,
                                                    schema.get_rules())
                        rset = session.execute(rql)
                        if rset:
                            for r, eclass in rset.rows:
                                if r:
                                    session.execute('SET X %(attr)s "%(r)s" WHERE X eid %(eclass)s'
                                                    % {'attr': attr[0], 'r': r, 'ecla
```

Hook B.2 – Sélecteur

```
class match_computed_attribute(EClassPredicate):
    """Returns 1 if the relation is in the formula of a  computed attribute."""
    def __call__(self, cls, req, rset=None, **kwargs):
        rtype = kwargs.pop('rtype')
        schema = req.vreg.schema
        for entity in schema.entities():
            for attr in entity.attribute_definitions():
                if attr[0].formula:
```

```
                    if rtype in attr[0].formula:
                        return 1
        return 0
```

Annexe C

Glossaire

Cloud computing technologie qui permet de mettre sur des serveurs localisés à distance des données de stockage ou des logiciels.

CSS3 langage servant à décrire la présentation des documents HTML et XML.

CSV format informatique ouvert représentant des données tabulaires sous forme de valeurs séparées par des virgules ou par un quelconque séparateur comme un point-virgule.

Cube composant logiciel de base de CubicWeb, comportant trois parties principales : le modèle de données, la logique métier, l'interface utilisateur, ainsi que des dépendances vers d'autres cubes.

CubicWeb framework logiciel libre implémenté par l'entreprise Logilab.

Debian système d'exploitation et une distribution de logiciels libres.

DOAP vocabulaire RDF décrivant les projets logiciels.

FOAF vocabulaire RDF permettant de décrire des personnes et les relations qu'elles entretiennent entre elles. C'est une application du Web sémantique.

Framework ensemble de composants logiciels permettant de créer l'architecture d'un logiciel.

Hadoop framework Java libre destiné à faciliter la création d'applications distribuées et échelonnables. Il permet aux applications de travailler avec des milliers de nœuds et des pétaoctets de données.

HTML5 cinquième version du format de données HTML conçu pour représenter les pages Web

Javascript langage de programmation de scripts utilisé dans les pages Web interactives

Mercurial logiciel de gestion de versions décentralisé qui administre les différentes versions d'un programme écrites par un ou plusieurs développeurs.

Moteur d'inférences logiciel ou composant logicile correspondant à un algorithme de simulation des raisonnements déductifs.

Ontologie Une ontologie est l'ensemble structuré des termes et concepts représentant le sens d'un champ d'informations. L'ontologie constitue en soi un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts.

Open data données diffusées de manière structurée selon une méthodologie et une licence ouverte garantissant son libre accès et sa réutilisation par tous, sans restriction technique, juridique ou financière.

Open source désigne un logiciel dont la license autorise la libre redistribution, l'accès au code source et la création des travaux dérivés.

Pattern variable valeurs de données qui peuvent correspondre avec n'importe quelle donnée ou variable.

Pyke Pyke est un moteur d'inférences de système expert, écrit en Python, qui introduit de la programmation logique inspirée de Prolog.

Python langage de programmation orienté objet.

RDF (Resource Description Framework) est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de ces descriptions.

Reasoner programme capable d'inférer des conséquences logiques à partir de faits et d'axiomes et de vérifier qu'une ontologie est cohérente.

Responsive Design notion de conception de sites Web qui permet à l'utilisateur de consulter le même site Web à travers une large gamme d'appareils (moniteurs d'ordinateur, smartphones, tablettes, TV, etc.) avec le même confort visuel.

RQL (Relationship Query Language) langage informatique d'exploitation de bases de données, utilisé par *CubicWeb*

SIOC vocabulaire utilisant RDF permettant de décrire des objets couramment utilisés sur les sites communautaires et leurs relations.

Sparql langage de requête et un protocole qui permet de rechercher, d'ajouter, de modifier ou de supprimer des données RDF disponibles à travers Internet.

Système experts outil capable de reproduire les mécanismes cognitifs d'un expert, dans un domaine particulier.

W3C (World Wide Web Consortium) est un organisme de normalisation à but non lucratif chargé de promouvoir la compatibilité des technologies.

Web sémantique Le Web sémantique est un mouvement collaboratif mené par le World Wide Web Consortium (W3C) qui favorise des méthodes communes pour échanger des données. Le Web sémantique vise à aider l'émergence de nouvelles connaissances en s'appuyant sur les connaissances déjà présentes sur Internet.

Yams signifie *Yet Another Magic Schema*. C'est un module python qui définit le schéma générique des relations et des entités de manière simple et générique.

Vsprint plage horaire hebdomadaire (les vendredi après-midi) où les membres du personnel travaillent en binôme sur un ensemble de tickets, généralement lié au développement d'outils internes.