

Travail de bachelor 2008

Filière Informatique de gestion

Web 3.0 : adaptation et personnalisation



Etudiant : Philippe Bonvin

Professeur : Mme Anne Le Calvé

Table des matières

TABLE DES MATIÈRES	2
1 INTRODUCTION	5
2 STRUCTURE DU DOCUMENT	5
3 DESCRIPTION DES NOTIONS	6
3.1 Evolution du web	6
Web	6
Web 2.0	6
Web 3.0 ou web sémantique	7
3.2 Ontologie	7
Les triplets	7
RDF	8
RDFS (RDF Schema)	8
OWL	8
Classe	9
Propriétés OWL	10
Domain and range	11
Raisonneur	11
SPARQL	11
Jena	12
3.3 Le projet Memoria Mea	12
Le moteur OntoMea	13
MemOnto	13
FOAF	13
GeoNames	13
3.4 User Modeling system	14
GenOUM	15
Différence entre adaptatif et personnalisable	15
4 TRAVAIL RÉALISÉ	16
4.1 Déroulement du projet (planning envisagé)	16
4.2 Analyse des fonctionnalités et de leur faisabilité	16
Comportements différents selon les connaissances de l'utilisateur sur le thème de la recherche	17
Requêtes se basant sur les habitudes de l'utilisateur	17
Zone de texte munie d'une fonctionnalité d'auto complétion désactivable	17
Zone qui affiche les résultats que l'on peut trier	18
Résultats différents suivant l'heure et la date du système, paramétrable	18

	Recherche allégée suivant les ressources mémoire et CPU libres de l'ordinateur, paramétrable.....	18
	Informations différentes selon la météo, paramétrable.....	18
4.3	Données et scénario	19
	Linked Data.....	21
	Utilisateur.....	22
	Station de ski.....	24
	Evaluations.....	26
	Météo.....	28
	Agenda	29
	Source des données.....	30
	Déductions	30
4.4	Choix de la technologie et des outils	31
	GWT Google Web Toolkit.....	32
	GWT designer.....	37
	GWT4NB.....	38
4.5	Développement et test	39
	Aperçu général.....	39
	Fonction Station.....	40
	Fonction Agenda.....	43
	Gestion du retour des résultats SPARQL.....	46
	Scénario : personnes et stations de ski	47
	Images associées Flickr.....	48
	Suggestions par météo.....	50
	Suggestions par évaluation.....	53
	Notification par mail.....	55
	Test	57
	Réalisation des fonctionnalités analysées	58
5	PROBLÈMES RENCONTRÉS.....	59
6	AMÉLIORATIONS FUTURES	60
7	CONCLUSION	61
8	BIBLIOGRAPHIE	62
9	TABLE DES ILLUSTRATIONS.....	64
10	GLOSSAIRE.....	66
11	DÉCLARATION SUR L'HONNEUR.....	67
12	REMERCIEMENTS.....	68

13 LISTE DES ANNEXES	68
13.1 Cahier des charges.....	68
13.2 Planning.....	68

1 Introduction

"Le fléau de mon existence est de faire des choses alors que je sais que l'ordinateur pourraient les faire pour moi." Dan Connolly, The XML Revolution.

Aujourd'hui l'utilisateur est considéré comme le point central des systèmes d'information. De plus en plus de systèmes s'adaptent aux interactions et au profil de l'utilisateur. Leur interface et/ou leur contenu peuvent être personnalisés selon l'utilisateur. Alors que le web 2.0 continue sa mise en place, tous les grands acteurs du web travaillent actuellement sur les nouvelles technologies nécessaires pour permettre aux données de prendre un sens dans le web : le **web 3.0** (web sémantique). La puissance et l'enjeu de ce web sémantique viennent de la mise en relation de la multitude de données existantes, et de leur valorisation par la déduction de nouvelles données et de nouvelles connexions. L'objectif de ce travail de diplôme est de mettre en œuvre un **système adaptatif et personnalisé** (*user modeling system*) utilisant la puissance des techniques du web 3.0. Le scénario utilisé est celui du projet Memoria Mea (www.memoria-mea.ch) de système de gestion d'information personnelle multimédia.

Les utilisateurs travaillent tous de manière différente, chaque utilisateur attend un résultat différent d'une requête. L'utilisateur perd beaucoup de temps pour configurer son programme de recherche chaque fois qu'il désire un autre résultat ou qu'il change ses habitudes. L'humain modifie fréquemment sa manière d'explorer des données. Un autre problème présent est qu'il cherche souvent des informations liées à d'autres sujets mais l'ordinateur ne le sait pas. Celui-ci contient énormément de données mais il ne sait pas vraiment les interpréter. De plus chaque personne étant différente, un même résultat ne peut satisfaire tout le monde. Dans le cadre de Memoria-Mea, nous souhaiterions pouvoir adapter les résultats de recherche à l'utilisateur.

Le travail à réaliser sera le développement d'un prototype d'un **système adaptatif et personnalisé** basé sur le **web sémantique**. Il sera fortement lié à l'**ontologie** modèle utilisateur **GenOUM** et au moteur sémantique **OntoMea** du Project **Memoria Mea**. Ci-dessous figure une description des notions indispensables à une bonne compréhension de la problématique du travail.

2 Structure du document

Suite à cette introduction, il me semble nécessaire d'expliquer la structure de ce dossier. Pour information, les chapitres suivent le planning envisagé du déroulement du projet.

Dans un premier temps, un chapitre intitulé Description des notions nous permet la compréhension des concepts nécessaires à l'ouvrage de mon travail. Ces notions figurant dans l'intégralité des chapitres de ce dossier, il est nécessaire de bien les comprendre. Ce chapitre correspond aux deux premières étapes de mon planning.

A ce stade, nous sommes bien immergés dans les thèmes du web sémantique et du "User modeling system", nous pouvons alors parler de la phase centrale de mon travail. Celle-ci se compose du choix des technologies et des outils, de l'analyse d'idées de fonctionnalités, du scénario, du développement et des tests. Les idées de fonctionnalités sont issues d'un "brainstorming" réalisé par mes propres soins. Elles comportent des idées regroupant des recherches sémantiques adaptables et personnalisables. Le développement des fonctionnalités et les tests seront réalisés suite à cette partie suivant la faisabilité. Le but de ce travail est de montrer les possibilités du web sémantique lié à un "User modeling system" afin de faire des recherches ; cela sous forme d'un prototype. C'est un "proof of concept" d'un modèle utilisateur de recherche sémantique. Ces chapitres correspondent aux phases 3 et 4 de mon planning.

Finalement le reste de ce rapport mentionne les problèmes rencontrés, les améliorations futures et une conclusion.

3 Description des notions

3.1 Evolution du web

Web

Le **World Wide Web** habituellement surnommé le web "est un système hypertexte public fonctionnant sur internet qui permet de consulter, avec un navigateur, des pages mises en ligne dans des sites." [Wikipedia]

Le web et internet sont souvent compris comme synonyme, ce qui est un abus de langage. Le web n'est qu'une application d'internet. D'autres applications d'internet sont les emails, les chats,...

Tim Berners-Lee et Robert Cailliau sont connus comme les fondateurs du web vers 1989 au CERN en Suisse. Dans les premier temps, une blague s'est rependue prétendant que WWW signifiait World Wide Wait, à cause de sa lenteur probablement.

Web 2.0

Le terme web 2.0 a été inventé pour désigner l'évolution du web. C'est un mot controversé puisque le web 1.0 n'a jamais officiellement été défini. Son principe est de rendre le web vivant et de permettre l'interaction entre les utilisateurs.

Avant cette évolution le web contenait des pages statiques rarement mises à jour. Puis avec le temps l'utilisateur est devenu de plus en plus impliqué, c'est ce que beaucoup de gens nomment le web 2.0. Comme bon exemple de cette évolution, on peut parler des réseaux communautaires, des wikis, des blogs, des possibilités de faire des commentaires, des votes,...

Web 3.0 ou web sémantique

Ce terme dérivé du web 2.0 ne va pas vraiment dans le même sens que celui-ci. On parle plutôt de web sémantique dont Tim Berners-Lee est l'inventeur.

Le web contient une masse énorme d'informations qui n'a pas forcément de sens pour un ordinateur. Le but du web sémantique est de donner un sens à toutes ces données. C'est comme si le web était considéré comme une gigantesque base de données. Pour pouvoir traiter ceci, des mécanismes particuliers sont mis en œuvre tels que l'utilisation d'ontologies décrites dans des langages spécifiques du XML (RDF, OWL).

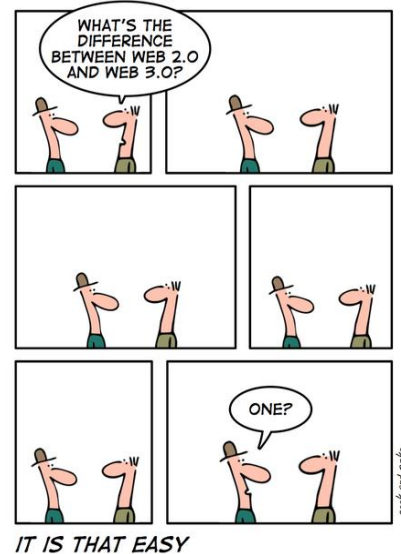


Figure 1 : <http://pages.usherbrooke.ca>

3.2 Ontologie

Une ontologie est utilisée pour modéliser le monde réel, ou plus particulièrement un domaine du monde réel. Elle permet de décrire un ensemble de concepts d'un domaine dans le but de lui donner un sens et de relier les concepts entre eux. Elle permet aussi de déduire de nouvelles connaissances à partir des informations disponibles, ce qui s'appelle des inférences. Plusieurs outils existent afin de créer et d'éditer des ontologies. Protégé est un éditeur open source gratuit d'ontologie. Dans le cadre de ce travail, les ontologies seront manipulées grâce à un moteur sémantique nommé OntoMea.

Les triplets

Le web sémantique utilise la forme de triplets pour structurer les données. Il permet de montrer les relations des ressources entre elles. Le concept de triplets est à la base du langage RDF. Ces triplets se composent des éléments suivants:

Un sujet qui est en général une instance. A savoir qu'une instance évoque une information précise. Par exemple pour une gestion des utilisateurs, Philippe serait une instance d'utilisateur. Un objet est l'élément décrit par le triplet. Pour

information chaque instance possède un URI définie permettant de distinguer la ressource de façon unique.

Un prédicat est un type de propriété applicable au sujet. Une propriété permet de mettre en relations des instances avec d'autres instances ou avec des littéraux. Un objet est une autre instance ou un littéral. Un graphe RDF est la reproduction graphique de triplets.

Ci-dessous figure un exemple:

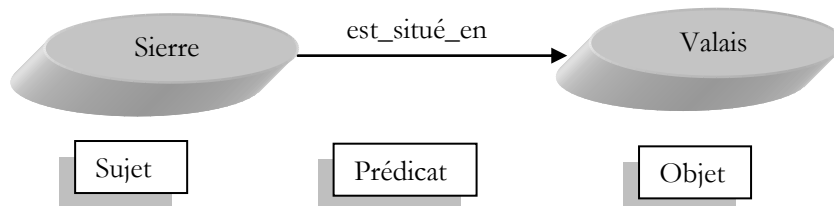


Figure 2 : Graphe d'un triplet

RDF

Resource **D**escription **F**ramework est un langage dérivé du XML qui est à la base du web sémantique. Il sert dans la majorité des cas à stocker des triplets.

Voilà ci-dessous le résultat au format RDF du triplet de l'exemple précédent :

```
<rdf:Description about="#sierre">
  <shema:cantons>Valais</shema:cantons>
</rdf:Description>
```

Figure 3 : Triplet au format RDF/XML

RDFS (RDF Schema)

RDFS est une expansion du langage RDF. Il y ajoute des représentations de classe, de sous-classe, de sous propriété, de domaine et de portée. Il est utilisé dans le langage OWL.

OWL

OWL (web ontology language) est un langage de la famille du XML qui permet de définir des ontologies structurées. Ce dialecte améliore l'interprétation du contenu web par l'ordinateur en supportant le XML, RDF et RDFS.

OWL dispose de trois sous langages différents :

- OWL/lite est une version allégée de OWL
- OWL/DL est une version plus complète. C'est la version la plus utilisée.
- OWL/Full est la version la plus complète.

Classe

Pendant la création d'une ontologie OWL, il est important de représenter les éléments nécessaires en classe. Dans un triplet le sujet et l'objet sont des instances de classe. RDFS, OWL disposent d'une syntaxe spéciale pour spécifier les caractéristiques d'une classe.

Ex:

- `rdfs:label` : le nom de la classe
- `rdfs:comment` : commentaires
- `rdfs:subClass` : pour indiquer un héritage
- `owl:versionInfo` : information sur la version
- `owl:disjointWith` : définit qu'une classe ne peut pas appartenir à une autre classe mentionnée par cette caractéristique
- ...

Le programme Protégé permet de gérer cela avec une interface graphique. En voici l'interface pour administrer les classes et leurs hiérarchies:

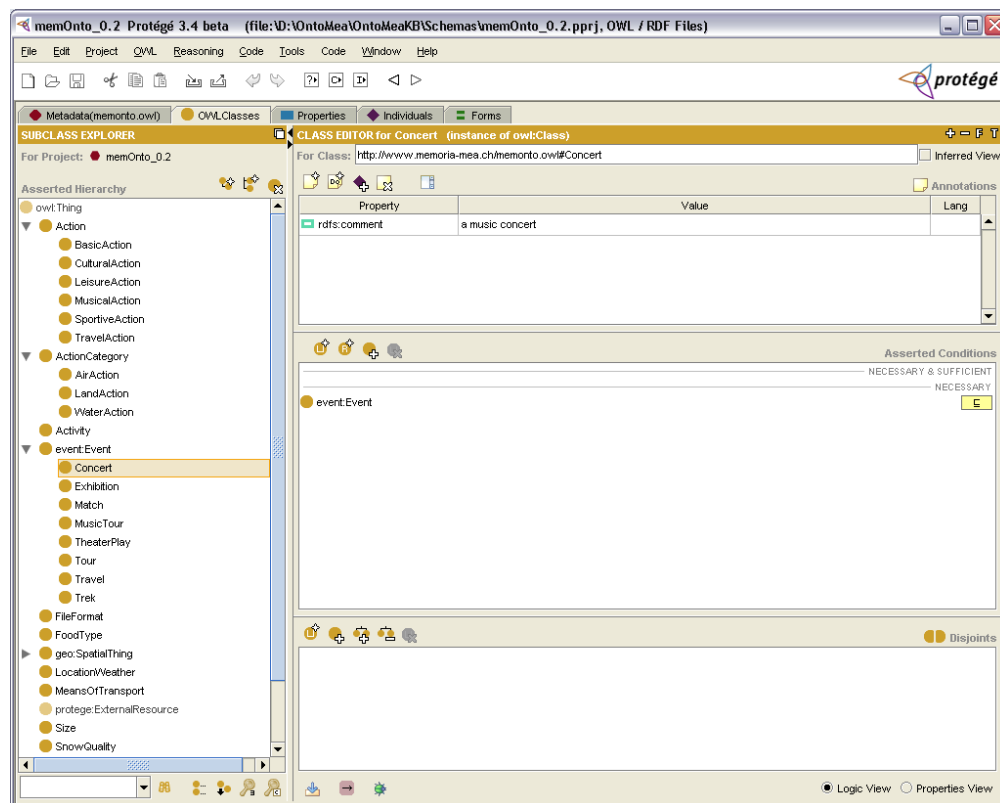


Figure 4 : Gestion des classes avec Protégé pour l'ontologie MemOnto

Propriétés OWL

Dans un triplet le prédicat est une propriété. La façon de gérer les propriétés est similaire à la gestion des classes. Une syntaxe spéciale existe pour en spécifier les caractéristiques.

Ex:

- `Owl:objectProperty` : nom de la propriété
- `rdfs:SubPropertyOf` : pour utiliser un héritage
- `owl:FunctionalProperty` : indique que l'ensemble des objets liés à un sujet par la propriété sont égaux
- `owl:InverseFunctionalProperty` : définit que l'ensemble des sujets liés à un objet par cette propriété sont égaux
- `owl:SymetricProperty` : établit que si un sujet A est attaché à un objet B par la propriété P alors B est uni à A par la propriété P
- `owl:TransitiveProperty` : démontre que si $A \rightarrow B \rightarrow C$ alors $A \rightarrow C$
- `owl:inverseOf` : permet d'indiquer une propriété inverse
- ...

Voici l'interface de Protégé pour gérer les propriétés:

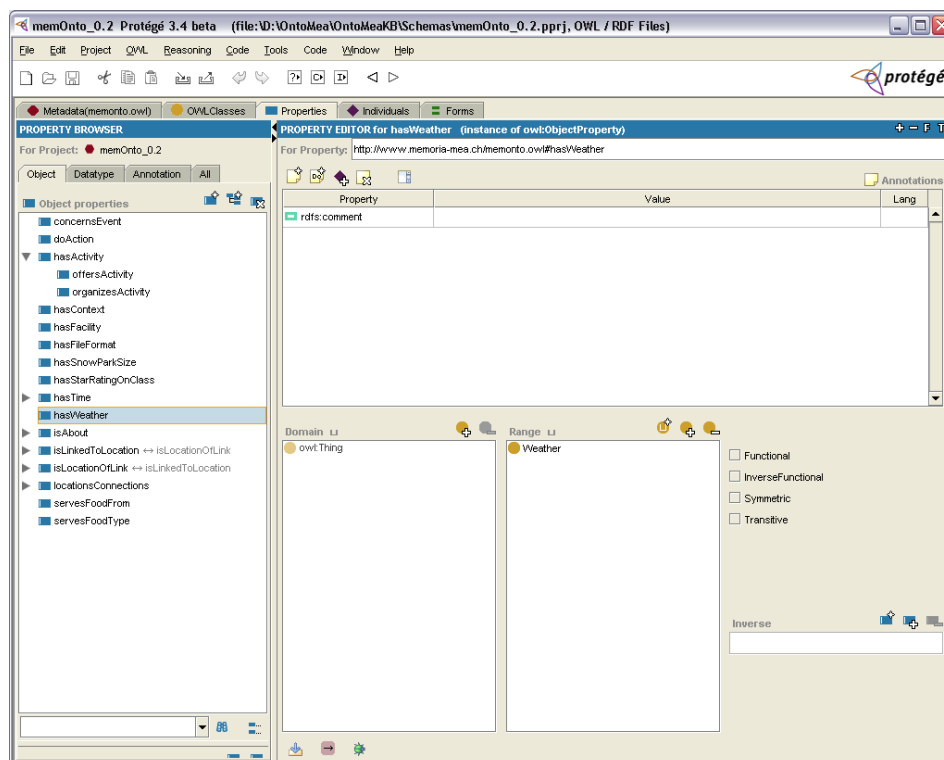


Figure 5 : Gestion des propriétés avec Protégé pour l'ontologie MemOnto

Domain and range

Chaque propriété dispose d'un domaine et d'une portée (Domain and range) afin de déterminer les inférences. Le domaine permet de définir, lors de l'inférence, à quelle classe le sujet d'une propriété va être lié. La portée précise lors de l'inférence à quelle classe l'objet va appartenir. Par exemple la propriété "habite" pourrait avoir comme domaine l'objet "pays" et comme portée l'objet "personne". En ajoutant le triplet "Philippe habite la Suisse". Alors Philippe sera interprété comme une instance d'objet "personne" et la Suisse comme une instance d'objet "pays".

Voici l'interface de Protégé qui permet de gérer ceci:



Figure 6 : Gestion du "Domain and range" par Protégé pour la propriété "doAction" de MemOnto

Raisonneur

Un raisonneur est un moteur d'inférences, il permet d'effectuer des déductions à partir des données de base. Actuellement, il en existe plusieurs : Pellet, Racer,...

SPARQL

SPARQL (Simple Protocol and RDF Query Language) est un langage pour effectuer des requêtes sur un graphe de données RDF. Il ressemble fortement au langage SQL, mais il se base sur les triplets. La meilleure manière de le présenter est de montrer un exemple :

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE{
    ?person foaf:name ?name.
}
ORDER BY ASC(?name)
```

Figure 7: Requête SPARQL

Une Requête SPARQL se compose des parties suivantes:

- PREFIX c'est le préfixe des URI des ressources à utiliser.
- La clause SELECT est un type de requête. Elle est analogue à celle du SQL. Elle permet de définir les ressources retournées dans le résultat de

la requête. Il existe plusieurs types de requête. ASK est un autre type de requête qui permet de tester si un motif d'interrogation a une solution ou non.

- La clause WHERE permet de filtrer les résultats, ceci est réalisé par des triplets. Il faut remarquer qu'ils doivent se terminer pas un point. Mais l'utilisation d'un point-virgule (;) à la fin d'un triplet autorise à ne pas avoir à préciser le sujet dans la ligne suivante.
- La clause ORDER BY qui est optionnelle permet de trier les résultats. ASC par ordre croissant et DESC par ordre décroissant.

Jena

Jena est un Framework Java pour réaliser des applications web sémantique. Il met à disposition des outils de programmation pour gérer RDF, RDFS, OWL et SPARQL. Il dispose aussi d'un moteur d'inférences basé sur des règles de déductions. C'est un projet Open Source développé par HP Labs Semantic Web Programme. Il est possible de le télécharger sous leur site officiel <http://jena.sourceforge.net>.

Ce Framework inclut :

- Une API RDF
- La possibilité de lire et d'écrire des données RDF dans les formats RDF/XML, N3 et N-Triples
- Une API OWL
- Le stockage en mémoire ou persistant d'ontologies
- Un moteur de Requêtes SPARQL
- Un moteur d'inférences

3.3 Le projet Memoria Mea

Memoria-Mea est un projet de gestion d'informations personnelles (PIM : Personal Information Management). Il vise à développer un système PIM pour gérer le contenu multimédia en aidant les utilisateurs dans l'organisation et le recouvrement d'informations à travers la collection entière de documents multimédia qu'ils ont rassemblés pendant des activités de vie quotidienne. Ce projet regroupe les 4 HES-SO suivantes : Valais, Fribourg, Arc, Genève et regroupe une vingtaine de chercheurs sur des technologies diverses. L'un des axes de ce projet est le stockage de données sémantiques et leur exploitation (via le moteur OntoMea).

Le moteur OntoMea

C'est un moteur sémantique de base de connaissances développé par l'HES-SO Valais dans le cadre du projet susnommé Memoria-Mea. Il se compose d'un moteur en Java et de clients de tests en .NET. Il est capable de traiter des ontologies stockées en local. Le moteur est basé sur la librairie Jena qui permet de traiter les ontologies. Les utilisateurs peuvent se servir de ce moteur pour consulter leurs données. OntoMea est accessible via différents web services, dont une possibilité via des requêtes SPARQL.

MemOnto

MemOnto est une petite ontologie OWL développée pour le projet Memoria-Mea. Elle permet de lier des ontologies déjà existantes comme FOAF, event, time, GeoNames. Elle permet aussi d'ajouter des notions qui sont introuvables dans ces ontologies. Par exemple, il y a quelques concepts/propriétés pour décrire n'importe quel fichier numérique et aussi des notions pour décrire des actions et des activités.

FOAF

FOAF est l'acronyme de **F**riend **O**f **A** Friend qui signifie en français l'ami d'un ami. C'est un projet qui vise à utiliser RDF afin de créer un type de document dédié aux personnes. FOAF est accessible à travers le monde, il permet de décrire chaque personne et les relations qu'elles entretiennent entre elles. Le site officiel est accessible sur cette adresse: <http://www.foaf-project.org>.

GeoNames

C'est une base de données sémantique accessible par internet dédiée à la géographie.

Chaque lieu possède des renseignements tels que la latitude, la longitude, l'altitude, la population, la région ou le canton, le code postal, le pays tout cela traduit en différentes langues

Voici ci-dessous une partie de la base GeoNames pour Sierre:

```
<rdf:RDF>
  <Feature rdf:about="http://sws.geonames.org/2658606/">
    <name>Sierre</name>
    <featureClass rdf:resource="http://www.geonames.org/ontology#P"/>
    <featureCode rdf:resource="http://www.geonames.org/ontology#P.PPL"/>
    <inCountry rdf:resource="http://www.geonames.org/countries/#CH"/>
    <population>14813</population>
    <wgs84_pos:lat>46.2919177499138</wgs84_pos:lat>
```

```
<wgs84_pos:long>7.53559112548828</wgs84_pos:long>
<parentFeature rdf:resource="http://sws.geonames.org/2658205/" />
<nearbyFeatures rdf:resource="http://sws.geonames.org/2658606/nearby.rdf" />
<locationMap rdf:resource="http://www.geonames.org/2658606/sierre.html" />
</Feature>
</rdf:RDF>
```

Figure 8 : Extrait GeoNames pour Sierre

Cette partie nous fournit des informations sur la ville de Sierre. Sa population est de 14'813 habitants. Ses coordonnées GPS sont : longitude 46.2919177499138° et latitude 7.53559112548828°. Elle contient aussi un lien avec les villes et commerces proches de celle-ci.

3.4 User Modeling system

C'est le principe d'avoir un système qui s'adapte aux besoins de l'utilisateur. Il emploie pour cela un *modèle utilisateur* (User model).

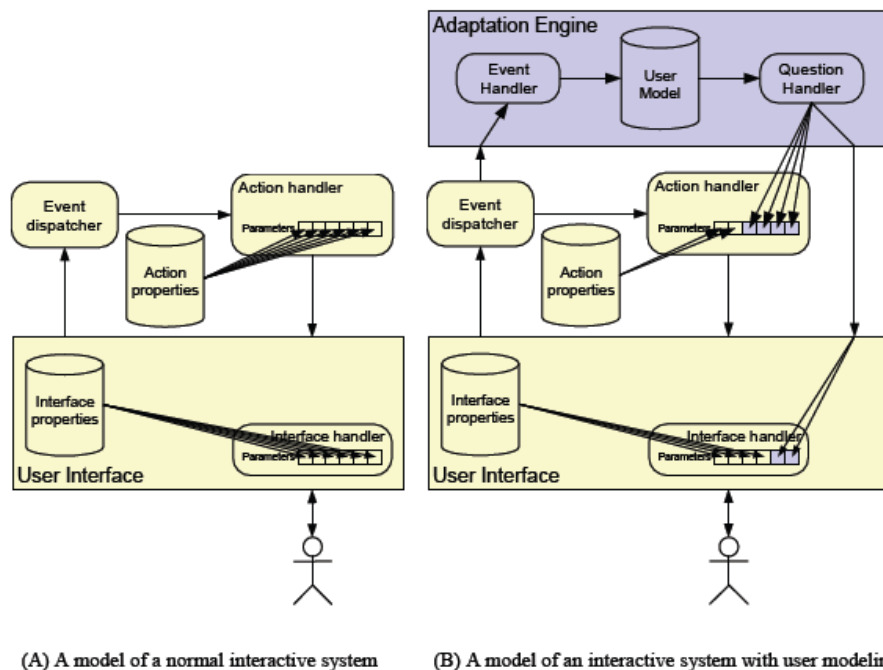


Figure 9 : Comparaison entre système classique et système avec modèle utilisateur¹

Le modèle utilisateur comprend des informations sur l'utilisateur d'un système telles que les buts, besoins, préférences ou même ses intentions. Certains modèles

¹ P.T. de Vrieze, P. van Bommel, Th.P. van der Weide - *A Generic Engine for User Model Based Adaptation* - Proceedings of the User Interfaces for All workshop – 2004.

utilisateurs plus avancés peuvent contenir des renseignements liés à l'état psychique, émotionnel, physique, le contexte dans lequel se trouve l'utilisateur etc.

Ces informations peuvent être acquises de plusieurs façons :

- explicite ou manuelle : l'utilisateur saisit son profil
- implicite ou automatique : les informations sont déduites de l'interaction de l'utilisateur avec le système.

Nombre de chercheurs se sont penchés sur la définition du modèle utilisateur et ce, dans de nombreux domaines (e-learning, interfaces adaptatives,...)

Grâce aux travaux de Liana Razmerita², la HES-SO Valais a pu définir son propre modèle utilisateur sémantique nommé GenOUM.

GenOUM

Generic ontology based User Modeling est une ontologie générique de système adaptatif et personnalisé (user modeling system). GenOUM permet de décrire le profil des utilisateurs, leurs connaissances, intérêts, buts, évaluations, comportement et personnalité.

Différence entre adaptatif et personnalisable

Adaptatif correspond à la particularisation du résultat aux exigences distinctives de l'utilisateur. C'est un procédé qui ne nécessite pas d'interaction avec l'utilisateur, il se fait automatiquement. Comme exemple, je pourrais citer les zones de recherche qui se pré-remplissent grâce à nos anciennes requêtes.

Personnalisable implique le principe de pouvoir donner une caractéristique unique à quelque chose. L'utilisateur doit fournir ses informations. Un exemple serait les barres d'outils de Word que l'on peut déplacer à sa guise.

² Liana Razmerita - *Modèle Utilisateur et Modélisation Utilisateur dans les Systèmes de Gestion des Connaissances: une Approche fondée sur les Ontologies* - Doctorat - Toulouse III, France 2003

4 Travail réalisé

4.1 Déroulement du projet (planning envisagé)

Le projet se découpe en cinq phases comme suit :

Phase 1 : Analyse de l'existant 15.09.2008 – 21.09.2008

- Web sémantique, ontologies, langages
- Projet Memoria-Mea
- User model et GenOUM

Phase 2 : Immersion 22.09.2008 – 28.09.2008

- Cahier des charges
- Prise en main des outils
- Petits tests

Phase 3 : Immersion 29.09.2008 – 05-10.2008

- -Compréhension d'OntoMea et de GenOUM
- -Analyse des fonctionnalités et de leur faisabilité

Phase 4 : Développement et test 06.10.2008 – 26.10.2008

Phase 5 : Rapport final 27.10.2008 - 10.11.2008

4.2 Analyse des fonctionnalités et de leur faisabilité

Cette partie expose une analyse d'idées de fonctionnalités regroupant la recherche sémantique adaptable et personnalisable. J'analyse les faisabilités de celles-ci en essayant de trouver une façon de les réaliser. Ces idées sont issues d'un "brainstorming" réalisé par mes propres soins. Ces fonctionnalités seront exploitées suivant les possibilités offertes par la technologie choisie et selon le temps imparti. Certaines de celles-ci risquent de ne pas être utilisées par mon prototype mais pourront être utiles pour de futurs projets.

Voici la liste de ces fonctionnalités:

Comportements différents selon les connaissances de l'utilisateur sur le thème de la recherche

Requêtes se basant sur les habitudes de l'utilisateur

Zone de texte munie d'une fonctionnalité d'auto complétion désactivable

Zone qui affiche les résultats que l'on peut trier
Résultats différents suivant l'heure et la date du système, paramétrable
Recherche allégée suivant les ressources mémoire et CPU libres de l'ordinateur, paramétrable
Informations différentes selon la météo, paramétrable

Figure 10 : Liste des fonctionnalités analysées

Comportements différents selon les connaissances de l'utilisateur sur le thème de la recherche

Un programme s'adaptant aux connaissances de l'utilisateur sur le thème de sa recherche serait une fonctionnalité très intéressante. En parcourant les possibilités offertes par l'ontologie GenOUM, j'ai pu remarquer que cette fonctionnalité serait envisageable. Grâce à l'ontologie GenOUM, il est possible de gérer des intérêts, des activités, d'évaluer des connaissances et encore bien d'autres choses qui pourront nous être utiles.

Requêtes se basant sur les habitudes de l'utilisateur

L'idéal serait d'avoir un programme qui s'adapte à nos habitudes. Une fois de plus, GenOUM nous permettrait de gérer les habitudes des usagers. Le seul problème qui se pose c'est de trouver un moyen d'enregistrer ses habitudes pour qu'il n'ait pas besoin de les saisir lui-même. En parlant avec mes responsables, nous avons imaginé la possibilité d'utiliser un ancien projet qui se nomme Chaméléon. Ce programme se base sur les logs d'un serveur web pour analyser les comportements utilisateurs. Etant donné que la saisie de données n'est pas le centre de mon travail, je n'ai pas analysé en détail cette solution.

Zone de texte munie d'une fonctionnalité d'auto complétion désactivable

En ce qui concerne les systèmes adaptatifs, des zones de texte suggérant des propositions lors du début de leur saisie sont très pratiques. Les propositions se basent sur les habitudes et le comportement de l'utilisateur. Un excellent exemple de ce genre de fonctionnalités, est la "awesome bar" du navigateur Mozilla firefox et celle du navigateur Chrome de Google.

Pour réaliser une telle zone de texte pour le langage Java, j'ai trouvé une classe qui surdéfinit la classe TextBox ce qui permettrait de lier une liste de suggestions à une zone de texte. D'autres technologies disposent de composants tout faits le

permettant. Au niveau des données, pour la liste des suggestions, GenOUM me permettrait d'utiliser les habitudes et le comportement des utilisateurs.

Zone qui affiche les résultats que l'on peut trier

Il est toujours agréable de pouvoir trier des résultats sur différents critères. Ce choix est réalisable soit au niveau de la requête soit plus tard lors de l'affichage dépendant du format de celui-ci. Pour la requête, il suffit de changer, par exemple, la condition ORDER BY d'une requête SPARQL. Au niveau de l'affichage, je pourrais entre autres, en Java, facilement reprendre une classe Java que j'ai découverte lors d'un ancien projet. Celle-ci surdéfinit l'objet Table et permet de trier cet objet pour chaque colonne.

Résultats différents suivant l'heure et la date du système, paramétrable

Un résultat différent suivant l'heure et la date du système pourrait être très agréable. Cela serait réalisable en exécutant une requête différente qui a comme paramètre la date et l'heure actuelle. Il est possible, notamment, en Java avec la fonction `Date.getTime()` ou en instanciant un nouvel objet `GregorianCalendar`, de récupérer la date et l'heure à un moment donné.

Recherche allégée suivant les ressources mémoire et CPU libres de l'ordinateur, paramétrable

Souvent les utilisateurs réalisent beaucoup de chose en même temps sur l'ordinateur. Il est d'ailleurs fort désagréable de devoir attendre. Une recherche allégée serait très appréciable dans ces conditions. Ce serait tout à fait possible de faire une recherche simplifiée quand l'ordinateur est surchargé. Il faudrait juste détecter les ressources mémoires utilisées et aussi celles employées par le processeur à un moment donné pour déclencher cette recherche abrégée. Il est partiellement possible de le réaliser en Java puisque les informations sur la mémoire sont récupérables par contre les ressources du processeur ne le sont pas. Il est cependant envisageable de faire une bibliothèque en langage C sous forme de dll permettant de connaître les ressources utilisées par le processeur. Par la suite JNI (Java Native Interface) nous permet d'accéder à cette dll.

Informations différentes selon la météo, paramétrable

Une météo liée à un résultat de recherche est pratique. Une telle fonctionnalité est tout à fait réalisable en utilisant un web service ou des flux RSS.

Dans un premier temps, j'ai fait une recherche pour trouver une façon d'acquérir des informations sur la météo actuelle et les prévisions. Un web service météo gratuit fourni par <http://www.webservicex.net>. Ce service ne m'a pas convaincu car il ne proposait la météo que pour une dizaine de villes différentes en Suisse

ce qui n'est pas assez complet. Puis j'ai obtenu un résultat semblable en essayant un autre web Service gratuit intégré directement dans l'outil de développement Java NEtBeans 6.1. J'ai alors demandé aux personnes de météo suisse.ch s'ils proposaient un service gratuit de ce genre. Suite à une réponse négative, Fabian Cretton m'a parlé d'un outil développé dans le cadre des projets de l'institut de recherche de l'HES-SO permettant d'extraire les informations d'un site web directement sous forme sémantique. Cet outil se nomme SemWeC (Semantic Web Crawler). Les données extraites sont sur les pages de prévisions détaillées par localité. Exemple pour Crans-Montana : http://www.meteosuisse.admin.ch/web/fr/meteo/previsions_en_detail/prevision_par_localite.html?language=fr&plz=montana&x=0&y=0 L'automatisation de cet outil est tout à fait possible. Mais dans le cadre de notre prototype, nous allons générer ces données manuellement.

4.3 Données et scénario

Pour la réalisation d'un prototype de système adaptatif et personnalisé basé sur le web sémantique, il est important de se donner un cadre afin de pouvoir montrer un exemple concret dans un domaine. Au vu de l'étendue des possibilités du web sémantique, ce chapitre sera consacré à l'explication du scénario et des données utilisés dans mon prototype.

Etant passionné de snowboard et de sport d'hiver, j'ai décidé de baser le scénario de ce travail sur ce thème. J'ai toujours rêvé d'une application qui m'indique la station idéale pour avoir les meilleures conditions par rapport à mes intérêts. J'ai alors pensé à enrichir les ontologies utilisées par OntoMea avec la possibilité de stocker des stations de ski, des avis d'utilisateurs et d'autres informations que je vais détailler dans les prochains paragraphes.

Il est à noter que certaines dates et certaines données météo paraissent incohérentes du fait qu'elles ont été créées manuellement pour la démonstration.

Il me semble important d'introduire deux mots anglais qui sont des centres d'intérêts.

Halfpipe : sorte de demi-tube en neige dans lequel les skieurs et snowboarders peuvent réaliser des sauts.

Snowpark : espace composé de différents sauts ainsi que de barres en métal permettant d'y réaliser des figures.

Voici à quoi ressemble un halfpipe :



Figure 11 : Exemple d'un halfpipe (image de <http://www.freestyleterrain.com>)

Voici à quoi ressemble un snowpark



Figure 12 : Exemple d'un snowpark (image de <http://www.skipass.com>)

Dans un premier temps, je vais parler des "Linked Data" qui peuvent être très utiles. Puis je vais expliquer les propriétés employées au niveau de l'utilisateur, des stations de ski, des évaluations, de l'agenda et de la météo. Par la suite, je vais expliquer d'où viennent ces données et les déductions qui pourront en être faites.

Voici un schéma qui présente les différentes données du scénario, liées aux principales ontologies : MemOnto, GenOUM, FOAF, GeoNames:

User Modeling Tailored results

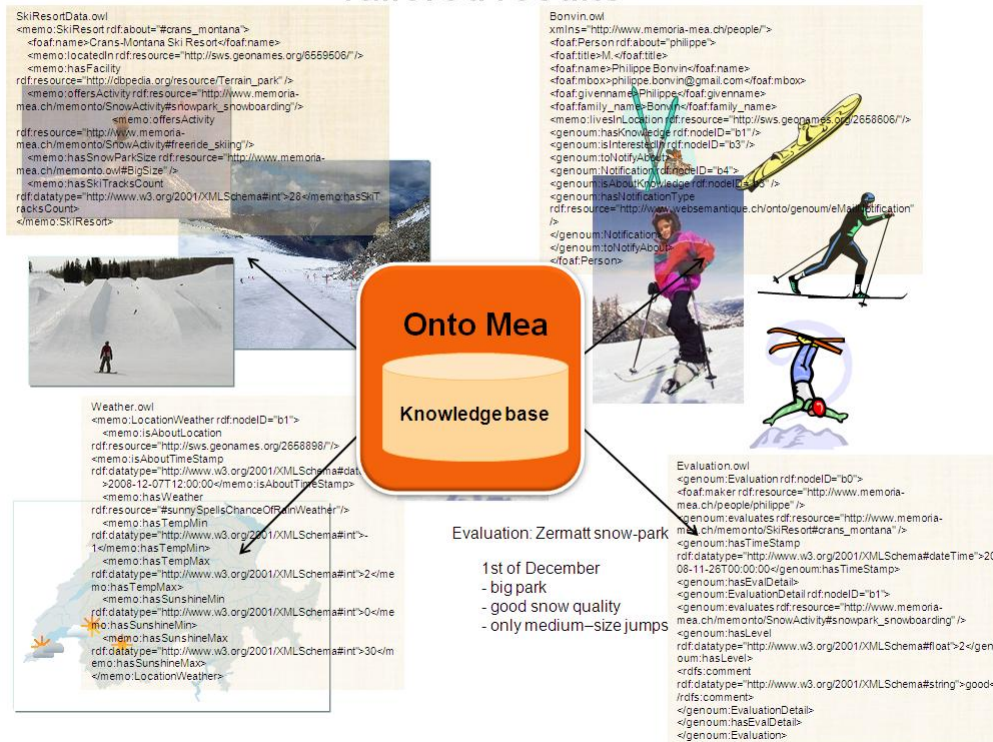


Figure 13 : Schéma représentant les données liées de mon application

Linked Data

Le premier besoin d'évolution du web de page connectée à un web de "Linked Data" (données connectées) est d'identifier chaque donnée du web à une ressource unique. Pour faire partie du web sémantique toutes les données, ressources (Data) doivent être identifiées par un URI.

De plus cela permet à l'ordinateur de savoir par exemple que l'information "Suisse", "Switzerland", "Die Schweiz" ou "瑞士" parle de la même ressource. Pour cet exemple l'URI pour la Suisse de la base GeoNames est: <http://sws.geonames.org/2658434/>.

Pour mon prototype, je vais utiliser les données de GeoNames, DBPedia.

GeoNames a déjà été introduit, c'est pour cela que je vais passer à l'explication de DBPedia. C'est l'effort d'une communauté pour extraire de façon structurée,

au format RDF, les informations de Wikipedia et de les rendre accessibles par le web. Cela permet d'y faire des requêtes sophistiquées et de les lier à d'autres informations.

Voici une représentation sous forme de schéma:

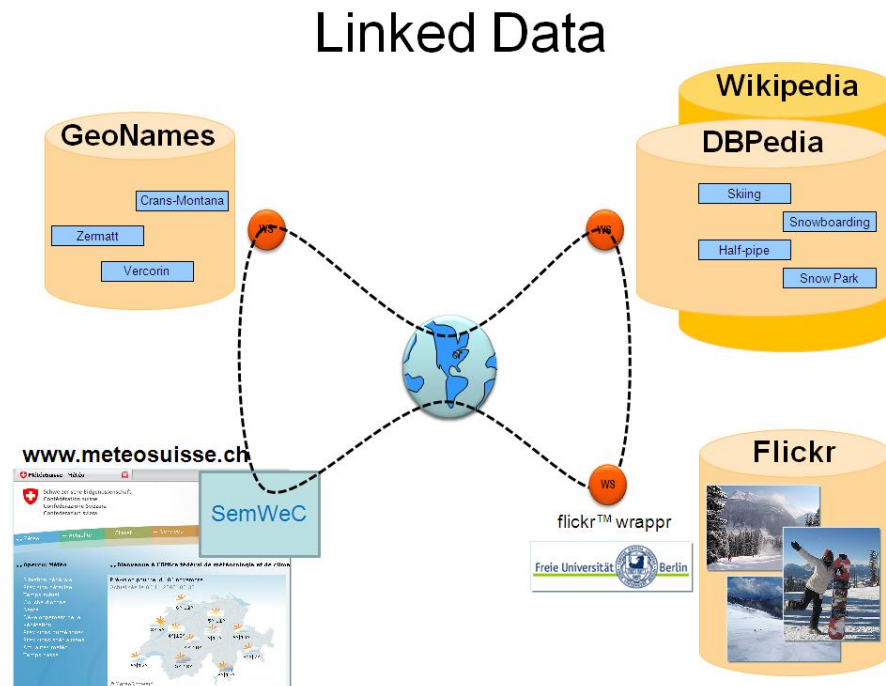


Figure 14 : Schéma des Linked Data

Utilisateur

Les informations pour l'utilisateur sont stockées dans des fichiers RDF. Je vais citer les informations les plus importantes:

La première ligne du fichier indique la version du format XML utilisé ainsi que son encodage. Les lignes suivantes servent à référencer les préfixes des ontologies utilisées (FOAF, MemOnto, GenOUM,...) et elles indiquent aussi l'URI de cette instance. Le but premier de GenOUM est de montrer le niveau de connaissance d'une personne. Ce niveau de connaissance est décrit par le concept "Knowledge" qui permet de spécifier sur quoi porte la connaissance (isKnowledgeAbout) ainsi que le niveau (hasLevel) de celle-ci.

Le reste du document contient des informations propres à l'utilisateur :

title : titre de la personne ex: M., Mme,...

name : le prénom et le nom de famille ex: Philippe Bonvin

- mbox : adresse email
- givenname : le prénom
- family_name : le nom de famille
- livesInLocation : l'URI du lieu d'habitation
- hasKnowledge : dans notre cas, c'est un nœud blanc qui pointe sur l'URI d'une connaissance ex: <http://dbpedia.org/resource/Snowboarding>. Un nœud blanc est utilisé pour ne pas répéter plusieurs fois la même ressource sans pour autant lui attribuer un URI spécifique. Ce nœud contient aussi le niveau de la personne dans cette connaissance. Celui-ci se situe de 0 à 20 d'après ce que nous avons défini.
- isInterestedIn : un nœud lié à l'URI d'une activité ex: www.memoria-mea.ch/memonto/SnowActivity#snowpark_snowboarding. Ce nœud renferme aussi une mesure pour indiquer le niveau de la personne dans cette activité. Ses valeurs peuvent varier comme celui des connaissances.
- toNotifyAbout : contient les URI du type d'activité et d'intérêt dont l'utilisateur veut être notifié ainsi que du type de notification. Dans notre scénario, on utilise seulement le mail, d'autres types de notifications pourraient être utilisés dans le futur.

Ci-dessous figure un exemple de données pour un utilisateur :

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xml:base="http://www.memoria-mea.ch/people/"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:memo="http://www.memoria-mea.ch/memonto.owl#"
xmlns:genoum="http://www.websemantique.ch/onto/genoum/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns="http://www.memoria-mea.ch/people/">
<foaf:Person rdf:about="philippe">
<foaf:title>M.</foaf:title>
<foaf:name>Philippe Bonvin</foaf:name>
<foaf:mbox>bonvphil@students.hevs.ch</foaf:mbox>
<foaf:givenname>Philippe</foaf:givenname>
<foaf:family_name>Bonvin</foaf:family_name>
<memo:livesInLocation rdf:resource="http://sws.geonames.org/2658606/" />
```



```

<genoum:hasKnowledge rdf:nodeID="b1" />
<genoum:isInterestedIn rdf:nodeID="b3" />
<genoum:toNotifyAbout>
<genoum:Notification rdf:nodeID="b4">
<genoum:isAboutKnowledge rdf:nodeID="b3" />
<genoum:hasNotificationType
rdf:resource="http://www.websemantique.ch/onto/genoum/eMailNotification"/>
</genoum:Notification>
</genoum:toNotifyAbout>
</foaf:Person>
<genoum:Knowledge rdf:nodeID="b1">
<genoum:isKnowledgeAbout rdf:resource="http://dbpedia.org/resource/Snowboarding" />
<genoum:hasLevel
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">20</genoum:hasLevel>
</genoum:Knowledge>
<genoum:Knowledge rdf:nodeID="b3">
<genoum:isKnowledgeAbout rdf:resource="http://www.memoria-
mea.ch/memonto/SnowActivity#snowpark_snowboarding"/>
<genoum:hasLevel
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">20</genoum:hasLevel>
</genoum:Knowledge>
</rdf:RDF>
  
```

Figure 15 : Fichier RDF pour un utilisateur

Dans ce fichier, nous pouvons voir les informations suivantes sur l'utilisateur Philippe. Sexe masculin, son prénom est Philippe, son nom de famille est Bonvin, son adresse email est bonvphil@students.hevs.ch, il habite Sierre, il fait du snowboard et plus précisément, il préfère le snowpark comme discipline. D'ailleurs, il aimerait être notifié par mail lorsque d'autres personnes pratiqueront cette discipline. Son niveau de snowboard et de snowpark se situe à 20, ce qui est un niveau excellent (même si ce n'est pas vraiment le cas, c'est pour l'exemple).

Station de ski

Les informations pour les stations de ski sont stockées dans un fichier RDF. Je vais en présenter les informations :

SkiResort	: l'URI de la station
name	: le nom de la station de ski

locatedIn	: l'URI du lieu GeoNames
hasFacility	: indique des URI de services offerts par la station. Ceux-ci proviennent de dbpedia.org. Ex: http://dbpedia.org/resource/Terrain_park
offersActivity	: énumère les URI d'activités, provenant de MemOnto, possibles dans la station. Ex: http://www.memoria-mea.ch/memonto/SnowActivity#snowpark_snowboarding .
hasSnowParkSize	: indique l'URI de la taille du snowpark. Ex: http://www.memoria-mea.ch/memonto.owl#BigSize
hasSkiTracksCount	: le nombre d'installations (téléskis, télésièges,...)
WebSiteURL	: le site internet officiel ex: www.mycma.ch
hasWebCamImageURL	: le lien vers l'image dynamique de la Webcam. Ex: http://webcams.bemore.ch/MyCMA/Images/cam2_fixe.jpg

Exemple d'une partie du fichier OWL pour la station Crans-Montana :

```

<memo:SkiResort rdf:about="#crans_montana">
<foaf:name>Crans-Montana Ski Resort</foaf:name>
<memo:locatedIn rdf:resource="http://sws.geonames.org/6559506/" />
<memo:hasFacility rdf:resource="http://dbpedia.org/resource/Terrain_park" />
<memo:offersActivity rdf:resource="http://www.memoria-mea.ch/memonto/SnowActivity#snowpark_snowboarding"/>
<memo:offersActivity rdf:resource="http://www.memoria-mea.ch/memonto/SnowActivity#freeride_skiing"/>
<memo:hasSnowParkSize rdf:resource="http://www.memoria-mea.ch/memonto.owl#BigSize" />
<memo:hasSkiTracksCount
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">28</memo:hasSkiTracksCount>
<memo:hasWebSiteURL
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">www.mycma.ch</memo:hasWebSiteURL>
<memo:hasWebCamImageURL
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.skiplan.com/webcam/montana/cam2_fixe.jpg</memo:hasWebCamImageURL>
</memo:SkiResort>
  
```

Figure 16 : Partie d'un fichier RDF pour une station de ski

Ce fichier nous informe que la station s'appelle Crans-Montana. Elle dispose de 28 remontés mécaniques. Cette station est intéressante pour faire du hors-pistes (freeride). Elle possède un snowpark de grande taille. Son site internet est www.mycma.ch. L'image dynamique de sa webcam se situe sur l'url suivante: http://www.skiplan.com/webcam/montana/cam2_fixe.jpg.

Evaluations

Les évaluations sont stockées dans un fichier RDF. En voilà la structure:

Evaluation rdf:nodeID : c'est l'identifiant de l'évaluation
 maker : l'URI de l'utilisateur faisant l'évaluation
 evaluates : l'URI de la station évaluée
 hasTimeStamp : date de l'évaluation
 hasEvalDetail : contient des évaluations de 0 à 2 des différentes caractéristiques de la station (URI). Il y'a plusieurs sortes de caractéristiques que je vais détailler dans le tableau suivant.

Caractéristiques possibles des évaluations	Correspondance des valeurs de 0 à 2
Snowpark	Niveau : 0-bad, 1-medium, 2-good
Halfpipe	Niveau: 0-bad, 1-medium, 2-good
Météo (Weather)	Niveau: 0-bad, 1-medium, 2-good
Qualité de la neige	0-icy, 1-slushy, 2-hard, 3-powder Traduction dans l'ordre: Glacé, soupe, dure, poudreux

Figure 17 : Tableau des caractéristiques possibles des évaluations

Exemple de l'évaluation de l'utilisateur Bonvin pour la station Crans-Montana :

```

<genoum:Evaluation rdf:nodeID="b0">
<foaf:maker rdf:resource="http://www.memoria-mea.ch/people/philippe" />
<genoum:evaluates rdf:resource="http://www.memoria-mea.ch/memonto/SkiResort#crans_montana" />
<genoum:hasTimeStamp
rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-11-
26T00:00:00</genoum:hasTimeStamp>
  
```

```

<genoum:hasEvalDetail>
<genoum:EvaluationDetail rdf:nodeID="b1">
<genoum:evaluates rdf:resource="http://www.memoria-
mea.ch/memonto/SnowActivity#snowpark_snowboarding" />
<genoum:hasLevel
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2</genoum:hasLevel>
<rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">good</rdfs:comment>
</genoum:EvaluationDetail>
</genoum:hasEvalDetail>
<genoum:hasEvalDetail>
<genoum:EvaluationDetail rdf:nodeID="b2">
<genoum:evaluates rdf:resource="http://www.memoria-mea.ch/memonto.owl#Weather" />
<genoum:hasLevel
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2</genoum:hasLevel>
<rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">good</rdfs:comment>
</genoum:EvaluationDetail>
</genoum:hasEvalDetail>
<genoum:hasEvalDetail>
<genoum:EvaluationDetail rdf:nodeID="b3">
<genoum:evaluates rdf:resource="http://www.memoria-
mea.ch/memonto.owl#SnowQuality"/>
<genoum:hasLevel
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">3</genoum:hasLevel>
<rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">powder</rdfs:comment>
</genoum:EvaluationDetail>
</genoum:hasEvalDetail>
</genoum:Evaluation>
  
```

Figure 18 : Partie d'un fichier RDF pour une évaluation

Cette évaluation pour le 26 novembre 2008 faite par l'utilisateur Philippe concerne Crans-Montana. L'évaluation pour le snowpark est bonne (good). La météo de cette journée était bonne (good). L'état de la neige était poudreux.

Météo

Les données météo sont reprises du site www.meteosuisse.ch grâce à un outil évoqué dans un chapitre précédent (SemWeC). Ces informations sont:

LocationWeather rdf:nodeID	: identifiant de cette prévision
isAboutLocation	: URI de la station de ski
hasTimeStamp	: date où la prévision a été effectuée. Cette date est utilisée lors de l'utilisation de SemWec pour mettre à jour les données.
isAboutTimeStamp	: jour concerné par la prévision
hasWeather	: URI de la météo. Ex: #mostlySunnyWeather
hasTempMin	: température minimale en °celcius
hasTempMax	: température maximal en °celcius
hasSunshineMin	: pourcentage minimal d'ensoleillement
hasSunshineMax	: pourcentage maximal d'ensoleillement
hasPrecipitationMin	: quantité de précipitations minimale en mm
hasPrecipitationMax	: quantité de précipitations maximale en mm
hasGutsOfWind	: probabilité en pourcent d'avoir du vent plus fort que 45 km/h

Voilà un exemple d'informations météo pour Crans-Montana pour le premier décembre:

```

<memo:LocationWeather rdf:nodeID="b4">
<memo:isAboutLocation rdf:resource="http://sws.geonames.org/6559506/" />
<memo:hasTimeStamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
> 2008-11-30T12:00:00</memo:hasTimeStamp>
<memo:isAboutTimeStamp rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
>2008-12-01T12:00:00</memo:isAboutTimeStamp>
<memo:hasWeather rdf:resource="#mostlySunnyWeather" />
<memo:hasTempMin
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">5</memo:hasTempMin>
<memo:hasTempMax
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">12</memo:hasTempMax>
<memo:hasSunshineMin
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">70</memo:hasSunshineMin>
  
```

```
<memo:hasSunshineMax
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">100</memo:hasSunshineMax>
<memo:hasPrecipitationMin
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</memo:hasPrecipitationMin>
<memo:hasPrecipitationMax
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</memo:hasPrecipitationMax>
<memo:hasGutsOfWind
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</memo:hasGutsOfWind>
</memo:LocationWeather>
```

Figure 19 : Partie d'un fichier RDF pour une information météo

Cette partie de fichier décrit les prévisions météo de Crans-Montana pour le premier décembre 2008 réalisées le jour d'avant. Le temps sera en grande partie ensoleillé. La température minimale sera 5° Celsius et maximal 12° Celsius. Le pourcentage d'ensoleillement sera au minimum de 70% et au maximum de 100%. La quantité de précipitations minimale sera de 0 mm et maximale 1 mm.

Agenda

Les données de l'agenda sont utiles pour savoir quelles personnes vont réaliser telle activité où et à quelle date. Cette partie va être utilisée pour les notifications. Une entrée d'agenda comporte les renseignements suivants:

event:Event rdf:nodeID : identifiant de cette événement

doneBy : URI de la personne concernée

place : URI de la station de ski

factor : moyens de transport ex: <http://www.memoria-mea.ch/memonto.owl#publicBus>

offersActivity : URI des activités proposées par la station

Time : indique l'heure de départ et de retour de la personne

Exemple d'une entrée d'agenda :

```
<event:Event rdf:nodeID="b0">
<memo:doneBy rdf:resource="http://www.memoria-mea.ch/people/philippe" />
<event:place rdf:resource="http://www.memoria-mea.ch/memonto/SkiResort#crans_montana"/>
<event:factor rdf:resource="http://www.memoria-mea.ch/memonto.owl#publicBus" />
<memo:offersActivity rdf:resource="http://www.memoria-mea.ch/memonto/SnowActivity#snowpark_snowboarding"/>
```

```
<event:time><tl:Interval>  
<tl:beginAtDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-  
12-02T08:00:00</tl:beginAtDateTime>  
<tl:endsAtDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-  
12-02T19:00:00</tl:endsAtDateTime>  
</tl:Interval></event:time>  
</event:Event>
```

Figure 20 : Partie d'un fichier rdf pour une entrée d'agenda

Cette entrée d'agenda nous informe que Philippe se rendra en bus à Crans-Montana le 2 décembre 2008 et que cette station propose un snowpark.

Source des données

Mon travail se situe au niveau de la partie de recherche adaptative sémantique. C'est pourquoi, je ne me suis pas attardé sur la façon de récolter les données. Des écrans de saisie sont inutiles. L'utilisateur n'a ni le temps, ni l'envie de saisir ces informations. D'ailleurs celles-ci existent sûrement déjà quelque part où elles peuvent être déduites par rapport aux comportements et aux habitudes de l'utilisateur. C'est pour cela que j'ai créé manuellement des échantillons pour réaliser mon prototype. Mais l'automatisation de la récolte des données est tout à fait possible. L'outil qui permet d'extraire les informations d'un site web directement sous forme sémantique utilisé notamment pour la météo peut nous servir pour récolter la plupart des données. Pour les évaluations et l'agenda, il est possible de les sortir d'un forum où les gens évaluent des stations de ski et indiquent leurs prochaines excursions. Les stations de ski peuvent être tirées d'un site qui répertorie celles-ci, un portail de station de ski par exemple. Les informations des utilisateurs peuvent être extraites d'un réseau social tel que Facebook. Nous pouvons même imaginer que certains réseaux sociaux et autres sites utiliseront la puissance du web sémantique. Dans ce cas, la reprise des informations est directement possible au bon format.

Déductions

Grâce à toutes ces données plusieurs déductions sont possibles. L'utilisateur peut rechercher les stations les mieux adaptées à ses besoins suivant son intérêt principal: halfpipe, snowpark, hors-pistes ou pistes. Il pourra aussi observer la météo de ces stations ainsi que les choisir suivant les avis les plus récents des utilisateurs concernant l'intérêt désiré. Il pourra aussi prévoir ses prochaines sorties en montagne en se basant sur les tendances des autres utilisateurs. Un système de notifications est aussi possible pour proposer aux utilisateurs des informations sur les nouvelles entrées d'agenda en relation avec leurs intérêts.

4.4 Choix de la technologie et des outils

Ayant la possibilité de choisir la technologie et les outils employés, j'ai consacré un certain temps à faire une sélection en essayant une série de possibilités, pour trouver la mieux adaptée. Je vais dans cette partie les expliquer.

Au début, je pensais soit faire mon application en Java Swing³, soit en C#(C Sharp) avec Microsoft Visual studio. La raison du choix Java Swing est qu'une partie du moteur OntoMea a été créé sous cette forme. L'autre solution en C# vient du fait qu'un client de test lié à OntoMea utilise cette technologie. Par ma préférence au logiciel Open Source, qui est pour moi plus évolutif comparé au logiciel propriétaire (privateur : définition selon Richard Stallman⁴), j'ai décidé d'utiliser le langage Java. J'ai tout d'abord implémenté des appels au web service d'OntoMea et tenté plusieurs web services météo.

Puis j'ai dû penser à l'interface graphique alors j'ai aussi décidé d'utiliser le plugin Visual editor d'Eclipse⁵ qui a été utilisé pour l'interface graphique d'OntoMea. Après m'être aperçu que ce plugin n'était pas très bien. Il n'est plus vraiment compatible avec les dernières versions d'Eclipse et le code qu'il génère est assez difficile à réutiliser. Alors j'ai décidé d'employer le programme NetBeans⁶ dans sa dernière version 6.1, qui dispose d'un éditeur visuel très performant.

Puis, en m'entretenant avec mes responsables, j'ai été guidé vers une solution de type web. En analysant ces possibilités, j'ai commencé par envisager de rester dans l'univers des possibilités de NetBeans et d'utiliser des pages JSF⁷. Puis je suis tombé sur GWT (Google Web Toolkit) qui est un Framework libre utilisé par Google pour ses applications web. Appréciant la puissance et la simplicité des outils Google, je l'ai directement essayé et j'ai opté pour cette technologie. En parallèle, j'en ai discuté avec des gens qui connaissent bien ce domaine mais personne n'avait vraiment testé le Framework GWT. Un collègue qui vient d'ouvrir une entreprise de développement informatique en Australie m'a conseillé d'utiliser Ruby on Rails⁸. Je n'ai pas voulu adopter cette technologie bien qu'elle soit très performante du fait que je n'ai pas connaissance du langage Ruby et que la date de fin du travail approche assez rapidement.

³ Swing est une librairie graphique du langage de programmation Java

⁴ Richard Stallman est le père fondateur des logiciels Open source (logiciels libres)

⁵ Eclipse est un environnement de développement intégré libre qui supporte entre autre le langage Java

⁶ NetBeans est un environnement de développement intégré libre qui supporte entre autre le langage Java

⁷ JSF est un Framework Java pour la création d'applications web.

⁸ Ruby on Rails est un Framework web libre écrit dans le langage Ruby

GWT Google Web Toolkit

Le Framework GWT permet de créer des pages dynamiques en Ajax⁹ sans avoir besoin de vraiment connaître ce langage. Le développement se fait en Java avec des composants semblables à ceux de Swing, puis au moment du déploiement, le code est traduit en application web utilisant les technologies Ajax. Celles-ci permettent d'avoir des pages internet qui ne rechargent que certaines parties de la page. Elles permettent aussi de faire des composants déplaçables via "drag and drop" ou même du multifenêtrage à l'intérieur de la même page web et encore plusieurs autres effets. Certaines personnes disent que "Ajax, ça nettoie les navigateurs", dans le sens de sa puissance. C'est un composant majeur de l'évolution du web (web2.0). En prenant du recul, j'en ris, car en choisissant cette technologie je fais du web3.0 avec du web2.0. Il faut préciser que dans le web 2.0 l'effort se situe au niveau de l'interface et de la provenance des données. Tout le monde peut alors publier des données. Tandis que le web 3.0 est une révolution pas directement visible pour l'utilisateur. C'est une façon différente de traiter les données.

On peut télécharger le Framework GWT à l'adresse suivante: <http://code.google.com/webtoolkit/>. C'est un logiciel gratuit et libre offert par Google. Ce Framework est disponible pour les systèmes d'exploitation: Windows, Mac OS X et Linux. Il est nécessaire préalablement d'avoir une version récente du Java SDK¹⁰ installée. GWT lui même ne nécessite aucune installation, il suffit de décompresser l'archive téléchargée. Ce dossier contient principalement 2 scripts qui peuvent être utiles pour générer des projets vierges avec la bonne structure. GWT propose deux manières pour produire un projet vierge, l'une pour travailler avec l'éditeur de son choix et l'autre avec le programme Eclipse. Les deux manières sont décrites ci-après.

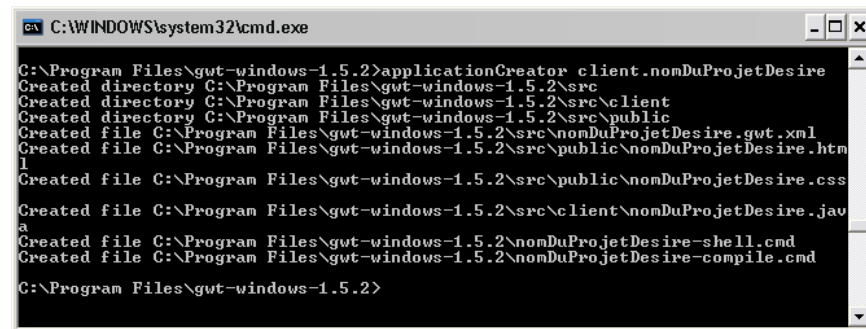
Pour l'éditeur de son choix, il suffit d'exécuter la commande depuis le répertoire décompressé:

```
applicationCreator client.nomDuProjetDesire
```

⁹ Ajax est une solution libre pour le développement d'applications web

¹⁰ Java SDK (software development kit.) est un kit de développement pour le langage Java

Voici le résultat depuis Windows:



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\gwt-windows-1.5.2>applicationCreator client.nomDuProjetDesire
Created directory C:\Program Files\gwt-windows-1.5.2\src
Created directory C:\Program Files\gwt-windows-1.5.2\src\client
Created directory C:\Program Files\gwt-windows-1.5.2\src\public
Created file C:\Program Files\gwt-windows-1.5.2\src\nomDuProjetDesire.gwt.xml
Created file C:\Program Files\gwt-windows-1.5.2\src\public\nomDuProjetDesire.html
Created file C:\Program Files\gwt-windows-1.5.2\src\public\nomDuProjetDesire.css
Created file C:\Program Files\gwt-windows-1.5.2\src\client\nomDuProjetDesire.java
Created file C:\Program Files\gwt-windows-1.5.2\nomDuProjetDesire-shell.cmd
Created file C:\Program Files\gwt-windows-1.5.2\nomDuProjetDesire-compile.cmd
C:\Program Files\gwt-windows-1.5.2>
```

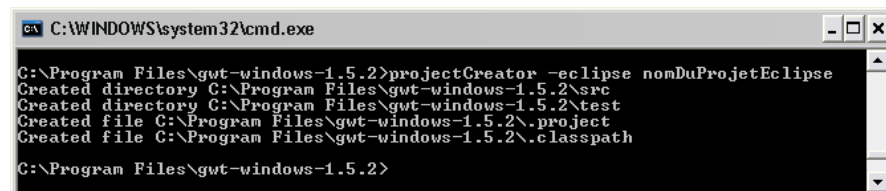
Figure 21 : Commande pour créer un Projet GWT

L'exécution de cette commande crée un projet vide avec lequel il est possible de démarrer. Il suffit juste maintenant de modifier à notre guise le fichier nomDuProjetDesire.java avec l'éditeur de son choix. Puis pour tester l'application, on peut utiliser le script "nomDuProjetDesire-shell.cmd". Ceci va ouvrir un outil qui sera décrit plus loin.

Pour créer un nouveau projet et l'utiliser directement avec Eclipse, on exécute deux commandes depuis le répertoire décompressé. La première est pour créer le projet Eclipse:

```
projectCreator -eclipse nomDuProjetEclipse
```

En voici le résultat:



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\gwt-windows-1.5.2>projectCreator -eclipse nomDuProjetEclipse
Created directory C:\Program Files\gwt-windows-1.5.2\src
Created directory C:\Program Files\gwt-windows-1.5.2\test
Created file C:\Program Files\gwt-windows-1.5.2\project
Created file C:\Program Files\gwt-windows-1.5.2\classpath
C:\Program Files\gwt-windows-1.5.2>
```

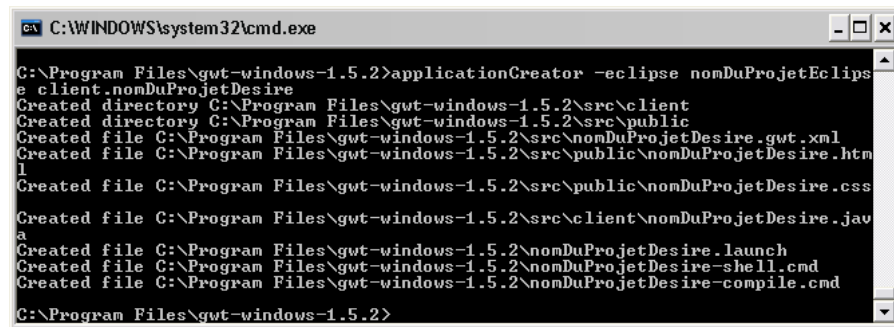
Figure 22 : Commande pour créer un Projet Eclipse pour GWT

L'exécution de cette commande crée les répertoires et les fichiers utiles pour un projet Eclipse.

Puis la deuxième commande sert à concevoir un projet GWT dans ce projet Eclipse:

```
applicationCreator -eclipse nomDuProjetEclipse client.nomDuProjetDesire
```

Voici le résultat:



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\gwt-windows-1.5.2>applicationCreator -eclipse nomDuProjetEclipse client.nomDuProjetDesire
Created directory C:\Program Files\gwt-windows-1.5.2\src\client
Created directory C:\Program Files\gwt-windows-1.5.2\src\public
Created file C:\Program Files\gwt-windows-1.5.2\src\nomDuProjetDesire.gwt.xml
Created file C:\Program Files\gwt-windows-1.5.2\src\public\nomDuProjetDesire.html
Created file C:\Program Files\gwt-windows-1.5.2\src\public\nomDuProjetDesire.css
Created file C:\Program Files\gwt-windows-1.5.2\src\client\nomDuProjetDesire.java
Created file C:\Program Files\gwt-windows-1.5.2\nomDuProjetDesire.launch
Created file C:\Program Files\gwt-windows-1.5.2\nomDuProjetDesire-shell.cmd
Created file C:\Program Files\gwt-windows-1.5.2\nomDuProjetDesire-compile.cmd

C:\Program Files\gwt-windows-1.5.2>
```

Figure 23 : Commande pour intégrer un projet Eclipse à un projet GWT

Maintenant le projet est prêt à être repris depuis Eclipse.

Il suffit de faire un clic droit dans le "package Explorer" d'Eclipse et de sélectionner Import...:

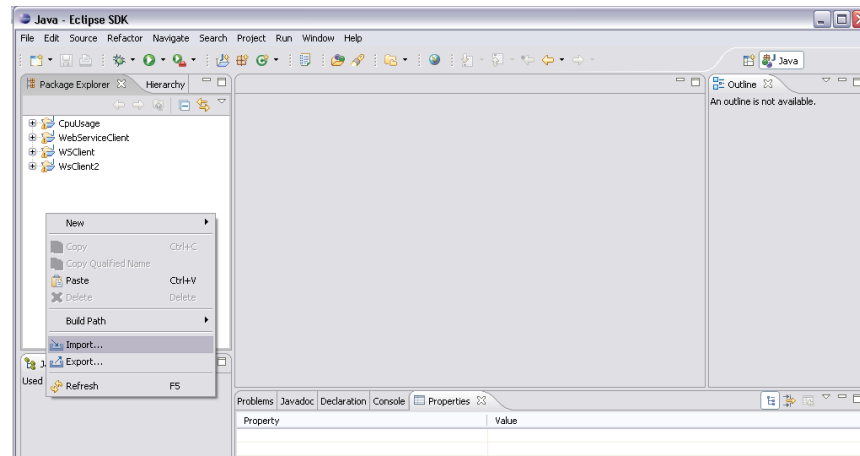


Figure 24 : Importer un projet existant à Eclipse step 1

Puis il faut choisir "Existing Projects into Workspace":

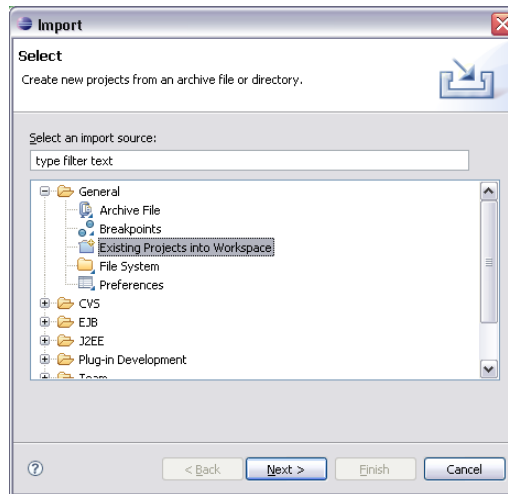


Figure 25 : Importer un projet existant à Eclipse step 2

Ensuite, il suffit de chercher le bon répertoire et nous sommes prêts pour travailler.

Pour tester l'application, il est possible de le faire directement depuis Eclipse comme un projet normal ou avec le script "nomDuProjetDesire-shell.cmd". Les deux manières donnent le même résultat, hormis que depuis Eclipse, il utilise la console de celui-ci alors qu'autrement, il ouvre une console externe.

Voilà ci-dessous l'outil fourni par Google qui permet de tester l'application en cours :

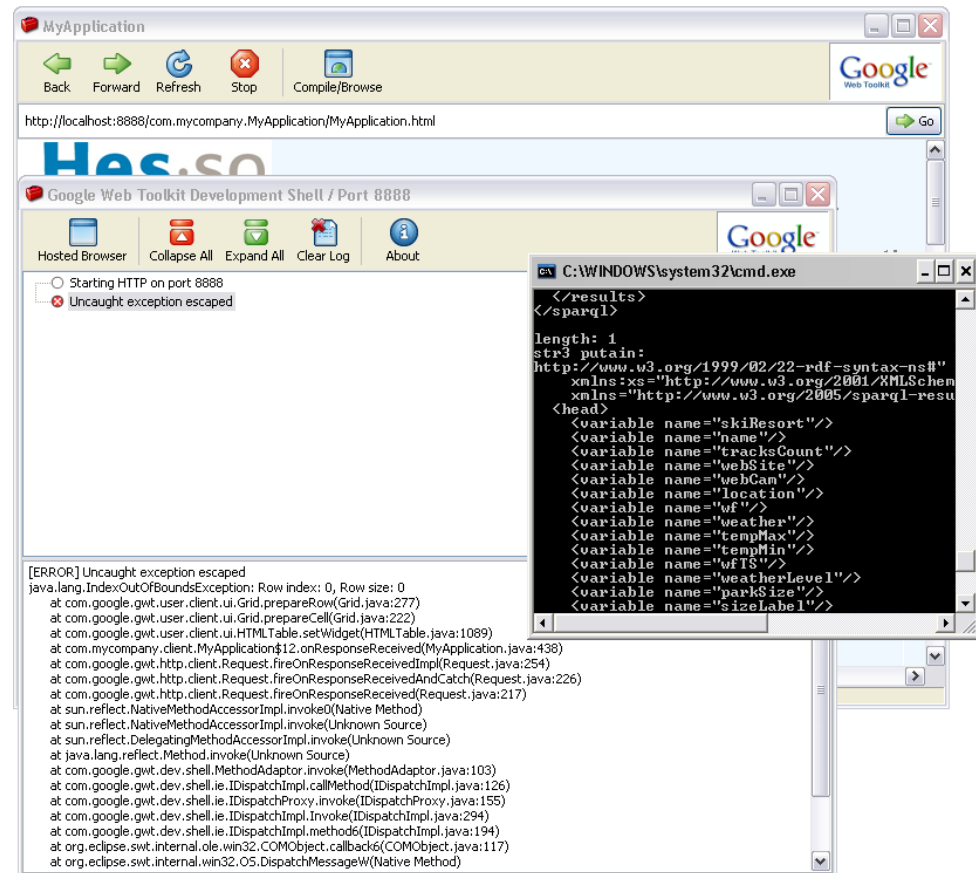


Figure 26 : Outil du Framework GWT

Cet outil est composé de deux ou trois fenêtres selon que la console est intégrée à Eclipse. Il y a une fenêtre qui se nomme "Development Shell". Elle permet de voir les erreurs et leurs détails, d'effacer le journal et d'ouvrir de nouveaux "Hosted Browser". Une autre fenêtre ressemblant à un navigateur se nomme "Hosted Browser". Il est possible d'en ouvrir plusieurs pour des tests. Celle-ci dispose, comme un navigateur, d'un bouton pour revenir à la page précédente et pour passer à la page suivante ainsi qu'un bouton pour rafraîchir l'affichage et un bouton pour stopper le chargement. Un bouton supplémentaire nommé "Compile/Browse" permet de compiler l'application et de l'ouvrir dans votre navigateur préféré. La dernière fenêtre est une fenêtre de commande MS DOS qui affiche les sorties consoles de l'application. Comme je l'ai déjà dit, si l'application est lancée depuis Eclipse, la console de celui-ci sera utilisée.

Cet outil est très pratique, il nous permet des tester le travail sans avoir besoin de le compiler.

Afin de développer une interface graphique avec ce Framework, je me suis intéressé à trouver un éditeur ou un plugin spécial pour GWT afin de voir ses possibilités. J'ai repéré deux plugins pour des environnements de développement intégré. GWT Designer est un plugin pour Eclipse et GWT4NB pour NetBeans. J'ai trouvé GWT designer bien plus complet que l'autre, mais par contre ce plugin est payant. J'ai profité de la version d'essai de 15 jours pour voir la syntaxe des composants afin de gagner du temps pour l'interfaçage. Je vais dans les prochains points détailler ces deux plugins.

GWT designer

GWT designer a été développé par l'entreprise Instantiations. Une version démonstration est téléchargeable depuis leur site: <http://www.instantiations.com/gwt designer/>.

Voilà un aperçu de ce plugin Eclipse:

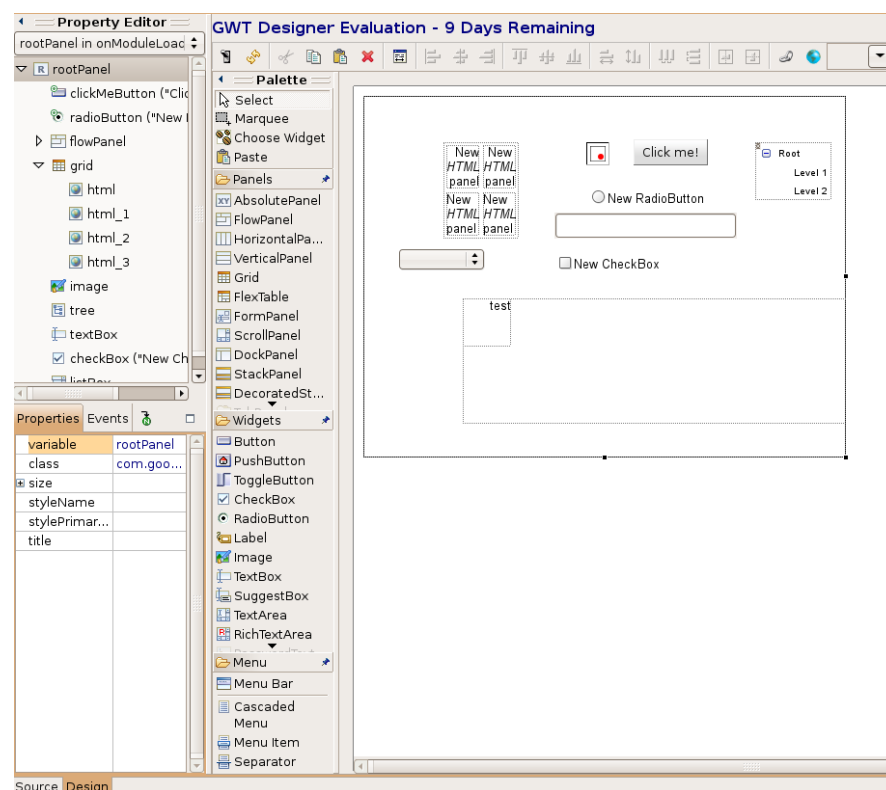


Figure 27 : Interface de GWT Designer

On peut remarquer que la palette d'outils est très complète. L'utilitaire permet de réaliser son application en WYSIWYG (What You See Is What You Get). Cela veut dire pouvoir accomplir une interface visuellement par des "drag and drop" (glisser/déplacer). Cet outil est très bien conçu, il offre la possibilité de déplacer

les objets ainsi que les aligner en toute simplicité. En plus, il permet de créer des projets vierges sans passer par les commandes vues précédemment.

GWT4NB

GWT4NB est un projet libre gratuit. Il est téléchargeable soit depuis NetBeans via le menu Tools puis plugins soit sur le site officiel: <https://gwt4nb.dev.java.net/>.

Voici un aperçu de la fonctionnalité de ce plugin:

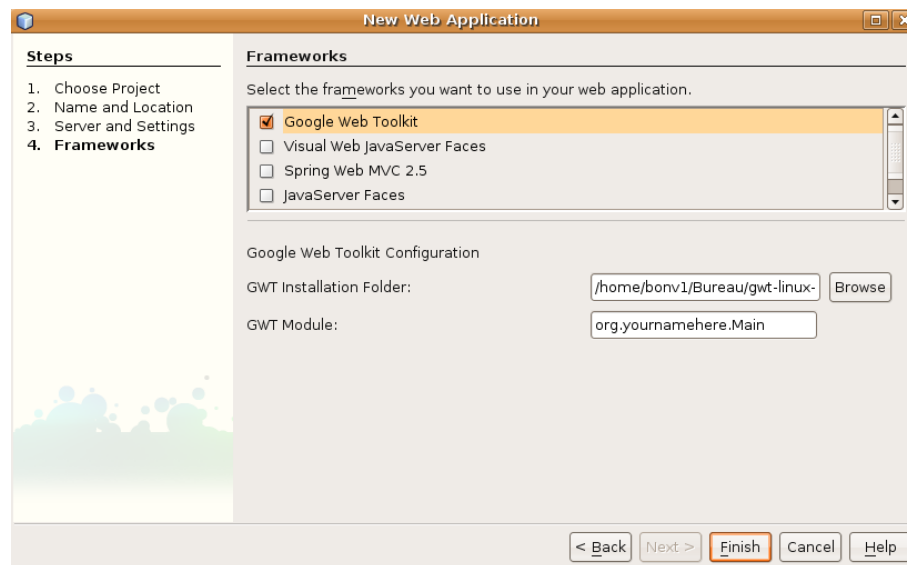


Figure 28 : Possibilité de créer un Projet GWT avec GWT4NB

Ce plugin autorise la création d'un nouveau projet structuré pour le Framework GWT. Il ne dispose pas de possibilité WYSIWYG comme le plugin précédent. Par contre il est gratuit et il permet d'intégrer un projet GWT directement avec le programme NetBeans sans devoir exécuter les commandes vues précédemment.

4.5 Développement et test

Aperçu général

Voici un aperçu de mon application qui détaille les différentes parties:

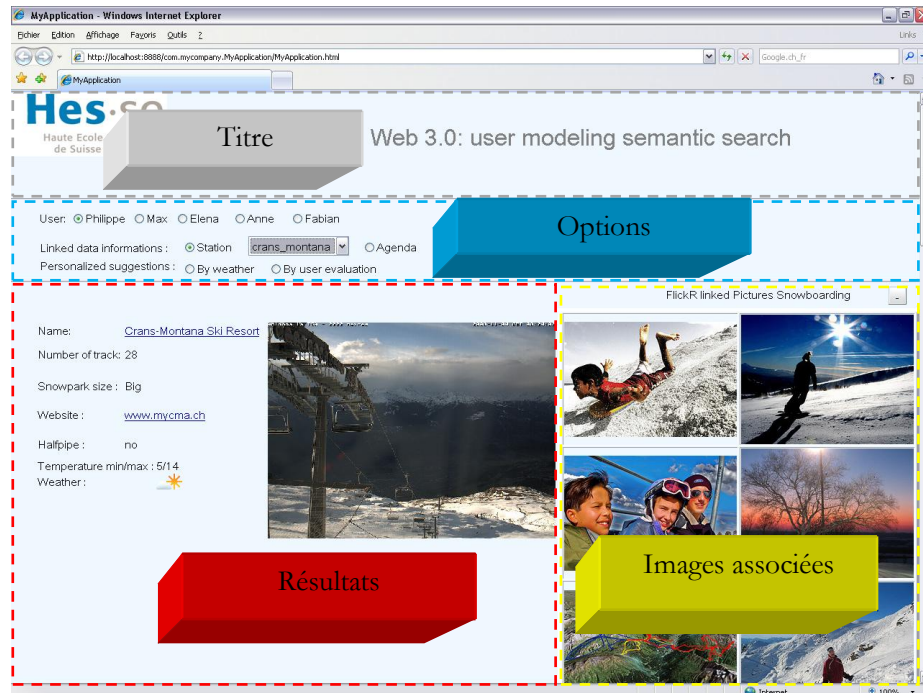


Figure 29 : Découpage des parties de l'application

Cet écran est découpé en quatre parties que je vais détailler.

La partie du haut en gris est la concerne le titre. Elle se compose du logo de l'école ainsi que du titre du rapport. Je ne vais pas m'attarder sur cette partie qui est statique.

La portion qui se situe en dessous en bleu est la partie des options. Elle se compose de 3 lignes. La première permet de changer d'utilisateur. Tout d'abord, j'avais conçu en Java une récupération du login du système Windows qui permettait de connaître l'utilisateur afin que l'application puisse s'adapter à celui-ci. En voici les lignes de code:

```
Properties p = System.getProperties();
String userName = p.getProperty("user.name");
```

Figure 30 : Code Java pour récupérer l'utilisateur Windows connecté

Pour la démonstration, j'ai pensé qu'il fallait une manière rapide pour montrer l'adaptation des résultats par rapport à l'utilisateur connecté. C'est pour cette raison que j'ai conçu 5 boutons radio pour les 5 utilisateurs du scénario. La

deuxième ligne est composée de deux boutons d'option ainsi que d'une liste déroulante. Le bouton d'option "Station" est lié avec la liste déroulante afin de présenter les détails d'une station. Le bouton d'option "Agenda" permet d'afficher bien évidemment l'agenda. La troisième ligne se compose de deux boutons radio permettant des résultats personnalisés pour l'utilisateur en fonction de la météo ou des évaluations des stations.

La partie rouge en dessous des options permet l'affichage des informations sur les stations, la météo et l'agenda.

La partie jaune à droite du résultat précédant permet d'afficher des photos liées à l'intérêt de l'utilisateur.

Les détails de ces fonctions seront expliqués dans les prochains chapitres.

Fonction Station

Le bouton d'option "Station" donne le même résultat pour tous les utilisateurs. Il permet d'afficher les détails de la station sélectionnée dans la liste déroulante. A savoir que celle-ci est alimentée par OntoMea par le biais d'un web service. Dans ce cas le service SPARQL de ce moteur est utilisé et il reçoit la requête suivant:


```
PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
SELECT * WHERE{
    ?skiResort a memo:SkiResort.}
ORDER BY ASC(?skiResort)
```

Figure 31 : Requête SPARQL pour lister toutes les stations de ski

Il est à noter que ces informations sont liées à l'ontologie MemOnto.

Voici un exemple d'affichage pour la station de Zermatt:

User: ☒ Philippe ☐ Max ☐ Elena ☐ Anne ☐ Fabian
 Linked data informations: ☒ Station zermatt ☐ Agenda
 Personalized suggestions: ☐ By weather ☐ By user evaluation

Name: [Zermatt Ski Resort](#)
 Number of track: 34
 Snowpark size: Medium
 Website: <http://www.zermatt.ch/>
 Halfpipe: yes
 Temperature min/max: 1/12
 Weather: 




Figure 32 : Détails pour une station: Zermatt

Sur la partie de résultat nous pouvons voir le nom de la station, le nombre d'installations, la taille du snowpark (s'il y en a un), le site internet, l'image dynamique de la webcam, si la station dispose d'un halfpipe ou non, la température minimale et maximale et une petite image de la météo. Ce résultat contient deux liens qui ouvrent des pages internet externes. Lors d'un clic sur le site web, il va s'ouvrir dans une nouvelle fenêtre. Pour le nom de la station, ce lien nous amène vers la page GeoNames. Ex:

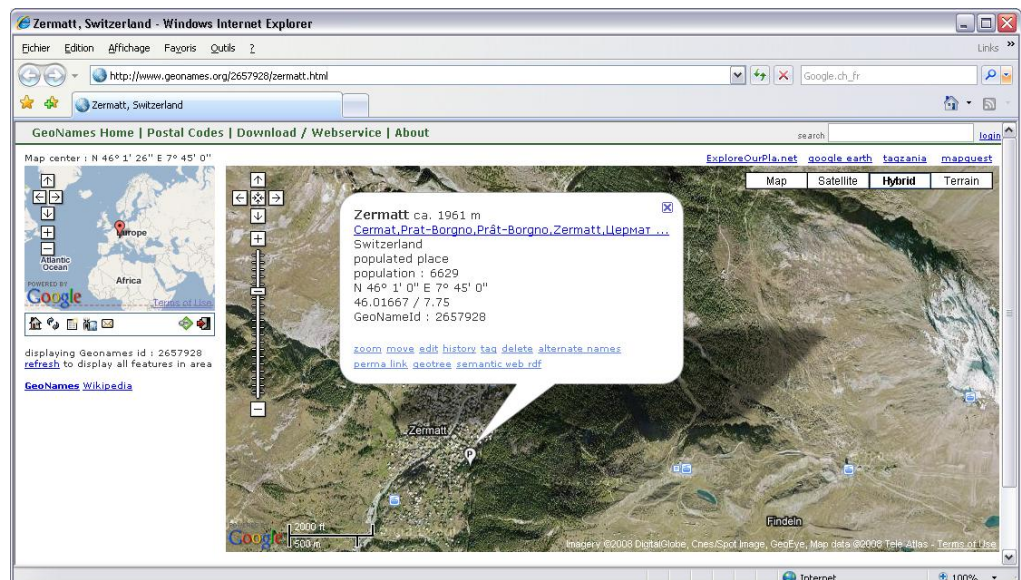


Figure 33 : Exemple de lien GeoNames pour zermat

Toutes ces informations sont issues de plusieurs requêtes SPARQL liant la météo avec la station. La première nous retourne les informations sur la station:

```

PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE { {
  <http://www.memoria-mea.ch/memonto/SkiResort#NomStation> foaf:name ?name;
  memo:hasSnowParkSize ?parkSize;
  memo:locatedIn ?skiResortLoc;
  memo:hasSkiTracksCount ?tracksCount;
  memo:hasWebSiteURL ?webSite;
  memo:hasWebCamImageURL ?webCam.
  ?parkSize rdfs:label ?sizeLabel. }
  UNION {
    <http://www.memoria-mea.ch/memonto/SkiResort#NomStation> memo:hasFacility
    ?facility} }
  
```

Figure 34 : Requête SPARQL pour récupérer les informations d'une station

Remarquons que ces informations sont liées aux ontologies MemOnto, FOAF. MemOnto contient les informations liées aux stations de ski, FOAF permet de retrouver le nom des stations. Le préfixe "rdfs: <http://www.w3.org/2000/01/rdf-schema#>" nous permet d'obtenir des labels à la place d'un URI. Par exemple dans cette requête la valeur de "?parkSize" est <http://www.memoria-mea.ch/memonto.owl#MediumSize>. Grâce à la ligne "?parkSize rdfs:label ?sizeLabel. " nous obtenons le label qui y correspond soit "Medium". Nous pouvons remarquer la partie "UNION" de la fin de la requête. Cette partie nous permet de ne pas avoir un doublon. Si une station dispose de plusieurs "memo:hasFacility" par exemple la station dispose d'un halfpipe et d'un snowpark alors la requête retournera deux fois le même résultat. Une fois le résultat sera mis avec le halfpipe et l'autre fois avec le snowpark comme si c'était deux stations différentes. C'est pour cette raison que j'ai utilisé le "UNION".

La deuxième requête retourne les informations météo pour la journée liées à la station désirée:

```

PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
  
```

```

SELECT * WHERE {
  ?wf a memo:LocationWeather.
  ?wf memo:isAboutLocation skiResortLoc.
  ?wf memo:hasWeather ?weather.
  ?wf memo:hasTimeStamp dateDuJour.
  ?wf memo:hasTempMax ?tempMax.
  ?wf memo:hasTempMin ?tempMin.
  ?wf memo:hasWeather ?weather.
  ?weather genoum:hasLevel ?weatherLevel.}
  
```

Figure 35 : Requête SPARQL qui retourne la météo pour une station

La zone "skiResortLoc" est reprise de la requête précédente et "dateDuJour" correspond bien évidemment à la date du jour.

"?weatherLevel" est utilisé pour afficher l'icone de la météo. Cette partie utilise l'ontologie GenOUM. Cette valeur varie de 1 à 8 et correspond aux petites vignettes suivantes:









Valeur	vignette
1	
2	
3	
4	
5	
6	
7	
8	

Figure 36 : Correspondance des valeurs pour la météo

Fonction Agenda

Le bouton d'option "Agenda" donne le même résultat pour tous les utilisateurs. Il permet d'afficher les futures excursions de ski ou de snowboard des autres utilisateurs.

Voici le résultat :

User: ☒ Philippe ☐ Max ☐ Elena ☐ Anne ☐ Fabian
 Linked data informations : ☐ Station ☐ crans_montana ☒ Agenda
 Personalized suggestions : ☐ By weather ☐ By user evaluation


Person name: Elena Mugellini

Location: [crans_montana](#)

Activity: Snowboarding, Free ride

Transport : Private car

Date : 2008-12-06

Weather : 

Temperture min/max :1/6


Person name: Max Salamin

Location: [crans_montana](#)

Activity: Snowboarding, Half-pipe

Transport : Public bus

Date : 2008-12-06

Weather : 

Temperture min/max :1/6

Figure 37 : Affichage de l'agenda

Cet écran présente deux détails de dates de l'agenda. Un détail est composé du nom de la personne, de la localisation, de la destination, de la date, du moyen de transport, de la météo et de la température minimale et maximale. A nouveau le lien sur le nom de la station nous ouvre GeoNames.

Ces informations sont issues d'une requête SPARQL:

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX tl: <http://purl.org/NET/c4dm/timeline.owl#>
PREFIX time: <http://www.w3.org/2006/time#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
SELECT * WHERE {
  ?event a event:Event;
  memo:doneBy ?person;
  event:place ?location;

```

```

memo:offersActivity ?activity;
event:time ?time;
event:factor ?transport.
?person foaf:name ?personName.
?activity rdfs:label ?activityLabel.
?time tl:beginAtDateTime ?startTS.
?transport rdfs:label ?transportLabel.
?time tl:endsAtDateTime ?endTS.
?activity memo:doAction ?action.
?action rdfs:label ?actionLabel.
?location memo:locatedIn ?geo.
?wf a memo:LocationWeather.
?wf memo:isAboutLocation ?geo.
?wf memo:hasTempMax ?tempMax.
?wf memo:hasTempMin ?tempMin.
?wf memo:isAboutTimeStamp ?wfTS.
?wf memo:hasWeather ?weather.
?weather genoum:hasLevel ?weatherLevel.
FILTER (?startTS >= "dateDuJour"^^xsd:dateTime)
FILTER (?startTS <= "dateDuJourPlusDeuxSemaines"^^xsd:dateTime)
FILTER(fn:substring(str(?startTS), 1, 10) = fn:substring(str(?wfTS), 1, 10))}
ORDER BY ASC(xsd:dateTime(?startTS))
  
```

Figure 38 : Requête SPARQL pour l'agenda

La zone " **dateDuJour** " correspond bien évidemment à la date du jour et la zone " **dateDuJourPlusDeuxSemaines** " est la date 15 jours après. Ceci est réalisé afin d'avoir un intervalle de date et de ne pas avoir trop de résultats. De plus les personnes prévoient rarement des sorties de ski plus de 2 semaines à l'avance. Cela pourrait par contre tout à fait être paramétrable dans une version future.

Nous pouvons remarquer que ces informations sont liées aux ontologies FOAF, tl, time, event, MemOnto, GenOUM. Au début de la requête, l'ontologie event est utilisée pour sortir les événements afin d'y extraire le lieu, la date et l'URI du moyen de transport. FOAF et MemOnto nous permettent ensuite de trouver les informations sur l'utilisateur. MemOnto permet aussi de lier la météo, la localisation, l'action (ex: ski) et l'activité (ex: halfpipe). Time et tl permettent de gérer l'espace temporel de l'événement. GenOum permet d'avoir la valeur pour la météo.

Le préfixe "xsd: <<http://www.w3.org/2001/XMLSchema#>>" est nécessaire pour la gestion de différents formats. Dans la requête ci-dessus, il nous permet de définir des dates "xsd:dateTime". Le préfixe "fn: <http://www.w3.org/2005/xpath-functions#>" est utile pour réaliser certaines fonctions "xpath". Dans notre cas, c'est pour réaliser un "substring" afin de relier la date de la météo à celle de l'agenda. Les deux premiers filtres de cette requête permettent de vérifier que les dates se trouvent dans les deux semaines à venir. Le troisième filtre fait correspondre les dates de l'agenda avec celles de la météo.

Gestion du retour des résultats SPARQL

Il me semble maintenant judicieux de parler de la gestion du retour XML du moteur OntoMea. Lorsque notre application envoie une requête SPARQL au moteur, celui-ci répond dans un format XML.

Afin de montrer un exemple simple, j'ai créé une requête pour lister les stations avec le nombre de remontées mécaniques:

```
PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
SELECT * WHERE {
    ?skiResort a memo:SkiResort;
    memo:hasSkiTracksCount ?numberOfTrack.}
```

Figure 39 : Requête SPARQL qui liste les stations et le nombre d'installations

A l'exécution de celle-ci, OntoMea donne un résultat au format XML. Voilà une partie de cette réponse:

```
...
<result>
  <binding name="skiResort">
    <uri>http://www.memoria-mea.ch/memonto/SkiResort#saas\_fee</uri>
  </binding>
  <binding name="numberOfTrack">
    <literal datatype="http://www.w3.org/2001/XMLSchema#int">22</literal>
  </binding>
</result>
<result>
  <binding name="skiResort">
    <uri>http://www.memoria-mea.ch/memonto/SkiResort#portes\_du\_soleil</uri>
  </binding>
  <binding name="numberOfTrack">
```



```

<literal datatype="http://www.w3.org/2001/XMLSchema#int">194</literal>
</binding>

</result>

...
  
```

Figure 40 : Résultat SPARQL de OntoMea

Nous pouvons remarquer que la réponse pour chaque station se trouve à l'intérieur des balises "<result> </result>". Puis les balises "<binding name=" indique le nom de l'information de la valeur qui suit. Toutes les réponses SPARQL de OntoMea ont cette structure, le seul changement réside dans les informations qu'elles contiennent. J'ai longuement réfléchi à une manière de gérer le stockage de ce retour. Dans un premier temps, j'ai imaginé faire des classes afin de réaliser ceci avec différents objets. Puis, j'ai remarqué qu'il fallait créer une classe distincte pour chaque résultat différent ce qui n'est pas très pratique. A ce moment là, j'ai pensé utiliser l'objet Java "HashMap" qui permet de stocker des clés liées à des valeurs. Cet objet est parfait pour ce genre de situation. J'ai donc employé un tableau de HashMap pour stocker les différentes informations qui sont utilisées lors de l'affichage des composants.

Scénario : personnes et stations de ski

Afin de comprendre les prochaines fonctionnalités qui réagissent de manière différente selon l'utilisateur, il me semble essentiel de présenter les particularités des personnes et des stations.

Il faut savoir que les personnes qui pratiquent intensément le ski ou le snowboard cherchent à s'améliorer dans une des disciplines spécifiques: hors-piste (freeride), snowpark, halfpipe ou pistes.

Voici un tableau qui résume les activités, discipline et l'intérêt de photos (Utilisé pour les images Flickr cf. chapitre suivant) :

Personne	activité	discipline	Intérêt photo
Philippe	Snowboard	Snowpark	Snowboarding
Max	Ski	Halfpipe	Ski
Elena	Ski	Freeride	Nature
Anne			Tango
Fabian	Ski	Piste	Taiji

Figure 41 : Tableau des caractéristiques des personnes

Par ailleurs, les stations de ski proposent ou non ces activités. Pour les activités Freeride et piste, il est certain que dans chaque station il est possible d'en faire mais certaines sont bien mieux équipées. Par exemple les Portes du Soleil comporte 194 remontées mécaniques ce qui justifie l'intérêt des pistes. Certains

autres choix ne sont pas vraiment réels mais il fallait bien faire une différence pour avoir des résultats diversifiés.

Voici un tableau qui récapitule les caractéristiques des stations:

Station	Snowpark	Halfpipe	Freeride	Piste
Crans-Montana	x		x	
Portes du Soleil	x	x	x	x
Saas-Fee	x	x	x	
Vercorin	x			
Zermatt	x	x		x

Figure 42 : Tableau des caractéristiques offertes par les stations

Images associées Flickr

Le panneau de droite présente des photos provenant d'un web service de Flickr. Flickr propose des prestations dans le domaine des photos, il propose d'ailleurs un service de photos sous forme sémantique. Je l'utilise afin de proposer des photos en relation avec les usagers. Ces photos changent en fonction de l'utilisateur suivant son intérêt de photos.

Voici quelques exemples.

Pour Elena:

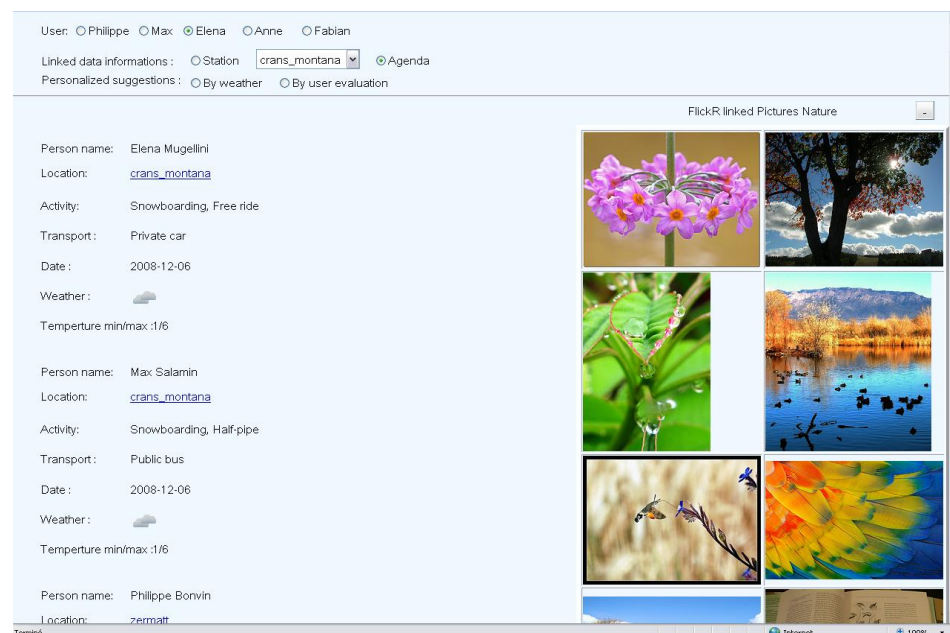


Figure 43 : Photos en relation avec l'intérêt d'Elena provenant de Flickr

Pour Anne:

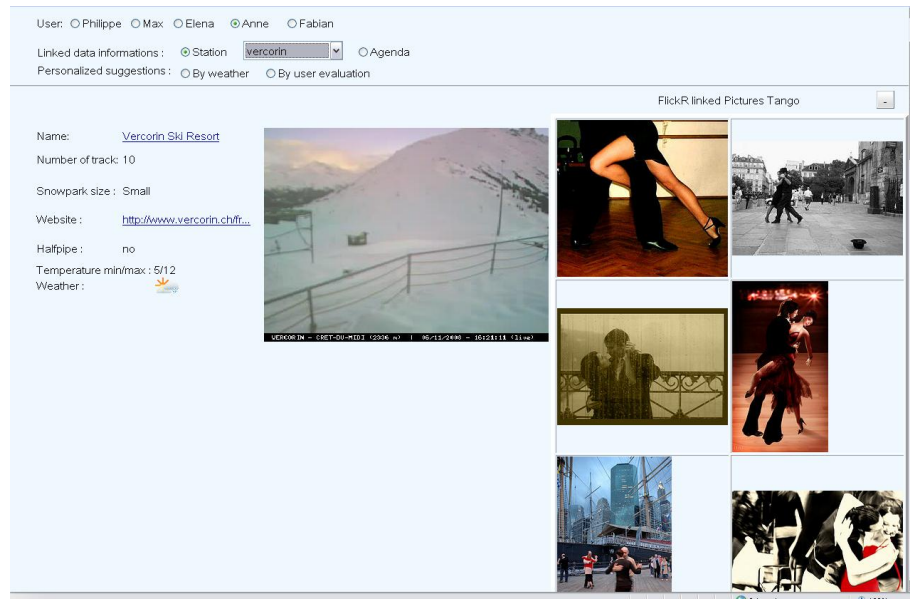


Figure 44 : Photos en relation avec l'intérêt d'Anne provenant de Flickr

Pour Fabian:

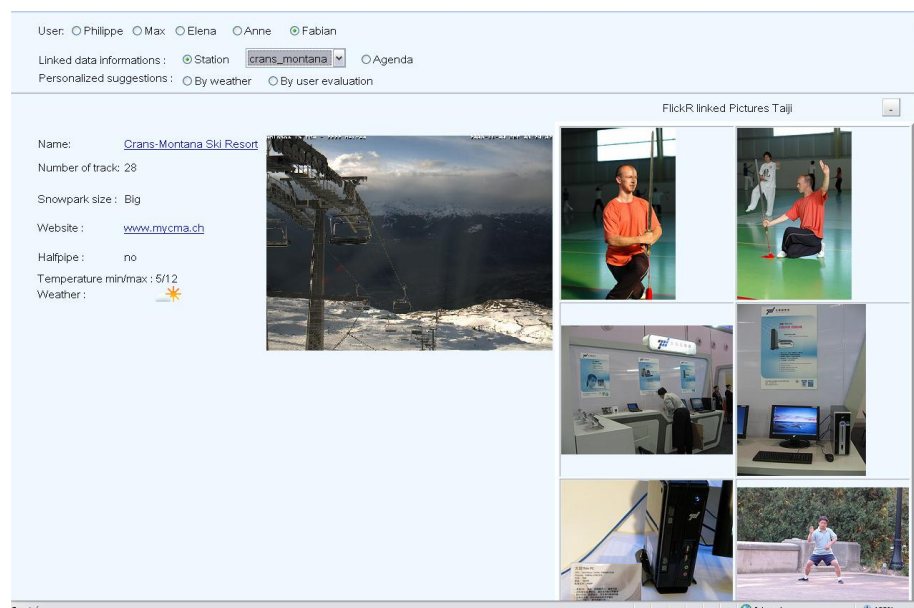


Figure 45 : Photos en relation avec l'intérêt de Fabian provenant de Flickr

On peut remarquer que ce panneau de photos est indépendant du reste des fonctionnalités. Il est possible de consulter l'agenda ou les informations sur les stations de ski en même temps.

Au sommet de ce panneau, il y a un bouton représentant un moins (-) qui permet de masquer ces photos. Lorsque celui-ci est activé, il se transforme alors en plus (+) qui lors d'un clic réaffiche ces photos.



Figure 46 : Bouton pour réafficher le panneau de photos Flickr

Dans un premier temps, j'ai voulu réaliser cette fonctionnalité dans le code Java GWT qui est en fait la partie cliente de l'application. Pour des raisons de sécurité, je n'ai pas pu le réaliser de cette manière. Afin d'y remédier, j'ai rajouté une nouvelle classe Java qui est un agent au niveau du moteur OntoMea. Le client fait un appel à cette nouvelle fonction de web service afin de pouvoir afficher ces photos. Cet agent a été créé principalement pour gérer les deux prochaines fonctionnalités qui concernent les suggestions.

L'application invoque ce service grâce à l'URL suivante:

<http://localhost:2020/agent?cmd=flickr&login=InteretUtilisateur>

Par cette communication l'agent connaît l'intérêt photo de l'utilisateur et il sait qu'il doit renvoyer le résultat de photos flickr.

Suggestions par météo


Cette fonctionnalité permet de lister les stations de ski triées par les conditions météo pour l'intérêt de l'utilisateur. Le résultat sera différent suivant l'utilisateur. Si celui-ci aime le halfpipe, il n'aura dans son résultat que des stations disposant d'un halfpipe et vice versa pour les autres intérêts.

Voici un exemple pour Fabian qui aime les pistes:

User: ☐ Philippe ☐ Max ☐ Elena ☐ Anne ☒ Fabian

Linked data informations: ☐ Station ☐ Agenda

Personalized suggestions: ☒ By weather ☐ By user evaluation

FlickrR linked Pictures Taiji 

Name: [Zermatt Ski Resort](#)


Number of track: 34


Snowpark size: Medium

Website: <http://www.zermatt.ch/>

Halfpipe: yes

Temperature min/max: 1/12

Weather: 



Name: [Les Portes Du Soleil](#)


Number of track: 194

Snowpark size: Big

Website: <http://www.portesdusoleil.ch/>

Halfpipe: yes

Temperature min/max: 6/12

Weather: 




Figure 47 : Résultat de la fonction de suggestion trié par météo pour Fabian

Remarquons que le programme lui propose seulement les deux stations qui coïncident avec son intérêt qui est de faire des pistes. Il est aussi intéressant de voir que ce résultat est classé par la météo. Zermatt apparait en premier du fait qu'il n'y a pas de précipitations au niveau de la vignette "Weather".

Voici un exemple d'une partie du résultat pour Philippe qui aime le snowpark:

Name: [Crans-Montana Ski Resort](#)


Number of track: 28


Snowpark size: Big

Website: www.mycma.ch

Halfpipe: no

Temperature min/max: 5/11

Weather: 



Name: [Vercorin Ski Resort](#)


Number of track: 10


Snowpark size: Small

Website: <http://www.vercorin.ch/fr...>

Halfpipe: no

Temperature min/max: 5/11

Weather: 



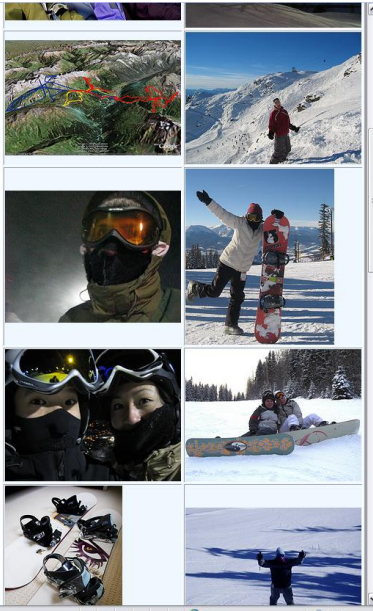


Figure 48 : Résultat de la fonction de suggestion trié par météo pour Philippe

Observons que les résultats sont classés par météo et que les stations présentées disposent toutes d'un snowpark puisque c'est l'intérêt particulier de Philippe.

Afin d'avoir un tel résultat j'ai créé un agent au niveau de OntoMea. Cet agent est situé dans la classe AgentService.java.

L'application invoque ce service grâce à l'url suivant:

<http://localhost:2020/agent?cmd=beststationweather&login=Utilisateur>

Par cette communication l'agent sait de quelle utilisateur il s'agit et qu'il doit lui renvoyer les résultats classés par météo.

Avec ces informations, l'agent va récupérer l'URI de l'intérêt de l'utilisateur avec une requête SPARQL que voici:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
SELECT * WHERE {
  <http://www.memoria-mea.ch/people/philippe>genoum:isInterestedIn ?aKnowledge.
  ?aKnowledge genoum:isKnowledgeAbout ?interests.
}
```

Figure 49 : Requête SPARQL qui retourne l'intérêt de l'utilisateur

Maintenant l'URI de cet intérêt est utilisé pour faire une autre requête qui nous permet d'avoir comme résultats uniquement les stations concernées. En voici le détail:

```
PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT * WHERE {
  ?skiResort a memo:SkiResort;
  memo:offersActivity ?interestURI;
  foaf:name ?name ;
  memo:hasSkiTracksCount ?tracksCount;
  memo:hasWebSiteURL ?webSite;
  memo:hasWebCamImageURL ?webCam;
  memo:locatedIn ?location.
  ?wf a memo:LocationWeather;
  memo:isAboutLocation ?location;
```

```
memo:hasWeather ?weather;  
memo:hasTempMax ?tempMax;  
memo:hasTempMin ?tempMin;  
memo:isAboutTimeStamp ?wfTS.  
?weather genoum:hasLevel ?weatherLevel.  
FILTER (?wfTS >= "dateD"^^xsd:dateTime && ?wfTS <= "dateF"^^xsd:dateTime).  
OPTIONAL {?skiResort memo:hasSnowParkSize ?parkSize.}  
OPTIONAL {?parkSize rdfs:label ?sizeLabel.}  
OPTIONAL {?skiResort memo:hasFacility ?halfPipe.  
FILTER (sameTerm (?halfPipe, <http://dbpedia.org/resource/Half-pipe>))}  
OPTIONAL {?skiResort memo:hasFacility ?snowPark.  
FILTER (sameTerm (?snowPark, <http://dbpedia.org/resource/Terrain_park>))}  
ORDER BY DESC(?weatherLevel)
```

Figure 50 : Requête SPARQL qui retourne les stations pour un intérêt

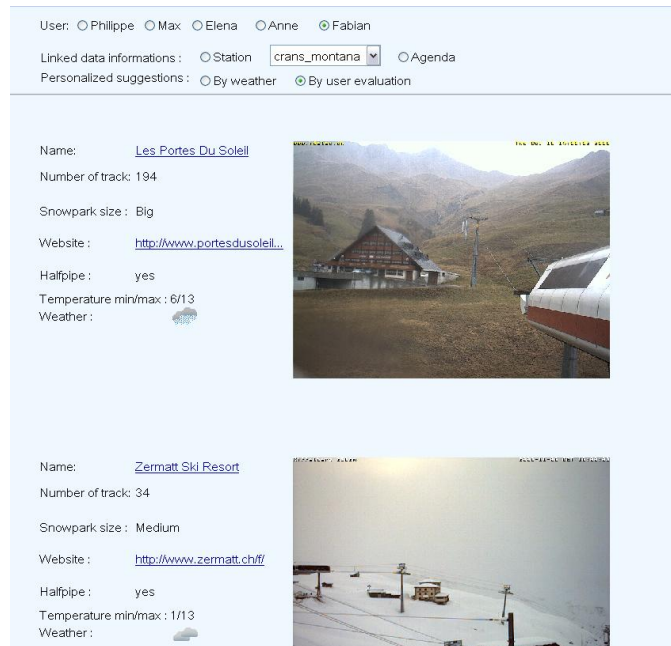
Les zones " dateD " et " dateF " correspondent à la date et l'heure du début de la journée et de fin de journée. Nous pouvons remarquer que la requête trie les résultats sur le champ "?weatherLevel" ce qui nous permet d'avoir la météo sélectionnée de la meilleure à la moins bonne. Cette valeur se situe entre 1 et 8. Le chiffre 1 étant la pire météo et 8 la meilleure. C'est ceci qui nous permet de faire ce choix. Je ne vais pas détailler le reste de la requête qui est similaire aux requêtes présentées dans ce document.

Suggestions par évaluation

Cette fonctionnalité est semblable à la précédente. Par contre son résultat est ordonné suivant la moyenne des 5 dernières évaluations de ces stations. Les valeurs des évaluations comptabilisées sont bien entendu propres à l'intérêt de l'utilisateur.

En reprenant l'exemple pour Fabian qui aime les pistes, on remarque que le classement a changé du fait que les évaluations pour les Portes du Soleil sont mieux cotées.

Voici les résultats:



User: ☐ Philippe ☐ Max ☐ Elena ☐ Anne ☒ Fabian

Linked data informations: ☐ Station: ☐ Agenda

Personalized suggestions: ☐ By weather ☒ By user evaluation

Name: [Les Portes Du Soleil](#)


Number of track: 194


Snowpark size: Big

Website: <http://www.portesdusoleil.ch>

Halfpipe: yes

Temperature min/max: 6/13

Weather: 



Name: [Zermatt Ski Resort](#)


Number of track: 34

Snowpark size: Medium

Website: <http://www.zermatt.ch/fr/>

Halfpipe: yes

Temperature min/max: 1/13

Weather: 




Figure 51 : Résultats de la fonction de suggestion triés par évaluation pour Fabian

Afin d'obtenir ces résultats, j'utilise le même agent que la fonction précédente avec un argument différent.

L'application invoque ce service grâce à l'url suivant:

<http://localhost:2020/agent?cmd=beststationeval&login=Utilisateur>

Par cette communication l'agent sait de quelle **utilisateur** il s'agit et qu'il doit lui renvoyer les résultats **classés par évaluation**.

Il réalise ensuite les mêmes requêtes que la fonction précédente. La différence est qu'à la place de renvoyer le résultat de la deuxième requête, il le stocke dans un objet "HashMap" qui est lui-même placé dans un "ArrayList"¹¹. Ceci me permet de parcourir les stations une à une afin de faire une moyenne des 5 dernières évaluations liées à l'intérêt de la personne.

Voici la requête qui sera effectuée pour chaque station de l'"ArrayList":

```

PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
  
```

¹¹ Un ArrayList est un Objet du langage Java semblable à un tableau ([]) qui permet de stocker un nombre indéterminé d'objets.

```
SELECT * WHERE {
  ?eval a genoum:Evaluation.
  ?eval genoum:evaluates skiResortURI.
  ?evalDet genoum:evaluates interstURI.
  ?eval genoum:hasEvalDetail ?evalDet.
  ?evalDet genoum:hasLevel ?value.}
ORDER BY DESC(xsd:dateTime(?evalDate))
LIMIT 5
```

Figure 52 : Requête SPARQL qui retourne les 5 dernières évaluations pour l'intérêt d'une station

Remarquons la reprise de L'URI de l'intérêt ainsi que celui de la station.

Ensuite la moyenne de ces résultats est ajoutée à l'"ArrayList". Maintenant il faut l'ordonner de manière croissante par rapport à cette moyenne. J'ai dû produire une nouvelle classe qui sur-définit l'objet "Comparator¹²" afin de permettre ce nouveau tri. Enfin, il ne reste plus qu'à envoyer le résultat au client. Pour ce faire, il m'a fallu reconstituer un fichier XML avec les données de cette "ArrayList". Celui-ci est transmis à la partie cliente afin d'afficher les résultats.

Notification par mail

L'entrée de nouvelles dates dans l'agenda fournit une notification par mail. Imaginons que Max rentre une nouvelle date dans l'agenda. Il ira à Crans-Montana le 3 décembre 2008 afin de s'entraîner à l'activité "snowpark". Son moyen de transport sera le bus. Lorsque le moteur OntoMea remarque une nouvelle entrée d'agenda, il va analyser l'activité, dans cet exemple, c'est le "snowpark". Il va envoyer un mail personnalisé à tous les utilisateurs concernés par cet intérêt.

Voici un exemple de mail que l'utilisateur Philippe a reçu:

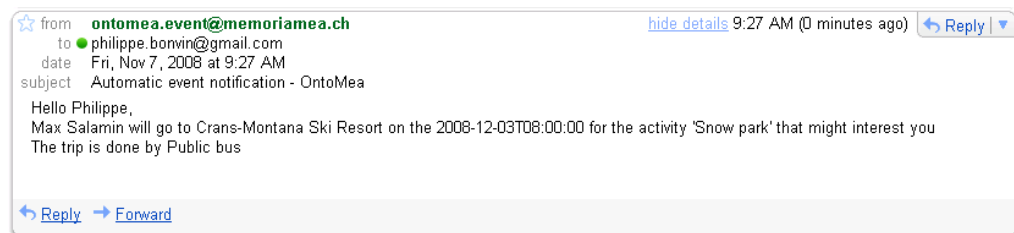


Figure 53 : Exemple de mail de notification

Pour réaliser ceci, j'ai dû créer un nouvel agent au niveau du moteur OntoMea. Il se nomme SnowAgent.java. Cette partie de code est exécutée lors du

¹² Comparator est un objet Java qui permet d'ordonner d'autres objets

chargement de données. Elle utilise plusieurs requêtes SPARQL afin d'obtenir ce résultat. La première permet de sortir les nouveaux événements afin d'en extraire les intérêts. En voici le détail:

```

PREFIX tl: <http://purl.org/NET/c4dm/timeline.owl#>
PREFIX event: <http://purl.org/NET/c4dm/event.owl#>
PREFIX memo: <http://www.memoria-mea.ch/memonto.owl#>
SELECT * WHERE {
  ?event a event:Event;
  memo:offersActivity ?notifActivity;
  memo:doneBy ?eventDoneBy;
  event:place ?eventPlace;
  event:time ?eventInterval;
  event:factor ?meansOfTransport.
  ?eventInterval tl:beginsAtDateTime ?eventDateTime.}
  
```

Figure 54 : Requête SPARQL qui retourne les événements pour la notification

Cette requête est semblable à celle de l'affichage. Elle sera exécutée pour toutes les nouvelles entrées d'agenda. Elle permet surtout de récupérer **l'intérêt pour la notification**. Celui-ci va être utilisé dans la requête suivante afin de trouver toutes les personnes intéressées:

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX genoum: <http://www.websemantique.ch/onto/genoum/>
SELECT * WHERE {
  ?persToNotify a foaf:Person;
  foaf:givenname ?givenName;
  foaf:mbox ?mbox;
  genoum:toNotifyAbout ?notif.
  ?notif genoum:isAboutKnowledge ?notifKnowledge.
  ?notif genoum:hasNotificationType ?notifType.
  ?notifKnowledge genoum:isKnowledgeAbout ?activityURI
  FILTER(?persToNotify != ?doneByURI)}
  
```

Figure 55 : Requête SPARQL qui retourne les personnes intéressées pour la notification

Cette requête permet de récupérer l'adresse e-mail et les informations utiles à la rédaction du mail de notification. La zone "**doneByURI**" correspond à l'URI de la personne qui a créé l'évènement. Ceci est utilisé dans un filtre pour que l'auteur de l'évènement ne reçoive pas ce mail.

Test

Dans un premier temps, j'avais prévu d'utiliser la bibliothèque de tests unitaires JUnit, afin de pouvoir vérifier chaque nouvelle fonctionnalité et modification. La raison pour laquelle je ne l'ai pas utilisée est que pour réaliser des tests unitaires il faut, avant de commencer le développement, connaître le résultat final. Mon application étant un "proof of concept", je ne connaissais pas exactement son aboutissement.

Afin de m'assurer du bon fonctionnement de l'application, j'ai testé toutes les requêtes SPARQL avec le client de test C# et j'ai utilisé l'affichage console pour m'assurer des retours des web services.

Voici une capture d'écran d'un test de requête grâce au client de test:

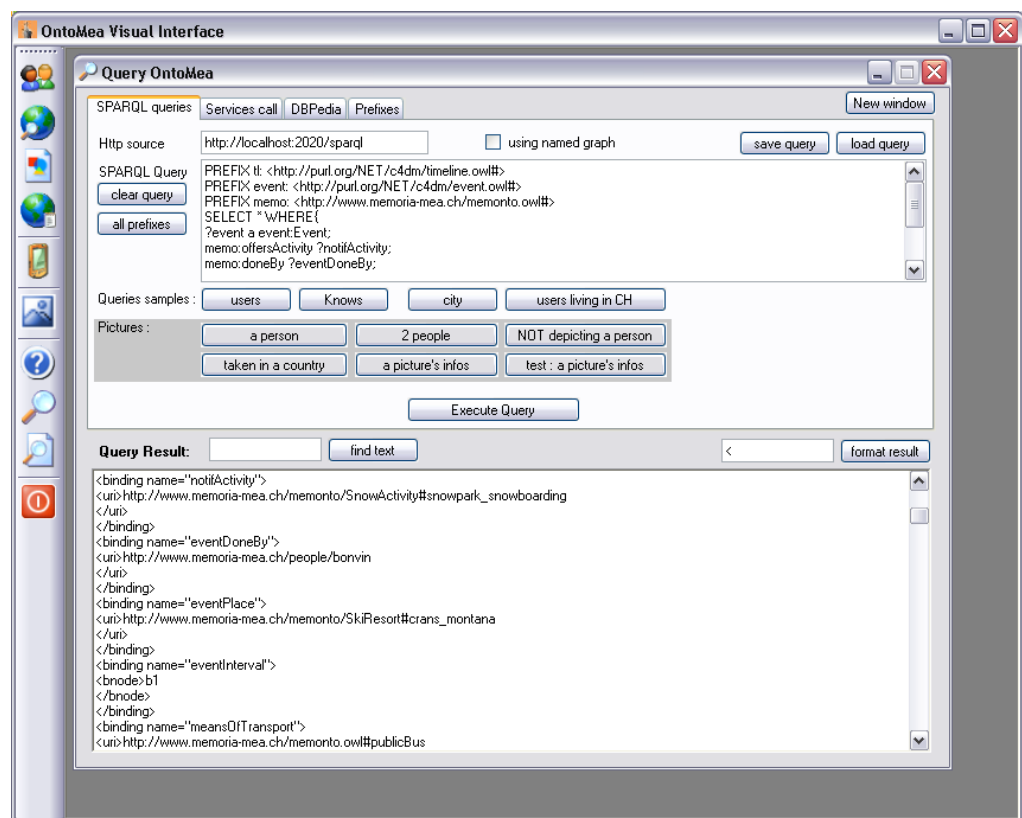
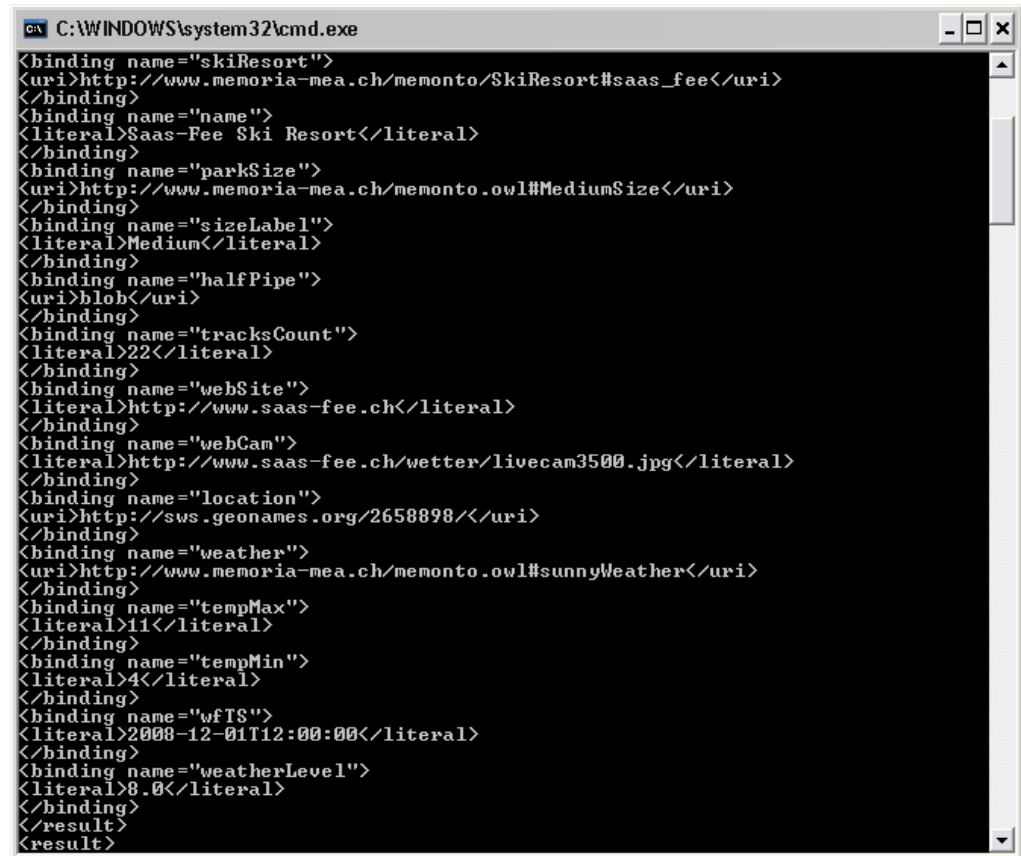


Figure 56 : Client c# pour tester les requêtes SPARQL

Voici une partie d'affichage d'un retour de web service dans une console:



```

C:\WINDOWS\system32\cmd.exe
<binding name="skiResort">
<uri>http://www.memoria-mea.ch/memonto/SkiResort#saas_fee</uri>
</binding>
<binding name="name">
<literal>Saas-Fee Ski Resort</literal>
</binding>
<binding name="parkSize">
<uri>http://www.memoria-mea.ch/memonto.owl#MediumSize</uri>
</binding>
<binding name="sizeLabel">
<literal>Medium</literal>
</binding>
<binding name="halfPipe">
<uri>blob</uri>
</binding>
<binding name="tracksCount">
<literal>22</literal>
</binding>
<binding name="webSite">
<literal>http://www.saas-fee.ch</literal>
</binding>
<binding name="webCam">
<literal>http://www.saas-fee.ch/wetter/livecam3500.jpg</literal>
</binding>
<binding name="location">
<uri>http://sws.geonames.org/2658898</uri>
</binding>
<binding name="weather">
<uri>http://www.memoria-mea.ch/memonto.owl#sunnyWeather</uri>
</binding>
<binding name="tempMax">
<literal>11</literal>
</binding>
<binding name="tempMin">
<literal>4</literal>
</binding>
<binding name="wfTS">
<literal>2008-12-01T12:00:00</literal>
</binding>
<binding name="weatherLevel">
<literal>8.0</literal>
</binding>
</result>
</result>
  
```

Figure 57 : Affichage d'un retour de web service dans une console

Réalisation des fonctionnalités analysées

Ce tableau récapitule les fonctionnalités et leur réalisation:

Fonctionnalités analysées	Implémentation
Comportements différents selon les connaissances de l'utilisateur sur le thème de la recherche	Réalisé et implémenté dans l'application
Requêtes se basant sur les habitudes de l'utilisateur	Partiellement implémenté selon l'idée de la fonction précédente
Zone de texte munie d'une fonctionnalité d'auto complétion désactivable	Implémenté dans l'application puis retiré car elle n'était pas vraiment utile

Zone qui affiche les résultats que l'on peut trier	Réalisé et implémenté
Résultats différents suivant l'heure et la date du système, paramétrable	Réalisé et implémenté
Recherche allégée suivant les ressources mémoire et CPU libres de l'ordinateur, paramétrable	Réalisé mais pas implémenté dans l'application. Car cela n'est pas indispensable à l'application.
Informations différentes selon la météo, paramétrable	Réalisé et implémenté

Figure 58 : Tableau de récapitulation des fonctionnalités et de leur réalisation

5 Problèmes rencontrés

J'ai eu un certain nombre de problèmes qui ont freiné l'avancement de mon travail, je vais en citer quelques uns.

Dans un premier temps, j'ai eu quelques ennuis avec les parties techniques du travail. Cela du fait que pendant les quatre premières semaines de mon travail, la personne qui s'occupait du support technique n'était malheureusement pas présente.

Puis, j'ai eu des difficultés pour invoquer des web services depuis Eclipse sans installer un serveur spécial. D'abord, j'ai essayé d'utiliser plusieurs librairies et projets dont "axis" et "jaxrpc". J'ai rencontré des complications de dépendance parce que je n'utilisais pas de serveur complet comme par exemple "glassFish". J'ai enfin trouvé une solution en utilisant le script "wsimport" qui a su me générer les parties clientes des web services. Pour finir, je n'ai pas utilisé cette façon car les web services météo étaient insuffisants pour mes besoins.

Un autre problème a été le temps d'adaptation au langage SPARQL.

Ayant la liberté de choisir la technologie, j'ai utilisé une grande partie du temps qui m'était imparti à essayer différentes solutions. J'ai donc réalisé les appels à OntoMea par plusieurs techniques avant de découvrir celle qui serait la mieux adaptée.

Le problème qui m'a sûrement fait perdre le plus de temps est le non fonctionnement des web services à cause de la configuration du firewall. La demande d'une configuration identique à l'ordinateur de Fabian ou tout fonctionne parfaitement a été faite à plusieurs reprises au service informatique de l'école qui dispose de la possibilité de le faire. A l'heure actuelle, cela ne fonctionne toujours pas. C'est pour cette raison que je dois désactiver les

services de sécurité de l'ordinateur afin de pouvoir travailler, ce qui est en soit contre le règlement.

Une autre chose qui m'a quelque peu gêné était le bruit dans la salle de travail attribuée. Premièrement, un chantier était situé juste à coté de la salle, ce qui nous a offert un fond musical très riche: bruit de perceuse, de marteau-piqueur ainsi que des vibrations assez importantes. Deuxièmement, cette salle a été attribuée pour des projets de groupe du module 675, ce qui a transformé l'univers de la salle de travail en une salle de conférence se rapprochant d'une cafétéria.

Pour conclure, les anciens diplômants disposaient de 600 heures pour accomplir leur travail. Ce temps est passé actuellement à 360 heures, soit presque la moitié moins et mes responsables semblent attendre de moi que je fournisse un travail équivalent à ce qui se faisait les années précédentes.

6 Améliorations futures

J'ai pensé à plusieurs améliorations futures dont je vais dévoiler une partie.

Il serait possible d'utiliser toutes les informations météo au niveau de l'affichage. Beaucoup de précisions météo sont disponibles au niveau des fichiers RDF, mais seulement une petite partie d'entre elles sont utilisées. L'affichage pourrait préciser le pourcentage d'ensoleillement minimal et maximal, la quantité de précipitation minimale et maximale et des informations sur le vent.

Une possibilité de paramétrer les intervalles pour l'affichage de l'agenda pourrait être très pratique.

Il serait très agréable de mieux tirer partie des possibilités offertes par les technologies Ajax, cela en ajoutant des effets de "drag and drop" et en utilisant un système de multifenêtrage ainsi que d'autres effets proposés par Ajax.

Une centralisation du moteur d'OntoMea sur un serveur distant afin d'étendre son utilisation au reste du monde serait parfaite.

Afin d'économiser des ressources lors des communications avec OntoMea, une utilisation du format de données générique JSON (JavaScript Object Notation) serait mieux adaptée.

Une des meilleures améliorations futures, dont j'ai eu l'idée, mais qui demanderait beaucoup de travail serait de rendre cette application utilisable pour n'importe quelle activité. Nous pouvons imaginer, cette même application pour gérer des lacs ou des plages afin de faire des sports aquatiques tels que la voile ou le kitesurf.

7 Conclusion

Ce sujet de travail de diplôme m'a plu énormément. . Il est vraiment important et intéressant de connaître les technologies du futur. Au début, j'étais un peu sceptique face au web sémantique, maintenant je vois vraiment sa puissance et ce qu'il nous permet de réaliser. Je pense par contre que le web sémantique est encore trop jeune, il va encore considérablement progresser dans le futur. Il est encore difficile de trouver de la documentation sur celui-ci et malheureusement trop peu de monde l'utilise actuellement. Mais je pense, comme disent les chercheurs, que d'ici quelques années, il sera énormément utilisé.

Dans un premier temps je n'étais pas convaincu de l'utilisation de l'application que j'allais faire. Maintenant, je me dis que si cette application disposait d'une météo et de données pertinentes et non de fichier créé par nos soins, je l'utiliserais tous les jours de l'hiver. C'est tellement ennuyeux d'avoir à parcourir plusieurs sites internet afin d'avoir les images des webcams, la météo qui y correspond, un agenda des futures excursions de nos collègues ainsi que des évaluations des différentes activités offertes par les stations. Cette application permet de regrouper toutes ces informations sur la même page, ce qui fait gagner beaucoup de temps.

La découverte de Framework GWT de Google que je ne connaissais pas auparavant m'a enthousiasmé.

Le concept le plus important pour moi, au niveau de ce travail, est de toucher à de nouvelles technologies et non de refaire un travail déjà effectué auparavant. J'en suis d'ailleurs satisfait.

Concrètement le fait de lier l'utilisation des ontologies GenOUM, MemOnto et autres à une application adaptive et personnalisée n'avait, semble-t-il, jamais été réalisé. Ce travail a prouvé que cela était concevable.

8 Bibliographie

Wikipedia *web* [en ligne]

http://en.wikipedia.org/wiki/World_Wide_Web

<http://fr.wikipedia.org/wiki/Oueb>

Wikipedia *web2.0* [en ligne]

http://en.wikipedia.org/wiki/Web_2.0

http://fr.wikipedia.org/wiki/Web_2.0

Web sémantique.ch *site dédié au Web sémantique* [en ligne]

<http://www.websemantique.ch/>

Memoria Mea.ch *Memoria Mea* [en ligne]

<http://www.memoria-mea.ch/>

Wikipedia *web3.0* [en ligne]

http://en.wikipedia.org/wiki/Web_3.0

http://fr.wikipedia.org/wiki/Web_3.0

Wikipedia *RDF* [en ligne]

http://en.wikipedia.org/wiki/Resource_Description_Framework

http://fr.wikipedia.org/wiki/Resource_Description_Framework

W3C *RDF* [en ligne]

<http://www.w3.org/RDF/>

W3C *RDFS* [en ligne]

<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>

Wikipedia *RDFS* [en ligne]

<http://en.wikipedia.org/wiki/RDFS>

http://fr.wikipedia.org/wiki/RDF_Schema

Wikipedia *Ontologie* [en ligne]

[http://fr.wikipedia.org/wiki/Ontologie_\(informatique\)](http://fr.wikipedia.org/wiki/Ontologie_(informatique))

[http://en.wikipedia.org/wiki/Ontology_\(information_science\)](http://en.wikipedia.org/wiki/Ontology_(information_science))

Websemantic.org *Ontologie* [en ligne]

<http://websemantique.org/Ontologie>

W3C *SPARQL* [en ligne]

<http://www.w3.org/TR/rdf-sparql-query/>

Wikipedia *SPARQL* [en ligne]

<http://en.wikipedia.org/wiki/SPARQL>

<http://fr.wikipedia.org/wiki/SPARQL>

Jena sourceforge.net *SPARQL* [en ligne]

<http://jena.sourceforge.net/ARQ/Tutorial/>

Joseki.org *A SPARQL Server for Jena* [en ligne]

<http://joseki.org/>

W3C *OWL* [en ligne]

<http://www.w3.org/TR/owl-guide/>

Wikipedia *OWL* [en ligne]

http://en.wikipedia.org/wiki/Web_Ontology_Language

http://fr.wikipedia.org/wiki/Web_Ontology_Language

Google *GWT* [en ligne]

<http://code.google.com/webtoolkit/>

DBPedia.org *DBPedia* [en ligne]

<http://dbpedia.org/About>

9 Table des illustrations

Figure 3 : Triplet au format RDF/XML	8
Figure 4 : Gestion des classes avec Protégé pour l'ontologie MemOnto.....	9
Figure 5 : Gestion des propriétés avec Protégé pour l'ontologie MemOnto	10
Figure 6 : Gestion du "Domain and range" par Protégé pour la propriété "doAction" de MemOnto	11
Figure 7: Requête SPARQL	11
Figure 8 : Extrait GeoNames pour Sierre	14
Figure 9 : Comparaison entre système classique et système avec modèle utilisateur	14
Figure 10 : Liste des fonctionnalités analysées	17
Figure 11 : Exemple d'un halfpipe (image de http://www.freestyleterrain.com)	20
Figure 12 : Exemple d'un snowpark (image de http://www.skipass.com)	20
Figure 13 : Schéma représentant les données liées de mon application.....	21
Figure 14 : Schéma des Linked Data.....	22
Figure 15 : Fichier RDF pour un utilisateur	24
Figure 16 : Partie d'un fichier RDF pour une station de ski	25
Figure 17 : Tableau des caractéristiques possibles des évaluations	26
Figure 18 : Partie d'un fichier RDF pour une évaluation	27
Figure 19 : Partie d'un fichier RDF pour une information météo	29
Figure 20 : Partie d'un fichier rdf pour une entrée d'agenda.....	30
Figure 21 : Commande pour créer un Projet GWT	33
Figure 22 : Commande pour créer un Projet Eclipse pour GWT.....	33
Figure 23 : Commande pour intégrer un projet Eclipse à un projet GWT	34
Figure 24 : Importer un projet existant à Eclipse step 1.....	34
Figure 25 : Importer un projet existant à Eclipse step 2.....	35
Figure 26 : Outil du Framework GWT.....	36
Figure 27 : Interface de GWT Designer.....	37
Figure 28 : Possibilité de créer un Projet GWT avec GWT4NB	38
Figure 29 : Découpage des parties de l'application.....	39
Figure 30 : Code Java pour récupérer l'utilisateur Windows connecté.....	39

Figure 31 : Requête SPARQL pour lister toutes les stations de ski.....	40
Figure 32 : Détails pour une station: Zermatt	41
Figure 33 : Exemple de lien GeoNames pour zermat.....	41
Figure 34 : Requête SPARQL pour récupérer les informations d'une station....	42
Figure 35 : Requête SPARQL qui retourne la météo pour une station.....	43
Figure 36 : Correspondance des valeurs pour la météo	43
Figure 37 : Affichage de l'agenda.....	44
Figure 38 : Requête SPARQL pour l'agenda	45
Figure 39 : Requête SPARQL qui liste les stations et le nombre d'installations.	46
Figure 40 : Résultat SPARQL de OntoMea.....	47
Figure 41 : Tableau des caractéristiques des personnes	47
Figure 42 : Tableau des caractéristiques offertes par les stations	48
Figure 43 : Photos en relation avec l'intérêt d'Elena provenant de FlickrR.....	48
Figure 44 : Photos en relation avec l'intérêt d'Anne provenant de FlickrR	49
Figure 45 : Photos en relation avec l'intérêt de Fabian provenant de FlickrR.....	49
Figure 46 : Bouton pour réafficher le panneau de photos FlickrR.....	50
Figure 47 : Résultat de la fonction de suggestion trié par météo pour Fabian....	51
Figure 48 : Résultat de la fonction de suggestion trié par météo pour Philippe.	51
Figure 49 : Requête SPARQL qui retourne l'intérêt de l'utilisateur	52
Figure 50 : Requête SPARQL qui retourne les stations pour un intérêt.....	53
Figure 51 : Résultats de la fonction de suggestion triés par évaluation pour Fabian.....	54
Figure 52 : Requête SPARQL qui retourne les 5 dernières évaluations pour l'intérêt d'une station.....	55
Figure 53 : Exemple de mail de notification.....	55
Figure 54 : Requête SPARQL qui retourne les événements pour la notification	56
Figure 55 : Requête SPARQL qui retourne les personnes intéressées pour la notification	56
Figure 56 : Client c# pour tester les requêtes SPARQL.....	57
Figure 57 : Affichage d'un retour de web service dans une console.....	58
Figure 58 : Tableau de récapitulation des fonctionnalités et de leur réalisation..	59

10 Glossaire

API	(Application Programming interface) c'est une interface de programmation applicative.
Axis	Framework écrit en Java dédié au web service.
DBPedia	DBPedia est un projet qui a été réalisé afin d'extraire les informations présentes dans Wikipedia dans un format structuré (RDF).
FOAF	Friend of a Friend projet du Web sémantique dédié aux personnes.
GeoNames	GeoNames est une base de données sémantique dédiée à la géographie.
Inférences	"L'inférence est une opération mentale qui consiste à tirer une conclusion (d'une série de propositions reconnues comme vraies). Ces conclusions sont tirées à partir de règles de base." [Wikipedia]
JENA	JENA est un Framework Java dédié au Web Sémantique.
Ontologie	"Par analogie, le terme est repris en informatique et en science de l'information, où une ontologie est l'ensemble structuré des termes et concepts représentant le sens d'un champ d'informations, que ce soient par les métadonnées d'un espace de noms, ou les éléments d'un domaine de connaissances. L'ontologie constitue en soi un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que les relations entre ces concepts." [Wikipedia]
OWL	"Web Ontology Language — dit OWL — est un dialecte XML basé sur une syntaxe RDF. Il fournit les moyens pour définir des ontologies Web structurées." [Wikipedia]
RDF	"Resource Description Framework (RDF) 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 telles descriptions. Développé par le W3C, RDF est le langage de base du Web sémantique. Une des syntaxes (sérialisation) de ce langage est RDF/XML." [Wikipedia]
RDFS	"RDF Schema ou RDFS est un langage extensible de représentation des connaissances. Il appartient à la famille des langages du Web sémantique publiés par le W3C. RDFS fournit des éléments de base pour la définition d'ontologies ou vocabulaires destinés à structurer des ressources RDF." [Wikipedia]
Web sémantique	"Le Web sémantique désigne un ensemble de technologies visant à rendre le contenu des ressources du World Wide Web accessible et utilisable par les programmes et agents logiciels, grâce à un système de métadonnées formelles, utilisant notamment la famille de langages développés par le W3C." [Wikipedia]

11 Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de diplôme ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de diplôme, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Fabian Cretton, Assistant, Institut Informatique de gestion, HES-SO Valais Wallis
- Anne Le Calvé, Professeur, Institut Informatique de gestion, HES-SO Valais Wallis

Signature :

12 Remerciements

Je remercie mon professeur Anne le Calvé et son assistant Fabian Cretton pour leur disponibilité et tout l'intérêt qu'ils ont mis dans la réussite de ce travail de Bachelor.

Mes remerciements vont aussi à mes parents qui ont eu la gentillesse de relire ce document.

13 Liste des annexes

13.1 Cahier des charges

13.2 Planning



Cahier des charges

Web 3.0 adaptation et personnalisation



Travail de Bachelor
– Sierre, septembre 2008 –
Philippe Bonvin

Professeur responsable : Mme Anne Le Calvé

Table des matières

TABLE DES MATIÈRES	2
1 INTRODUCTION	3
1.1 Description	3
1.2 Web sémantique ou web 3.0	3
1.3 Ontologie.....	3
1.4 Le moteur OntoMea	4
1.5 Le projet Memoria Mea.....	4
1.6 User modeling system	4
1.7 GenOUM	5
2 PROBLÉMATIQUE	5
3 SOLUTION	6
3.1 Travail à réaliser	6
Réalisation.....	6
3.2 Planning envisagé.....	7

1 Introduction

1.1 Description

"Le fléau de mon existence est de faire des choses alors que je sais que l'ordinateur pourraient les faire pour moi." Dan Connolly, The XML Revolution.

Aujourd'hui l'utilisateur est considéré comme le point central des systèmes d'information. De plus en plus de systèmes s'adaptent aux interactions et au profil de l'utilisateur. Leur interface et/ou leur contenu peuvent être personnalisés selon l'utilisateur. Alors que le web 2.0 continue sa mise en place, tous les grands acteurs du web travaillent actuellement sur les nouvelles technologies nécessaires pour permettre aux données de prendre un sens dans le web : le **web 3.0** (web sémantique). La puissance et l'enjeu de ce web sémantique viennent de la mise en relation de la multitude de données existantes, et de leur valorisation par la déduction de nouvelles données et de nouvelles connexions. L'objectif de ce travail de diplôme est de mettre en œuvre un **système adaptatif et personnalisé** (*user modeling system*) utilisant la puissance des techniques du web 3.0. Le scénario utilisé est celui du projet Memoria Mea (www.memoria-mea.ch) de système de gestion d'information personnelle multimédia.

Le travail à réaliser sera le développement d'un prototype d'un **système adaptatif et personnalisé** basé sur le **web sémantique**. Il sera fortement lié à l'**ontologie GenOUM** et au moteur **OntoMea** du Project **Memoria Mea**. Ci-dessous figure une très brève description des notions indispensables à une bonne compréhension de la problématique du travail.

1.2 Web sémantique ou web 3.0

Le web contient une masse énorme d'information qui n'a pas forcément de sens et de liens pour un ordinateur. Le but du web sémantique est de donner un sens à toutes ces données. C'est comme si le web était considéré comme une gigantesque base de données. Pour pouvoir traiter ceci, des mécanismes particuliers sont mis en œuvre telles que l'utilisation d'ontologies décrites dans des langages spécifiques (RDF, OWL).

1.3 Ontologie

Une ontologie permet de décrire un ensemble de concepts d'un domaine dans le but de leur donner un sens et de les relier entre eux. Elle permet aussi de déduire de nouvelles connaissances à partir des informations disponibles, ce qui s'appelle des inférences. Dans le cadre de ce travail, les ontologies seront manipulées grâce à un moteur sémantique nommé OntoMea.

1.4 Le moteur OntoMea

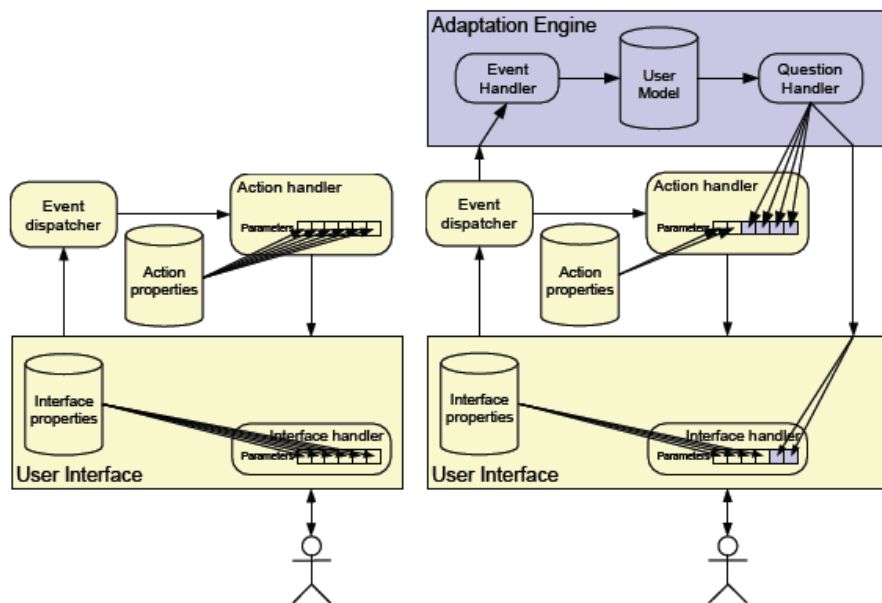
C'est un moteur sémantique de base de connaissances développé par l'HES-SO Valais dans le cadre du projet nommé Memoria-Mea. Il se compose d'un moteur en java et de clients de tests en .NET. Il est capable de traiter des ontologies stockées en local. Le moteur est basé sur la librairie Jena qui permet de traiter les ontologies. Les utilisateurs peuvent se servir de ce moteur pour consulter leurs données via des requêtes SPARQL.

1.5 Le projet Memoria Mea

Memoria-Mea est un projet de gestion d'informations personnelles (PIM : Personal Information Management). Il vise à développer un système PIM pour gérer le contenu multimédia en aidant les utilisateurs dans l'organisation et le recouvrement d'informations à travers la collection entière de documents multimédia qu'ils ont rassemblés pendant des activités de vie quotidienne. Ce projet regroupe les 4 HES-SO suivantes : Valais, Fribourg, Arc, Genève et regroupe une vingtaine de chercheurs sur des technologies diverses. L'un des axes de ce projet est le stockage de données sémantiques et leur exploitation (via le moteur OntoMea).

1.6 User modeling system

C'est le principe d'avoir un système qui s'adapte aux besoins de l'utilisateur. Il utilise pour cela un *modèle utilisateur* (User model).



(A) A model of a normal interactive system

(B) A model of an interactive system with user modeling

Figure 1 : Comparaison entre système classique et système avec modèle utilisateur¹

¹ P.T. de Vrieze, P. van Bommel, Th.P. van der Weide - *A Generic Engine for User Model Based Adaptation* - Proceedings of the User Interfaces for All workshop – 2004.

Le modèle utilisateur comprend des informations sur l'utilisateur d'un système telles que les buts, besoins, préférences ou même intentions de l'utilisateur. Certains modèles utilisateurs plus avancés peuvent contenir des informations liées à l'état psychique, émotionnel, physique, le contexte dans lequel se trouve l'utilisateur etc.

Ces informations peuvent être acquises de plusieurs façons

- explicite ou manuelle : l'utilisateur saisit son profil
- implicite ou automatique : les informations sont déduites de l'interaction de l'utilisateur avec le système.

Nombre de travaux se sont penchés sur la définition du modèle utilisateur et ce, dans de nombreux domaines (e-learning, interfaces adaptatives,...)

Grâce aux travaux de Liana Razmerita², la HES-SO Valais a pu définir son propre modèle utilisateur sémantique nommé GenOUM.

1.7 GenOUM

Generic ontology based User Modeling est une ontologie générique de système adaptatif et personnalisé (user modeling system). GenOUM permet de décrire le profil des utilisateurs, leurs connaissances, intérêts, buts, évaluations, comportement et personnalité.

2 Problématique

Les utilisateurs travaillent tous de manière différente, chaque utilisateur attend un résultat différent d'une requête. L'utilisateur perd beaucoup de temps pour configurer son programme de recherche chaque fois qu'il désire un autre résultat ou qu'il change ses habitudes. L'humain change fréquemment sa manière d'explorer des données. Un autre problème présent est qu'il cherche souvent des informations qui sont liées à d'autres sujets mais l'ordinateur ne le sait pas. Celui-ci contient énormément de données mais il ne sait pas vraiment les interpréter. De plus chaque personne étant différente, un même résultat ne peut être satisfaisant pour deux personnes différentes. Dans le cadre de Memoria-Mea nous souhaiterions pouvoir adapter les résultats de recherche à l'utilisateur.

² Liana Razmerita -*Modèle Utilisateur et Modélisation Utilisateur dans les Systèmes de Gestion des Connaissances: une Approche fondée sur les Ontologies* - Doctorat - Toulouse III, France 2003

3 Solution

Pour résoudre cette problématique une des solutions serait de réaliser un programme de recherche qui s'adapte automatiquement à l'utilisateur.

Cet outil pourrait directement dépendre du web sémantique, afin de pouvoir gérer des relations entre les différentes données et rendre les données compréhensibles par l'ordinateur pour avoir de meilleurs résultats dans les recherches.

3.1 Travail à réaliser

Le projet se divise en trois parties distinctes:

1. Une phase d'analyse et de compréhension afin de bien être imprégné des thèmes web sémantique ainsi que du user modeling system.
2. Une partie d'analyse de ce qui a déjà été fait afin de bien comprendre où va se greffer le prototype. Le travail déjà réalisé est OntoMea et GenOUM.
3. Une phase de développement et de test décrite ci-dessous.

Le prototype sera une extension d'OntoMea permettant la recherche d'informations personnelles sur son propre ordinateur. Ce sera une recherche sémantique se basant sur l'ontologie GenOUM. Les tests seront effectués durant tout le développement afin d'être sûr de sa bonne intégration.

Réalisation

- Scénario de "proof of concept" d'un modèle utilisateur sémantique
- Simulation de données pertinentes pour le scénario
- Interface permettant la recherche et la visualisation des résultats
- Prototype d'un système sémantique de modèle utilisateur dans le cadre de Memoria Mea
- Analyse des idées de fonctionnalités suivantes :
 - Comportements différents selon les connaissances de l'utilisateur sur le thème de la recherche
 - Requête se basant sur les habitudes de l'utilisateur
 - Zone de texte muni d'une fonctionnalité d'auto complétion désactivable
 - Zone qui affiche les résultats que l'on peut trier
 - Résultats différents suivant l'heure du système, paramétrable
 - Recherche allégée suivant les ressources mémoire et CPU libres de l'ordinateur, paramétrable
 - Informations différentes selon la météo, paramétrable
- Développement des fonctionnalités selon leur faisabilité

3.2 Planning envisagé

Le projet se découpe en cinq phases comme suit :

Phase 1 : Analyse de l'existant 15.09.2008 – 21.09.2008

- Web sémantique, ontologies, langages
- Projet Memoria-Mea
- User model et GenOUM

Phase 2 : Immersion 22.09.2008 – 28.09.2008

- Cahier des charges
- Prise en main des outils
- Petits tests

Phase 3 : Immersion 29.09.2008 – 05-10.2008

- -Compréhension d'OntoMea et de GenOUM
- -Analyse des fonctionnalités et de leur faisabilité

Phase 4 : Développement et test 06.10.2008 – 26.10.2008

Phase 5 : Rapport final 27.10.2008 - 10.11.2008

Planning

Date	Etape	Détails
15.09.2008	-Analyse de l'existant	-Attribution du travail -Séance de début -Lecture de doc sur les RDF -Installation du logiciel Protégé
16.09.2008		-Lecture de doc sur OWL -Tutoriel Pizza sous Protégé -Installation du raisonneur Racer
17.09.2008		-Finition du tutoriel Pizza sous Protégé -Lecture d'un partie de la thèse de Razmerita -Exploration du cd distribué -Exploration de memoria-mea.ch et websemantique.ch
18.09.2008		-Séance -Lecture de GenOum.doc -Test de Google desktop -Lecture d'anciens TB
19.09.2008		-Découverte du Framework Jena -Lecture d'anciens TB -Découverte de l'utilisation d'OntoMea -Ouverture du projet OntoMea sur Eclipse et analyse du code source -Ouverture et analyse de Genoum avec Protégé -Test de Protégé 4.x beta
20.09.2008		
21.09.2008		
22.09.2008	-Cahier des charges -prise en main des outils -petits tests	-Création du Planning -Lecture d'anciens TB -Petit brain Storming d'idées pour améliorer une recherche -Recherche de documentation pour l'autocomplétition de TextBox en Java -Début de la rédaction du cahier des charges
23.09.2008		-Rédaction du cahier des charges -Etude du code d'OntoMea -Essai d'ouverture du GUI avec visual editor Eclipse -Lecture de différentes documentations
24.09.2008		-Rédaction du cahier des charges -Séance -Correction du cahier des charges et du planning -Test de l'interface GUI en c# d'OntoMea

25.09.2008		<ul style="list-style-type: none"> -Ouverture du GUI en c# d'OntoMea avec Visual Studio -Analyse de code source -Début d'une fenêtre de recherche -Correction du cahier des charges
26.09.2008		<ul style="list-style-type: none"> -Continuation du petit test d'une fenêtre de recherche -Visite de http://www.joseki.org d'où vient une bonne partie du code source d'OntoMea -Découverte de l'utilisation des Web service D'OntoMea (HTTP (GET and POST) implémentation of the SPARQL Protocol)
27.09.2008		
28.09.2008		
29.09.2008	<ul style="list-style-type: none"> -Analyse de OntoMea et de GenOUM -Analyse de la faisabilité des fonctionnalités 	<ul style="list-style-type: none"> -Tuto et apprentissage SPARQL -Continuation du petit test de recherche -Tentative d'enrichir les fichiers d'utilisateur *.rdf afin de faire de nouvelles requêtes
30.09.2008		<ul style="list-style-type: none"> -Apprentissage SPARQL -Analyse des faisabilités -Essai d'analyse de ressources CPU utilisées en C injecté dans Java par JNI
01.10.2008		<ul style="list-style-type: none"> -Séance -Recherche de widget météo intégrable à Java -Test Java pour récupérer des variables systèmes -Test Java de ressources CPU utilisées -Analyse de LARQ pour faire du Free texte indexing -Analyse des faisabilités
02.10.2008		<ul style="list-style-type: none"> -Début de rédaction de la structure du rapport -Recherche de web services météo -Analyse des faisabilités -Création d'un web service client en Java
03.10.2008		<ul style="list-style-type: none"> -Recherche d'une librairie pour WS client -Analyse des faisabilités -Installation de netBeans 6.1 -Découverte de web Service de localisation et de météo
04.10.2008		
05.10.2008		
06.10.2008	Développement et test	<ul style="list-style-type: none"> -Découverte de la raison du non-fonctionnement des web services (Firewall) -Test de axis 1 et 2 pour la partie cliente des WS (pas convainquant) -Création d'un batch wsimport -Test de plusieurs web services météo

07.10.2008	<ul style="list-style-type: none"> -Rédaction d'une partie du rapport -Customising de recherche -Refactoring de mon début de code
08.10.2008	<ul style="list-style-type: none"> -Séance -Test de JSF avec Netbeans 6.1 -Discussions avec Fabian et démonstration de ce que j'ai déjà fait -Test de Google web toolkit -Découverte de GWT designer
09.10.2008	<ul style="list-style-type: none"> -Découverte du plugin GWT4NB pour faire GWT sous NetBeans -Test de GWT designer qui est mieux que GWT4NB mais payant -Relecture de GenOUM.doc
10.10.2008	<ul style="list-style-type: none"> -Recherche pour intégrer des widget Igoogle avec GWT -Utilisation des web services de OntoMea avec GWT -Recherche des bons imports pour certaines fonctions avec GWT -Lecture de doc: GenoUM, GWT,... -Rédaction d'une partie du rapport
11.10.2008	
12.10.2008	
13.10.2008	<ul style="list-style-type: none"> -Rédaction d'une partie du rapport -Test de Ruby et Ruby on Rails -Web service météo pour GWT -Rédaction d'un petit scénario
14.10.2008	<ul style="list-style-type: none"> -Recherche d'un bon web service météo complet, mail à météo suisse.ch -Discussion du scénario avec Fabian -Génération de données pour le scénario
15.10.2008	<ul style="list-style-type: none"> -Injection d'HTML dans du Java afin d'avoir des infos météo -Séance -Génération de données pour le scénario -Rédaction d'un partie du rapport -Tentative de récupérer le nom de session depuis GWT -Parssage de fichier XML avec GWT
16.10.2008	<ul style="list-style-type: none"> -Parssage de fichier XML avec GWT -Stockage des résultats SPAQL dans un HashMap -Affichage de webcam depuis GWT -Essai d'affichage graphique extensible dans des tableaux
17.10.2008	<ul style="list-style-type: none"> -Parssage de fichier XML avec GWT -Stockage des résultats SPAQL dans un HashMap -Essai d'affichage graphique extensible dans des tableaux -Récupération des données météo
18.10.2008	
19.10.2008	

20.10.2008		<ul style="list-style-type: none"> -Récupération des données sation -Refactoring de code -Affichage d'une station -Possibilité de rechercher plusieurs stations
21.10.2008		<ul style="list-style-type: none"> -Refactoring de code -Séance -Explication du scénario dans le rapport -Ajout de bouton radio sur l'interface
22.10.2008		<ul style="list-style-type: none"> -Ajout et compréhension du début d'un agent -Création d'un agent -Essai de refactoring de code -Affichage de plusieurs sations sous forme de tableau -Génération de données météo
23.10.2008		<ul style="list-style-type: none"> -Utilisation des évaluation des utilisateurs dans mes requêtes -Correction d'une requête SPARQL qui était très lente -Récupération de la date system pour une requête SPARQL -Ajout de l'agenda
24.10.2008		<ul style="list-style-type: none"> -Création d'un agent intelligent qui fait la moyenne des évaluations -Algo de tri d'un arrayList d'HashMap -Refactoring -Génération d'XML envoyé aux client -Débuggage de problèmes de fichiers
25.10.2008		
26.10.2008		
27.10.2008	Rapport final	<ul style="list-style-type: none"> -Changement de sens du tri d'arrayList -Résolution de bugs -Rédaction du rapport final -Travail sur la partie de l'agenda
28.10.2008		<ul style="list-style-type: none"> -Rédaction du rapport final -Travail sur la partie de l'agenda -Ajout de commentaire sur le code -Refactoring
29.10.2008		<ul style="list-style-type: none"> -Rédaction du rapport final -Ajout de label à la place des URI dans les différentes recherches -Ajout d'images pour la météo
30.10.2008		<ul style="list-style-type: none"> -Préparation des notifications par mail pour les intérêts liés aux nouvelles dates de l'agenda -Rédaction du rapport final -Ajout d'un panel avec des image de Flickr liées aux intérets -Débugage -Redisign de l'interface

31.10.2008		-Rédaction du rapport final -Finalisation du prototype -Modification des données pour avoir des résultats distincts -Finition de la partie de notification
01.11.2008		
02.11.2008		
03.11.2008		-Dernières petites retouches du prototype -Rédaction du rapport final -Explication des fichiers de données utilisés par mon prototype
04.11.2008		-Rédaction du rapport final -Petites corrections dans les fichiers de données
05.11.2008		-Rédaction du rapport final -Rédaction du résumé du travail
06.11.2008		-Rédaction du rapport final -Séance
07.11.2008		-Rédaction du rapport final
08.11.2008		
09.11.2008		
10.11.2008		-Rendu