Semantic exploitation of engineering models: application to petroleum reservoir models

Laura Silveira Mastella

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

MINES ParisTech

Ecole doctorale n° 084: Sciences et technologies de l'information et de la communication

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École nationale supérieure des mines de Paris

Spécialité "Informatique Temps Réel, Automatique, Robotique"

présentée et soutenue publiquement par

Laura SILVEIRA MASTELLA

le 02 Mars 2010

Semantic exploitation of engineering models: application to petroleum reservoir models

Directeur de thèse : **Michel PERRIN**
Co-encadrement de la thèse : **Yamine AÏT AMEUR** (LISI / ENSMA)

Jury

M. Djamal BENSLIMANE, Professeur, LIRIS, Université Claude Bernard, Lyon 1	Rapporteur
M. Aris OUKSEL, Professeur, University of Illinois at Chicago	Rapporteur
Mme. Mara ABEL, Professeur, Universidade Federal do Rio Grande do Sul	Examinateur
M. Olivier CORBY, Chargé de Recherche, INRIA Sophia Antipolis	Examinateur
M. Jean-François RAINAUD, Ingénieur de Recherche, Institut Français du Pétrole	Examinateur

MINES ParisTech
Centre de Géosciences

T H E S E

Remerciements

Mes premiers remerciements s'adressent à mon directeur de thèse Michel Perrin. Je le remercie pour nos riches discussions, grâce auxquelles il a pu transmettre sa connaissance en Géologie à une interlocutrice informaticienne. Je le remercie pour ses précieux conseils, aussi bien sur le plan humain que scientifique, pour ses encouragements et son soutien total dans les moments délicats de ma thèse. Je le remercie aussi pour les nombreuses et attentives relectures de chacun des chapitres. Merci enfin d'avoir été toujours présent et disponible malgré sa récente retraite professionnelle.

Je remercie également Jean François Rainaud, mon tuteur industriel à l'IFP, pour m'avoir fait profiter de sa connaissance et de son enthousiasme. Merci pour sa patience lors de nos discussions et pour ses idées apportées en vue d'une application industrielle de mes travaux.

Je remercie aussi Pierre Ferry Forgues et tous les collègues du département informatique et mathématiques de l'IFP pour m'avoir accueillie dans des conditions de travail privilégiées.

J'exprime toute ma reconnaissance au Professeur Yamine Aït-Ameur, qui a accepté d'encadrer ce travail en cour de route, malgré son emploi du temps chargé. Je le remercie pour le regard pragmatique et compréhensif qu'il a porté sur ma thèse, et qui m'a permis de mettre en relation les différents aspects de mon travail.

Merci également à tous les membres du LISI pour m'avoir accueilli et intégrée au sein du groupe. Un grand merci à Stéphane Jean pour m'avoir consacré du temps ainsi que pour ses conseils et suggestions concernant l'implémentation de mon travail.

Je remercie le personnel de l'École des Mines de Paris pour m'avoir assisté dans toutes les tâches administratives du doctorat. Je remercie également la fondation CAPES, institution faisant partie du ministère de l'Éducation du Brésil, qui m'a financée pendant toute la durée de la thèse.

Je remercie les Professeurs Djamal Benslimane et Aris Ouksel pour avoir accepté d'évaluer ce travail doctoral, et pour avoir formulé des remarques très encourageantes sur mon manuscrit.

Je remercie aussi Olivier Corby pour avoir accepté d'être membre du jury en tant qu'examinateur.

Je remercie particulièrement la Professeur Mara Abel, à qui je dois ma formation et grâce à qui ce travail a été initiée. Merci d'être présente avec moi à la fin de cette thèse, et surtout de me faire l'honneur de présider mon jury.

Pour finir, un tendre merci à Erwan, avec qui j'ai mutuellement partagé les joies et les péripéties d'un travail de doctorat. J'ai aussi une pensée à tous les amis qui de près ou de loin ont rendu plus agréable la réalisation de ce travail, et à ma famille, qui attend avec impatience la fin de ma thèse, pour que je puisse enfin retourner auprès d'eux.

To the reader

This thesis is a work in computer science and is naturally addressed to computer scientists. Earth sciences and oil reservoir studies also have a significant place in the following text since this engineering field is the one that was chosen as a use case for illustrating the semantic approach developed in the work. For these reasons, my wish is that the following text be understood both by open minded computer scientists and by open minded geologists.

For being scientific sound, a doctoral work should not be oversimplified. Considering the challenge of making our text understandable to two scientific communities, that have little in common, our choice has been:

- to write the text in an explicit but not simplified way both for the "computer science" and for the "geological" parts,
- to explain all technical words by means of footnotes and of a glossary.

I apologize in advance for the difficulties that the reader will no doubt find when reading the parts of the work that do not refer to his/her field of expertise. I hope however that most of these difficulties will be overcome and that the following text will provide a common field of reflection both to Computer scientists and to Exploration geologists.

Laura Mastella

Abstract

This work intends to propose innovative solutions for the exploitation of heterogeneous models in engineering domains. It pays a special attention to a case study related to one specific engineering domain: petroleum exploration . Experts deal with many petroleum exploration issues by building and exploiting three-dimensional representations of underground (called earth models). These models rest on a large amount of heterogeneous data generated every day by several different exploration activities such as seismic surveys, well drilling, well log interpretation and many others. Considering this, end-users wish to be able to retrieve and re-use at any moment information related to data and interpretations in the various fields of expertise considered along the earth modeling chain.

Integration approaches for engineering domains needs to be dissociated from data sources, formats and software tools that are constantly evolving. Our solution is based on semantic annotation, a current Web Semantic technique for adding knowledge to resources by means of semantic tags. The "semantics" attached by means of some annotation is defined by ontologies, corresponding to "formal specifications of some domain conceptualization". In order to complete engineering model exploitation, it is necessary to provide model integration. Correspondence between models in the ontology level is made possible thanks to semantic annotation. An architecture, which maps concepts from local ontologies to some global ontology, then ensures that users can have an integrated and shared global view of each specific domain involved in the engineering process.

A prototype was implemented considering the seismic interpretation activity, which corresponds to the first step of the earth modeling workflow. The performed experiments show that, thanks to our solution, experts can formulate queries and retrieve relevant answers using their knowledge-level vocabulary.

Keywords : Model integration and interoperability, Ontologies, Ontology-based databases, Meta-modeling, Semantic annotation, Petroleum reservoir modeling.

Table of Contents

Part I Semantic based solutions for management and exploitation of engineering models

Résumé étendu

1 Contexte de développement du travail présenté

1.1 Introduction

Commencée en octobre 2006 et soutenue en mars 2010, la présente thèse de doctorat a été réalisée dans le cadre du Doctorat “Informatique Temps Réel, Automatique et Robotique” de l’École des Mines de Paris. Elle a été dirigée par Michel Perrin, Professeur à l’École des Mines de Paris, et co-dirigée par Yamine Aït-Ameur, Professeur à l’École de Mécanique et d’Aéronautique (ENSMA) à Poitiers). Durant toute la durée de mon travail, j’ai été localisée à l’Institut Français du Pétrole (IFP) à Rueil-Malmaison, ce qui a permis une bonne synergie avec les utilisateurs finaux des outils que j’ai développés.

Cette thèse se propose d’aborder l’*exploitation des modèles d’ingénierie hétérogènes* et traite, en tant qu’étude de cas, une *application au domaine de l’exploration de pétrole*.

1.2 Les modèles d’ingénierie

Un modèle est une représentation abstraite et simplifiée d’un phénomène ou d’une suite d’actions appartenant au monde réel. En ingénierie, la construction et la manipulation de modèles constituent des activités essentielles dans la mesure où l’intérêt principal se porte des cycles de vie: cycle de vie d’un logiciel, cycle de vie d’objets ou de pièces mécaniques (pièces d’avions, de voitures), cycle de vie d’un réservoir pétrolier et autres. Ces cycles sont appréhendés au travers de modèles relatifs à des domaines très variés : génie logiciel, ingénierie des transports aériens ou terrestres, matériaux, génie civil, ingénierie environnementale, ingénierie de pétrole. La plupart de ces modèles s’appuient par ailleurs sur des données nombreuses et hétérogènes.

Dans la plupart des cas, les raisonnements qui ont servi à la construction d’un modèle demeurent implicites. Ces éléments et la sémantique qui les sous-tend restent pour ainsi dire cachés derrière le modèle lui-même et sont au mieux accessibles aux seuls auteurs de la modélisation. Une raison à cela tient au fait que, jusqu’à une période récente, la culture des entreprises privilégiait les résultats plutôt que le capital de connaissances expertes qui les a produits. De ce fait, les actes d’interprétation (autrement dit

les “opinions”) au travers desquels les experts expriment leur savoir au service de la construction d’un modèle n’ont été jusqu’à présent, sauf exceptions, ni répertoriés ni mémorisés. Le résultat de la mise en œuvre d’un procédé d’ingénierie se trouve ainsi être trop souvent un simple document en langage naturel envoyé aux divers membres d’une équipe. Un tel document, par essence ambigu, ne permet généralement pas que les ingénieurs de cultures différentes rassemblés autour d’un projet comprennent la signification des éléments de modèles qu’ils manipulent. Les ingénieurs doivent donc consacrer des temps de travail très importants pour rechercher les informations sur lesquelles se sont basés les projets passés et pour traduire “manuellement” les données correspondantes produites la plupart du temps par des outils de modélisation hétérogènes. Enfin, lorsqu’il s’avère nécessaire d’échanger des données entre ces outils, les modules de traduction sont le plus souvent codés en dur par les fournisseurs de logiciels, au sein de produits logiciels intégrés. La traduction opérée porte alors seulement sur les formats de données, ce qui ne permet pas d’assurer l’interopérabilité et la communication entre modèles.

Actuellement, les ingénieurs sont placés devant le défi de pouvoir accéder à toutes les informations utiles dans leur domaine, afin de prendre des décisions adéquates. Il devient par conséquent essentiel que les experts réunis autour d’un projet, porteurs de connaissances hétérogènes, soient à même de partager les diverses “opinions” qu’ils produisent. Par ailleurs, cette approche d’intégration d’ingénierie doit être dissociée aussi bien des sources et formats de données que des outils, car ceux-ci évoluent constamment. Pour coopérer, les ingénieurs doivent par conséquent s’accorder sur un *vocabulaire commun* de communication et ils doivent être également en mesure de décrire la *signification* des données et des formats de données mis en œuvre dans des modèles. La correspondance entre modèles doit donc être basée sur des *descriptions signifiantes*, qui doivent être isolées de la partie physique des modèles.

1.3 La modélisation géologique pour l’exploration pétrolière

Le domaine qui a été choisi dans ce travail pour illustrer les problématiques qui viennent d’être exposées est celui qui s’intéresse à la *construction de modèles géologiques en vue de l’exploration pétrolière*. Nous allons donc exposer brièvement en quoi il consiste.

Les experts géologues et les ingénieurs en charge de l’exploration pétrolière élaborent leurs stratégies de prospection en se basant sur des représentations tridimensionnelles du sous-sol appelées modèles géologiques. Ces modèles reposent sur un grand nombre de données hétérogènes générées au fur et à mesure de la conduite de l’exploration par des activités telles que la prospection sismique, les forages, l’interprétation des logs de puits.

La chaîne de modélisation qui est mise en œuvre pour la production de modèles géologiques est représentée sur la Figure 1. Elle a pour objectif final la construction d’un *modèle de réservoir* intégré, sur lequel les utilisateurs finaux pourront s’appuyer pour évaluer la quantité de pétrole et/ou de gaz potentiellement présente dans un réservoir. A fin de construction d’un tel modèle, les professionnels des différentes disciplines de l’industrie pétrolière et des géosciences rassemblent leur connaissance en vue d’ interpréter et de modéliser les données brutes acquises dans les champs pétroliers. L’hétérogénéité de ces donnée, celle des formats sous lesquels elles sont représentées, ainsi que l’hétérogénéité des outils logiciels, du vocabulaire et des domaines d’expertise mis en œuvre rend la tâche d’intégration des connaissances particulièrement difficile dans ce contexte.

1.3.1 Cas d'étude concernant l'interprétation sismique

La modélisation géologique commence par la définition d'un *prospect*, c'est-à-dire d'un volume géologique d'intérêt. Les données prises en compte au départ sont principalement celles acquises au moyen de la prospection sismique (Figure 1 (1)) et celles qui résultent de l'exploitation des renseignements fournis par les sondages (Figure 1 (2)).

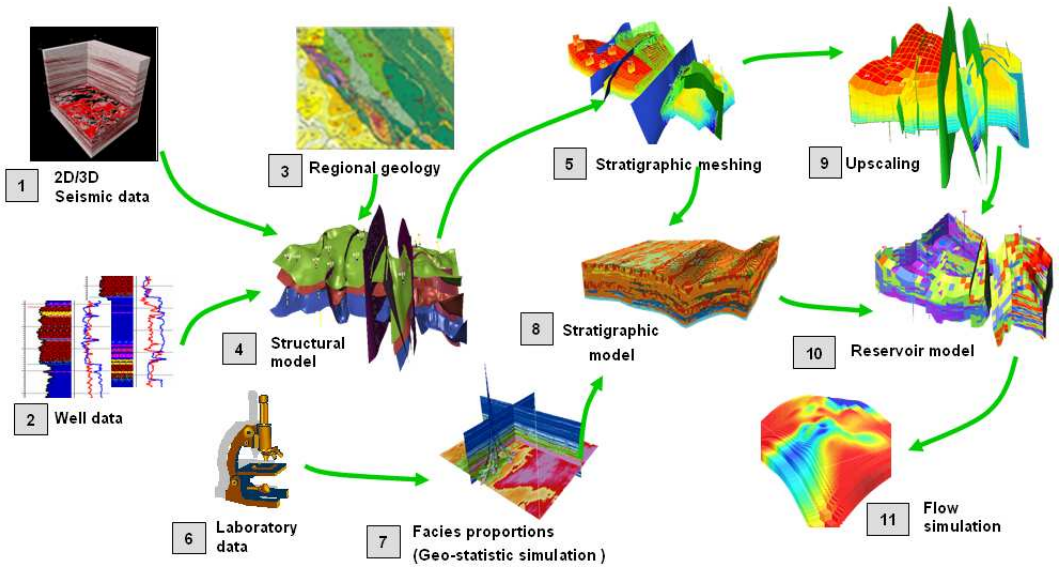


Figure 1: La chaîne de modélisation géologique

La technique de prospection sismique consiste à enregistrer à l'aide de nombreux capteurs, les échos résultant de la réflexion sur des horizons géologiques d'ondes acoustiques artificiellement produites en mer ou sur la surface terrestre. Suite à des opérations plus ou moins complexes de traitement du signal, il est possible d'obtenir une *image sismique* 2D ou 3D, qui permet de visualiser les portions d'horizons géologiques (*réflecteurs*) au niveau desquels se sont produites les réflexions enregistrées. En s'aidant de moyens informatiques, les géologues et géophysiciens procèdent ensuite à une interprétation de cette image sismique. Cette tâche *d'interprétation sismique* consiste à identifier des "motifs" correspondant à divers types de surfaces géologiques (telles que des horizons ou des failles) ou à des assemblages de surfaces plus ou moins complexes correspondant à des objets sédimentaires spécifiques (par exemple des chenaux ou des dômes de sel). L'interprétateur décrit une *scène géologique* en identifiant les objets qui sont présents ainsi que leurs relations mutuelles. Ceci exige de prendre en compte, en sus des données sismiques, les données fournies par les sondages, qui sont les seules aptes à fournir des informations exactes sur la localisation spatiale des horizons géologiques détectés. Au cours de leur travail d'identification d'objets géologiques, les interprétateurs emploient un vocabulaire spécifique qui est celui relatif aux divers sous-domaines qui sont pris en considération : géologie, sismique, analyse des données de puits.

La prospection fournit de manière permanente des données nouvelles, qui sont objets de nouvelles interprétations. Un grand nombre de données hétérogènes sont ainsi produites, que les utilisateurs souhaitent pouvoir éventuellement rechercher et réutiliser à tout moment.

À l'aide d'outils logiciels spécifiques (appelés *géomodeleurs*), les géologues assemblent les surfaces géologiques identifiées suite à l'interprétation sismique en vue de construire le **modèle structural** du prospect (Figure 1 (4)), qui servira à son tour de base pour la construction de modèles plus sophistiqués aptes à fournir des informations sur la nature et la répartition spatiale des roches présentes dans le réservoir (**modèle stratigraphique**, Figure 1 (8)) et sur les propriétés physiques qui en découlent (porosité, perméabilité (**modèle de réservoir**, Figure 1 (10)). Ce dernier modèle est celui qui est utilisé par les ingénieurs de réservoir pour simuler le migration des hydrocarbures fluides au sein du sous-sol et pour estimer la quantité exploitables de réserves présentes dans le réservoir et la qualité de ces réserves (pétrole lourd ou léger, gaz).

1.3.2 Conditions requises pour une modélisation géologique pilotée par la connaissance

Compte tenu de l'état de l'art en matière de modélisation géologique, il existe actuellement de nombreuses questions auxquelles il n'est pas facile de répondre de manière simple. Les utilisateurs finaux d'un modèle déterminé aimeraient par exemple identifier les données à partir desquelles le modèle a été construit et accéder à toutes les informations relatives à la localisation géométrique des données et aux modalités de leur interprétation.

Les plaintes les plus fréquentes des modeleurs tiennent au fait que les modèles courants ne fournissent aucune représentation explicite des objets et des relations géologiques. Les objets géologiques tels que les horizons ou les failles sont actuellement identifiés uniquement par l'intermédiaire de leurs représentations visuelles dans les modèles et il ne leur correspond généralement aucune représentation symbolique, pas même une représentation sous forme de classes à l'intérieur du code. Les relations chronologiques et topologiques entre objets ne sont également accessibles qu'au travers des représentations visuelles observables à l'écran.

Une approche nouvelle de modélisation géologique pilotée par la connaissance a été proposée, il y a quelques années Rainaud et al. (2005). Elle considère que l'identité des objets et des relations géologiques doit être "préservée tout au long de la chaîne de modélisation". Cette approche suppose que les interprétations géologiques (c'est-à-dire l'identification d'objets et de relations par les experts) soient rendues explicites et soient enregistrées à chaque étape du processus de modélisation. Les professionnels souhaitent en effet pouvoir, à n'importe quelle phase de la chaîne de modélisation, poser des questions liées aux objets géologiques et à la gestion de données. Un interprétateur peut désirer identifier par exemple les réflecteurs qui sont intersectés par un puits X dans le modèle. La réponse à cette question exige que soient croisées des informations relatives à des objets issus de deux activités différentes: l'*interprétation sismique* d'une part et la *corrélation entre puits* d'autre part. Actuellement il n'est pas possible de répondre à ce type de question, puisque il n'existe aucun moyen permettant de corréler les données relatives aux diverses interprétations produites le long de la chaîne.

En résumé, afin de définir une approche de modélisation géologique pilotée par la connaissance, il est nécessaire:

- de garder la mémoire des données/interprétations/modèles attachés à un prospect,
- de garder la mémoire du contexte relatif à chaque interprétation (opérée par qui ?, quand ?, où ?,

avec quoi ?),

- d'expliciter la corrélation entre les objets dans les divers modèles.

1.4 Objectif du présent travail

L'objectif de ce travail est de proposer une solution pour l'intégration et l'exploitation des modèles d'ingénierie hétérogènes, afin d'offrir une vue cohérente des différents domaines auxquels chacun de ces modèles est relié et de permettre ainsi l'émergence de connaissances nouvelles indispensables pour les ingénieurs. Notre contribution est double.

En premier lieu, nous proposons un cadre général pour l'intégration des modèles d'ingénierie. Pour cela :

- nous proposons un modèle d'annotation qui permet d'enrichir les modèles d'ingénierie avec de la sémantique;
- nous avons développé une opération d'alignement d'ontologies basée sur une relation partielle de subsomption *is-case-of* qui permet l'intégration de domaines *sémantiquement indépendants*;
- nous avons conçu et réalisé un prototype incorporant le modèle d'annotation et l'opération *is-case-of* ;
- nous démontrons que, dans le cas où les modèles d'ingénierie sont annotées par des experts, il devient possible de les requêter en utilisant le vocabulaire du domaine d'expertise.

Le présent travail se focalise, en second lieu, sur un domaine d'ingénierie particulier, la modélisation géologique, et examine dans ce cas comment la connaissance scientifique peut être modélisée et comment le cadre d'intégration préalablement défini peut être appliqué dans le cas d'étude. Pour cela:

- nous avons défini une étude de cas basée sur une description formalisée de l'activité de modélisation géologique;
- nous avons construit des ontologies de domaine concernant les secteurs des géosciences liés à l'étude de cas;
- nous avons proposé une architecture qui offre aux utilisateurs la possibilité de croiser l'information attachée aux différentes données et interprétations prises en compte dans les diverses étapes de la modélisation et de parfaire ainsi la connaissance globale relative au réservoir modélisé.

Pour traiter le problème de l'exploitation sémantique des modèles d'ingénierie, le présent travail s'est attaché à mettre en œuvre ou à adapter différentes approche basées sur les *ontologies*, l'*annotation sémantique*, les techniques de *méta-modélisation* et d'*intégration d'ontologies*:

- nous avons employé les ontologies pour formaliser et partager la connaissance au sujet des domaines d'application ;
- l'annotation sémantique a été utilisée pour lier les modèles, les outils et les interprétations à la connaissance globale concernant le domaine ;

- l'intégration d'ontologies s'est avérée nécessaire afin de produire une vue globale des différente connaissances relatives aux sous-domaines pris en considération ;
- des techniques de méta-modélisation ont finalement été employées afin de créer de nouvelles primitives pour les modèles d'ingénierie et l'annotation des modèles.

L'articulation de toutes ces stratégies conditionne l'architecture d'intégration à base d'ontologies que nous allons décrire à la suite.

2 Apport du présent travail concernant le management de la connaissance en ingénierie

2.1 Contexte du développement: OntoDB et OntoQL

Dans le cadre de notre travail, nous avons considéré deux critères fondamentaux pour déterminer le choix d'un système de base de données dans lequel développer l'approche proposée.

Le système choisi doit, en premier lieu, pouvoir contrôler un volume considérable d'information dans la mesure où les domaines d'ingénierie font appel à une quantité importante de données. Notre choix s'est donc porté vers *bases de données basées à ontologies* (BDBO) , qui permettent de tirer partie de représentations basées sur des ontologies tout en conservant les avantages liés aux caractéristiques des bases de données (telles que l'extensibilité, la sécurité, etc.).

En second lieu, il est important de pouvoir faire évoluer le méta-modèle qui sous-tend la base de données choisie, afin d'être en mesure de représenter éventuellement d'autres méta-modèles (par exemple un méta-modèle d'*annotations*). Comparée à d'autres BDBOs, l'architecture d'OntoDB est la seule qui accepte d'être modifiée en augmentant les primitives originales. Pour cette raison, nous avons donc choisi de développer ce travail en utilisant la BDBO OntoDB (Dehainsala, 2007).

2.1.1 L'architecture d'OntoDB

L'architecture d'OntoDB est basée sur le système de base de données relationnelles PostgreSQL [1]. Le modèle OntoDB comporte quatre parties associés (*cf.* Figure 2).

La partie montrée sur la Figure 2 (1) correspond au catalogue du système et est traditionnellement disponible dans tout le SGBD, à savoir le **catalogue du système**, qui contient les tables de système qui sont utiles pour la gestion des données. La partie **méta-schéma** (Figure 2 (2)) contient les primitives du méta-méta-modèle d'OntoDB: *ENTITY* et *ATTRIBUTE* qui, en termes de base de données, correspondent à deux tables. Dans OntoDB, ces primitives sont utilisées pour construire les langages d'ontologies dans OntoDB mais aussi éventuellement d'autres langages.

[1] PostgreSQL (ou Postgres) est un système de gestion de base de données (SGBD) en source libre et ouverte (http://www.postgresql.org/).

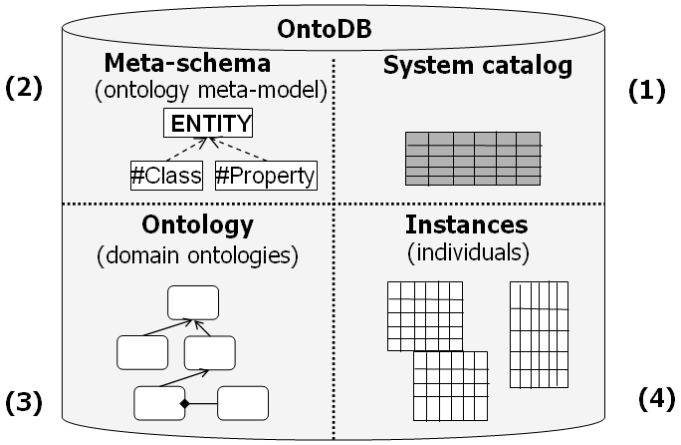


Figure 2: L'architecture d'OntoDB

La troisième partie présente dans OntoDB est appelée **Ontologie** (Figure 2 (3)). C'est le lieu dans lequel nous décrivons la structure des ontologies de domaine, c'est à dire, les concepts et les relations attachés à tel ou tel domaine spécifique. En dernier lieu, les instances des ontologies considérées sont stockés dans la partie **Instance** (Figure 2 (4)). Chaque instance est stockée dans la table qui représente sa classe mère.

2.1.2 Le langage d'exploitation OntoQL

Le langage d'OntoQL a été proposé par Jean et al. (2007) en vue d'exploiter la BDBO OntoDB. OntoQL a une syntaxe semblable au langage SQL, et fournit des outils pour la définition et la manipulation de données, et aussi pour l'interrogation des données dans les trois niveaux d'OntoDB, du niveau *logique* jusqu'au *méta-méta-modèle*. En conséquence, il est possible d'employer le langage OntoQL pour travailler non seulement avec les ontologies de domaine et leurs instances mais également avec le méta-méta-modèle d'OntoDB. OntoQL permet notamment d'accroître le nombre de primitives d'une ontologie en utilisant la clause *CREATE ENTITY* afin de créer des entités supplémentaires.

Par exemple, afin d'ajouter le constructeur de restriction OWL *#AllValuesFrom* au noyau de OntoDB, nous pouvons concevoir les expressions suivantes:

```
CREATE ENTITY #Restriction UNDER #Class (
  #onProperty REF(#Property))

CREATE ENTITY #AllValuesFrom UNDER #Restriction (
  #allValuesFrom REF(#Class))
```

La première instruction crée une nouvelle entité *#Restriction* qui hérite de l'entité *#Class*. Cette entité a l'attribut *#onProperty*, qui fait une référence à l'entité *#Property*. La deuxième instruction crée l'entité *#AllValuesFrom* en tant que sous-entité de *#Restriction*. L'attribut *#allValuesFrom* indique la classe (dont le type est une référence à l'entité *#Class*) dont les instances de la restriction prennent leurs valeurs, pour la propriété définie dans l'attribut *#onProperty*.

Cette approche permet de représenter des ontologies dans OBDB quel que ce soit le méta-modèle qui les

sous-tend. Elle permet également la création de primitives entièrement nouvelles.

2.2 Une proposition de modèle d'annotation pour les modèles d'ingénierie

Pour remédier au manque de connaissances explicites dans des domaines d'ingénierie tels que, par exemple, la prospection pétrolière, nous proposons dans ce travail l'utilisation d'une approche d'*annotation sémantique*. Celle-ci vise à assigner des significations explicites aux objets qu'un expert identifie dans les modèles. L'annotation est par ailleurs considérée comme une *entité de niveau supérieur*, indépendante du modèle d'ontologie. Cette proposition est spécifiée en termes d'un *méta-modèle d'ingénierie* et d'un *méta-modèle d'annotation*. Les méta-modèles proposés sont des conceptualisations abstraites qui peuvent être implémentées avec n'importe quel langage.

Nous estimons que, pour attacher une information sémantique à des fichiers de données d'ingénierie, il convient d'*extérioriser* le modèle dans lequel des données sont organisées. Nous proposons que la structure de données des fichiers soit capturée (manuellement ou automatiquement) et exprimée dans un modèle de données réduit, homogène, et formel qui rassemble l'ensemble des éléments de base qui donnent accès aux données. Cependant, il n'est pas souhaitable de représenter les modèles d'ingénierie en utilisant des primitives conçues pour les ontologies (comme *owl:Class* dans le langage OWL ou *rdfs:Property* dans le langage RDFS) dans la mesure où les modèles d'une part et les ontologies d'autre part sont de "nature différente". Nous proposons donc que des primitives spécifiques, différentes des primitives d'ontologies, soient employées pour représenter les modèles d'ingénierie. La stratégie classique pour représenter différents modèles consiste à produire un méta-modèle supérieur et à considérer chaque modèle individuel comme une instance du méta-modèle en question. Ainsi, nous proposons que les primitives utilisées pour représenter les modèles d'ingénierie soient formalisées sous la forme d'un *méta-modèle d'ingénierie*.

2.2.1 Le méta-modèle d'ingénierie

Le méta-modèle d'ingénierie est l'ensemble minimale des caractéristiques nécessaires pour décrire de façon uniforme les modèles d'ingénierie (nom de fichier, identification, principaux objets composés, et ainsi de suite.).La Figure 3 illustre la structure du méta-modèle d'ingénierie proposé.

L'élément *DataElement* est la super-entité abstraite qui regroupe toutes les autres entités du méta-modèle. *DataClass*es sont les primitives de modélisation des catégories de donnée. Les instances de *DataClass* ont chacune une identité et elles peuvent être organisées dans une hiérarchie de spécialisation/généralisation au moyen du lien *subtype_of*. Les attributs des *DataClass*es sont définis grâce à l'entité *DataAttribute*, qui peut avoir une cardinalité minimale et maximale (champs *min* et *max*) et un champ *range* dont le type est défini par l'entité *DataType*. Les *DataAssociation*s traduisent des relations binaires entre des *DataClass*es. Les instances de *DataAssociation* sont des liens entre des instances de *DataClass* qui n'ont pas ni état ni identité. *DataAssociation* est reliée à deux entités *DataAssociationEnd*, qui spécifient :à quelles *DataClass*es l'association est reliée, la multiplicité (champs *min* et *max*) et le type de l'association (champ *aggregationType*) à chacun des points d'extrémité. Le type distingue des associations d'agrégation et de non-agrégation. Ces éléments sont les blocs constitutifs qui permettent

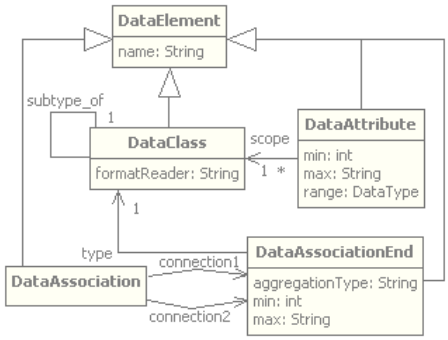


Figure 3: Méta-modèle relatif aux modèles d'ingénierie

représenter n'importe quelle donnée utilisée dans des modèles d'ingénierie.

En formalisant le méta-modèle d'ingénierie, il devient possible de représenter n'importe quel modèle d'ingénierie comme une instance de ce méta-modèle.

2.2.2 Le méta-modèle d'annotation

Afin de rendre la connaissance explicite dans un modèle d'ingénierie, il est nécessaire de "lier" ce modèle aux concepts d'ontologie qui décrivent sa sémantique, ce qui peut être fait par le biais d'une *annotation sémantique*. Nous proposons que l'annotation soit considérée comme une *entité de niveau supérieur*, ayant ses propres attributs, et soit ainsi séparée du modèle ontologique et de l'entité annotée. L'entité d'annotation doit être liée aux concepts des ontologies, en général. Un concept d'ontologie doit pouvoir être lié à une entité d'annotation, quelle que soit la forme sous laquelle il est représenté: *owl:Class* dans le langage OWL ou *rdfs:Property* dans le langage RDFS.

L'entité d'annotation doit être définie au même niveau que les primitives d'ontologie *owl:Class* dans le langage OWL ou *PLIB:Class* de PLIB. La Figure 4 illustre la notion de méta-modèle d'annotation. Les primitives des différents langages de description d'ontologies sont représentées dans la partie gauche de la figure. Dans la partie droite figurent les primitives des ressources: documents, vidéos, images, web-services et celles des modèles d'ingénierie (l'entité *DataElement*). Dans la mesure où les ressources du type documents, vidéos ou images disposent déjà des frameworks qui permettent leur annotation, nous sommes plus particulièrement concernés dans notre cas par l'application du *méta-modèle d'annotation* aux modèles d'ingénierie.

Dans ce cas, l'entité *Annotation* crée un lien entre la primitive des concepts d'ontologie et la primitive *DataElement* au moyen des relations *annotates* et *isAnnotatedBy*. Ces relations ont une cardinalité multiple, ce qui signifie qu'un élément d'annotation peut annoter des entités de modèles d'ingénierie multiples avec un même concept d'ontologie, ou employer des multiples concepts d'ontologie pour annoter un même élément de modèle, ou encore que ces deux cas de figure peuvent se produire simultanément. L'entité *AnnotationProperty* est la primitive qui crée les propriétés d'un élément d'annotation, au moyen du champ *property*. Au moyen de la primitive *AnnotationProperty* il est possible de définir autant de

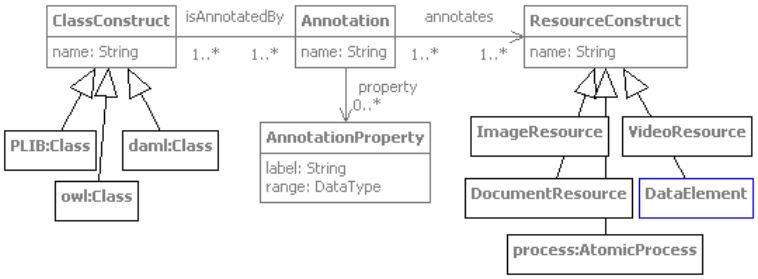


Figure 4: Méta-modèle pour l'annotation des modèles d'ingénierie

propriétés que nécessaire pour un type donné d'annotation, par exemple, des propriétés permettant de stocker une version d'annotation donnée ainsi que son statut, l'activité dans laquelle cette annotation a été produite, les outils employés dans l'activité correspondante, etc.

2.3 Proposition pour l'annotation de modèles dans OntoDB

Le méta-schéma du système d'OntoDB a été augmenté à l'aide des primitives proposées dans le Méta-modèle d'Ingénierie et dans le Méta-modèle d'Annotation. Pour cette opération, nous avons employé le langage d'exploitation OntoQL, qui permet de manipuler l'ensemble de primitives définies dans le méta-schéma d'OntoDB. Les expressions d'OntoQL utilisent la clause *CREATE ENTITY* pour créer les primitives du Méta-modèle d'Ingénierie dans le méta-schéma d'OntoDB.

Dans le détail, la première étape consiste à modifier le méta-schéma d'OntoDB afin d'ajouter les entités du Méta-modèle d'Ingénierie, comme cela a été illustré sur la Figure 3). L'entité ***#DataElement*** est créée indépendamment de l'entité d'ontologie *#Class*, ayant un attribut *#name* du type chaîne de caractères. L'entité ***#DataClass*** est créée en tant qu'entité-fille de *#DataElement*, et hérite ainsi l'attribut *#name*. L'attribut *#subtype_of* fait référence à une autre entité *#DataClass*. Les autres entités du Méta-modèle ont été créées de la même manière.

```
CREATE ENTITY #DataElement (
  #name STRING )

CREATE ENTITY #DataClass UNDER #DataElement (
  #subtype_of REF (#DataClass)),
  #formatReader STRING
```

L'étape suivante consiste à modifier le méta-schéma d'OntoDB afin d'ajouter les primitives du Méta-modèle d'Annotation, comme illustré sur la Figure 4. L'entité ***#Annotation*** est créée indépendamment de l'entité d'ontologie *#Class*, ayant un attribut *#name* du type chaîne de caractères. L'attribut *#annotates* fait référence à l'entité *#DataElement* du Méta-modèle d'Ingénierie, et l'attribut *#isAnnotatedBy* fait référence à l'entité d'ontologie *#Class*. Ceci établit le lien entre une entité du méta-modèle d'ingénierie et une entité du méta-modèle de l'ontologie. L'attribut *#property* fait référence à l'entité *#Annotation-Property*, ce qui permet d'ajouter des informations plus contextuelles à l'annotation. L'entité ***#AnnotationProperty*** est créée ayant un attribut *#name* du type chaîne de caractères.

```
CREATE ENTITY #Annotation (
  #name STRING,
  #annotates REF (#DataElement),
  #isAnnotatedBy REF (#Class),
  #property REF (#AnnotationProperty))

 CREATE ENTITY #AnnotationProperty (
  #label STRING,
  #range REF(#PrimitiveType))
```

La Figure 5 illustre l'architecture d'OntoDB après extension de son méta-schéma avec les nouvelles primitives.

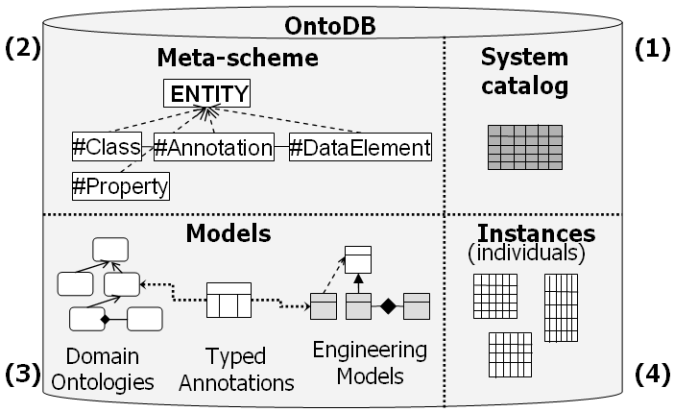


Figure 5: Architecture d'OntoDB augmentée par l'ajout des primitives des méta-modèles d'ingénierie et d'annotation

Une fois exécutés, les scripts en OntoQL modifient le méta-schéma d'OntoDB (partie 2 de la Figure 5) en ajoutant de nouvelles entités et attributs, soit respectivement des instances de *#ENTITY* et *#ATTRIBUTE*. Dans la partie 3 de la Figure 5 les primitives du Méta-modèle d'Ingénierie permettent de définir les *modèles* qui ne sont pas des ontologies. Par ailleurs, les primitives du Méta-modèle d'Annotation permettent de définir des *annotations* qui font le lien entre les modèles d'ingénierie et les ontologies de domaine. Grâce aux nouvelles entités ajoutées à son méta-schéma, OntoDB peut donc stocker dans une seule et même base de données, les données de modèles d'ingénierie et leurs annotations à base ontologique. Ce méta-schéma sera utilisé pour créer les modèles et les annotations utiles pour l'étude de cas considéré dans cette thèse.

2.4 Intégration des modèles d'ingénierie

Les domaines d'ingénierie dépendent de disciplines très diverses, qu'il n'est pas facile d'intégrer. Le présent travail n'a pas pour but d'intégrer divers schémas se rapportant à des entités du monde similaires (ce que ferait, par exemple, une ontologie d'information sur des livres), mais divers domaines a priori indépendants les uns des autres (comme la mécanique et l'électronique, ou la modélisation 3D et la géologie) chacun de ces domaines étant décrit par sa propre ontologie.

2.4.1 Architecture pour l'intégration des modèles d'ingénierie

Nous allons décrire ici une *architecture pour intégration des modèles d'ingénierie à base ontologique*. Dans cette architecture les modèles d'ingénierie sont caractérisés comme des *sources de données locales*; leur représentation comme instances du Méta-modèle d'ingénierie sont appelées *vues locales*; les domaines d'expertise sont décrits par des *ontologies locales* et le domaine fédérateur par une *ontologie globale*; la connexion des vues avec leur signification ontologique est fournie par les *annotations*; la correspondance entre les ontologies locales et l'ontologie globale sont désignés comme des *alignements local-global*. Par ailleurs, des requêtes peuvent être définies aussi bien sur les ontologies locales que sur l'ontologie globale. L'architecture, illustrée sur la Figure 6, peut être décrite comme suit.

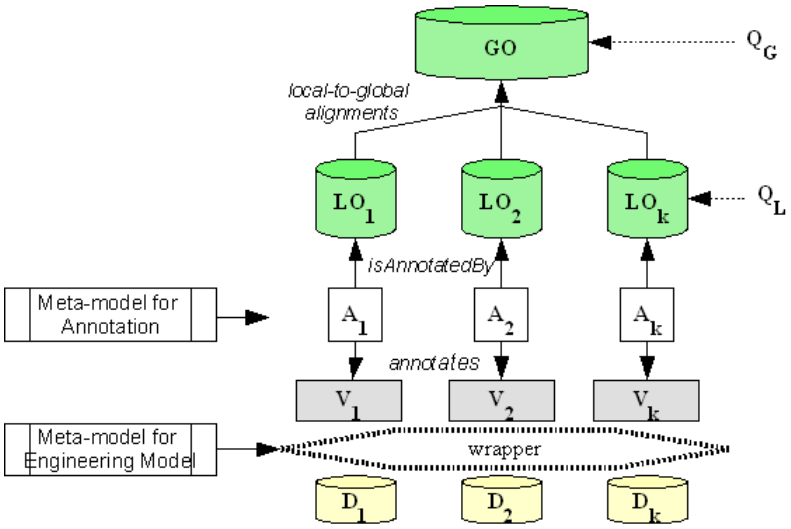


Figure 6: L'architecture pour l'intégration des modèles à base ontologique

Les sources de données locales (D_I) sont *enveloppées* dans un langage unifié, fourni par le Méta-modèle d'ingénierie. Le résultat (les représentations formelles des modèles d'ingénierie) sont les vues locales (V_I). Les *ontologies locales* (LO_I) sont définies. Chaque LO décrit le vocabulaire utilisé dans un domaine spécialisé. Les sources locales sont reliées à leur LO respectif au moyen d'*annotations sémantiques* (A_I). L'annotation peut être manuelle ou semi-automatique. Les sources locales et les ontologies locales ne sont pas modifiées dans ce processus. Une ontologie globale (GO) est définie. Le vocabulaire décrit dans la GO est partagé par les divers domaines locaux. Un ensemble de liens de correspondance est défini entre les concepts de la GO et les concepts des LO (LO-GO). Le processus d'alignement des ontologies doit être manuel, puisque les diverses ontologies ne se rapportent pas au même domaine de spécialisation. Les structures des LO et de la GO ne sont pas modifiées dans ce processus. Quand une nouvelle source de données locale doit être intégrée, les différentes étapes précédentes sont répétées. Les nouvelles annotations (liens entre les données et les ontologies) et les alignements LO-GO sont ajoutés à la base de connaissance sans que la structure originale des données et des ontologies soit changée.

Quand un utilisateur formule des requêtes en termes des concepts de la GO (Q_G), la question est propagée aux ontologies locales en appliquant les alignements LO-GO. L'utilisateur peut également formuler des requêtes en termes des concepts des LO (Q_L). Les résultats des requêtes sont intégrés par le système et

présentés à l'utilisateur.

2.4.2 Relations de correspondance entre les LO et la GO

Pour créer une correspondance entre les domaines locaux et le domaine global, il nous est apparu préférable de définir un ensemble de *relations de subsomption* des concepts des LOs vers des concepts de la GO, plutôt que de définir des *relations d'équivalence*. Ceci tient compte de la nature des ontologies impliquées, qui décrivent généralement différentes perspectives relativement à des domaines qui sont très semblables.

La relation typique de subsomption est la relation *is-a* , qui implique la transmission totale des propriétés de la classe mère vers la classe fille. A l'inverse, la relation *is-case-of* est une relation de subsomption qui n'est pas associée à un mécanisme d'héritage de propriétés. Cette relation est proposée comme une relation de subsomption explicite dans le *modèle d'ontologie PLIB* (Pierra, 2004). Si une classe B donnée est déclarée comme étant *un cas d'*une classe A, il est nécessaire d'importer explicitement de A les propriétés dont on souhaite que B hérite. Ceci permet d'importer des propriétés sans devoir faire des doublons des classes ou des propriétés. On peut assurer ainsi un degré d'indépendance plus élevé des ontologies issues de différentes sources, ayant éventuellement des cycles de vie différents. La relation *is-case-of* entre deux concepts, comme *(B is-case-of A)*, exprime la situation où l'expert interprète le concept *B* comme étant un cas spécifique du concept *A*. Cela signifie que des instances du concept *B* sont également considérés comme étant des instances du concept *A*, même si ces deux concepts ne partagent pas les mêmes propriétés.

Il apparaît ainsi opportun de définir en plus des relations *classiques* d'alignement entre les concepts de différents ontologies, des alignements basés sur des *relations de subsomption*, et plus spécifiquement, la relation *is-case-of* , qui permet de préserver l'indépendance entre les ontologies aussi bien que leur interdépendance.

Par ailleurs, la relation *is-case-of* permet que l'établissement de correspondances entre les concepts des ontologies soit effectué selon un mode *a posteriori* , c'est-à-dire une fois que les ontologies ont déjà été définies. Cette approche qui travaille à partir d'ontologies déjà existantes s'oppose au mode *a priori* (Bellatreche et al., 2004) dans lequel les correspondances sont définies durant la *conception des ontologies* elles mêmes et sont *incluses dans* la définition de ces mêmes ontologies.

2.5 Approche proposée pour l'intégration de modèles dans OntoDB

Considérons un concept qui est cas d'un autre concept, nous désignerons le premier comme *concept englobé* et le second comme *concept englobant*. Utilisant ces définitions, nous proposons deux types d'opérations *is-case-of* pour OntoQL: le ***a priori case-of*** et le ***a posteriori case-of*** . La grammaire et la sémantique du langage de définition de données d'OntoQL ont été modifiées pour prendre en considération la hiérarchie des concepts *is-case-of* lors d'une requête.

2.5.1 A priori case-of

L'opérateur *a priori case-of* est prévu pour être employé au moment de la création d'un concept d'ontologie. Dans ce cas, le concept englobé définit explicitement les propriétés à importer du concept englobant. Considérons par exemple que les concepts *A* et *B* existent déjà, avec leurs propriétés respectives *a1* et *b1*, comme illustré sur la Figure 7. On peut alors créer le concept *C* comme cas des deux concepts *A* et *B* important respectivement les propriétés *a1* de *A* et *b1* de *B*.

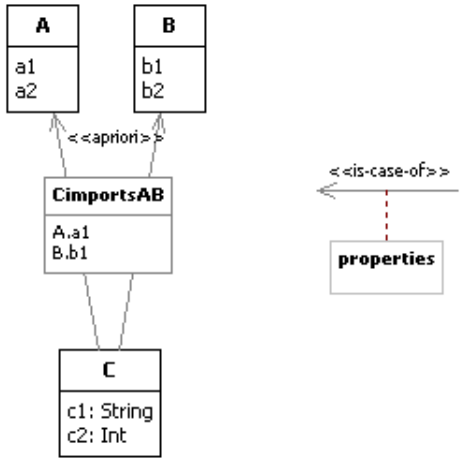


Figure 7: Exemple de *a priori case-of*

Dans ce cas, l'expression valide d'OntoQL pour créer le concept *C* comme cas de *A* et de *B* sera:

```
CREATE #Class C ISCASEOF (A, B) (
  IMPORTS(A.a1, B.b1)
  PROPERTIES (c1 STRING, c2 INT))
```

Cette expression crée un concept *C* qui est un cas des concepts *A* et de *B*, et importe la propriété *a1* de *A* et la propriété *b1* de *B*. Le concept *C* spécifie aussi ses propres propriétés *c1* et *c2*.

2.5.2 A posteriori case-of

La relation *a posteriori case-of* a un comportement particulier dans la mesure où elle est créée *après* que les concepts de l'ontologie aient été définis. Dans ce cas, la définition originale des concepts utilisés ne peut pas être changée. Le concept englobé ne peut donc pas importer des propriétés du concept englobant.

Pour illustrer ceci, considérons le cas où les concepts *A*, *B* et *C* existent déjà, avec leurs propriétés respectives, comme cela est montré sur la Figure 8.

Dans ce cas, une expression OntoQL valide pour la création d'une relation *a posteriori case-of* entre le concept *C* et les concepts *A* et *B* pourra être:

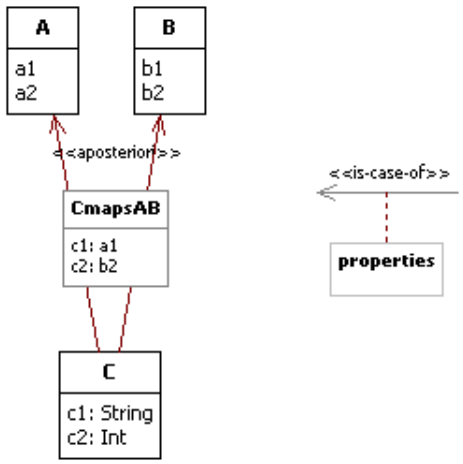


Figure 8: Exemple de *a posteriori case-of*

```
CREATE #AposterioriCaseOf C CASEOF (A, B)
WITH (C.c1 MAP A.a1, C.c2 MAP B.b1)
```

Cette expression crée une instance de l'entité *#AposterioriCaseOf* qui lie le concept *C* aux concepts *A* et *B*. Elle relie la propriété *c1* de *C* à la propriété *a1* de *A* et également la propriété *c2* de *C* à la propriété *b1* de *B*.

À l'origine, une requête OntoQL prend seulement en considération la hiérarchie de classes classique (c.-à.-d., la relation *is-a*). La clause *SELECT* a dû être modifiée afin d'inclure également la hiérarchie de classes *is-case-of* . Nous avons décidé de proposer deux opérateurs qui permettent à l'utilisateur de requêter explicitement les classes *is-case-of* . Ces opérateurs, une fois ajoutés à une requête OntoQL, modifient l'opération *SELECT* comme suit.

- *WITH APRIORI*: active la recherche pour des classes *a priori case-of* .
- *WITH APOSTERIORI* : active la recherche pour des classes *a posteriori case-of* .

Nous pouvons illustrer la modification de la clause *SELECT* d'*OntoQL* comme suit. Considérons la configuration de la base de données ci-dessous, dans laquelle la classe *Student* est un cas de la classe *Person*.

```
CREATE #Class Person (
  PROPERTIES (name STRING, age INTEGER, profession STRING, email STRING))

CREATE #Class Student ISCASEOF Person (
  IMPORTS (name, age)
  PROPERTIES (registrationID INTEGER))
```

La requête OntoQL ci-après se sert du quantificateur *WITH APRIORI* pour choisir des instances de la classe *Person*, des sous-classes de *Person* et des classes qui sont de cas de *Person*.

```
SELECT name, profession FROM Person WITH APRIORI
```

Le résultat renverra des instances des classes *Person* et *Student*. La propriété *name* sera évaluée pour les instances des deux classes, mais la propriété *profession* va résulter *NULL* pour les instances de *Student*, puisque cette propriété n'est pas définie dans *Student* et n'est pas importée par la classe *Student*.

Grâce à la relation *is-case-of* , il est possible d'adresser, dans une même requête, des concepts qui n'avaient pas été originalement définis dans une même hiérarchie. La relation *is-case-of* permet également de définir des concepts sans se servir du *multi-héritage*. Un concept peut hériter d'un super-concept et être aussi un cas de plusieurs autres concepts. Ces possibilités ont une grande importance lorsqu'on a affaire à des ontologies "fortement typées", telles que celles qui sont mises en places dans les bases de données relationnelles.

3 Étude de cas concernant l'interprétation sismique pour l'exploration pétrolière

3.1 Définition d'ontologies

Une partie significative de cette thèse a été consacrée à la définition d'ontologies pour décrire les connaissances liées aux disciplines relatives à l'étude de cas. La caractérisation des réservoirs de gaz et de pétrole est basée sur l'expertise des professionnels de diverses disciplines reliées aux géosciences. Pour que ces experts puissent échanger entre eux au travers d'outils de modélisation informatisés, il est nécessaire qu'ils s'accordent une représentation commune de la connaissance concernant les *objets géologiques* modélisés. Au vu de ce besoin et de l'état de l'art dans le domaine, nous avons été amenés à proposer une ontologie globale, l'*ontologie de la géologie de base*, qui se rapporte aux objets géologiques utilisés dans les modèles géologiques. Nous proposons par ailleurs de détailler cette ontologie de la géologie de base en diverses **sous-ontologies**, en vue de décrire et de relier les unes aux autres l'ensemble des entités géologiques qui doivent être prises en considération pour la modélisation géologique.

L'ontologie de la géologie de base est centrée sur le concept de *GeologicalObject* qui peut être très diversifié (une unité sédimentaire stratifiée, un récif, un diapir, une faille, sont autant de GeologicalObjects.). La partie centrale de l'ontologie de la géologie de base est montrée sur la Figure 9. Cette ontologie est par ailleurs détaillée dans Perrin et al. (2008)).

La prise en compte des modalités complexes de description du temps géologique est un autre élément essentiel pour la représentation des connaissances utiles à la modélisation. Pour cette raison, nous avons été amenés à proposer des modèles d'ontologies pour décrire le temps géologique et ses relations de datation avec les objets géologiques.

3.1.1 Ontologie du temps géologique

La hiérarchie des périodes géologiques est décrite dans les échelles de temps stratigraphiques. Pour représenter ces échelles, nous avons défini une ontologie du temps géologique. Celle-ci est centrée sur deux concepts principaux: *GeochronologicUnit*, qui représente les *intervalles* de temps géologique, et *GeochronologicBoundary*, qui représente les *limites* entre ces intervalles. Les deux concepts s'opposent

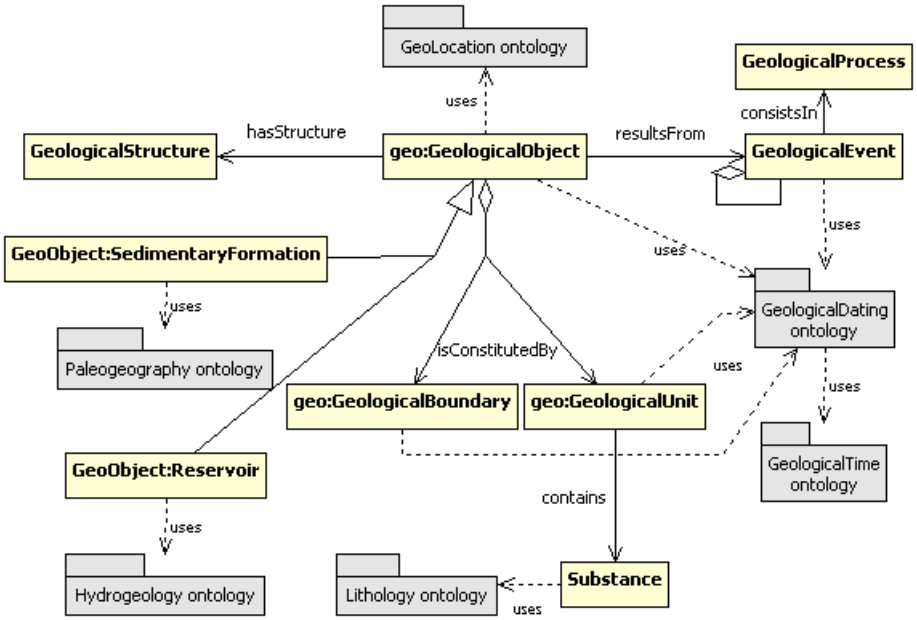


Figure 9: L'ontologie de la géologie de base

puisque les GeochronologicUnit> correspondent à des intervalles de temps de durée significative tandis que les GeochronologicBoundary> ne sont que des *instants* n'ayant aucune durée temporelle. L'ontologie de temps géologique est représentée sur la Figure 10.

3.1.2 Ontologie de datation géologique

Le processus de *datation géologique* consiste à affecter un "âge" à tout objet géologique. L'*ontologie de datation géologique* que nous proposons et qui est illustrée sur la Figure 11, définit des concepts abstraits qui permettent de faire le lien entre les concepts de l'*ontologie du temps géologique* et de l'*ontologie de la géologie de base*. Les concepts importés de l'ontologie du temps géologique ont le préfixe *GeoTime*, et ceux importés de l'ontologie de la géologie de base, le préfixe *BasicGeo*.

3.1.3 Ontologies décrivant des modèles d'activités géologiques

Les professionnels engagés dans la modélisation géologique souhaitent avoir accès à la totalité de la connaissance attachée aux objets qu'ils modélisent (unités et limites stratigraphiques, failles individuelles et réseaux de failles, etc.). Pour répondre à ce besoin, nous avons été amenés à définir des ontologies locales représentant les concepts attachés aux activités spécifiques développées le long de la chaîne de modélisation géologique, telles l'**interprétation sismique**, ou la **description des puits**. Les concepts de l'ontologie de la sismique (Figure 12a) ont été définis dans le travail de doctorat de Philippe Verney (Verney, 2009). L'ontologie des puits (Figure 12b) a été définie en se reprenant un ensemble de concepts

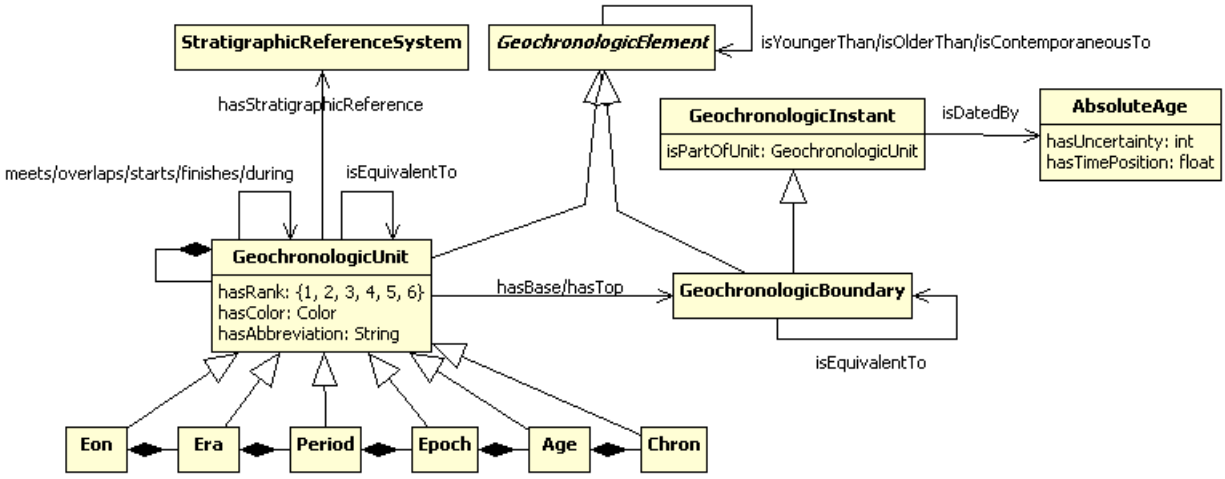


Figure 10: L'ontologie du temps géologique

géologiques détaillés dans la norme WITSML [2] relative aux forages pétroliers.

Au total, 5 ontologies de domaine ont été définies relativement aux disciplines des géosciences considérées dans ce travail. Ces ontologies détaillent 151 concepts et 137 propriétés. L'entrepôt utilisé pour le stockage de ces ontologies est la version étendue de la base de données à base ontologique OntoDB.

4 Validation du travail réalisé

4.1 Définition du use case considéré

Afin de valider les propositions de ce travail, nous avons choisi de considérer les activités relatives à l'interprétation sismique, qui correspondent à la première phase de la chaîne de modélisation géologique. Nous considérerons ici les données obtenues par l'interprétation sismique du bloc Alwyn, qui correspond à des données relatives à un champ situé dans la Mer du Nord fournies par Total UK.

Pour interpréter ces données, l'expert géologue émet un certain nombre d'hypothèses, qui lui permettent d'identifier puis d'assembler les objets géologiques qui entreront dans le modèle qu'il construit. Toutefois, ces interprétations ne sont pas stockés dans le modèle, ce qui a pour conséquence de rendre difficile la vérification des connaissances introduites dans le modèle. Pour préciser les choses dans notre cas d'utilisation, nous avons rassemblé quelques questions pour lesquelles les experts souhaitent obtenir des réponses. Nous en donnons ci-dessous une liste abrégée.

Q_1 - Quels horizons sont plus jeunes que Lias, et plus anciens que Crétacé ?

[2] WITSML (*Wellsite Information Transfer Standard Markup Language*) est une norme XML, développée par Energistics, pour l'échange de données entre les organismes dans l'industrie pétrolière (voir http://www.witsml.org/).

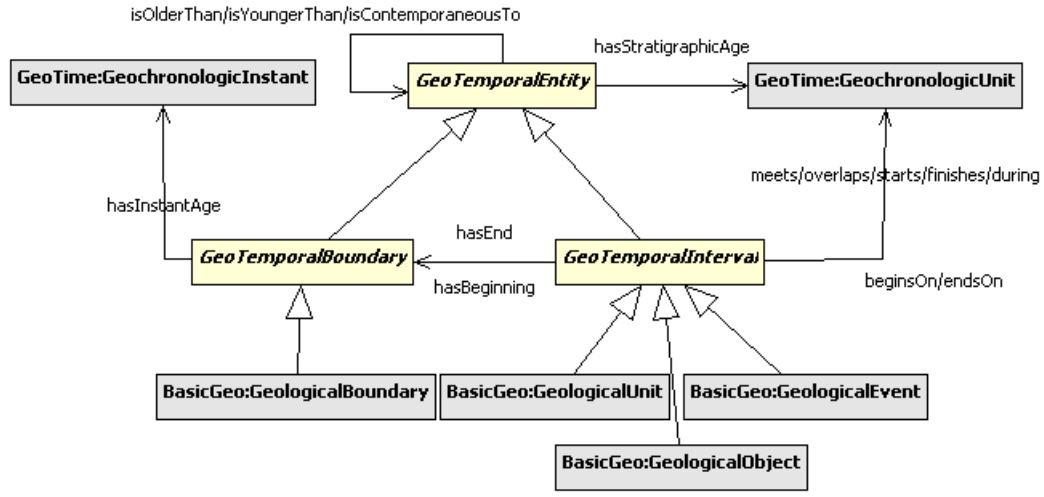


Figure 11: Ontologie de datation géologique

Q$_2$ - De quelle image sismique provient l'horizon BCU ?

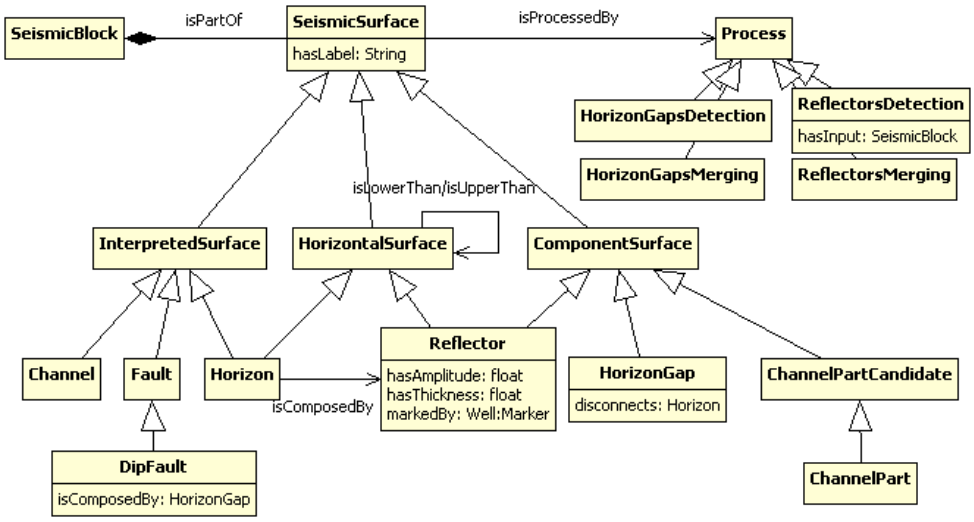
Q$_3$ - Quels horizons sont plus jeunes (ou plus anciens) que Top Dunlin ?

Pour être en mesure de répondre à ces questions, nous avons représenté les interprétations concernant les objets géologiques comme des *annotations sémantiques*. Les questions ci-dessus deviennent des questions sémantiques relatives à ces annotations.

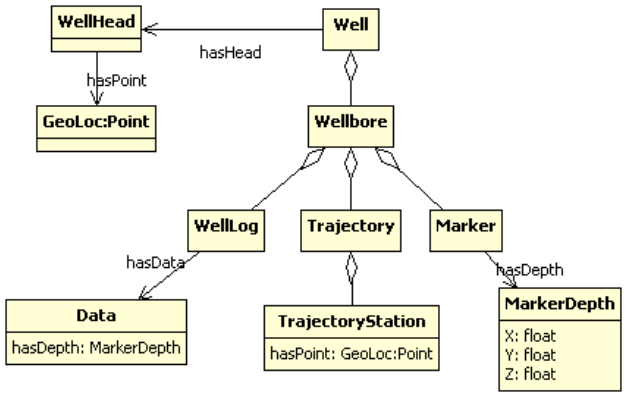
4.2 Approche mise en œuvre

La première étape de l'approche que nous proposons dans ce travail basée sur des annotations relatives à l'interprétation sismique, consiste à représenter tous les formats de données utilisés dans l'activité d'interprétation sismiques comme des *instances du méta-modèle d'ingénierie*. L'étude de cas considérée fait appel à des fichiers dans les formats LAS (pour les données de puits), DAT (pour les marqueurs), SEG-Y (pour les blocs sismiques) et PLO (pour les surfaces). Les modèles sismiques ont été codés dans OntoDB utilisant les primitives du méta-modèle d'ingénierie ajoutées dans OntoDB. Les types de modèles (comme par exemple, le type *PLOFile*, ont été créés avec l'entité *#DataClass*. Les fichiers de données réelles produits par l'interprétation sismique sont représentés à la suite comme des *instances* des modèles de données sismiques (par exemple *Top_Dunlin.plo*).

Les interprétations sont ensuite stockées comme des *annotations* utilisant les concepts des ontologies locales (ontologie de la sismique et ontologie des puits). Le modèle d'*annotation sismique* a été codé en utilisant la primitive du méta-modèle d'annotation ajoutée dans OntoDB : *#Annotation*. L'annotation sismique définit les propriétés *author*, *date*, *amplitudeThreshold* et les propriétés *name*, *isAnnotatedBy* et *annotates*. Les instances d'annotation font le *lien* entre les modèles d'ingénierie et les objets géologiques interprétés (concepts des ontologies).



(a) Ontologie de la sismique



(b) Ontologie des puits

Figure 12: Ontologies locales pour la modélisation géologique

Les liens d'annotation sont codés dans OntoDB sous forme d'instances du modèle d'annotation sismique. La Figure 13 illustre la structure finale des modèles d'ingénierie, des ontologies et des annotations relatives à l'étude de cas.

4.2.1 Exemple de requête et de résultats

Les données et les interprétations relatives à l'étude de cas ayant été stockées dans la même base de données que les ontologies locales et globales, nous pouvons maintenant fournir des réponses à l'utilisateur concernant les questions formulées. A titre d'exemple, nous présentons à la suite la requête OntoQL correspondant à la question Q_3 ainsi que les résultats obtenus.

Si l'on analyse la question posée, on constate qu'elle est ambiguë car le terme *Top_Dunlin* désigne à

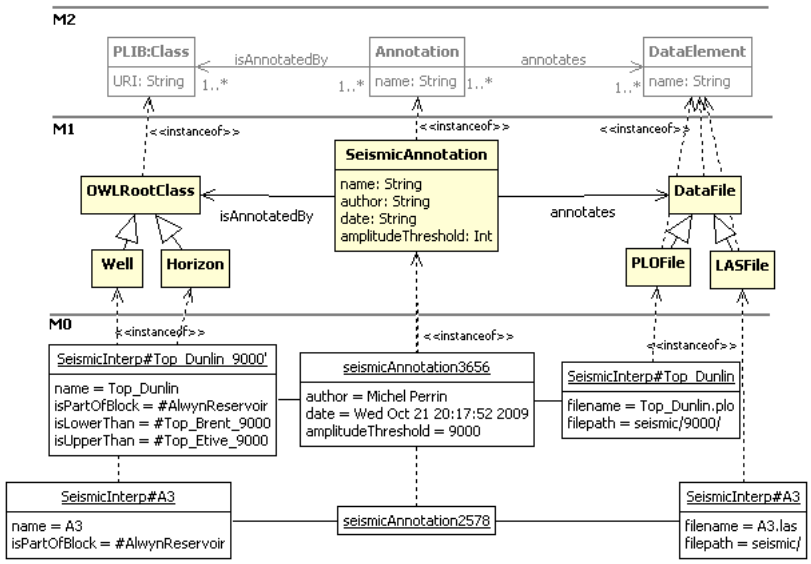
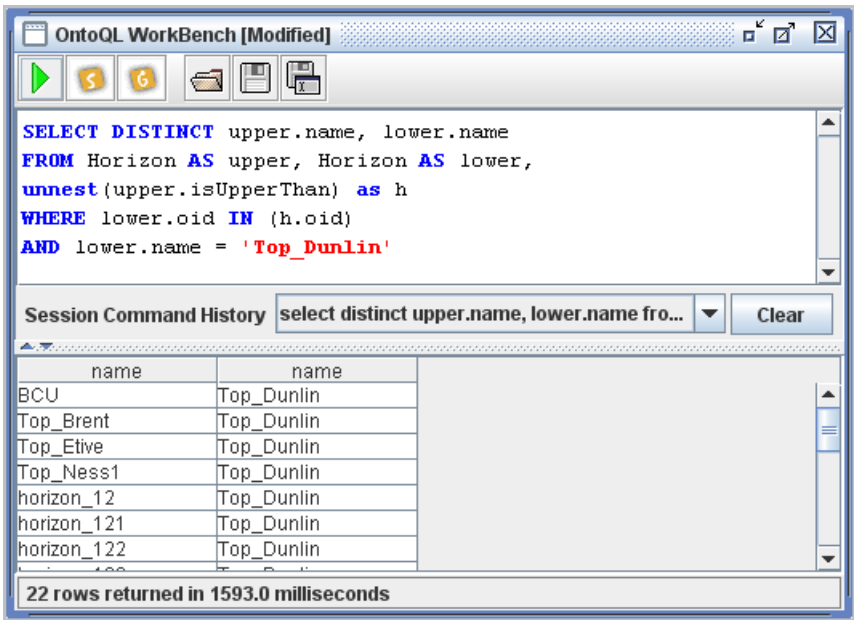


Figure 13: Étude de cas en interprétation sismique

Table 1: Q_3: Quels horizons sont plus jeunes (ou plus anciens) que Top Dunlin ?

```
SELECT DISTINCT upper.name, lower.name
FROM Horizon AS upper, Horizon AS lower, unnest(upper.isUpperThan) as h
WHERE lower.oid IN (h.oid)
AND lower.name = 'Top_Dunlin'
```



la fois une instance d'*horizon géologique* et une instance d'*horizon sismique* (l'horizon sismique qui a été interprété comme correspondant à l'horizon géologique Top Dunlin). On peut donc envisager de répondre à la question en recherchant les noms des horizons répondant à la question dans l'univers de la géologie en considérant l'ontologie de la géologie de base (GO) et ses liaisons avec l'ontologie de la sismique (LO) ou bien, plus simplement, en limitant la recherche au domaine de la sismique (LO). Nous examinerons successivement ces deux éventualités en commençant par la seconde.

Le libellé de la question posé fait référence aux relations *isOlderThan*/*isYoungerThan* qui sont définies dans la seule ontologie de la géologie de base. Pour pouvoir être traitée dans le cadre de la seule ontologie de la sismique (LO), la question doit donc au préalable être traduite dans un "vocabulaire" qui soit celui de cette ontologie. Cette traduction devra être faite par l'utilisateur qui pourra poser que les relations *isLowerThan*/*isUpperThan* appartenant à la LO peuvent être considérées comme des équivalents des relations *isOlderThan*/*isYoungerThan* définies dans la GO.

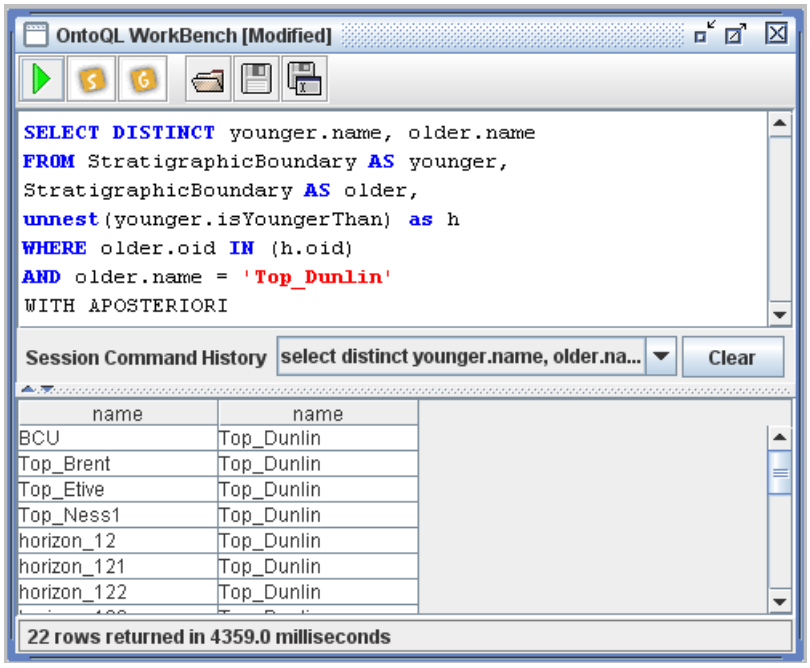
Dans le cas inverse, où l'on décide de travailler non pas à l'échelle de la seule LO mais dans un univers global impliquant à la fois l'ontologie de la géologie et celle de la sismique, il devient possible de conserver la question posée sous sa forme initiale. Cette possibilité est intéressante car les utilisateurs souhaitent de préférence formuler leur requête en utilisant un vocabulaire "géologique" faisant référence aux relations *isOlderThan*/*isYoungerThan*. Dans ce cas, pour répondre à la question posée, nous devons en premier lieu établir des relations *is-case-of* entre les concepts de l'ontologie locale et les concepts de l'ontologie globale. Ainsi les concepts *seismic:Horizon* et *seismic:Reflector* sont des *a posteriori case-of* du concept *geo:StratigraphicBoundary*. Les propriétés *isLowerThan* et *isUpperThan* relatives aux concepts *seismic:Horizon* et *seismic:Reflector* seront respectivement reliées aux propriétés *isOlderThan* et *isYoungerThan* relatives au concept *geo:StratigraphicBoundary*, ces paires de propriétés renseignant les unes comme les autres l'*ordre d'apparition* des objets. Au moyen de la relation *is-case-of* , les exemples des concepts *seismic:Horizon* et *seismic:Reflector* seront également considérés comme des instances du concept *StratigraphicBoundary*. L'utilisateur sera ainsi en mesure de retrouver les objets qui sont un cas de *StratigraphicBoundary* dans les divers sous-domaines auxquels la modélisation fait appel et notamment, en considérant notre exemple, dans le domaine de l'interprétation sismique.

Dans ces conditions, la question posée devient maintenant acceptable sous sa forme initiale qui fait appel au vocabulaire "géologique" de l'utilisateur. La requête OntoQL donne comme résultat toutes les instances de *StratigraphicBoundary* plus les instances des concepts qui sont un *a posteriori case-of* de *StratigraphicBoundary*, et qui sont plus jeunes que l'horizon Top Dunlin.

Les tableaux 1 et 2 montrent les résultats obtenus en mettant respectivement en œuvre l'une ou l'autre des deux démarches décrites ci-dessus. La question posée étant en substance la même dans les deux cas, les résultats obtenus sont bien entendu identiques (correspondant à un même ensemble de 22 horizons identifiés). Toutefois, la seconde démarche qui recherche à la fois dans deux ontologies en faisant appel à des relations *a posteriori case-of* s'avère plus coûteuse en temps (temps d'exécution de plus de 4 millisecondes contre 1,6 millisecondes dans le cas de la première démarche). Cette différence dans les temps d'exécution est le prix à payer pour permettre à l'utilisateur d'obtenir de manière automatique les réponses qu'il attend en ayant formulé sa requête dans son langage, celui de la géologie. Toutefois, dans le cas particulier considéré, cette automatisation pose elle-même question du point de vue de l'utilisateur. Il peut en effet exister des cas où le *mapping isOlderThan* → *isLowerThan* et *isYoungerThan* → *isUp-*

Table 2: Q_3: Quels horizons sont plus jeunes (ou plus anciens) que Top Dunlin ?

```
SELECT DISTINCT younger.name, older.name
FROM StratigraphicBoundary AS younger,
    StratigraphicBoundary AS older,
    unnest(younger.isYoungerThan) as h
WHERE older.oid IN (h.oid) AND older.name = 'Top_Dunlin'
WITH APOSTERIORI
```



perThan devra être remplacé par le *mapping* inverse *isOlderThan* → *isUpperThan* et *isYoungerThan* → *isLowerThan*. Ce sera le cas notamment lorsque les strates géologiques auront été retournées sous l'effet de la tectonique. Pour cette raison, il semble judicieux de laisser à l'utilisateur le choix au cas par cas du type de *mapping* à réaliser. Il est prévu que cette éventualité soit étudiée lors de travaux ultérieurs.

5 Conclusion

Ce travail propose des solutions innovantes en vue de l'exploitation des modèles d'ingénierie hétérogènes. Parmi les multiples problèmes liées à la gestion des modèles d'ingénierie, nous avons plus particulièrement examiné les questions suivantes:

- **Annotation sémantique des modèles d'ingénierie**: Le modèle d'annotation proposé permet d'expliciter les interprétations qui concernent l'*identification* des objets dans les différents sous-domaines relatifs à un modèle d'ingénierie donné.
- **Intégration d'ontologies**: Selon leur niveau d'expertise, les experts peuvent identifier tel ou tel

objet comme *général* (c'est-à-dire relatif à un domaine d'intérêt général pour le modèle d'ingénierie considéré) ou comme *spécifique* à un sous-domaine déterminé. Compte tenu de cette dualité, il a été nécessaire de proposer des solutions permettant d'aligner des ontologies sémantiquement indépendantes.

- **Représentation et persistance**: Grâce à des techniques de méta-modélisation, il a été possible de produire une représentation uniforme des ontologies, des données et des annotations. Toutes ces représentations ont été stockées dans une base de données à base ontologique, qui assure l'extensibilité de la proposition.
- **Requêtes**: Des solutions ont été proposées qui permettent que des modèles annotés par des experts en référence à des ontologies de domaine, puissent être requêtés par des utilisateurs en utilisant les concepts de ces mêmes ontologies, c'est à dire en employant le vocabulaire de leurs domaines d'expertise.
- **Application au domaine de la prospection pétrolière**: Les approches proposées dans ce travail ont été appliquées à un cas d'utilisation réel, qui concerne l'interprétation de données sismiques relatives à la prospection pétrolière. Les expérimentations réalisées prouvent que, grâce à l'approche proposée, les experts peuvent, en utilisant le vocabulaire de leur domaine d'expertise, formuler des questions et obtenir des réponses appropriées.

Le travail actuel doit être considéré un cadre qui fournit des solutions pour la gestion de connaissance en ingénierie à un "premier niveau" en assurant la conservation de l'identité des objets du domaine. Le travail offre diverses perspectives pour de travaux futurs ouvrant la voie au développement de solutions plus ambitieuses. Ces travaux à venir pourront concerner aussi bien l'ajout de nouvelles connaissances dans les ontologies définies que le perfectionnement du prototype développé. Par ailleurs, les approches proposées dans ce travail sont potentiellement applicables à d'autres domaines, dont les activités sont basées sur des modèles d'ingénierie.

Chapter 1

Introduction

1 Integration of heterogeneous models in engineering domains

Building and manipulating models constitute the bulk of engineering activities. Generally speaking, a model is an abstract, simplified representation of the essentials of a real world phenomenon. Models (*cf.* Section 1.1) are the means by which engineers organize, use, and communicate knowledge about products (their structure, their manufacture, their maintenance) (Olsen et al., 1995) or also about real-world domains (their objects, their behavior).

In many fields, methodologies for developing models were established long ago. Consequently, when there was no need to integrate models, professionals have tended to use different terminology and incompatible data formats for building models. As a result, many heterogeneous models were originated from various engineering tools, with no integrating framework beneath and no explicit correspondence between the model elements.

Today, models are specified by means of various modeling languages which each have their own syntax. They are manipulated by various engineering tools, which are usually proprietary and do not interoperate (Tudorache, 2008). Engineers spend much of their working time in search of knowledge used in past projects and in manually translating the various models produced by different tools. When there is the need of data exchange between tools, it is often hard-coded as built-in translators by the vendors. But mere format translation does not enable interoperation and communication between models.

Many aspects of the rationale behind a model remain implicit. Only the engineers, who actually built it, understand the hidden semantics of the model. One reason for this is that, until now, companies hardly saw knowledge as an important asset. As a consequence, the knowledge underlying decisions made by professionals was not well organized or not kept at all. The main output of an engineering procedure is often a piece of paper that is mailed or faxed to the other team members (Gruber et al., 1992). The consequence is that currently engineers from different domains often do not understand the meaning of model elements.

At present, engineers are faced with the challenge of having access to all information about their domain, in order to make well-informed decisions. Modelers from different disciplines must be able to share their

diverse views of the world. However, they face problems that make integration difficult to support. When dealing with information integration, the typical solution is to provide a uniform interface to a collection of heterogeneous information sources, giving users the illusion that there is a centralized and homogeneous information system. This approach works well for activities in which the sources are static, but it has clear limitations in engineering domains (Olsen et al., 1995). New tools and new ways of using them constantly come out so that users will ask for information exchange that was not anticipated by the tool integrators.

The integration approach for engineering domains needs to be dissociated from data sources and formats and from tools, which are constantly evolving. For cooperation, engineers need to agree upon a *common vocabulary* for communicating and also need to describe the *meaning* of data and data formats within models. Correspondence among models should be operated by means of *meaningful descriptions*, which should be detached from the physical part of the models.

1.1 Definitions of a model

There is generally little agreement about what a model exactly is and what it is not. A consensual definition seems to be that given by Rothenberg (1989):

> "Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality."

According to their characteristics, models can be classified as *deterministic* vs. *stochastic*, *static* vs. *dynamic*, model for *simulation* vs. model for *optimization*, models of *phenomena* vs. models of *data*, and so on. Some classification that is of interest for engineering domains consists in dividing into *iconic*, *analogical* and *symbolic* models (Hubka, 1984).

- **Iconic models** are faithful representations of reality, generally involving a change of scale (down or up). Iconic models may be represented in 2D (e.g. photos, drawings) or 3D (e.g. miniaturized model of a building). A common example is *physical mock-ups*, which have the same appearance as the original to be studied. Figure 1.1 shows a full-scale engineering mock-up of an aircraft featured for wind tunnel tests (NASA's Langley Research Center[3]).
- **Analogical models** imitate the real system by analogy rather than by replication (as in the iconic model) and have only certain properties similar to the original. They can be built through visualization (e.g. the color coding of a geographical map, diagrams) or simulations (e.g. a wind tunnel build to observe the aerodynamics properties of vehicles).

[3] http://oea.larc.nasa.gov/PAIS/Partners/

Figure 1.1: Full-scale mock-up of an aircraft

- **Symbolic models** represent the real world using a language made up of symbols (numbers, words, variables, mathematical or logical symbols). *Mathematical models* are often employed for engineering analysis. They are used in place of real-world experiments (e.g. equations that represent the working model of a falling rock) but might require some engine to produce understandable results (e.g. a simulation program on a computer).

1.2 Examples of engineering domains and models

Engineering domains are domains whose main interest is the life cycle of something: the life cycle of a software product, the life cycle of mechanical artifacts (e.g. planes, cars), the life cycle of a petroleum reservoir, and so on. These domains often rely on various engineering models to handle the different aspects of the life cycle of the object of interest.

Engineering domains rely on *models* which are indispensable tools for studying real-world phenomena and also for creating and enhancing technologies. Some examples of engineering domains are: civil engineering, aerospace engineering, automotive industry, environmental engineering, material research, drug design, manufacturing, petroleum engineering, biomedical engineering. In its second part, the present work will take special consideration for the domain of *petroleum engineering* and in particular for the field of *characterization of oil and gas reservoirs*. The main objective is building an integrated *reservoir model*, which is the support for quantifying the amount of oil and/or gas present in a petroleum reservoir. For this, professionals from different disciplines of petroleum industry and geosciences gather their knowledge for interpreting and modeling the data acquired from the petroleum field. The heterogeneity of the various types of data models, software tools, vocabulary and expertise makes integration a difficult task in this context. The activity of reservoir modeling will be detailed in Chapter 6.

All these activities and domains are the subjects of a huge amount of heterogeneous models and data. These domains have been called by Ludäscher et al. (2006) *complex multiple-world scenarios*. These are domains in which knowledge from very different fields of expertise is required to articulate meaningful queries across disciplines (or within different micro-worlds of a single discipline). “In *complex multiple-worlds* scenarios there are often latent links and connections between disparate data sources. Through these implicit knowledge structures, the various pieces of information can be "glued" together to help

answer scientific questions. Making explicit these knowledge structures is therefore a prerequisite for connecting the underlying data." (Ludäscher et al., 2006).

Engineering data can be expressed in various types of models: UML schemas, database tables, programming units (such as classes of `Java` or `C` programming languages), numerical models, graphical visualizations of multiple data sets. In this work, the term *model* make reference to the "amalgam" of data that is generated by some software tool, and that is used to run simulations about the domain in study or perform verification and validation.

1.3 Handling and making semantics of models explicit

At present, there is an overall consensus that activities that rely on strategic knowledge, such as engineering modeling, must include knowledge engineering as a modeling activity. Stefik (1986) proposed the paradigm of *knowledge medium*, meaning that knowledge, when explicitly represented, could be used as a communication medium among people and their programs. Moreover, many previous researches about information sharing in engineering systems adopted the idea of *knowledge media* that consists in considering an explicit representation of the knowledge related to a given domain.

The SHADE project (Gruber et al., 1992) proposes to share engineering knowledge through the internet using a variety of technologies such as agents, subscription systems and notification systems. Each agent has a different problem-solving-knowledge, while they are all in relation with a shared knowledge-base between the different tools. The SHARE project (Toye et al., 1993) employs a wide range of information exchange technologies (such as electronic mail service and file servers) in order to help engineers and designers to achieve a "shared understanding" of the mechanical design process. PACT (Cutkosky et al., 1993) is a concurrent engineering infrastructure that makes use of agent-based technology to integrate existing multi-tool systems.

Some of the above cited frameworks do try to make knowledge explicit, but in fact knowledge is only shared in the *level of tools*. The languages employed for homogenizing data representation and agents, are also those that are set up for searching for information produced by the various technologies. Moreover, these frameworks do not enable users to query sources, most of information retrieval being made *on-demand* (by subscription to a notification service). Finally, adding yet other tools that encapsulate data in their own format increases heterogeneity in the domain of interest.

In the case of engineering models, the current issue is not that of sharing *data*, but of sharing *understanding* about data. Engineering models contain significant information about the expertise domains to which they are related, but this information remains hidden and cannot be recovered. This is due to the fact that the way in which data are organized does not follow the understanding of domain experts. For these reasons, it appears to us that the problem of heterogeneity in engineering domains still subsists.

We believe that the solution for this issue can be found at first by adding a level of *semantics* over the engineering models. The last decade has seen the emergence *ontologies* as tools for providing an explicit and formal definitions of specific domains (Gruber, 1995). Since ontologies can be developed for formalizing semantics in engineering domains, it becomes possible to access engineering models through ontology concepts, i.e. at the *knowledge-level*. The issue is then to attach ontologies concepts

to the engineering models. For this end, we propose to use *semantic annotation*, which is a current Web Semantic technique for adding knowledge to resources by means of semantic tags. Thanks to *model-based semantic annotations*, we intend to make explicit the expert knowledge which is currently enclosed in the model. There is no current technique for completing models by formal comments or explanations, or for attaching more semantics to the technical data produced by modeling tools. We thus consider the approach of *model-based semantic annotations* to be a contribution of this work.

A second step for complete exploitation of engineering models consists in providing *model integration*. Correspondence between models should be ensured in the *ontology level*, in order to be dissociated from data sources, formats, languages and tools. An architecture that maps concepts from *local ontologies* to a *global ontology* guarantees that the users will have an integrated and shared global view of each specific domain involved in the engineering process. The mapping relation used for creating a correspondence between global and local ontologies is the *is-case-of* relation. In contrast with the classical subsumption relationship (the *is-a* relationship), which creates hierarchies (the is-a relation), the is-case-of relation only creates *partial inheritance* hierarchies between concepts (not all properties are imported, only those explicitly chosen). This "light" subsumption relation is appropriate to be used for creating correspondences between concepts of different ontologies, without having to actually merge them.

The case study which is considered in this work concerns the field of petroleum reservoir engineering, and, in particular, the activity of 3D seismic images interpretation. A 3D seismic image provides a visual representation of earth subsurface based on seismic reflection data. Interpreting a seismic image consists in building a description of a "geological scene" by identifying the objects that are present and how they are related each with the others. This object identification task is performed by experts, who use a specific vocabulary associating terms issued from various subdomains: geology, seismics, well drilling. Prospection permanently provides new data, which are objects of new interpretations. A large amount of heterogeneous data is generated by the seismic interpretation activity and by many others in the field of petroleum reservoir engineering, which end-users wish to be able to retrieve and re-use at any moment.

In this work, the main concepts and relationships related to the technical of the specific domains related to seismic interpretation were formalized as ontologies (a Basic Geology ontology, a Seismics ontology a Well ontology). These ontologies were designed in the OWL/RDF language, and persisted in the OntoDB ontology-based database, in which the proposals of this work were implemented for validation.

2 Research questions

The main question investigated by this thesis is:

> *How can heterogeneous engineering models be integrated and exploited so as to offer a coherent view of different domains and allow the emergence of new knowledge that is relevant for the engineers ?*

This question evokes the exploration of some specific points:

- **How can complex scientific or technical knowledge be formalized independently of any specific need?**
- **How can the semantics of engineering models be externalized and formalized?**
- **How to extract and represent correlations among the elements of different engineering models?**
- **How to query the engineering models using meaningful vocabulary?**

3 Contributions

The goal of this research work is to improve the automation of data integration in complex models such as engineering models relying on explicit knowledge across various domains and making such models exploitable for *domain experts* rather than for the computer science experts.

This thesis provides a twofold contribution to the related domains. First, it proposes a general-purpose integration framework for engineering models. Secondly it studies in a particular engineering domain (reservoir modeling) how can complex scientific knowledge be modeled and how the general integration framework formerly defined for engineering models in general may be applied in this case.

Contribution to semantic-based integration.

- proposal of a comprehensive annotation model enabling engineering models to be enriched in semantics;
- development of an ontology alignment operation based on a partial subsumption relation – the *is-case-of* operation – that makes possible the integration of *semantically unrelated* domains;
- design and implementation of a prototype incorporating the annotation model and the *is-case-of* operation;
- demonstration that in the case when engineering models are annotated by experts, they can be queried using significant vocabulary.

Contribution in the earth modeling activity in petroleum engineering.

- set up of a case study resting on a formalized description of the earth modeling activity;
- construction of ontologies for the geosciences fields related to the case study;
- proposal of an architecture which will allow users to cross and exchange information issued from models performed at various steps in the workflow and which thus increases the overall understanding about the reservoir to be modeled.

Some of the contributions of this thesis have been the subjects of several publications:

- Preliminary ideas on knowledge management for earth modeling: (Mastella et al., 2007a,b; Perrin et al., 2007, 2008).

- Results of the e-Wok Hub project: (Ait-Ameur et al., 2008; Rainaud et al., 2008).
- Proposal of ontology-based annotation for engineering models: (Mastella et al., 2008a,c,b, 2009b,a).

4 Working environment

This work was developed in the context of a doctoral thesis registered in the École Nationale Supérieure des Mines de Paris (ENSMP), in close collaboration with two other institutions: the Institut Français du Pétrole (IFP) and the Laboratoire d'Informatique Scientifique et Industrielle (LISI) of the l'École Nationale Supérieure de Mécanique et d'Aérotechnique (ENSMA).

The thesis is registered in the ENSMP's doctoral school named *Sciences et Technologies de l'Information et de la Communication* (STIC), in the specialty *Informatique temps réel, Robotique et Automatique*. The work is supervised by Professor Michel Perrin (ENSMP, Department of Geosciences).

The IFP is the institution that provides the case study for this thesis work. Jean-François Rainaud is a Project Manager of the division *Technologies Informatiques et de Mathématiques Appliquées* (DTIMA) who works on approaches of knowledge-driven reservoir modeling.

Finally, the laboratory LISI/ENSMA is a computer science team that develops research in database, knowledge engineering, workflows. Professor Yamine Aït-Ameur, director of the LISI, is the supervisor of the computer science part of this work.

This thesis is complementary to the thesis of the doctorate student Nabil Belaid, from the laboratory LISI/ENSMA. The present work handles the *static* part of the problem, that is, *how the elements in study will be represented* within the architecture; while Nabil Belaid's thesis deals with the *dynamic* part of the problem, that is, *how the elements in study are produced*.

5 Organization of the thesis

This thesis is organized in three parts.

Part I is dedicated to general issues concerning knowledge management in engineering systems. Chapter 2 discusses related work and provides background for our research in four areas: ontologies, semantic annotation, information integration, and meta-modeling. Chapter 3 introduces the ontology-based database system (OntoDB) and the exploitation language (OntoQL) used in this work.

Part II present the two main contributions of this thesis: Chapter 4 presents the proposed **model for semantic annotation of engineering models** and shows how this model is implemented in OntoDB by extending its basic constructs. Chapter 5 describes the **architecture for ontology-driven integration** designed in this work. It presents the *is-case-of* subsumption relation, which is used to interrelate different ontologies, and describes how the OntoQL language was extended with a *is-case-of* operator.

Finally, Part III examines the issues related to geological knowledge formalization and to geological modeling. Chapter 6 provides an overview of the application domain of this work: the earth modeling work-

flow for petroleum exploration. It identifies the major challenges concerning knowledge-management in the process of creating geological models of the earth underground. Chapter 7 describes the geosciences ontologies that were developed: the Basic Geology ontology, which is the top-level ontology for the case study; the Geological Time and Geological Dating ontologies, which represent the temporal aspects of geological objects; and the local ontologies of Seismic interpretation and Well identification, which describe the objects of these specific domains in the workflow. Chapter 8 shows by means of a case study how the developed proposals were implemented and validated. The case study evaluated queries that are typically posed by domain experts about data and models when working in the earth modeling workflow and showed that it is possible to return significant information that was not possible to make explicit before.

The final chapter concludes by summarizing the results and contributions of this thesis and by pointing out possible future work.

Part I

Semantic based solutions for management and exploitation of engineering models

Chapter 2

State of the Art concerning the Approaches Studied in this Work

This chapter is composed of five sections, in which we introduce all the necessary definitions and sum up approaches and methods that are relevant for the proposal of this work. References will be given for a more detailed presentation.

In the following section, we present the definitions of *ontologies* that we find in the literature and discuss the characteristics that distinguish an ontology from other computer models, notably database models. We present in Section 2 the idea of semantic annotation as a means for semantic enrichment of data resources. In Section 3, we analyze approaches for integrating heterogeneous resources based on their semantics. Section 4 introduces metamodeling approaches as a means of making different models to communicate.

1 Ontologies

The term "ontology" has been widely used in computer knowledge-based systems in the last years. Ontology in its origins, is a branch of philosophy that deals with the nature and the organization of reality, the *Science of Being*.[4] The term was borrowed by Linguistics, where an ontology (with lowercase 'o') *describes the meaning of terms and categories used for linguistic description*. Gruber (2008) tells that the AI community came to use the term ontology in the 1980's to refer to both a theory of a modeled world and a component of knowledge systems. And then, in the early 1990's, ontologies were identified as a key component for creating interoperability standards. At that moment, a globally accepted definition of an ontology in computer science was forged:

> "An ontology is an explicit specification of a conceptualization". (Gruber, 1995)

A *conceptualization* refers to an abstract model of the things that are assumed to exist in some area of interest (objects, relations). An *explicit specification* means that the concepts and relationships in the

[4] From the volume IV of the *Metaphysics* manuscripts, written by Aristotle.

abstract model are given explicit names and definitions. An ontology is a symbolic model of the concepts of some domain (distinctively from other kinds of models, such as numerical models). It represents a domain conceptualization by means of words and of their meanings.

During the last two decades a large number of works have been produced on ontologies: methodologies for building ontologies, frameworks for ontology implementation, evaluation of ontologies, languages for ontologies, among plenty of others. Gruber (1993, 1995); Uschold and Gruninger (1996); Guarino (1998); Noy and McGuinness (2001) are just few of the pioneers authors that considered the use of ontologies in computer science. For a global view of the subject, an extensive bibliography on ontologies, studied from different perspectives, was assembled by Carrara and Guarino (1999). However, this list must be updated with the most recent proposals for ontologies.

1.1 Ontology conceptualization

During the conceptualization of an application domain, the *modeling decisions* (i.e. whether the *things* of the domain will be represented as concepts, instances, attributes) depend, almost inevitably, on the *users*, *goals* and *purposes* of the ontology. In the first methodology proposed for developing ontologies, Uschold et al. (1995) stated that an essential step before building an ontology is to clarify why the ontology is being created, and in which scope it is intended to be used. For example, the conceptualization of a classroom for an architecture project will be different of the conceptualization of the same classroom to some course scheduling system that allows rooms to be assigned to courses.

In spite of the variety of definitions for the term *ontology*, there seems to be some consensus on what an ontology conceptualization should be constituted of. The kinds of things represented by an ontology are called *concepts*, which are groups of individuals that share the same characteristics. The concepts of an ontology are organized by a partial order relation called *is-a*. This relation creates hierarchies where super-concepts are more general than sub-concepts (taxonomy). For example, in an ontology that describes persons, the concept Person is more general than the concept Woman (*Woman is-a Person*). Another type of organization of ontologies is named partonomy, a lattice of concepts organized by the *is-part-of* relation. The fact that a Wheel is part of a Car can be expressed as *Wheel is-part-of Car*. One could also state that an Arm is part of a Person, or still, a Person is part of some Society. But the notion of parthood in each of these examples is different: being a *part* can be interpreted as being just a independent component part, or being an inseparable constituent of the whole. This is better explained in studies of part-whole relations (mereology) such as in Winston et al. (1987).

Each concept in an ontology is described by its attributes. For example, the concept *Person* can have the attributes name, age, marital status. The *is-a* and *is-part-of* relations are useful for the organization of the concepts, but these relations are not *specific* to some domain. Besides these relations, the concepts in an ontology can be associated through relationships that are more significant to the domain of interest. For example, the relationship *marriedTo*, in which a Person can be married to another Person. Moreover, one can restrict the cardinality of the relation *marriedTo* as one to one. Finally, when we describe a specific occurrence of some concept, and give a specific value to its attributes, we are defining an *instance* of this concept. An ontology together with a set of concrete instances constitutes a *knowledge base*. The following toy example presents a simplified ontology of Person (Figure 2.1).

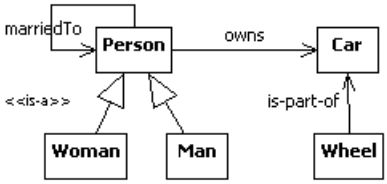


Figure 2.1: A simplified ontology that defines the concept Person

Nevertheless, the terminology used for referring to ontological *things* is not a consensus. Here are some of the most used terms and their equivalences:

- "Concept" ≈ "Class" ≈ "Type" ≈ "Category" ≈ "Entity" ≈ "Role";
- "Instance" ≈ "Individual" ≈ "Object";
- "Property" ≈ "Relation" ≈ "Association" ≈ "Slot" ≈ "Attribute".

In the present work, we will use the terms *concept* for referring to the type of objects in the world, *relation* to denote an association between two concepts, *attribute* refers to the characteristics of the concepts that can be valued, and *instance* is the occurrence of a concept. Also, we will use *is-a* to denote the subsumption relation that creates hierarchies of concepts (also known as *subClassOf* relation). The relation between a concept and the object that is an occurrence of that concept is called *instance-of*.

1.2 Ontology languages

There are many forms of specifications that different people term *ontologies*. What distinguishes different approaches is the degree and manner of formalizing terms. Different formalization levels gives rise to a continuum of kinds of ontologies (Uschold and Gruninger, 2004), illustrated on one-dimension diagram of Figure 2.2.

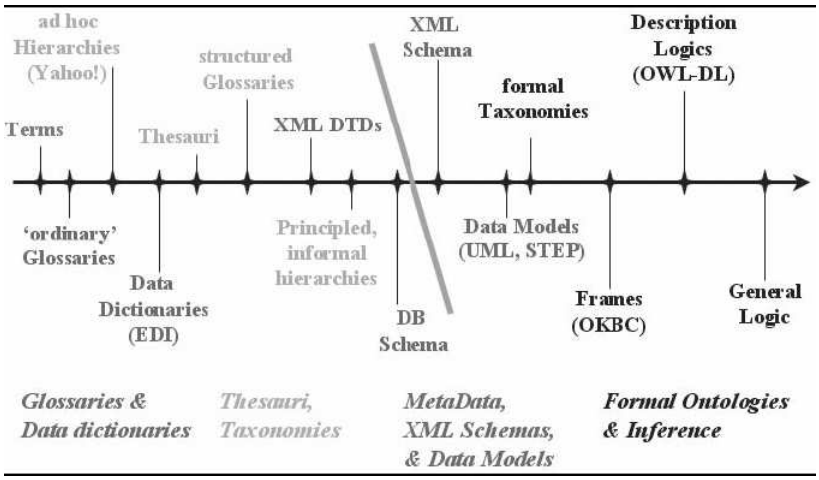


Figure 2.2: Spectrum of kinds of ontologies

The horizontal axis stands for the variety of entities formalized by the related approach. At one extremity of the axis, we have glossaries and data dictionaries, which only present definition for terms, with little or no specification of relationships among them. After, we have taxonomies of terms, which present no other semantic relation than the *is-a* relation. Despite that, these ontologies can already provide enough semantics for some applications, such as navigation support. At the other end of the spectrum, we have very formalized theories, which can be useful for the *AI community*, for representing the static knowledge about the domain on top of which inferences have to be executed. As we move along the continuum, the ambiguity is reduced and the degree of formalization increases. Ontologies in the extreme right of the axis are more suitable for being processed by automated reasoning engines. With respect to *database systems*, ontologies have the function of specifying a data modeling representation at a level of abstraction above specific database designs (Gruber, 2008). They are at some point of the spectrum, in the middle way between strong formalization and strong expressiveness, where *structure* is important. The focus of the various approaches shown in the spectrum can be completely different. For example, ontologies developed for natural language processing seems to be nearly useless for database integration, and conversely (Pierra, 2003).

Lately, we have been seeing an exponential growth of ontologies for *Semantic Web* applications. The current globally accepted ontology standards have a direct application to the *Semantic Web* . Researchers discuss, however, whether these ontologies are suitable for being used in domains which are essentially data-centric and which thus require more structured ontologies and do not rely upon inference and deduction, such as technical/industrial fields. We describe in the next sections two different *visions* in the ontology research area: the development of ontologies for the cutting edge area of *Semantic Web* and the development of ontologies that can be used as conceptual schemas to data-centric applications.

1.2.1 Ontologies for the *Semantic Web*

The exponential growth of the resources in the World Wide Web has increased the availability of electronic information, but it has made information more difficult to find and organize information. The so-called *Semantic Web* (Berners-Lee, Hendler, and Lassila, 2001) aims to overcome this problem by providing machine-readable semantic descriptions to Web resources.

> "The *Semantic Web* provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. [Currently,] we don't have a web of data. Because data are controlled by applications, and each application keeps it to itself".

The above definition is given by the *Semantic Web* activity within the W3C (World Wide Web Consortium), [5] which is the group responsible for determining standards of the Web. With the current popularity of the Internet as communication medium, most of the information is shared on the Web, but it is not described in an *intelligent* way. The consequence is that huge time is lost in human work for extracting relevant information from the useless. In order to fill this gap, researchers have been proposing technologies for providing descriptions that complement the content of Web documents in a machine-readable format. As a result, the information added with a semantic markup is involved in a context that enables

[5] http://www.w3.org/

the machine to have the correct interpretation about it. This allows to obtain more meaningful results from researches.

An important assumption of the *Semantic Web* is that resources are uniquely identified by URIs (Uniform Resource Identifiers). URIs are defined in (Berners-Lee et al., 1998) as "compact strings of characters for identifying an abstract or physical resource". URIs are meant to uniquely identify all kinds of resources on the web, which may be anything, a person, an institution, the relation that a person has with an institution, and also an electronic document, a service, a Web page, an entire collection of pages. Almost every *Semantic Web* standard that is built makes an ubiquitous use of URIs. This is fundamental to keep the idea of the Web as an information space rather than a computing program.

1.2.1.1 Resource Description Framework

The Resource Description Framework (**RDF**, (Klyne and Carroll, 2004)) was the first W3C recommendation for describing information that is implemented in web resources. RDF provides a basic syntax whose building block is a *triple*. A triple is a statement of the form *[subject, predicate, object]*, where *predicate* (also called property) denotes a relationship between *subject* and *object*. For example, in the following triple

```
[http://www.w3.org/People/Berners-Lee/, hasAuthor, "Tim Berners-Lee"]
```

the subject is the URI *http://www.w3.org/People/Berners-Lee/*, the predicate is the property *hasAuthor* and the object is the literal "Tim Berners-Lee". This triple states that Tim Berners-Lee is the author of the cited web page. RDF-based Ontologies are generally serialized in XML (eXtensible Markup Language) using the full RDF syntax. In this work, however, when describing RDF-based statements, we will adopt the Turtle syntax (Beckett and Berners-Lee, 2008), a concise RDF serialization alternative to RDF/XML, for the sake of readability. The previous example is serialized as follows:

```
:hasAuthor rdf:type rdf:Property .
:http://www.w3.org/People/Berners-Lee/ rdf:type rdf:Description .
:http://www.w3.org/People/Berners-Lee/ :hasAuthor "Tim Berners-Lee" .
```

Although the RDF data model provides a simple way for describing resources, it doesn't allow defining *classes* of resources. Those are defined using RDF Schema, as it will be explained in this section.

1.2.1.2 RDF named graphs

The subject of one RDF statement (triple) may be the object of another RDF statement. A set of linked statements forms a graph. RDF proposes a reification mechanism which allows one to objectify (i.e. assign a URI to) a single statement and use it as resource in other triples. RDF graphs assigned to an URI are called **named graphs** (Carroll et al., 2005). An example of named graph is shown below.

```
:G1 { :http://www.w3.org/People/Berners-Lee/ :hasAuthor "Tim Berners-Lee" .
      :http://www.w3.org/People/Berners-Lee/ :creationDate 2009-06-10 .
      :http://www.w3.org/People/Berners-Lee/ :language "English" . }

:G2 { :G1 :date "2009-07-10" .
      :G1 :creator "Laura M." . }
```

All triples originating from a specific source (Berners-Lee's home page) are grouped under a named graph which is associated with an URI (*G1*). It is possible, then, to assert other statements about the named graph by creating triples with the graph URI as subject, such as `[G1, creator, "Laura M."]`, which defines a creator for the graph *G1*. A second graph can be created (*G2*) that group statements that refer to the first graph, and so on. This particularity of RDF is useful in application domains like provenance tracking, versioning, access control and signing RDF. However, this approach allows an infinite number of different ways of modeling some domain and the inference on named graphs can be non-deterministic (that is, at each node, in order to move forward to the next step of inference, the inference mechanism needs to make a choice between various possible nodes).

1.2.1.3 RDF Schema RDF Schema (**RDFS**, (Brickley and Guha, 2004)) semantically extends RDF enabling to create *classes of resources*. It provides means to describe domain specific vocabularies in RDF. Figure 2.3 shows an UML diagram that represents only the basic constructs of RDFS model.[6]

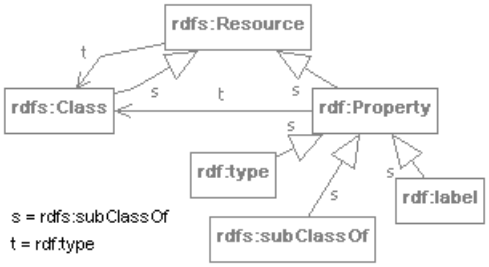


Figure 2.3: Simplified RDFS model

The following example shows how to define concepts, such as Person, using the RDFS construct *rdfs:Class* (line 1). With the *rdfs:subClassOf* property, it is possible to create hierarchies of classes (line 4), and later on, to define an instance of an RDFS class (line 6). Properties representation was also extended. We can specify the domain and range of a property, e.g. the domain of the property *hasAuthor* is *WebPage* and its range is the concept *Author* (line 9).[7]

```
1  :Person rdf:type rdfs:Class .
2
3  :Author rdf:type rdfs:Class ;
4    rdfs:subClassOf :Person .
5
6  :_BernersLee rdf:type :Author .
7
8  :hasAuthor rdfs:domain :WebPage ;
```

[6]UML (Unified Modeling Language, (OMG, 2008)) is a standardized general-purpose modeling language that includes a set of graphical notations for representing class-diagrams (*cf.* Appendix A).

[7]In order to express the validity context of ontology properties, the OWL language borrowed terms from the study of mathematical functions, such as *domain* and *range*. The *domain* are the concepts for which a property is defined, and the *range* are the concepts into which the results of a property are constrained to fall. Despite the fact that the correct nomenclature for *range* would be *codomain* for the intended meaning, in this work we will also refer to the *domain* and the *range* of some property.

```
9           rdfs:range :Author .
```

All these extensions increase the expressivity of the representation when compared to the RDF model. However, RDFS lacks some representation possibilities such as cardinality constraints, logical operations over classes, negation, properties of relations, among others. These possibilities have been specified by the OWL language.

1.2.1.4 Web Ontology Language Web Ontology Language (**OWL**, (Harmelen and van, 2004)) semantically extends RDFS and offers a rich set of modeling constructors. OWL was sanctioned by the W3C as the *standard ontology language* for the Web. There are three "species" of OWL ontologies, depending on the constructs they employ.

- **OWL Lite** is used for representing ontologies that basically need a classification hierarchy and simple constraints (no more than 0 or 1 cardinality values for properties). Tool support for OWL Lite ontologies is easy to be provided.
- **OWL Full** is used in the other extreme, for representing ontologies that require maximum expressiveness from constructs and which do not care for computational guarantees or decidability. A class can be simultaneously treated as a collection of individuals and as an individual itself.
- **OWL DL** is finally used for representing ontologies that make use of all OWL language constructs, but under some restrictions (for example, a class cannot be an instance of another class, like in OWL Full ontologies). OWL DL is equivalent to a description logics[8] in terms of representation power.

For ontologies that fall into the scope of OWL-DL, we can use a reasoner to infer information that is not explicitly represented, by performing some verifications such as subsumption, equivalence, consistency and instantiation testing. Classes and Properties are the basic building blocks of the OWL language. But OWL proposes an extensive set of constructs, with which is possible to create meaningful assertions about concepts of the ontology. To cite some, OWL allows to combine two or more classes with intersection or union operators, make quantifier or cardinality restrictions, or state algebric properties (inverse, symmetric, transitive) about the ontology properties.

The following example shows the concepts Man and Woman defined as *owl:Class*es (lines 6 and 10). They are subclasses of Person (line 7) and they are also disjoint (line 8), that means, they have no common instances. It is also possible to declare that two concepts are equivalent, such as Person and Human in line 4.

```
1  :Person rdf:type owl:Class .
2
3  :Human rdf:type owl:Class ;
4    owl:equivalentClass :Person .
5
6  :Man rdf:type owl:Class ;
7    rdfs:subClassOf :Person ;
8    owl:disjointWith :Woman .
```

[8] Description logics is a family of representation formalisms that resembles to first-order logics (see: http://www.dl.kr.org/).

```
9
10  :Woman rdf:type owl:Class ;
11    rdfs:subClassOf :Person .
```

Besides the languages presented in this section, other languages have also been used for supporting semantic markup, but they have been already deprecated.

1.2.2 Ontologies for data intensive applications

Ontologies have been seen lately as a synonym for *data models*, despite the fact that the two approaches are significantly different. Data models (or data schemas) represent the structure and integrity of the data elements for an application that is going to be developed (Spyns et al., 2002) and are originally used for describing information systems and databases. Generally, in data models, the only data that are described are those that are relevant for the envisaged goal. The data schema is optimized for the target application and data must respect the definitions and *constraints* defined in the conceptual schema. This last item (constraints) is often a source of misunderstanding concerning the use of ontologies as data models.

In data models for relational databases, constraints over the schema (*integrity constraints*) are usually interpreted as checks used to verify whether the information satisfies certain conditions. OWL axioms may look like integrity constraints, but they are interpreted under first-order semantics and not as checks. An example taken from (Motik et al., 2009) describes an application in which each person is required to have a social security number. In a relational database, this requirement would be represented as a *null rule* defined over the social security column, disallowing inserts or updates of rows containing a null (the absence of a value) in that column. In OWL, it is possible to express a similar statement, using a cardinality restriction:

```
:Person rdf:type owl:Class ;
[ rdf:type owl:Restriction ;
  owl:onProperty :hasSSN ;
  owl:cardinality "1" ].
```

However, the OWL representation results in a different behavior: if the instance of Person does not present a value for the SSN attribute, it is only considered to have an unknown number, it does not raise an error. Motik, Horrocks, and Sattler (2009) are currently working on the proposition of an extension of OWL with integrity constraints.

Some kinds of domains present characteristics that are not easily implemented by *Semantic Web* languages. Engineering domains, for example, require specific validations that include numerical comparison (the internal diameter of some piece is smaller than its external diameter) or operations (such as currency conversion). The *Semantic Web* languages have a poor, not to say absent, representation of numeric expressions, since they are not meant to deal with these kinds of constraints.

Semantic Web standards are expected to markup existing resources providing enough formalization to the Web data so that machines are able to execute reasoning and inferences. On the contrary, data models are designed to provide a structured repository to huge quantity of data that is to be created. They rely on strong typing of the described objects in order to define a relevant and optimal schema for storing and

retrieving data.

In practice, the differences in the goals of *Semantic Web* and data-intensive systems lead to differences also in modeling style (Wang et al., 2006). A developer of ontologies for the *Semantic Web* thinks in terms of necessary and sufficient conditions to define a concept. For developing a data models, on the other hand, the problem is addressed from another angle: one should decide what are the implications of being a member of some concept. We present here an ontology model that is oriented towards a data-centric characterization of the domains, the PLIB model.

1.2.2.1 The PLIB model The PLIB model (detailed in Pierra (2003)) was initially conceived as an approach for exchanging and integrating automatically engineering component databases. PLIB has given place to a set of norms ISO in the *Parts Library* series (ISO 13584). Various domain ontologies following that model were developed, the first one describing the main categories of electrical components (IEC 61360-4 :1998). The whole set of PLIB ontologies can be found at the PLIB home page.[9]

The PLIB ontology model does not focus at all on the equivalence between concepts. It offers, conversely, features that allow to precisely define concepts in order to provide non redundant (canonic) vocabulary. PLIB is said to be oriented to technical domains, since it supports most functionalities currently used in engineering (Dehainsala et al., 2007): A property value may depend from its evaluation context, thus a property may be a function; the property value may be associated with a measure unit; an object must be characterized by one single class. PLIB model is also particularly fit to express numerical properties and integrity constraints.

A PLIB ontology allows the description of concepts, relations, attributes, domain types and instances. In order to be able to refer to concepts in a no-ambiguous and multilingual fashion, each ontology element is associated to an unique identifier, called BSU code (Basic Semantic Unit). Figure 2.4 shows a UML (Unified Modeling Language) diagram that represents a simplified model of PLIB. Concepts are represented by the primitive *Class*, which is identified by a BSU class. Both relations and attributes are represented by the primitive *Property_DET*. Relations are properties whose domain is another concept, and attributes are properties that have a data type as domain (the primitive *DataType* represent both types of domains). A *Class* is characterized by a list of *Property_BSU*, each one uniquely identifying a *Property_DET*. Each *Class* can have a super-class.

The PLIB model implements two types of subsumption between classes: one is the classical inheritance relation and the other is an *import* relation. The inheritance relation (*is-a*) is inversely represented in PLIB by the *is-superclass* relation between classes. Stating that *A is-superclass B* is the same as saying that *B is-a A*. The more specific concept (*B*) inherits all properties that characterizes the super concept. The *its-superclass* relation organizes the concepts of PLIB in hierarchy of classes.

The definition of an import relation has to do with the goal of PLIB model at the development of vast ontologies (which cover all the technical field). To this end, PLIB offers mechanisms for achieving *modularity of ontologies*. This mechanism interfaces an ontology to another ontology, relating a class to a *case-of superclass* and imports the properties of the latter one into the first one. This operator is called *is-case-of* : it defines a subsumption relation that is not associated to inheritance of properties. The class

[9] http://www.plib.ensma.fr/

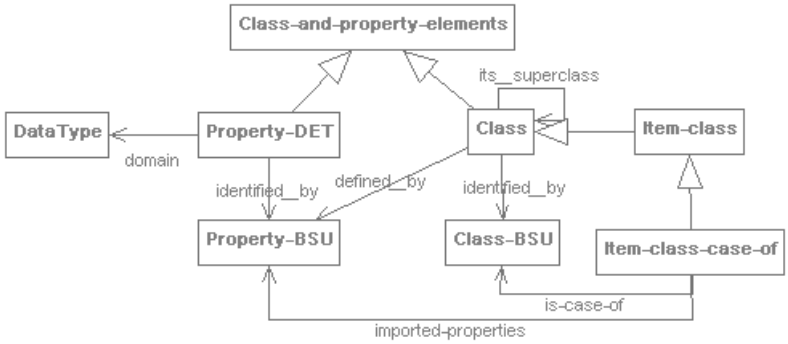


Figure 2.4: Simplified PLIB Model

declaring itself *a case of* another, must explicitly import the needed properties from the class of which it is a case. In particular, it allows to import properties without needing to duplicate class or property definitions. This difference gives a higher degree of independence between ontologies from different sources, and that have different life cycles. For this reason, the *is-case-of* relation is employed in PLIB models to make the articulation of one ontology to the other.

In the UML diagram of Figure 2.4, the primitive *Item_class_case_of* is *a class that is case of* another class. The class of which it is a case, is defined by the attribute *is_case_of* (which makes a reference to *Class_BSU*), and the list of properties that are imported from that class is defined by the attribute *imported_properties* (which makes a reference to *Property_BSU*). The following example presents the creation of an *is-case-of* relation using the PLIB model.

Considering a concept which is case of another concept, we call the first one as *subsumed concept* and the second one *subsuming concept*. There are two approaches for setting up *is-case-of* relation:

1. **the *a priori* approach**, in which the relation is created during the *ontology design time*. An ontology concept is created as being *case of* another concept. The subsumed concept explicitly choses the properties to be imported from the subsuming concept.
2. **the *a posteriori* approach**, in which the relation is created subsequently to the design of the ontology. After the creation of the ontology concepts, an *is-case-of* relation is created between two concepts. Some properties of the subsumed concept (which already exist) are *mapped* to the chosen properties of the subsuming concept.

The *is-a* relation is a typical *a priori* relation, i.e., it is defined at the design time and is also embedded in the definition language. The *is-case-of* relation, on the contrary, can be used as an auxiliary subsumption relation, defined in *a posteriori* fashion.

Example. Lets consider the Person ontology as a shared ontology among different domains. In order to define an ontology for the domain of a University, the concepts Student and Professor need to be created. The Person ontology is to be re-used, and the concept Person itself is the better choice for subsuming the concepts Student and Professor. However, some basic properties of the concept Person may not be interesting for the University ontology, such as the relation *marriedTo*. In this case, we are able to import

just the properties that are important for the domain being represented. The following code (represented using EXPRESS language[10]) shows the use of the import operation between PLIB ontologies.

```
ENTITY Student ;
  is_case_of : Person;
  imported_properties : firstname, lastname, age;
  registrationID : INTEGER;
END_ENTITY ;
```

The Student concept is declared to be *a case of* the concept Person by means of the *is-case-of* relation. Some of the properties of Person are imported (firstname, lastname and age), by means of the *imported_properties* attribute. The other properties of Person are not copied into the concept Student, which can define its own properties, such as *registrationID*, whose domain type is an integer type. This is an example of *a priori case-of* , since the *is-case-of* relation was defined during the creation of the concept Student.

The ontologies produced when applying the PLIB model intend to enable data characterization and exchange, and not deduction. In contrast with *Semantic Web* languages such as OWL, which support the description logics representation paradigm, the PLIB model is more likely to support the *Frames paradigm*. A frame is a knowledge representation formalism that provides a concise structured representation of an object or a class of objects and of their characteristics and relations (Fikes and Kehler, 1985). The major differences between Frames and OWL are described in the work of Wang et al. (2006). We list as follows some features of the PLIB model which are alike those of Frames (next to a description of the same feature in the RDF/OWL model).

- Properties must be explicitly attached to a concept at the broadest context where they have a precise meaning. In RDF/OWL, properties can be used with any class or individual.
- Strong domain types are assigned to attributes. RDF/OWL specify types of data using a simplification of concrete domains.
- Nothing can be entered into a PLIB model until there is a place defined for it (closed world). Theoretically anything can be entered as an RDF/OWL instance (open world).

The comparison presented here between PLIB and OWL models is just a small part of a ample discussion about the differences between ontological and data-centric models. Even though triple-based databases are flexible and portable, databases are still recognized as the best technology for an efficient management of very large quantities of data. In the next section, we present *ontology-based databases*, which are current proposals for guaranteeing the efficiency of the representation while keeping also the richness of an ontology.

1.3 Ontology-based databases

An ontology-based database (OBDB) is a data source that contains ontologies, a set of data, and links between data and the ontological elements (Dehainsala et al., 2007). Many OBDB have been proposed

[10]EXPRESS is part of the STEP standard (ISO 10303), and is in widespread use to define data models for large-scale industrial applications, including manufacturing, engineering, defense, oil rigs, processing plants (Schenck and Wilson, 1994).

lately, including: Sesame (Broekstra et al., 2002), RDF Suite (Alexaki et al., 2001), Jena (Wilkinson et al., 2003), OntoDB (Dehainsala et al., 2007), 3Store (Harris and Gibbins, 2003), OntoMS (Park et al., 2007) among many others. According to Jean (2007), they differ concerning some criteria :

- the supported ontology model;
- the database schema employed to store *ontologies*;
- the database schema employed to store *instances*;
- the mechanism employed to define links between data and ontologies.

Fankam et al. (2008) propose three classifications of OBDBs, using the above mentioned criteria.

1.3.1 Type 1 OBDBs

Type 1 OBDBs represent information in a schema composed of a unique triple table with three columns which respectively represent the subject, the predicate and the object *(subject, predicate, object)*, pretty much like RDF triples. This triple-based schema is used to describe both ontologies and instances, and is completely independent from the structure of the domain ontologies to be stored, since all will be stored in a same generic triplet structure. Figure 2.5 presents an extract of the triple table that stores the information related to the Person ontology.

TRIPLES		
Subject	**Predicate**	**Object**
Person	rdf:type	rdfs:Class
Student	rdf:type	rdfs:Class
Student	rdfs:subClassOf	Person
student#1	rdf:type	Student
student#1	firstname	"John"

Ontology

Instances

Figure 2.5: Type 1 OBDB: triple table

Both the structure of the ontology (*Student, rdfs:subClassOf, Person*) and the instances (*student#1, firstname, "John"*) are represented as triples. This type of representation is particularly adopted by Jena and 3Store OBDBs.

1.3.2 Type 2 OBDBs

Type 2 OBDBs store ontology descriptions and instance data in two distinct schemas. Contrary to Type 1 OBDBs, the schema for storing the ontologies *depend* on the ontology model (RDF, OWL, PLIB), as a consequence, the ontology primitives are represented as tables (*table per class* representations). For storing instances, some alternatives have been proposed. Instances can be stored as *triples*, just like Type 1 OBDBs (Jena and 3Store). Another option is to represent the instances identification in an *unary table*, and the values of their properties in a *binary table*. This approach is followed by Sesame and RDF Suite. Figure 2.6 presents an example of type 2 OBDBs.

Class	
ID	Name
1	Person
2	Student

Student
ID
student#1
student#2

firstname	
ID	value
student#1	"John"
student#2	"Paul"

Figure 2.6: Type 2 OBDB: one table per primitive and separated instances

In this example, ontology classes are stored using a *table per class* schema (*Student*), and instance data are represented using a binary representation that links to the table *Student* via the value of the *ID* column.

1.3.3 Type 3 OBDBs

The OntoMS and OntoDB OBDBs propose a representation that depends on the *structure of the domain ontology*. Each domain concept is represented as a table, which has one column for each used property, and the instances of this concept and its property values are stored as lines in the concept table. For multivalued properties, its values can be represented either by using the tables types introduced by SQL 99 (approach followed by OntoDB), or by creating new tables for these properties (approach followed by OntoMS). Figure 2.7 illustrates the Type 3 representation.

Student	
ID	firstname
student#1	"John"
student#2	"Paul"

Figure 2.7: Type 3 OBDB: one table per class and integrated instances

The table *Student* has one column to represent the instance ID and one column to the property *firstname*. All the instances that have values for this property are represented together in this table.

Both in Type 1 and 2 approaches, instances are not organized following the structure proposed by the domain ontology (the way concepts are organized and the relations between them). The notion of data schema as in traditional databases is not present. Their choice of decomposing the property values and storing them separately from the instance identifier tries to reflect the flexibility of representation of languages such as RDFS and OWL. The instances of those languages are not strongly typed. Nevertheless, in the case of applications that need to build their databases taking as reference domain ontologies, these ontologies need to reflect a relational structure and follow the strong-typing assumptions. That is the proposal of the Type 3 ontology-based databases.

1.4 Ontology-based exploitation languages

In order to query the content of ontology-based databases, traditional query languages (such as SQL[11]) are inappropriate, since they do not take into consideration ontological descriptions (synonymous names, comments, illustrations) for each table and for the related properties (Jean et al., 2005). We will present in this section some of the query languages that are suitable for handling ontology-based content. In order to provide examples to the query languages, we will increase the Person ontology with instances of Car (*_car1*, that has red color) and Person (*_paul*, who owns *_car1*), as expressed in the following code:

```
:_car1 rdf:type :Car ;
     :hascolor "Red" .

:_paul rdf:type :Person ;
    :owns :_car1 .
```

1.4.1 OWL Query language

OWL Query Language (OWL-QL) is a candidate standard language and protocol for querying *Semantic Web* resources represented in OWL language (Fikes et al., 2004). An OWL-QL query pattern contains an union of OWL statements (a conjunction) in which some URI references or literals have been replaced by variables. OWL-QL enables clients to designate which variable or set of variables must be bound to the query pattern: *must-bind*, *may-bind*, and *do-not-bind* variables, which are disjoint sets each with the others. By adjusting the variable for binding, OWL-QL can answer questions such as "What resources make the query pattern true" or "Is the query pattern true". This is quite flexible and allows for sophisticated queries. As an example, the query pattern for the question "Who owns a red car?" would be expressed by the code (a) (in an adapted Turtle syntax). The variable *?p* is in the must-bind list.

```
owl-ql:queryPattern {
  var:c :hascolor "Red" .
  var:c rdf:type :Car .
  var:p :owns var:c .
}

owl-ql:mustBindVars {
  var:p . }
```

(a) OWL-QL query

```
owl-ql:answer {
  owl-ql:binding-set {
    var:x rdf:resource :Paul . }

  owl-ql:answerPatternInstance {
    :_car1 :hascolor "Red" .
    :_car1 rdf:type :Car .
    :_paul :owns :_car1 .
}}
```

(b) OWL-QL answer

Figure 2.8: OWL-QL patterns

The answer for an OWL-QL query contains two entries: a binding set, with the values for the must-bind list of variables, and an *answerPatternInstance* tag, which contains the query pattern with the variables filled in. The answer "Paul owns a red car" is expressed by the code (b).

[11] SQL (Structured Query Language) is a database language designed for management and retrieval of data in relational databases.

The expressive power of OWL-QL is however limited compared to other languages, such as SPARQL. Indeed, it doesn't support complete Boolean filters (negation, disjunction), set-based operations (union, intersection, difference), arithmetic operations and comparison on data values (greater then, equal) and non polymorphic queries (Jean et al., 2005).

1.4.2 SPARQL language

The SPARQL query language (Seaborne and Prud'hommeaux, 2008) is a standard for posing queries over RDF-based repositories (it includes OWL ontologies). The data model of SPARQL is based on RDF triples and its syntax is similar to SQL. A SPARQL triple can include variables to indicate data items that will be returned by a query. SPARQL supports disjunction in the query and also provides optional variable binding.

The following query searches all instances that are linked by the property *owns* to an instance of the class *Car* whose value of property *hascolor* is *"Red"*. More complex expressions are possible to be written, since SPARQL is also able to query RDF graphs.

```
SELECT ?person FROM <Person.owl>
{ ?person :owns ?car .
  ?car rdf:type Car .
  ?car :hascolor "Red" }
```

The problem of SPARQL is that it is dependant on the RDF triple model to be interpreted. Database back-ends that provide SPARQL engines over their data (such as Jena) first translate the data to RDF triples representation, then, queries are evaluated mostly in the RDF part of the engine, rather than in the underlying database. This misses an opportunity to let the database perform much of the work using its built-in query optimizer (Harris and Shadbolt, 2005).

1.4.3 OntoQL language

OntoQL is a language proposed by Jean et al. (2006b) to exploit the OntoDB database. OntoQL is not only a query language, but also a data definition (DDL) and data manipulation (DML) language. This illustrates the biggest difference between OntoQL and other ontology-based query languages. OntoQL allows creating, altering and dropping concepts of ontologies (classes, properties) as well as definition fields of these concepts (name, definition).

The following expression creates an ontology concept named *Student* as a subclass of *Person*, whose properties it inherits. The *PROPERTIES* clause allows to define the attributes *registrationID* and *registeredIn*, which links the concept *Student* to the concept *University*.

```
CREATE #CLASS Student EXTENDS Person
PROPERTIES(registrationID Integer, registeredIn University);
```

The query language part of OntoQL is designed as an extension of SQL to query ontologies, their contents, or both ontologies and contents stored in an OBDB. An example of query that retrieves content is given in code (a). It searches the instances of Person that own a red car. In addition to that, *ontology*

queries allows to retrieve descriptive information about the structure of the ontology. Code (b) searches for the English name of the class which French name is *"Université"*.

```
SELECT * FROM Person
JOIN Car
ON Person.owns = Car.ID
WHERE Car.hascolor = "Red" ;
```

(a) OntoQL query on content.

```
SELECT #name[EN]
FROM #Class
WHERE #name[FR]="Université" ;
```

(b) OntoQL query on ontology.

Figure 2.9: OntoQL queries

The result of an OntoQL query is, as in the relational model, tuples containing the values that returned from query processing. OntoQL differs from the semantic web languages in the sense that it originates from databases approaches. However, this is also its weak point, since OntoQL may not be able to fully address features of ontologies that are build in an RDF/OWL paradigm, such as OWL's notion of equivalence between concepts.

2 Semantic annotation approaches

Annotation can be defined as the process of adding comments or making notes on something. Such notes, when they can be retrieved by other persons, are a means of disseminating useful information. The purpose of annotating can be: explaining, interpreting, giving opinion, or describing some resource.

Buneman et al. (2005) say that in every area of science, much investigation depends on databases in which experimental evidence has been stored. This evidence is typically some form of interpretation of the data, and annotation is an increasingly important part of that interpretation.

> "In the scientific community, the focus of annotation is on *description or interpretation* from trusted sources that may inform further interpretation or research, possibly performed by others in other organizations or outside the community."
>
> (Buneman, Bose, and Ecklund, 2005)

The *Semantic Web* (introduced in Section 1.2.1) depends essentially on the easy creation, integration and use of semantically described data. The process of attaching semantic descriptions to Web resources is called *semantic annotation* or *ontology-based* annotation. The general process of semantic annotation involves linking a Web page or some elements inside it to the ontology concepts that better express the *meaning* of the resource. These ontologies must be defined in a Web-enabled ontology language (e.g. OWL, RDFS).

For those involved with the *Semantic Web* , the goal of annotating a web resource is to specify machine-processable meaning for it (Zuo and Zhou, 2003). This goal can be generalized for other communities than the *Semantic Web* . For those working with digital images, annotating an image often means identifying a section of the image to comment upon, providing geolocation or simply a caption. Annotating a

film clip means attaching text or audio descriptions, facts or interpretation to different temporal sequences of video.

What is more important is that the semantic annotations enable many new types of applications: highlighting, indexing and retrieval, categorization, generation of more advanced metadata, smooth traversal between unstructured text and available relevant knowledge (Kiryakov et al., 2004).

2.1 Resources to annotate

When talking about annotating resources, it is important to differentiate the various types of resources:

- **Unstructured data.** According to Blumberg and Atre (2003), the term *unstructured* refers to the fact that no identifiable structure within this kind of data is available. In textual documents, a word is simply a word, there is no data type definition. A limitation of this kind of data is that no controlled navigation within unstructured content is possible. Examples for unstructured data are documents in a file folder, videos, images, e-mail files, word-processing text documents, post-script documents, slide show presentations, JPEG and GIF image files, and MPEG video files.

- **Semi-structured data.** They are data that do not neatly fit into the relational model (Sperberg-McQueen, 2005). There follows an overall implicit structure, a loose hierarchical representation of the data, but may have some irregular structures. The primary source of semi-structured data is not XML, but rather existing reports and business documents. XML just provides a natural representation for semi-structured information. In the case when unstructured data provide information such as author and time of creation, this information can be easily stored in a relational database management system. This means that while image or video data cannot fit neatly into structure, its metadata can.

- **Structured data.** They are data that follow a predefined schema. The schema formally defines the type and structure of data and its relations. We can recognize two types of structured data:

    - Implicitly structured data. The inherent structure needs to be inferred from patterns, such as text position. One example are files that organize the different types of information in different columns. The software tools that use these types of files must know the way data are organized so as to exploit the information.

    - Explicitly structured data. Most of the data that are used by engineering tools are organized according to an explicit data model. The typical example are data inside a relational database system, but also proprietary file formats and programming code.

Semantic annotation is likely to be applicable to any sort of resource: from non-structured (textual web pages, regular documents, images, videos), to structured (databases). However, from a comparative analysis of several semantic annotation projects, available in the survey of Uren et al. (2006), we understand that the most significant works in semantic annotation concern the annotation of semi-structured and

unstructured documents, such as HTML[12] pages and XML[13] documents. This approach has limited usefulness for companies that keep their knowledge in documents of different formats, including word processor files, spreadsheets, graphics files and complex mixtures of different formats.

Furthermore, most of those works do not propose a solution for models issued from different fields, such as *engineering models*. The closer one can get of annotation of computer-based models are works that intend to annotate parts of programming code and proposals that add semantic annotation to databases.

2.2 Semantic annotation tools and frameworks

Due to our interest in works that converge to some computer-model annotation approach, this section will be divided in two parts: (i) proposals of semantic annotation that consider the classical semi-structured and unstructured resources to be annotated, such as Web pages, multimedia resources, and also e-mails and web-services; and (ii) works that aim to annotate structured resources. More on annotation tools can be found in the Annotations at *Semantic Web* site,[14] which gathers information about the topic of *annotation and authoring for the Semantic Web* .

2.2.1 Semantic annotation of semi-structured and unstructured resources

Uren et al. (2006) differentiate general *frameworks for annotation*, which could be implemented differently by different tools, from specific *annotation tools*, which can produce annotations that reference an ontology.

The W3C project *Annotea* (Kahan et al., 2002), and the *CREAM project* (Handschuh et al., 2001) developed at the University of Karlsruhe, are general frameworks for annotation.

The target of Annotea's annotations are documents in web-native formats such as XML and HTML. The annotation structure is defined in a RDF-based schema, and annotation is stored in annotation servers as metadata over the document. XPointer is used for locating the annotations in the annotated document.[15] CREAM framework annotates Web pages, but also considers the possibility of annotating the databases from which web pages are generated (the *deep web*[16]). To this end, they use the term *relational metadata* to denote annotations that contain relationship instances. Both Annotea and CREAM use XPointers to locate annotations, which restricts the annotation to web-native formats such as XML and HTML.

There are basically two categories of annotation tools: the tools that support *manual* annotation and the tools that provide *automatic* annotation. Manual tools are similar to purely textual annotation tools but provide some support for ontologies. Automatic tools are those that use rules or wrappers written by hand that try to capture known patterns for the annotations; or that learn how to annotate (supervised or

[12] HTML (HyperText Markup Language) consist of a set of tags that provides structure and format to web pages.

[13] XML (Extensible Markup Language) is a specification for creating *custom* markup languages for adding meta-information to text documents.

[14] http://annotation.semanticweb.org/

[15] XPointer (XML Pointer Language, (Grosso et al., 2003)) is a W3C recommendation for addressing components of an XML based schema. XPointer allows a link to point to specific parts of an XML document, based in the structure of the document.

[16] Deep web refers to WWW content that is not indexed by standard search engines.

not) (Uren et al., 2006).

SHOE Knowledge Annotator (Heflin and Hendler, 2000) was one the earliest systems for manually adding semantic annotations to web pages, but it did not have a browser to display the Web pages. Using Amaya (Quint and Vatton, 1997), on the contrary, the user can annotate Web documents (it produces Annotea RDF mark-up) in the same tool they use for browsing.

Several works were also proposed concerning the annotation of multimedia content. M-OntoMat Annotizer (Bloehdorn et al., 2005) is built over CREAM framework and supports manual annotation of image and video, using low level features that describe the contents of the visual objects. Vannotea (Schroeter et al., 2003) is a prototype which supports the annotation of audiovisual documents by extending the W3C RDF-based annotation model Annotea.

One not very common application is to annotate *mappings*. The work of Zhdanova and Shvaiko (2006) declare that the [manual] matching of two or more heterogeneous ontologies is often subjective, depending on the application. They present the term "annotated mapping", which is defined as follows: "Annotation of a mapping element generally contains its usage-related characteristics. Repositories of *annotated mapping* elements are collections of mapping elements annotated with values corresponding to characteristics, such as provenance and tool usage".

For other type of resources: SMORE (Kalyanpur et al., 2004) allows mark-up of emails as well as images, HTML and text. More recently, concerning the annotation of Web-services, METEOR-S Web Service Annotation Framework (Patil et al., 2004) allows semi-automatic mark-up of Web service descriptions with ontologies. SAWSDL (Farrell and Lausen, 2007) is a W3C recommendation that allows description of additional semantics of WSDL (Web Services Description Language) components.

2.2.2 Semantic annotation of structured resources

Data models explicitly determine the structure of data, which in this case is known as structured data, but one same structure can have different meanings in different domains.

Some approaches exploit semantic enrichment of structured data by applying annotation to enterprise reports or business models. Diamantini and Boudjlida (2006) propose to annotate enterprise strategic data models. Enterprise reports are considered as being instances of the enterprise model and are annotated using an enterprise ontology. The work of Lin et al. (2006) proposes semantic annotation of business process templates, in order to improve the intelligent re-use of former processes in each new activity.

Annotation of databases, for example, enables semantic tags to be attached to database entities at various granularities, e.g. at the table, tuple, column, cells or the result of a query. DBNotes (Bhagwat et al., 2005) is currently a relational database system where every column of every tuple in every relation can be associated with zero or more annotations. In DBNotes, an annotation can be a comment about data such as their correctness, quality or sensitivity. MONDRIAN (Gasevic et al., 2005) introduces an annotation mechanism for relational databases that is capable of annotating both single values and associations between multiple values. Works on semantic annotation of databases represent a key research subject for domains that handle huge amount of data, such as scientific databases, for example, as well as many other database applications.

2.3 Other issues about annotation

Other issues that should be considered in the subject of semantic annotation are: how is an annotation positioned inside the resource? Which are the properties of this annotation? Which kinds of atomic elements can be marked-up? How to formulate queries over annotations?

2.3.1 Annotation positioning in resources

A Microsoft study on annotating digital documents (Brush et al., 2001) explains the challenge to save the correct position of an annotation in digital documents that are frequently modified. Some approaches can be taken to solve the problem.

- Considering that the document will not change (case of post-script documents). In such systems, annotations are positioned using very simple means, such as character offsets or page numbers plus a (x,y position).
- Limiting the places were annotations can be placed in the document. CoNotes system (Davis and Huttenlocher, 1995) requires inserting special HTML-like markup tags into a document before it can be annotated.
- Using positioning algorithms. A number of systems allow annotations to be placed anywhere within the document. In order to identify the annotated point some store a combination of annotated text and surrounding text, some store key words of the annotated text, others calculate a hash signature from the annotated text.

For annotation of HTML or XML documents, *XPointer* is normally used to indicate where an annotation should be attached to a document. This is the case for the Annotea and CREAM frameworks.

2.3.2 Model for structuring annotations

Few annotation systems provide rich contextual information for an annotation. There is no standardized model for the structure of an annotation.

The Annotea framework (Kahan et al., 2002) defines an RDF based annotation schema for describing annotations as metadata. The Annotea annotation schema proposes one *Annotation* class, which has an *annotates* property for annotating a resource, such as a Web page or even another annotation. *Annotation* objects typically include a small set of core properties, such as: author, created, modified (which are sub-properties of the Dublin Core properties *dc:creator* and *dc:date*).[17] Figure 2.10 (from the slides of the Annotea presentation[18]) illustrates the Annotea annotation schema.

Zheng (Zheng et al., 2006) looks to solve one of the main problems of collective writing tools: the impossibility of creating meta-comments. A meta-comment is a comment that has a substructure (called

[17] The Dublin Core initiative (Core, 2000) standardizes terms that define the characteristics of published resources (*cf.* Section 1.2.5 of Chapter 4).

[18] http://www.w3.org/2002/Talks/0511-annotea/

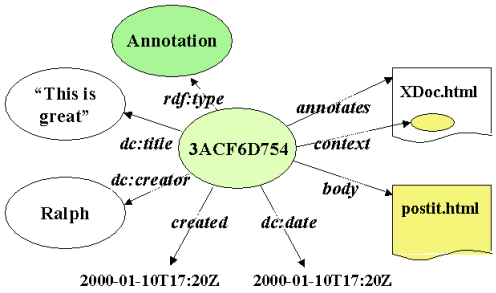


Figure 2.10: Structure of Annotea annotation schema

bundled annotation), indicating the list of annotations to which it refers, and these in turn may have anchors into the document. The work proposes an annotation data model in which every annotation has a set of mandatory and optional attributes. Mandatory attributes are the creator of the annotation, a timestamp, reviewing status (unread/read and accepted/rejected), and an anchor (the annotation's location and range relative to the document).

Works on annotation of ontologies can give some insights about how the schema of the annotation can be extended in order to annotate different types of resources.

Vrandecic et al. (2006) claim that in OWL in contrast with ontology entities, axioms cannot be annotated in any definite way. However, in order to formalize and share information like *trust*, *provenance*, or *confidence*, one must not only be able to consider entities, but also to annotate ontology axioms. (Vrandecic et al., 2006) thus propose to extend the metamodel of the OWL DL language in order to enable annotation of ontology axioms.

Providing a structure for annotation allows to state richer metadata about annotated resource.

2.3.3 Querying annotations

It is no use making annotations without providing the ability of retrieving them.

In the technical report of Buneman et al. (2005), it is said that users need to be able to query annotations for two distinct purposes: (i) to locate annotations where the annotation values themselves are of interest ("Show me all annotations made in October 2009"); and (ii) to locate annotations where the associated data values are of interest ("Show me all the annotations associated with the following data file").

iMONDRIAN (Geerts et al., 2006) offers an annotation-aware query algebra which the authors have shown to be both complete (it can express all possible queries over the class of annotated databases) and minimal (all the algebra operators are primitive). The iMONDRIAN algebra is able to answer queries that have the two possible purposes described above.

Annotea annotations can be retrieved using *Algae* RDF-based query language as well as its successor, SPARQL. Amaya client (Quint and Vatton, 1997) have chosen to optimize the query operation by adding a simple forward-chaining inference mechanism to the annotation server. This way, inference rules can be evaluated to determine the validity of some statement about the annotated resources during Algae query execution.

3 Semantic-based integration approaches

The integration of data issued from autonomous and heterogeneous sources is still a significant problem for an important number of applications. There exist, nowadays, various types of scenarios which demand integration: databases, information systems, programming codes, calendars, e-mails, enterprise catalogues, documents, and, more recently, peer-to-peer systems, and web-services. The goal of integrating data, in all these scenarios, is to provide the user an uniform access to information from different sources. But the integration will be performed differently depending on (i) the characteristics of the resources we need to integrate and (ii) on what the final user expect from the integration (unique access interface or complete merge). Bergamaschi et al. (1999) claims that the first step in an information integration process is to determine if the sources contain semantically-related information, that is, information related to the same or similar real-world concepts.

If the different data sources correspond to the same domain of interest, one possible integration problem is the *syntactic heterogeneity*, when the information are not represented using the same language. Another integration problem is the *semantic heterogeneity*. It happens when the same world entities are modeled differently by the designers of the different data sources. One example is the temperature concept, which can be represented in Fahrenheit by one system and in Celsius by another. Also, different online bookstores (such as Amazon[19] or Barnes & Noble[20]) may chose different attribute names to represent a book: writer or author, topic or subject, and so on. The objective is to identify the correspondences between the different structures. The research in database integration targets basically this kind of problem when performing *schema matching*.

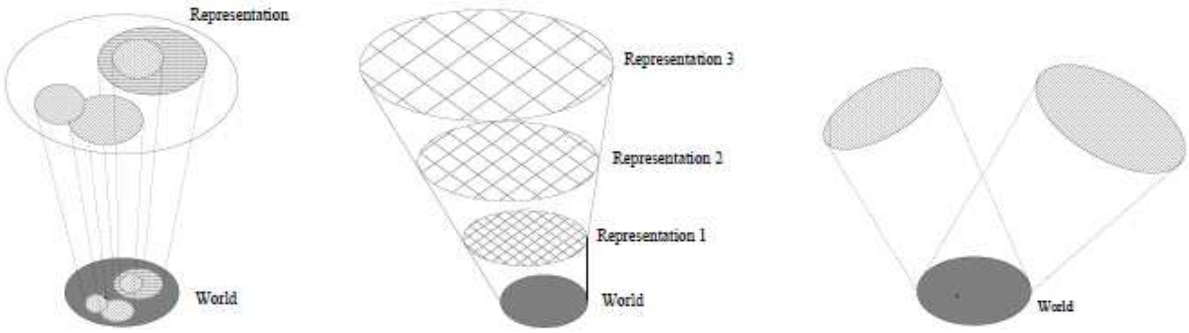
3.1 Types of heterogeneity

Considering an AI perception, the semantic heterogeneity between two representations may vary on account of the *dimension* of the representations (Benerecetti et al., 2000).

- **Partial representation** corresponds to the case when representation *covers a subset* of a more comprehensive domain of interest. In Figure 2.11a, the small circles represent different portions of the same domain of interest (the circle below). Each portion can overlap or be included in other portions. An example could be the domain of medicine, and the ontologies that represent medicine specializations.
- **Approximate representation** corresponds to the case when representation *abstracts* some aspects of a given domain of interest. In Figure 2.11b, the circles above are representations of the world at different levels of approximation (or granularity). The description of the human body, for example, can be done at the level of cells, or at the level of limbs.
- **Perspectival representation** finally corresponds to the case when representation encodes a spatio-temporal, logical, cognitive or functional *point of view* on a domain of interest (Figure 2.11c). Environmental engineering produces various models of a same area, which differ in their purposes,

[19] http://www.amazon.com/
[20] http://www.barnesandnoble.com/

for instance: soil dynamics model, vegetation dynamics model, radiation model, hydrological model, and so on.



(a) Partiality (b) Approximation (c) Perspective

Figure 2.11: Three dimensions of heterogeneity (adapted from Benerecetti et al. (2000))

Many authors stipulate that the use of ontologies for expressing the semantics of the sources is a possible approach to overcome the problem of semantic heterogeneity. When ontologies are used to describe the heterogeneous data sources, we are talking about *ontology-based integration*. On the one hand, this approach solves the problem of making explicit the semantics of data. On the other hand, it just moves the problem onward, because if we use different ontologies to describe data sources, these ontologies will probably have to be integrated themselves. In this situation, we have to deal with *ontology matching*.

Some surveys have been organized that focus on the various techniques of semantic integration. Rahm and Bernstein (2001) and Doan and Halevy (2005) review database schema matching approaches; Wache et al. (2001) and Noy (2004) focus on ontology-based approaches for information integration; while Kalfoglou and Schorlemmer (2003) focus on current state of the art in ontology matching.

3.1.1 Terminology

Inspired from the glossary proposed by Euzenat and Shvaiko (2007), we provide here a terminology with the definition of the terms that will be used in the present work. Some terms apply only to ontologies, while others are also used for databases.

Integration is the general process of providing unique vision/access to heterogeneous information. Data integration, information integration and semantic integration are considered to be synonyms in this work.

Database integration is the integration process when applied to databases.

Ontology-based integration is the integration process when applied to data sources that are described by ontologies.

Matching is the process of finding correspondences between entities of different schemas or ontologies.

Correspondence/Mapping is the relation that holds between entities (classes, instances, properties) of different schemas. The term *mapping* is defined by Euzenat and Shvaiko (2007) as the directed, oriented version of *alignment*. Other authors do not differentiate *correspondence* from *mapping*.

Alignment is the output of the matching process: a set of correspondences between the entities of two or more schemas.

Ontology merging is the creation of a new ontology from two, possibly overlapping, source ontologies. The initial ontologies remain unaltered.

We explain in further sections the main current semantic integration approaches.

3.2 Database integration approaches

The problem of database integration has been studied since the early 1980s. Database integration systems must provide the solutions to the following issues: (i) identification and specification of semantic correspondences between schemas, (ii) providing an integrated view to the information from different databases, (iii) management of the data update on local sources and of the effect on the global schema. The work of Bellatreche et al. (2006) proposes a classification of database integration systems using three orthogonal criteria.

1. **Data representation**. The middleware that gives access to the data sources is materialized or virtual.
    - *Materialized*. The data of local sources are physically centralized in a repository. The advantage is that the user queries directly the centralized data and not the original sources. On the other hand this approach requires an additional storage and maintenance cost because any modification in the local sources must be reflected on the central repository.
    - *Virtual*. The data remain in the local sources. A mediator combine data from different data sources, maintaining a global schema, mappings between the global and source schemas and handling user queries. An adapter (*wrapper*) encapsulates the data sources and translate them to a uniform language. The problems are related to how to build the integrated schemas and how to map the terms of the user query to the local schema.
2. **Direction of mapping**. The direction of the set of correspondences between the global and local schemas.
    - *Global-as-view* (GaV). The global schema is defined as a view over the local schemas. This facilitates the query reformulation by simply replacing the global predicates by their local definition. However, there is a cost for the maintenance of the global schema and the mappings when the local sources are modified.
    - *Local-as-view* (LaV). This approach supposes the existence of a global schema and it defines views over the local schemas in terms of the global schema. Queries on the global schema must be rewritten to the local sources. LaV approach allows to easily add new sources, and is easier to maintain than GaV, but has low query performance when users make complex queries.

3. **Mapping automation**. The integration process is manual, automatic or semi-automatic.
    - *Manual*. Manual strategies are found in the first generations of integration systems. The administrator must know the semantic of the schemas to be integrated and build the mediators and mappings. The heterogeneity can also be handled in the level of the query language, by the users themselves. The manual integration becomes an almost impossible task when the sources are many and often evolve.
    - *Semi-automatic and automatic*. When it is not possible to manually perform the integration, the system needs to decide if any two elements refer to the same real-world concept. To this end, a number of techniques has been developed to find *schema matching*, which base on characteristics of the data structure to infer the semantics of data.

3.2.1 Schema matching

Doan and Halevy (2005) separate schema matching techniques into two groups: rule-based and learning-based solutions. A **rule-based solution** computes similarity based on rules that exploit the characteristics of the schema, such as element names, data types, and integrity constraints. One example of a very naïve rule is: *two elements match if their names are synonyms*. Rules are a very intuitive and powerful formalism for capturing knowledge about how to match schemas. The main disadvantage of rule-based techniques is that they are not suitable to exploit data instances, which means they loose information that would aid the matching process, such as values format or words frequency.

In **learning-based solutions**, learning approaches from the AI community, like neural network and Naive Bayes, are applied to exploit both schema and data information. The key idea is that a matching system must be able to learn from the past matches, to predict successfully matches for subsequent, unseen matching scenarios (Doan and Halevy, 2005).

Rahm and Bernstein (2001) describe a largely-orthogonal classification criterion for matching techniques, which considers the nature of the features that are analyzed in order to perform the matching.

- *Instance vs. schema*: wheter the technique considers schema information or data contents.
- *Element vs. structure matching*: match can be performed for individual schema elements, such as attributes, or for complex schema structures.
- *Linguistic-based vs. constraint-based*: match can be based on names and textual descriptions of schema elements or based on keys and relationships.
- *Matching cardinality*: the technique may define a specific cardinality for each mapping relation, yielding four possible cases: *1:1*, *1:n*, *n:1*, *n:m*.
- *Auxiliary information*: matchers that rely not only on the input schemas but also on auxiliary information, such as dictionaries, previous matching decisions, and user input (semi-automatic).

Works on schema matching propose approaches based on the characteristics of the schema: similarity of the features (name, description, type, structure), linguistic contents, and so on. Thus for example, some of the most used techniques look for common substrings (e.g. <phone> and <telephone>) or for

strings with similar sound (e.g. <4U> and <for you>) or expand abbreviations (e.g. <P.O> and <Post Office>) (Giunchiglia and Shvaiko, 2003). The syntactic features of different schemas are analyzed in order to infer the semantics of the involved elements, identify elements that have the same meaning and create the mapping between them. However, even if we can get to integrate data objects from the different models, there is a semantic problem: the models do not speak with each other, because the *semantics* of data is not integrated. This is the reason why many researchers are using ontologies for describing the semantics of the information sources and to make the contents explicit. Although the nature of the schema change from databases to ontologies, some of the basic problems with respect to integration remain the same.

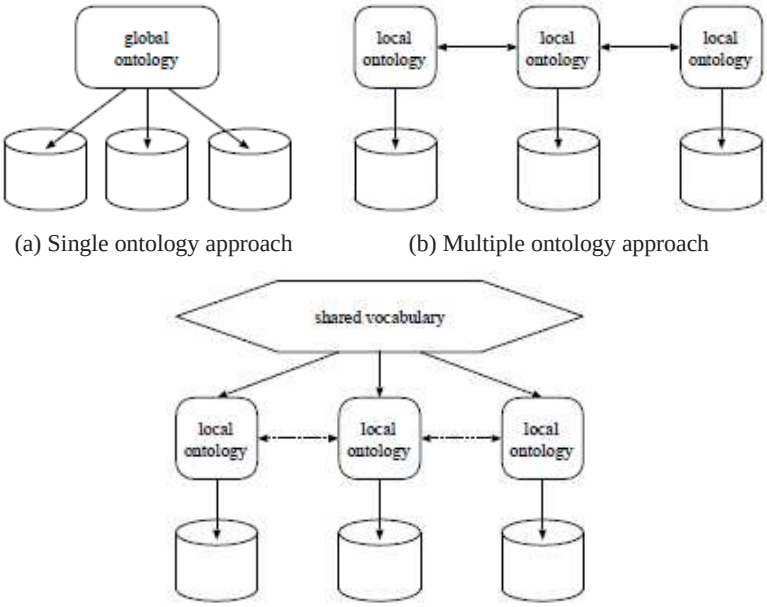
3.3 Ontology-based integration approaches

Considering the integration of heterogeneous data sources, ontologies can be used for the explicit description of the semantics of the information sources. There are, however, different ways of how ontologies can be employed. Wache et al. (2001) proposes three architectures for applying ontologies for data integration.

- In the **single ontology approach**, each information source is related to one same global domain ontology, which provides a shared vocabulary for the specification of the semantics (Figure 2.12a). This approach should be applied when all information sources to be integrated provide nearly the same view on a domain, since a new source cannot bring new or specific concepts without requiring change in the global ontology.
- In the **multiple ontologies approach**, each information source is described by its own independent ontology (Figure 2.12b). In practice, it is difficult to compare different source ontologies without a common vocabulary, and inter-ontology mappings need to be defined. However, this solutions leads eventually to facing semantic heterogeneity problems.
- **Hybrid approaches** were developed to overcome the drawbacks of single and multiple ontology approaches. Each information source is described by its own ontology, and all ontologies are built using basic primitives described in a global shared vocabulary (Figure 2.12c). This approach brings advantages when the ontologies need to be developed from scratch. Existing ontologies cannot be reused or they need to be rewritten to refer to the shared vocabulary.

3.3.1 Connecting ontologies to information sources

The connection between an ontology and the information source it describes can also be seen as a *mapping*. There are some possible ways to establish a connection between ontologies and information sources. The TSIMMIS system (Chawathe et al., 1994) produces an ontology that is a simple one-to-one *copy of the structure of the database*. In addition to it, further definitions of concepts can be included, in order to enrich the ontology structure. But the classical approach of linking ontologies to data sources is still the use of *annotations*, which has become prominent with the *Semantic Web* requirements. Semantic annotation on the Web allows to associate ontology elements to parts of a web page.



(a) Single ontology approach (b) Multiple ontology approach

(c) Hybrid ontology approach

Figure 2.12: Three ways for connecting ontologies to information sources (adapted from Wache et al. (2001))

3.3.2 Ontology matching

Although the objective of ontology-based systems is to decrease semantic heterogeneity, ontology development itself leads to a large number of ontologies covering the same domain or overlapping domains. As a consequence, in order to make two or more ontology-based systems to interact, *the ontologies need to be matched*.

Creating mappings is an error-prone activity, even for humans. Many current tools propose semi-automated mapping, interacting with experts. These approaches are the most adequate for applications where accuracy is important (Uschold and Gruninger, 2004). Approaches that perform *fully automatic mapping*, combining various techniques, usually employ ontologies that tend to be less formal, and the results are much more error prone.

Ontology matching techniques do not vary much from schema matching ones (*cf.* Section 3.2.1). The difference is that once the ontology matching is performed, the resulting ontology alignments are commonly formalized, using a proper alignment format. Several techniques of ontology or schema matching have been produced in the previous decades. The book of Euzenat and Shvaiko (2007) presents a mostly complete survey on the frameworks and tools devoted to ontology and schema matching.

3.3.2.1 Representing ontology mappings Another significant issue is how to represent mappings produced by matching processes. This concern is due to the fact that mappings can be reused for other applications. In ontologies represented in OWL, it is possible to use primitives from the language itself for expressing correspondences between concepts. In particular, the primitives *owl:equivalentClass* and *owl:equivalentProperty* are suggested to be used for relating elements in ontologies that describe the

same domains. However, there are some drawbacks in using embedded primitives of OWL for expressing mappings (Euzenat and Shvaiko, 2007): it forces the use of a particular language for representing the ontologies (OWL), and it mixes correspondences and definitions.

Noy (2004) discuss other approaches of mapping representation that externalize mappings in relation to the ontologies to be mapped. Mappings can be expressed as a set of *bridging axioms* in first-order logic relating classes and properties of the two source ontologies. C-OWL (Contextualized OWL, (Bouquet et al., 2003)) is an extension of OWL to express bridge rules between ontologies. An alternative approach for bridges is to create an ontology for defining the structure of the mappings. Correspondences between two ontologies will be represented as *instances of the mapping ontology*. The MAFRA framework uses the Semantic Bridging Ontology (SBO, (Maedche et al., 2002)), which is an ontology of mapping constructs and transformation functions to transfer instances from one ontology to another.

A final approach consists in representing mappings as *views*, similarly to the database integration approaches of global-as-view (GaV) and local-as-view (LaV). The Ontology Integration Systems framework (OIS, (Calvanese et al., 2001)) is composed of a global ontology and a set of disjoint local ontologies, defined using Description Logics. The elements of one ontology are mapped into a *view* of the other ontology. The direction of this view can be global or local centric, depending on which ontology is used as query model. In other words, the mappings are expressed as views (DL expression) over the ontologies.

4 Fundamentals of metamodeling

If a model is an abstraction of the real world, as defined in Section 1.1 of Chapter 1, then a *metamodel* is yet another abstraction, which highlights properties of the model itself.[21] We say that a model conforms to its metamodel in the same way we say that a map conforms to its legend, i.e. that the map is written in the (graphical) language defined by its legend, or that a program conforms to the grammar of the programming language in which it is written (Bézivin, 2005).

Metamodeling is an attempt to adequately model all aspects of any given modeling technique. Metamodels have proved popular in explaining and communicating constructs of some modern modeling techniques, for example, workflow models, object-oriented schemas, and even ontologies (Davies et al., 2003). In summary, a metamodel can be considered as an explicit description (constructs and rules) of how a model is built.

For this end, Model-driven engineering (MDE) is a discipline in software engineering that focuses on creating models or abstractions. MDE technologies aim to address the lack of integrated view of software applications (Schmidt, 2006). A particular variant of MDE trend is the Model-Driven Architecture ® (MDA ™) initiative. Model-driven architecture is a software design approach launched by the Object Management Group (OMG) in 2001. The most important standards launched by MDA for realizing MDE principles are:

[21] The prefix *meta-* is commonly used to indicate the rise of one layer. The *metamodel* layer is the layer on top of the *model* layer.

- the Meta Object Facility (MOF, (OMG, 2006)), which provides a four-level architecture for creating metamodels; and,
- the Unified Modeling Language (UML, (OMG, 2008)), a modeling language for analysis, design, and implementation of software-based systems using a set of graphical notation techniques.
- the XML Metadata Interchange (XMI, (OMG, 2009)). Commonly used as an XML interchange format for UML models.

The real power of MDE strategy comes from the possibility of stratifying a given system in its models and metamodels. This allows building coordination between models, based on different metamodels. MOF architecture can support the role of providing *metamodel stratification*.

4.1 Metamodel stratification with MOF

The Meta-Object Facility (MOF) is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. The MOF framework is based on an architecture with four layers: meta-metamodel, metamodel, model, user objects. The roles of these layers are summarized in the table pictured on Figure 2.13 (adapted from (OMG, 2008)).

Layer	Description	Elements
M3 meta-metamodel (e.g. MOF layer)	The *meta-metamodel* layer is the foundation of the metamodeling architec-ture. The primary responsibility of this layer is to define the language for specifying a metamodel. A meta-metamodel can define multiple metamodels.	MetaClass, MetaAttribute, MetaOperation
M2 metamodel (e.g. UML primitives)	A *metamodel* is an instance of a meta-metamodel. The primary responsibility of the metamodel layer is to define a language for specifying models.	Class, Attribute, Operation
M1 model (e.g. Data model)	A *model* (a.k.a. *Metadata*) is an instance of a metamodel. The primary responsibility of the model layer is to define a language that describes a specific information domain.	StockShare, askPrice, sellLimitOrder, StockQuoteServer
M0 user objects (e.g. values)	User objects (a.k.a. *user data*) are instances of a model. The primary responsibility of the user objects layer is to describe the actual values in a specific information domain.	Acme_Software_Share_98789, 654.56, sell_limit_order

Figure 2.13: Four-layer metamodeling architecture

A *stratified* (layered) metamodel architecture such as that of MOF is a proven methodology for defining the structure of complex models that need to be reliably stored, shared, manipulated and exchanged (Kobryn, 1999). In non-fixed layer metamodeling architectures, on the contrary, classes and their instances are intermixed and can be directly related. Some arguments against unstratified models are discussed in works such as Pan and Horrocks (2002) and Nejdl et al. (2000). Section 4.3 will present a current discussion in the *Semantic Web* community about the ability of dealing with metamodeling in ontology languages. The advantage of stratified models is, thus, shown in practice.

4.2 Model transformation

Kleppe et al. (2003) define *model transformation* as automatic generation of a *target model* from a *source model*, according to a set of transformation rules that describe how constructs in some source language can be transformed into constructs in some target language. Figure 2.14 shows the classical schema for model transformation.[22] Model transformation consists in writing transformation rules ($Rules_{Tr}$) that maps from the source model (M_a), conforming to a metamodel MM_a, to the corresponding target model (M_b), that conforms to a metamodel MM_b. The transformation rules are defined according to a model transformation language MM_{Tr}, and the metamodels are conform to a metametamodel MMM (e.g. MOF).

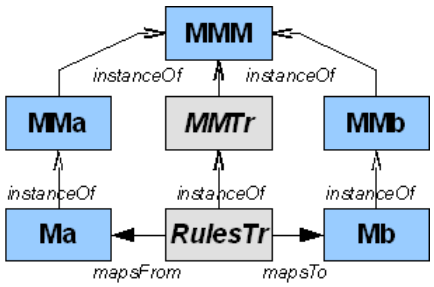


Figure 2.14: Model transformation between source and target models

When we are transforming a source model into a target model and when the two are expressed using the same metamodel, the transformation is called *endogenous*. If the metamodels of the two models are different, the transformation is called *exogeneous* Mens and Van Gorp (2006). Examples of transformation are: class models into schema models, ontology models into relational models, and so on.

4.3 The problem of metamodeling for ontologies

Metamodeling has become a common discussion in the ontology community. In Welty and Ferrucci (1994) the authors point out the usefulness of metamodeling in applications that need to express situations in which classes can also be seen as instances. Another advantage of metamodeling for ontologies is that it becomes possible to interchange models that are described in different languages.

4.3.1 Metamodeling in non-fixed layer architecture

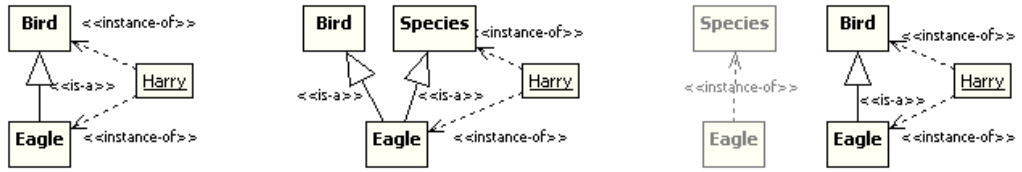
The following example, originally presented in (Welty and Ferrucci, 1994), explains the duality between classes and instances in ontologies.

Example. Let us consider a hierarchy comprising the concept Bird representing the set of all birds, and the concept Eagle, and let us state that all eagles are birds (*Eagle is-a Bird*). The individual Harry is an instance of the concept Eagle. Using the transitivity property of the *is-a* relation, we are able to derive the fact that Harry is also [an instance of] a Bird, and this is a valid conclusion in this domain (Figure 2.15a).

[22] Adapted from Eclipse ATL/Concepts wiki page: http://wiki.eclipse.org/ATL/Concepts/.

Consider adding to this domain the object Species, the class of animal species (Figure 2.15b). The intuitive representation of this object would be to make Species a superclass of Eagle (*Eagle is-a Species*). However, in this case, the application of transitivity property produces undesirable results: the eagle Harry will be an instance of Species.

Another approach to modeling these objects would be to represent Species as a class which has Eagle as an instance (Figure 2.15c). But this conceptualization needs to be placed in another level of interpretation from the level where Eagle is considered to be a class. To this end, the modeling language needs to be able to differentiate the semantics of instances, classes and metaclasses.



(a) Correct representation and inference

(b) Unwanted inference: Harry is an instance of Species

(c) Breaking it up into two interpretation layers

Figure 2.15: Example of metamodeling in ontological hierarchy

RDFS, as a schema layer language, has a non-standard and non-fixed layer metamodeling architecture, which makes some elements in the RDFS specification to have dual roles (*cf.* RDFS model on Figure 2.3 on page 40). RDFS entities are used to define both other RDFS modeling primitives (such as in the triple *[rdf:Property rdfs:subClassOf rdfs:Resource]*) and domain ontology entities (such as in the triple *[:Woman rdfs:subClassOf :Person]*).

For example, the properties *rdfs:subClassOf*, *rdf:type*, *rdfs:range* and *rdfs:domain* are used to define both other RDFS modeling primitives and domain ontology entities.

Languages such as OWL Lite and OWL DL are based on the metamodeling architecture defined by RDFS language, therefore these languages have similar problems of non-fixed layer metamodeling architecture. OWL Lite and OWL DL do not allow one to add other entities to their metamodel. With OWL Full we are able to change the meta-layers of OWL adding or modifying entities.

However, when we modify the metamodel of OWL, the ontology looses some guarantees which OWL DL and OWL Lite provide for reasoning systems, and which constitute an advantage when developing OWL ontologies. Notably, modifications in the OWL metamodel may create an *undecidable model*.[23] Motik (2007) proves that the practice of metamodeling in OWL language leads to the problem of undecidability of basic inference, due to the free usage of the built-in vocabulary .

[23]Decidability is a term that comes from Logic. In computer science, a *decidable* set is any set of elements for which there exists an algorithm that will determine whether any element *is* or *is not* within the set in a finite amount of time. The classes of an ontology may be undecidable if they are described in the OWL-Full language.

4.3.2 Fixed layer architectures for *Semantic Web* languages

It is well accepted that *Semantic Web* languages do not facilitate metamodeling aspects, since their metamodel is embedded in the language, and cannot be modified. For example, if one needs to add measuring units to OWL properties, they have to be represented as a class, instead of being added as a new ontology primitive. For this objective, some alternative approaches have been proposed for handling metamodeling in ontology languages. In Pan and Horrocks (2003) the authors propose a fixed layer metamodeling architecture for RDFS called RDFS(FA). It is similar to the metamodeling architecture of UML: RDFS(FA) divides up the universe of discourse in different layers, in which the built-in modeling primitives of RDFS are separated. In practice, the architecture consists of four layers: the meta-language layer, the language layer, the ontology layer and the instance layer. However, they are criticized because in theory there can be a infinite number of layers in the metamodeling architecture. Motik (2007) present a proposal of *contextual semantics* for OWL. They follow the same principles of RDFS(FA) by strictly separating the modeling primitives from ontology and instance layers, and proposes that modeling primitives should be interpreted depending on the context. In practical terms, the use of some URI in order to identify an individual has no effect on the meaning of a class that uses the same URI as identifier. In this situation, rules applied to a *class* do not affect their *instance* interpretation.

There also exists a line of related work relying on the metamodeling features of MOF. The most important proposal is the Ontology Definition Metamodel (ODM, (Brockmans et al., 2004)), which defines a metamodel for the OWL-DL built on the top of the MOF framework. They split the modeling primitives of ontologies in the different layers of MOF, which allow them to provide an UML profile for OWL ontologies. Figure 2.16 illustrates this approach.

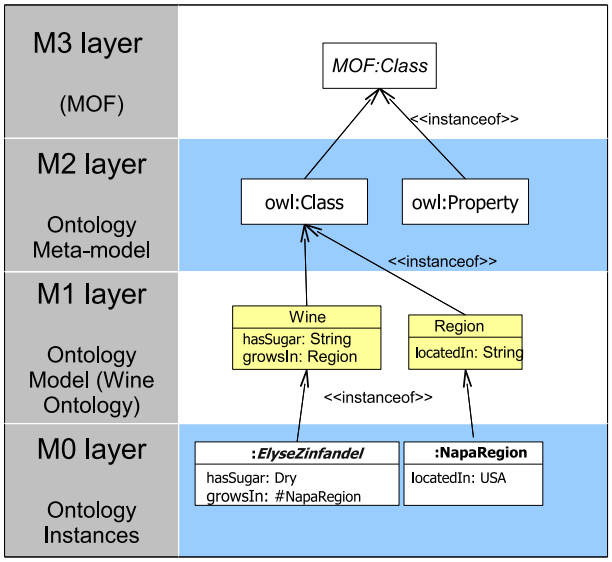


Figure 2.16: Wine ontology represented as an instance of the Ontology Definition Metamodel

The bottom layer represents the information to be described: a wine called Elyse Zinfandel, which grows in the Napa region. The model layer contains the definition of the required structures, i.e. the ontology concepts *Wine* and *Region*. The set of all concepts and relations represented in the model layer defines the Wine ontology. The metamodel defines the constructs in terms of which the model is expressed.

In this example, we are describing an ontology, which is expressed in terms of concepts and relations, by creating instances of the respective meta entities. Ontology concepts are represented as instances of the entity *owl:Class*, and ontology relations as instances of the entity *owl:Property* in the metamodel. Finally, these entities are themselves instances of the *MOF:Class* entity, in the metametamodel layer. Vrandecic et al. (2006) propose an extension of ODM that addresses the requirements for *annotating ontology elements*. In their metamodel, the authors reify axioms as OntologyElement, which become, then objects that can be directly referenced using a URI, and can, finally, be annotated with metadata. This work equips the OWL language with the ability to state facts about any ontology elements, including axioms. Using the ODM profiles makes it possible to transform different ontology models, subject that will be tackled in future sections.

5 Conclusion

Having analyzed the concepts of models, ontologies, semantic enrichment and integration of heterogeneous models, we will demonstrate how we used or adapted these approaches in order to address the issue of semantic exploitation of engineering models.

- We used ontologies for formalizing and sharing the knowledge about the application domains;

- Semantic annotation approach is employed for linking models, tools, interpretations to the global and shared knowledge;

- Semantic integration has proved necessary in order to create a global view of the different knowledge of sub-domains of a global domain;

- Finally, metamodeling techniques were used in order to create new primitives for engineering models and annotation.

The articulation of all these strategies is the basis of an architecture for ontology-based integration of engineering domains, that will be described in Chapter 5.

Chapter 3

Context of the Development: OntoDB and OntoQL

1 Introduction

The implementation considered for our approach must be able to represent the following elements: (i) data models and their objects; (ii) ontologies and their instances and (iii) annotations of data objects by instances of ontology. In order to explain how the proposals of this work were developed, it is necessary to introduce the infrastructure where semantic-based solutions will be implemented. In the context of our work, we have two fundamental criteria for choosing the implementation infrastructure. Firstly, it must be able to manage a huge amount of information, since an important quantity of data is currently available in engineering domains. It is natural, then, to rely on a solution that supports a database infrastructure, such as *ontology-based databases* presented in the previous chapter. These architectures deal with issues of ontology representation while taking advantage of characteristics of databases (such as scalability, safety, etc).

Secondly, support for metamodel evolution is important, since we need to extend this infrastructure to represent other data containers than the ontology metamodel (e.g the *annotation* meta-model). In comparison to other OBDBs, the OntoDB architecture is the only one that accepts to be modified by increasing the original primitives. For these reasons, this work will be implemented in the OntoDB ontology-based database, which fulfills the main criteria needed for this work, as we will detail further on.

2 The OntoDB architecture

OntoDB is designed by a layered approach on top of the relational database system PostgreSQL,[24]. The database model of OntoDB consists on four related parts (*cf.* Figure 3.1).

[24]PostgreSQL (a.k.a. Postgres) is a free and open source database management system (http://www.postgresql.org/).

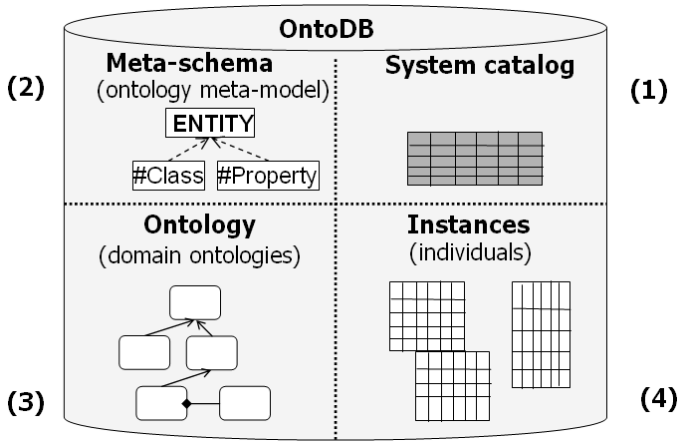


Figure 3.1: OntoDB architecture

2.1 Part 1: System catalog

The part shown on Figure 3.1(1) is the traditional part available in all DBMSs, namely **system catalog**, which contains system tables that are useful for data management. The system catalog structure describes tables, foreign keys, data types, etc., and it varies from one DBMS vendor to the other. The other parts of the architecture of OntoDB are analogous to the OMG's Meta Object Facility, which propose four superposed layers that represent four levels of abstraction of information: metametamodels (M3 layer), metamodels (M2 layer), metadata (M1 layer) and data (M0 layer) (OMG, 2006).

2.2 Part 2: Meta-schema

The **meta-schema** part (Figure 3.1(2)) corresponds to the layers M3 and M2 of MOF. The meta-schema structure (M3 layer) is based on a meta-meta model mainly composed of upper level constructs *ENTITY* and *ATTRIBUTE*, which, from a database point of view, correspond to two tables. These tables store the ontology model (M2 layer) used to define *ontologies*. OntoDB was first designed for the PLIB ontology model, which was stored in the meta-schema part. It has, then, been extended so as to store constructors of other ontology models, such as OWL. As a result, OntoDB provides basic ontological constructs, like *Class*, *Property*, *DataType*, which corresponds to instances of *ENTITY*. Ontological constructs are described by fields, such as *code*, *definition*, *version*. These fields are instances of the primitive *ATTRIBUTE*. Figure 3.2 illustrates the basic ontological constructs proposed by OntoDB.

The construct *Class* stands for ontological elements that represent concepts and categories of objects (such as *owl:Class* from the OWL language or *PLIB:Class* from the PLIB model). The construct *Property* stands for characteristics of ontological concepts and relationships between them (such as *rdf:Property* from the RDF language, or *owl:DataType* from the OWL language or *PLIB:Property* from the PLIB model). Properties can have as range a datatype that may be either primitive types (integer, real, string, boolean), association type (*RefType*) or collection types (*CollectionType*).

New ontology constructs (e.g. *owl:Restriction* and *PLIB:MeasureType* can be added into the meta-schema of OntoDB. Technically, a new construct is added as a row in the *ENTITY* or in the *ATTRIBUTE*

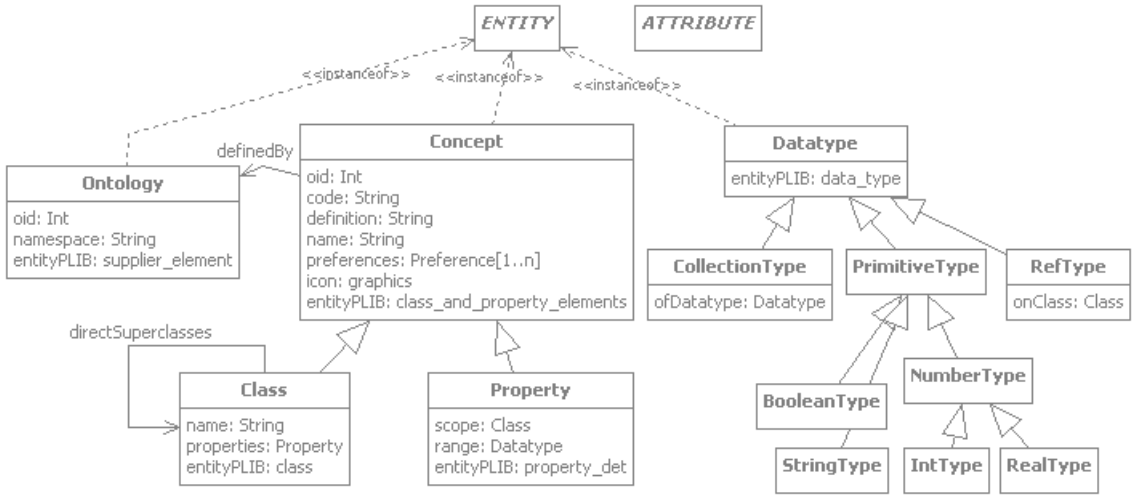


Figure 3.2: Ontological constructs of the OntoDB meta-schema

table, and a new table is added in the ontology part (described below) so as to store instances of the new constructs. This capability of OntoDB of extending its metamodel makes it possible to store ontologies specified in different ontology languages.

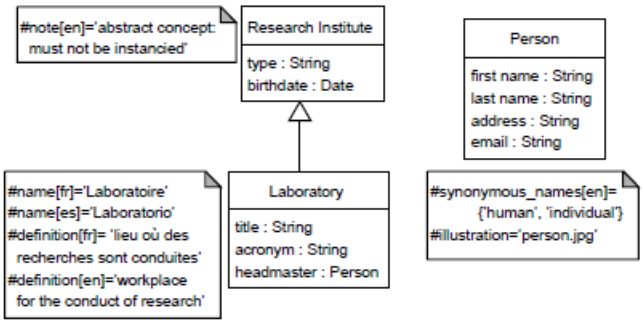
2.3 Part 3: Ontology

The third part of OntoDB, named **ontology** (Figure 3.1(3)), corresponds to the layers M2 and M1 of MOF. This part allows to store ontologies (M1 layer) as *instances of ontological constructs* (M2 layer). For example, each ontology concept is an instance of the *Class* construct of the ontology metamodel, and each ontology property is an instance of the *Property* construct of the ontology metamodel.

Figure 3.3 illustrates how an ontology is represented in the internal tables of the OntoDB architecture.

Figure 3.3a presents a toy example of an ontology that describes concepts related to research institutions (adapted from (Jean et al., 2006b)). Figure 3.3b stands for the internal table of OntoDB (*OntoDB:Class*) that describes the information about each concept as one line in the table. For example, the concept *Laboratory* is described by its oid code (‘3’), name (“Laboratory”) and the oid codes of the concepts that are directed superclasses of *Laboratory* (the oid code ‘2’ corresponds to the concept *ResearchInstitute*).

Figure 3.3c shows properties being represented as lines in the *OntoDB:Property* table. The property *firstname* has the concept *Person* as scope and its range is the data type *String*. The scope of the properties *title* and *headmaster* is the concept *Laboratory*, but the range of the second is the concept *Person*. The property *headmaster* is an *association property*, i.e. it defines a relation between two concepts.



(a) Research ontology

OntoDB:Class					
oid	name[fr]	name[es]	superclasses	synonimous[en]	definition[en]
1	Person		NULL	'human', 'individual'	
2	ResearchInstitute		NULL		
3	Laboratory	Laboratorio	2		'workplace for..'

(b) Concepts represented in the *OntoDB:Class* table

OntoDB:Property			
oid	name	scope	range
1	firstname	1	String
2	title	3	String
3	headmaster	3	1

(c) Properties represented in the *OntoDB:Property* table

Figure 3.3: An example of ontology represented in OntoDB tables

2.4 Part 4: Instance

Finally, instances of ontologies are stored in the **instance** part (Figure 3.1(4)), which stands for the layers M1 and M0 of MOF. The structure of this part (M1 layer) is built from a *subset of concepts and properties* of the ontology. This subset constitutes the logical schema of data, and represents the data structure in which instances (M0 layer) will be organized. Each ontology concept is transformed in a table in the logical schema, and the user determines which available properties are needed to describe instances of that concept. Then, a table view (called here an *EXTENT* of the concept) is derived to store instances. This approach of representing instances is called *horizontal approach*: one table is created for each ontological class. This representation is able to well scale when numerous properties per instances are used (Dehainsala et al., 2007).

For example, in order to build the logical schema of the ontology on Figure 3.3a, one could choose the concept *Laboratory*, with properties *title* and *headmaster*; and the concept *Person*, with *firstname*, *lastname* and *email* as properties. Tables *E_Laboratory* and *E_Person* will be created as an *EXTENT* of the chosen concepts, along with the chosen properties. The resulting logical schema is shown on Figure 3.4 (adapted from (Jean et al., 2006b)).

As a consequence, many different logical models may be derived/related to the same ontology. This pos-

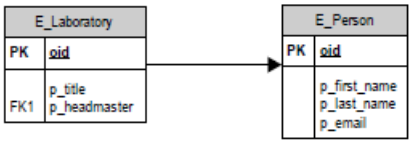


Figure 3.4: Possible logical schema for the Research ontology of Figure 3.3a

sibility promotes a database approach preserving compatibility with Relational Database Management Systems (RDBMS) and promotes semantic integration of OBDBs by offering an ontology for different logical models (Jean et al., 2006a).

3 The OntoQL exploitation language

In order to exploit the OntoDB OBDB, the OntoQL language has been proposed by Jean et al. (2007). OntoQL language has a syntax similar to SQL, and provides Data Definition, Manipulation and Query Languages at three layers of OntoDB, from the *logical* level to the *meta-schema* level. Consequently, it is possible to use OntoQL language for defining, manipulating and querying not only domain ontologies but also the meta-schema of OntoDB (M2 layer).

3.1 OntoQL language for the ontology and instance parts

OntoQL allows to create, alter and drop elements of domain ontologies (concepts, properties) and also to create and manipulate instances and finally to query ontologies.

3.1.1 OntoQL Data Definition Language (DDL)

The DDL allows to create, alter and drop concepts of domain ontologies. Ontology concepts are created in OntoQL using the basic constructors *#Class* and *#Property* (presented in Section 2.2). To illustrate, let's consider the following OntoQL expression:

```
CREATE #CLASS Laboratory UNDER ResearchInstitute (
  DESCRIPTOR(#name[fr,es] = ("Laboratoire", "Laboratorio"),
    #definition[fr] = "lieu ou des recherches sont conduites",
    #definition[en] = "workplace for the conduct of research")
  PROPERTY(title STRING, headmaster REF(Person), acronym STRING) );
```

This OntoQL expression uses the *CREATE #CLASS* clause to create an ontology concept *Laboratory* in the default language (English); and uses the *UNDER* clause to declare it as a subclass of concept *ResearchInstitute*. Thus, *Laboratory* inherits the properties of *ResearchInstitute*.

The *DESCRIPTOR* clause is used to describe a concept using fields declared in the ontology metamodel. In the previous example, this clause describes the concept using the field *name*, for names in other languages (e.g. French and Spanish) and the field *definition*, for the concept meaning. The *PROPERTY*

clause creates properties attached to the concept. In the example above, the *PROPERTY* clause creates the properties *title*, *headmaster*, *acronym* as instances of the *#Property* construct. The range of the property *headmaster* makes a reference to the concept *Person* using the operator *REF*. Note that the distinction between entities from the ontology metamodel and entities of the user ontology is carried by the attributes prefix '#' (*cf.* Section 3.1.3).

A concept can be modified using the *ALTER* clause. The operator *ADD* in the following expression modifies the concept *Person* by adding a new property *age*, while the operator *DROP* erases this property from the concept.

```
ALTER Person ADD age INT;

ALTER Person DROP age;
```

The DDL enables to create an extent of concepts and properties created in a domain ontology. An extent can be attached to a concept by means of the clause *CREATE EXTENT*, as in the following expression:

```
CREATE EXTENT OF Laboratory (title, headmaster);
```

Notice that the extent do not declare the *acronym* property. Thus, this property will not be valued in the content part. When executed, this expression creates a logical schema, as the one presented on Figure 3.4, to store instances of this concept in the logical level.

3.1.2 OntoQL Data Manipulation Language (DML)

DML operators are provided to create, update and delete instances of ontology concepts. The following clause creates a new instance of concept *Laboratory*. Notice that if the extent of the concept was created with the latter OntoQL expression, the following insertion would not be accomplished, since property *acronym* would not exist in the extent table.

```
INSERT INTO Laboratory (title, acronym)
VALUES ("Laboratoire d'Informatique Scientifique et Industrielle", "LISI");
```

The instruction *UPDATE* allows to modify instances, adding new values to properties or deleting values. The following expression modifies the headmaster of the laboratory whose acronym is "LISI", by retrieving the identifier of the desired person.

```
UPDATE Laboratory SET headmaster =
  (SELECT oid FROM Person as p WHERE p.firstname="Yamine")
WHERE acronym = "LISI";
```

Finally, instruction *DELETE* is used to delete instances of some concept. For example, the following expression deletes instances of concept *Person* that do not have a last name.

```
DELETE ONLY(Person) WHERE lastname IS NULL;
```

This example shows that object-oriented constructors are available in OntoQL. Indeed, the keyword *ONLY* restricts the operation (in this case, the *delete* instruction) *only* to the instances of the declared

concept. The instances of subclasses of the concept are not concerned. When an operation is applied in cascade over the subclasses of a concept, we call this a *polymorphic* operation. The keyword *ONLY* defines an *non-polymorphic* operation, and is mostly used in *SELECT* clauses.

3.1.3 OntoQL Data Query Language

The data query language part of OntoQL is designed as an extension of SQL to query ontology *content* stored in OntoDB. Querying content with OntoQL is similar to a classical SQL query and does not rely on any specific logical database model. Therefore, two applications sharing a common ontology will have the right to run a common query even if the underlying logical database models are different. The *SELECT* clause in OntoQL defines projection on properties defined on a concept. As in SQL, the input and the output of an OntoQL query is a relation. The following queries search for the acronyms of all laboratories. Because names of concepts and properties can be defined in different natural languages, a given query can be written in any of these languages. The two queries below are equivalent, but the first query uses the English names of concepts and properties, while the second query uses their French version:

```
SELECT acronym FROM Laboratory

SELECT acronyme FROM Laboratoire
```

Furthermore, like SQL, OntoQL allows queries to be nested in the clauses SELECT, FROM or WHERE; it is equipped with aggregate operators (count, sum, avg, min, max), sorting (ORDER BY) and set operations (UNION, INTERSECT, EXCEPT, GROUP BY).

3.2 OntoQL language for the metamodel part

The DDL, DML and DQL of OntoQL allow users to define ontologies and their instances. OntoQL proposes also a language for defining, manipulating and querying its own primitives, described in the meta-schema part (Section 2.2). OntoQL handles the built-in basic constructors of ontology models (*cf.* Figure 3.2). This set can be extended with other entities using the *CREATE ENTITY* clause and enriched with new attributes. For example, in order to add the OWL restriction constructor *AllValuesFrom* to the core model of OntoQL, we can design the following expressions (adapted from Jean (2007)):

```
CREATE ENTITY #Restriction UNDER #Class (
  #onProperty REF(#Property))

CREATE ENTITY #AllValuesFrom UNDER #Restriction (
  #allValuesFrom REF(#Class))
```

The first instruction creates a new entity *#Restriction* that inherits from the entity *#Class*. This entity has the attribute *#onProperty*, which makes a reference to the entity *#Property*. The second instruction creates the entity *#AllValuesFrom* as a sub entity of *#Restriction*. The attribute *#allValuesFrom* indicates the class (its type is a reference to the entity *#Class*) from which the instances of the restriction take their values for the property defined in the attribute *#onProperty*.

All the operators for data manipulation (INSERT, UPDATE, DELETE) can also be employed for metamodel and ontology model manipulation. An INSERT instruction executed over an entity creates instances of this entity. Considering the following expressions:

```
INSERT INTO #Class (#name)
VALUES("Laboratory")

INSERT INTO #Property (#name, #scope, #range)
VALUES('acronym', 'Laboratory', 'String')
```

The first expression creates an instance of the entity *#Class* named Laboratory. The second creates an instance of *#Property*, that is, a *property* whose scope is the concept *Laboratory* and the range is type String. These operations are equivalent to creating a concept with its properties in the classical way, using the *CREATE #Class* clause, as shown in Section 3.1.

The query part is a SQL-like language to query the core model of OntoQL. An special symbol is used to directly address the elements of the core model: the '#' symbol, as we see in the following query:

```
SELECT p.#name FROM #Property as p, #Class as c
WHERE p.#scope = c.#oid AND c.#name[fr] = "InstitutDeRecherche";
```

This query returns the names in the default language of the properties of the concept named "InstitutDeRecherche" in French. Notice that *#name* is an internal field of both the entities *#Class* and *#Property*, so it has to be referenced unambiguously, using *path expressions* (like it is done in *p.#name* and *c.#name*).

3.3 OntoQL tools

OntoQL expressions can be created and executed using the command line interface **OntoQLPlus**. The expression can be edited directly in the text area. OntoQLPlus provides syntax highlighting, history of the executed commands, possibility of executing scripts of commands, and translation of an OntoQL query to a query in SQL or SPARQL. on Figure 3.5, some OntoQL data definition expressions were already executed in the OntoQLPlus interface. The OntoQL query in the last line
`SELECT title, acronym FROM Laboratory` searchs for all the titles and acronyms of laboratories. The result of an OntoQL query is displayed in columns.

4 Conclusion

The current management of ontology-based data does not satisfy the performances and reliability requirements necessary for many applications, especially data-intensive applications. OBDB have been defined to solve this problem providing database architectures to store ontologies and their instances. OntoDB is a OBDB that differs from other OBDB in two ways:

1. under some assumptions it stores ontology instances in an horizontal representation that provides better performance when numerous properties are used in queries;

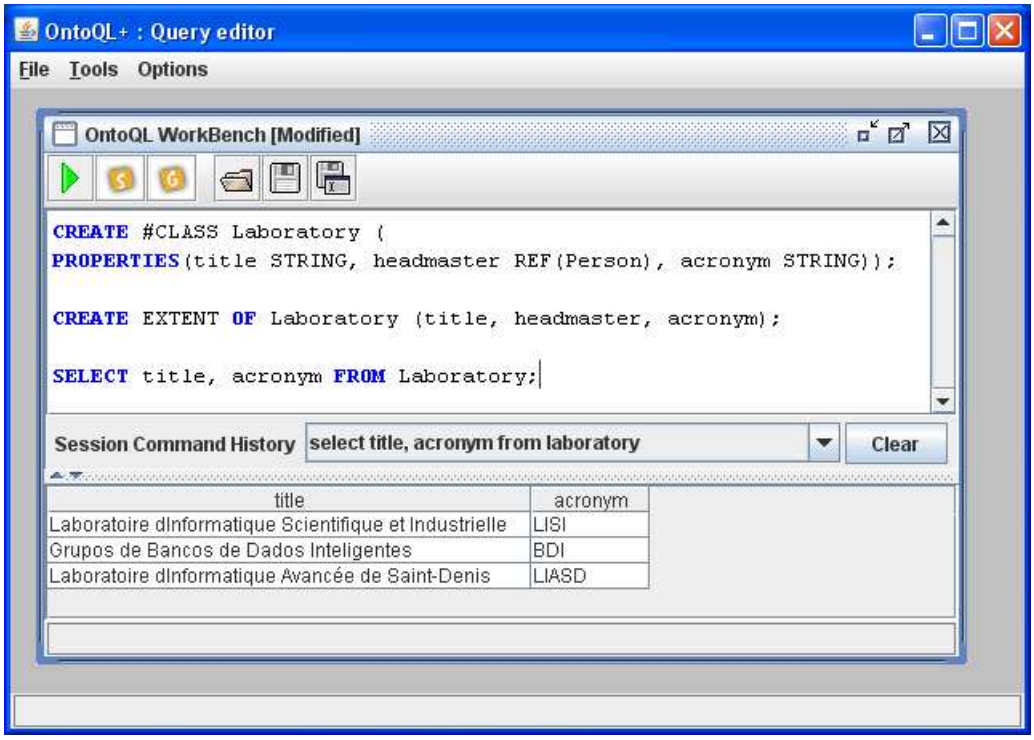


Figure 3.5: OntoQLPlus: a visual interface for OntoQL language

2. it provides a meta-schema part that can be altered and increased. The importance of this capability for the work performed in this thesis will be clarified in further sections.

To query ontologies, a new generation of languages, called Ontology Query languages have been defined (see: Section 1.4 of Chapter 2). These languages offer the possibility to query ontologies and their content. OntoQL is an ontology query language based on SQL that provide definition, manipulation and query languages for both data and ontology. OntoQL bases on a core model containing primitives that are common to different ontology models (such as *#Class* and *#Property*). This core model can be structurally extended by adding new constructs so as to take into account particularities of other ontologies. This allows representing ontologies in OBDB whatever the ontology metamodel and also creating entirely new primitives that are not related to the ontology primitives. It this allows to fully exploit the meta-schema of OntoDB.

Chapter 4 and Chapter 5 of this work describe how the OntoDB OBDB and the OntoQL exploitation language have been used and extended for validating a proposed approach for handling models heterogeneity in engineering domains.

Part II

Contribution: Knowledge Explicitation in Engineering Models

Chapter 4

A Model of Annotation of Engineering Models

1 The issue of making knowledge explicit in Engineering Models

As it will be explained in later chapters (*cf.* Section 1 of Chapter 6), petroleum companies are subjected to significant issues related to capturing the users expertise. The knowledge corresponding to the experts' interpretation about raw data is not made explicit inside models: it is generally embedded in technical data stores and files or it remains hidden in reports or in experts' brains. As a consequence, interpretation remains in most cases inaccessible for being recovered by professionals. Thus it is generally impossible to characterize the content of a given interpretation and to determine when, how, why and by whom it was created.

These issues are shared by other domains that carry out activities that are based on engineering models. Previous chapters explained that engineering models are constituted of pieces of raw data that are assembled by professional that are experts in the interest field. When building the model, these professionals identify significant objects inside the data sets. The identified objects are the result of a complex task of interpretation (*cf.* Section 1.2 of Chapter 6) accomplished by professionals that have years of experience in the field. However, depending on the level of expertise of these professionals, and also on the quality and quantity of contextual information that is available about the data, each expert can provide different interpretations about the same data set. And, currently, it is not possible to maintain these different interpretations in an explicit way, coupled with the data. Here lies the need for a way of enabling professionals to preserve their interpretation.

The proposal of this work for addressing the lack of explicit knowledge in engineering domains such as the petroleum exploration activity is inspired by the *semantic annotation* approach. We aim to allow explicit meanings to be assigned to objects that an expert identifies inside data. Annotation is then considered as a *top-level entity*, independent of the ontology model. This proposal is specified in terms of a *metamodel for annotation*, based on the Meta Object Facility (MOF). The goal of this work being the annotation of engineering models, we also apply metamodeling strategies in order to propose a *metamodel for representing engineering models*. The proposed metamodels are abstract conceptualization that can be grounded to any language. This chapter describes a semantic-annotation-based approach to make explicit the knowledge from engineering domains.

Please note that the models presented in this work are UML class diagrams. The actual metamodels were formalized in EXPRESS language (Schenck and Wilson, 1994) and implemented using the OntoQL language, as shown in Section 2.

Also, in this work the terms *data model*, *data organization*, *metadata*, *file format* are employed meaning *the description of the way data are structured or organized*. For this reason, we refer to these expressions interchangeably throughout this work.

1.1 Making engineering models explicit

The main goal of this section is explaining (i) how to create explicit and formal representations of the data models used by engineering models and (ii) represent files as instances of their data models.

1.1.1 Analysis of how engineering models can be annotated

As explained in Section 2 of Chapter 2, semantic annotation (or ontology-based annotation) is a current semantic Web technique for adding knowledge to resources by means of semantic tags, which are previously formalized by means of an ontology model. Inspired on this approach, we propose that engineering models should be *tagged* with ontology concepts so as to allow explicit meanings to be assigned to objects that an expert identifies inside data.

From the state of art on semantic annotation, one can figure out that no framework or technique for annotating objects inside computer-based models is proposed. We are aware of the fact that ontology-based annotation is aimed to overcome issues related to the *Semantic Web* paradigm. Indeed, this technique was created to attach machine-readable semantic descriptions to Web resources. In consequence, even ontology languages have evolved in such a way that they are compatible with the format in which Web items are represented. The resources that store information in the Web are identified by global identifiers (URIs), the same type of identifiers that are used by the languages used for developing ontologies for the Web, such as RDFS and OWL. It becomes evident that resources that are not suitable to be represented using the Web standards cannot have benefit of the use of semantic Web proposals. This concerns the data sets manipulated in engineering domains. For these reasons, it is necessary to conceive of a new approach of semantic annotation, which captures also the resources not compatible with the Web formats.

We are interested in annotating data files that are not yet handled by other semantic annotation techniques, that is, *structured* data files. When dealing with data that are part of computer-based models, we are not allowed to change the data structure, because this would mean also change the associated computer-based systems that use the data. The data files could, then, be easily exported to a text format (CSV, XML) and read into a relational database; or transformed in RDF triples and read into a RDF database. However, this method does not keep the original format in which data were organized inside the file: it transforms data either in simplified format either in triples. We are interested in storing data using their original format in order to perform *data transformation* between the various formats, which is a very demanded task in engineering domains.

We have decided to “wrap” the information from the data files using a common formalized language.

This language needs to be able to represent the various data standards and formats of data. It must, then, be defined on an upper information layer from all the possible data formats. To this end, we have made appeal to MDE metamodeling strategies (*cf.* discussion on Section 4 of Chapter 2), as explained next.

1.1.2 Metamodeling techniques applied to engineering models

We believe that, in order to attach semantic information on engineering data files, we need to *externalize* the model in which data are organized. We propose that the data structure of files be captured (manually or automatically) and expressed in a reduced, homogeneous, formal data model that captures all the basic elements on which data inside the files is based. The elements that are essential for accessing the information within a file are basically: the *file identification* (file name and its physical location) and the *attributes* that organize the data inside the file. These elements can be captured from the metadata associated to the file. If the file has a file reader, the metadata of one file is written inside it, otherwise, the metadata is stored externally to the file itself.

The question that follows is: in which language should engineering metadata be represented? Considering that we intend annotating metadata with ontologies, we could consider representing file formats as ontologies. However, it is not desirable to represent engineering data models using constructs designed for building ontologies (such as *owl:Class* in OWL language or *rdf:Property* in RDF language) since data and ontologies have different "nature". The reason is that we do not expect to have, for engineering data models, features currently proposed for ontologies, such as subsumption between concepts or inference. Considering these criteria, we propose that specific constructs, different from the ontology constructs, should be used for representing the metadata of engineering models.

In MDE, the classical strategy for representing different models consists in producing a top-level metamodel and in considering each individual model as an instance of the super metamodel. Czarnecki and Helsen (2003) consider that defining a precise metamodel is a prerequisite for performing transformations between models. We propose then that the constructs for representing the metadata of engineering models should be formalized as an *Engineering Metamodel*.

1.1.3 The Engineering Metamodel

Considering what has just been said, the next step of this work should concern the issue of generalizing the structure of engineering data models into an *Engineering Metamodel* that should be able to communicate with all individual models. The Engineering Metamodel is actually the minimum necessary set of features that allows a uniform description of the engineering models (file name, identification, main composite objects, and so on.). Figure 4.1 illustrates the structure of the Engineering Metamodel.

The element *DataElement* is the abstract super-entity of all other entities of this metamodel, and it has a field *name*. *DataClass*es are first-class modeling constructs. Instances of *DataClass*es (at M1-layer) have identity and can be organized in a specialization/generalization hierarchy by means of the *subtype_of* link. The optional field *formatReader* may be used for specifying the reader that is associated to the data format. The attributes of some *DataClass* are defined with the entity *DataAttribute*, which can have a minimum and maximum cardinality (fields *min* and *max*) and a field *range*, which is defined by the

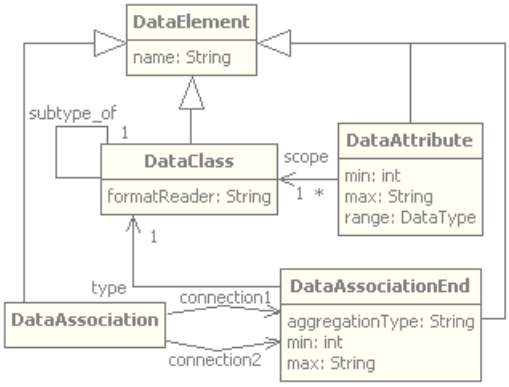


Figure 4.1: The metamodel for engineering models

DataType entity.

*DataAssociation*s reflect binary relationships between *DataClass*es. Instances of *DataAssociation* at the M1-layer are links between *DataClass* instances and which have neither a state nor an identity. This entity is connected to two entities *DataAssociationEnd*, which specify: to which *DataClass*es the association is connected, the multiplicity (fields *min* and *max*) and the type of the association (field *aggregationType*) in each of the end points. The type distinguishes between aggregate and non-aggregate associations. These features are the building blocks that allow to represent any data artifact used in engineering models.

Having the metadata of the file formalized, it is possible to represent (i) all kinds of engineering metadata as instances of the Engineering Metamodel, (ii) all files that use the same metadata as *instances of the engineering metadata*. This approach follows the MDE strategies of *representing data as instances of their metadata* explained in a previous chapter. Figure 4.2 presents the Engineering Metamodel at the M2 layer in the MOF architecture with four metalayers.

The classical four layer metadata architecture has several advantages over simple modeling approaches (OMG, 2008). It can allow different kinds of metadata to be related, new kinds of metadata to be added incrementally, and it can support interchange of arbitrary metadata (models) between parties that use the same metamodel. This favors the generalization of tool-specific data formats and the homogenization of data representation.

1.1.4 Data transformation using the Engineering Metamodel

According to approaches of model transformation (explained in Section 4.2 of Chapter 2), the Engineering Metamodel can be used as common metamodel for transformations between different models. The model transformation consists in writing transformation rules that receive elements from the source model and produce the corresponding target model. This is shown on Figure 2.14 on page 64.

An example of this operation is the case when an engineer, using some specific software tool, needs to convert data files from one model to another. Since all the engineering models are represented conforming to the Engineering Metamodel, the model transformation here is an example of an *endogenous*

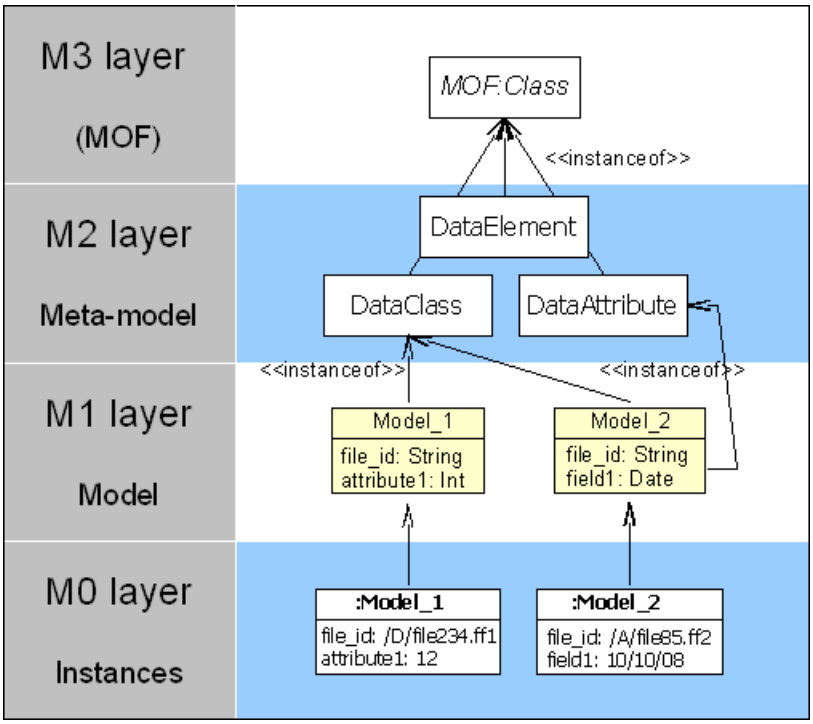


Figure 4.2: Four-layer metadata architecture for the Engineering Metamodel

transformations. From Figure 2.14 on page 64, the metamodels MM_a and MM_b are replaced by the Engineering Metamodel. Model M_a is the model of the source file, and model M_b is the model of the target file. Rules are defined, according to some model transformation language, that maps the elements of the source file model to the target file model.

Furthermore, it is possible to transform models described in terms of the Engineering Metamodel into models that adopt another metamodel, such as ontology models. This is a case of *exogenous* transformations. Figure 4.3 depicts the idea of model transformation between engineering models and ontologies using the Engineering Metamodel.

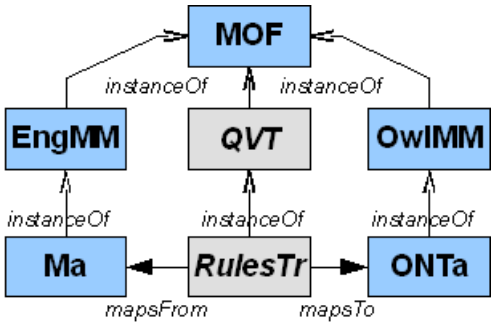


Figure 4.3: Model transformation between engineering models and ontologies

Explanation. The source model is some model M_a used in engineering models, which is conform the Engineering Metamodel ($EngMM$). The target model is an ontology model ONT_a, which can be described

according to the OWL language (*OwlMM*) or another ontology language. These two metamodels (Engineering Metamodel and OWL language) can be described in terms of the MOF language. The model transformation consists in writing transformation rules ($Rules_{Tr}$) in some model transformation language (such as QVT[25]) that produces an ontology from the elements in the engineering model.

The model transformation presented here will be implemented as an approach of *model annotation*, and will be detailed in Section 2.

1.1.5 Example of models represented using the Engineering Metamodel

Let us illustrate the use of the Engineering Metamodel by taking a real example of file format from an engineering domain, extracting its metadata and producing instances of it.

```
#aP 2001  1  1 IGS97 HLM  IGS

+ 28  1  2  3  4  5  7  8  9...
+ 21 22 23 24 25 26 27 28 29 30 31
/* Orbit combination from weighted average

*  2001  1  1  0

P 1 26496.42   -199.76  -1969.65 158.84
P 2 8382.84 -24819.96   2891.41   -319.34
P 3 12055.36 13380.80 -19579.69     71.30
P 4 -2613.85 -22464.07  13821.99    633.04
…
P 31 25564.24 5376.01  -5908.91     24.61
EOF
```

Figure 4.4: Pictorial representation of the file *igs12502.sp3*

Example. In GPS (Global Positioning System) tools, there are many different file formats to represent way-point, track, and route information. The site of the National Geodetic Survey (NGS),[26] which defines and manages an American coordinate system, have some sets of GPS data available. The SP3 format (Standard Product # 3 Orbit Format) keeps information about satellite coordinates. Its metadata (described in the file header) defines the kind of information provided in each set of lines. The first line (starting by the symbol #aP) give contextual information about the file, such as the starting date of the orbit (01/01/2001), the coordinate system (IGS97), orbit type (HLM), agency (IGS), the number of satellites and their identifiers (lines starting by '+'). After that, each line (starting by 'P') gives coordinates information for each of the satellites, within some epoch.

The metadata of the SP3 format (simplified) can be formalized using the Engineering Metamodel as illustrated on Figure 4.5.

Having the SP3 metadata formalized, it is possible to describe every SP3 file as an *instance of the SP3 metadata*. We chose one file, named *igs12502.sp3*, collected from the NGS site, in order to use as

[25]The Query/View/Transformation (QVT) language is a declarative language standardized by OMG for model-to-model transformations.

[26]http://www.ngs.noaa.gov/

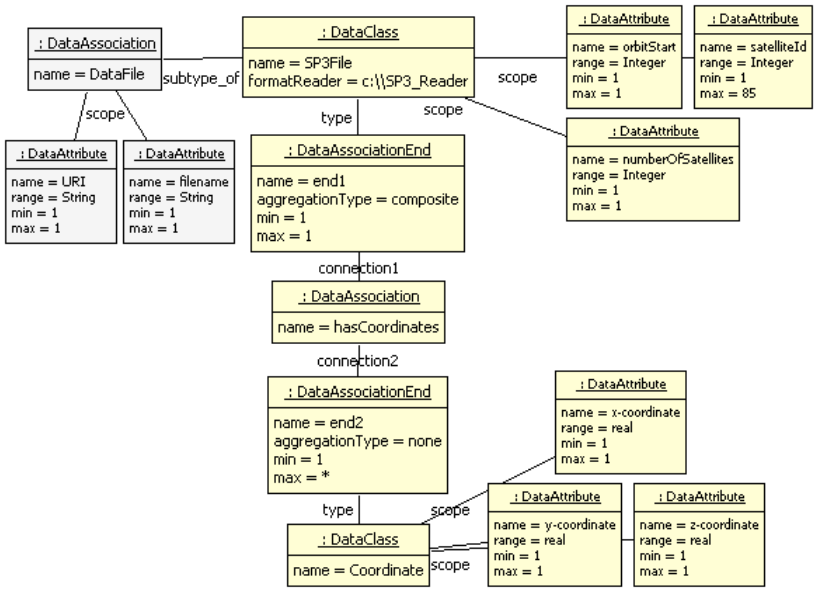


Figure 4.5: SP3 metadata described in terms of the Engineering Metamodel

an example (*cf.* Figure 4.4). Figure 4.6 shows an instance of the SP3 metadata, which stands for the *igs12502.sp3* file.

The attributes of this instance are valued with data obtained from the actual *igs12502.sp3* file.

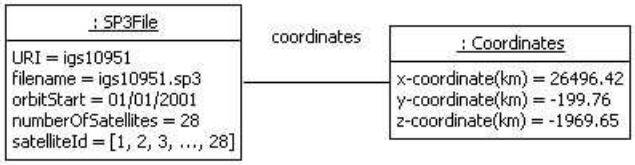


Figure 4.6: File *igs12502.sp3* described as an instance of the SP3 metadata (in terms of the Engineering Metamodel)

The practical advantage of having instances that stand for actual files is the possibility of exploitation of the information included in the instance, independently of the system that originated the file.

1.2 Emergence of knowledge in engineering models

At this point, we recall the original objective of the work described in this chapter: to annotate engineering models with ontology concepts. We have now a formal representation of the engineering models. In order to make knowledge explicit in engineering models, it is necessary to "link" them somehow to the concepts of the ontology that describes the engineering domain, that is, to perform *semantic annotation* of engineering models.

To explore this goal, we have first identified the main types of annotation processes that can be carried out by the users. We have, then, established the requirements for engineering models annotation. We have defined more precisely what constitutes an annotation in this context, and which are its properties. We end this section by presenting an annotation-based metamodel for engineering models and an example of its application.

1.2.1 Annotation processes

In order to understand the types of annotations that we needed to take into consideration, we observed the professionals from the reservoir characterization domain, during the construction of reservoir models. Based on this case study, it was possible to understand the way these users work with the data files, how and when they extract and save information on these files. According to the type of task and the computer-based tool used, we could distinguish three scenarios for producing annotations of engineering models:

- **white box annotation**: the annotation system is integrated to the modeling tool. The annotation system knows the associated ontology, so, when the user produces an interpretation about the engineering model, the annotation system automatically creates instances of annotation referring to the URI of the concepts of the ontology that are associated to the user's interpretation. The annotation system links the annotation to the data set concerned.
- **black box annotation**: when using a proprietary modeling tool, it is not possible to integrate the annotation system into it. The annotation must then be carried out in an interactive way by the user: while building the engineering model the user, simultaneously, produces annotations about the data and manually attaches the annotations to the data files. This corresponds to the practice of generating *experience reports* (text documents) that explicate the interpretation. The difference is that these documents are in natural language, and cannot be processed later on by automatic software tools, while ontology-based annotations are explicit and formal.
- **intrusive annotation**: when the engineering model is previously interpreted, in some open-format data file, the annotation system examines the data files associated and discovers, by using heuristic rules, which objects must be annotated. The system, then, associates these objects to the correspondent ontology concepts, and produces automatic annotations. The link to the original data source is maintained by the annotation.

1.2.2 Requirements for annotation of engineering models

We have defined some requirements for annotation that reflect the need when annotating engineering models.

1. **Support multiple annotations on the same model**. In engineering domains, data sets are evaluated by professionals that have different levels of expertise. The interpretation depends on the objective which the user has in mind, which will define the granularity, precision and context of the information needed. Consequently, each professional may have a different opinion about the

data. It means that for one same data set, different users may probably add different annotations to explain its meaning.

2. **Support annotations of multiple models with one same ontology concept**. This behavior is due to the fact that data sets issued from different models may be just different representations of the same domain concept.
3. **Preserve the original data file**. It is not desirable to add the semantic *tags* directly inside the data artifact. In the situation where there exist multiple opinions about the same data set, simple comments fields directly added to the data set would undesirably increase the attributes of the data artifact. Furthermore, we do not want to modify the original data for the sake of the interest of end-users.
4. **Allow different granularities of annotation**. The expert may want to give an opinion about the entire files, or just about a particular location within the file.
5. **Support attributes on annotation**. Attributes such as author, timestamp, version, are critical features needed from and by end-users.
6. **Support annotation on relations**. From Section 1 we have concluded that two types of opinions can be given by experts: opinions about *object identification* (which is encompassed by annotating the data sets) and about *relationships between objects*. In current annotation models, one can annotate ontology entities, but there is no defined way to annotate ontology statements. This is not sufficient for a complete formalization and sharing of data's interpretation.
7. **Support queries on annotations**. Users need to be able to query annotations for two distinct purposes: (i) to find annotations where the annotation values themselves are of interest; and (ii) to find annotations where the associated data values are of interest (*cf.* Section 2.3.3 of Chapter 2).

All these characteristics lead us to think that it is better to have annotations existing in a different space than that of the ontology, so that they may not modify the behavior of the domain model. Therefore, we propose to consider annotations as "stand-off" item, which will remain some way external to the annotated entities.

1.2.3 The Annotation Metamodel

In this section we introduce a metamodel that addresses the requirements for annotating engineering models identified in the previous sections. In this metamodel, annotation becomes a *top-level entity*, with its own attributes, separated from the ontological model and from the entity being annotated.

The annotation element need to be linked to concepts of ontologies, in general. An ontology concept represented with the OWL language should be able to be linked to an annotation entity as much as a concept represented in the PLIB model. We need to define the annotation entity in the same level as the ontology constructs *owl:Class*, from OWL and *Class* from PLIB.

Therefore, the annotation entity needs to be proposed in the metamodel level. For this, we propose a *Metamodel for Annotation* that aims to link the annotation to ontology constructs in one side, and link to a *metamodel of possible annotated resources* in the other side. But why a *metamodel of resources*?

The goal is to propose a general framework of annotation, where any ontological construct can be linked, through the annotation entity, to any type of resource that can be annotated.

Figure 4.7 illustrates the idea of the general metamodel for annotation: in the left side we have all the constructs for building ontology concepts from different ontology languages. In the right side we have the constructs for resources such as documents, videos, images, and also Web-services (the *process:AtomicProcess* construct) and computer-based models (the newly created construct DataElement - *cf.* Section 1.1.2). Resources such as documents, videos and images dispose already of frameworks for being annotated. We are interested in the application of the *Metamodel for Annotation* to engineering models. Therefore, we restrict the type of resource only to the constructs of engineering models (*DataElement* entity).

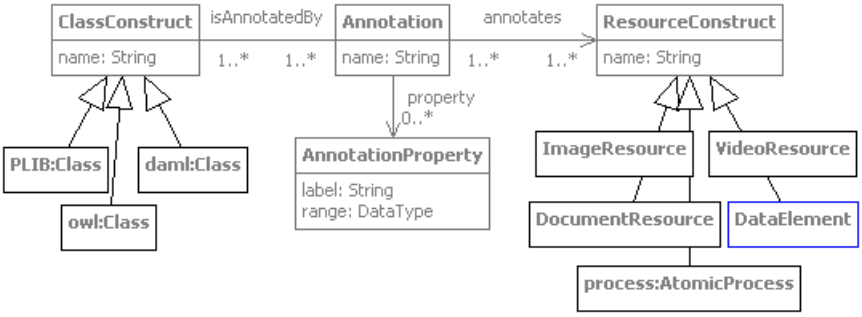


Figure 4.7: Metamodel for annotation of engineering models

Explanation. The *Annotation* entity creates a link between the construct of ontology concepts and the *DataElement* construct through the relations *annotates* and *isAnnotatedBy*. These relations have multiple cardinality, which means that one annotation element can either annotate multiple data elements with one same ontology concept, or use multiple ontology concepts to annotate one same data element, or both. The *AnnotationProperty* entity is the construct that should be used to provide all types of properties to the annotation, by means of the field *property*.

We compare here the present proposal with the annotation metamodel proposed in the work of Vrandecic et al. (2006). Vrandecic et al. extend the Ontology Definition Metamodel (*cf.* Section 4.3 of Chapter 2) with the constructs *AnnotateableElement*, *Annotation*, and *AnnotationPropertyValue*. All ontology elements (classes, properties) become a subtype of *AnnotateableElement*, that is, they are possible to be annotated. An *Annotation* can be either a data value, an URI or an individual. It means that an ontology element will be annotated with an instance. The structure of an annotation is the triple *AnnotateableElement AnnotationPropertyValue Annotation*. The difference to the present proposal is that the metamodel of Vrandecic et al. can be used just for annotating ontology elements, while the Annotation Metamodel proposed in this work is suitable for annotating any kind of resource.

1.2.4 Example of annotation of engineering models

In order to illustrate the application of the Metamodel for Annotation, we take as example the activity of climate modeling, described as follows.

Example. The Earth System Curator project develops a metadata formalism for describing the resources used in climate simulations, presented in Murphy et al. (2008). The metadata for climate simulations include classes that describe the gridded data used in numerical climate models, called Gridspec metadata.[27] The Gridspec describes the discretized data that are the output of the simulation. And this domain also suffers from the problem of having a wide and increasing diversity of model grids used, and the absence of a standard representation of grids. The authors from the Curator project claim that it is rather difficult to perform comparative analyses of data from disparate model grids. However, the Curator project focus on describing the technical details of climate modeling components. They affirm that *the ability to formally describe the scientific aspects of models is appealing because it opens the door to scientific not just technical compatibility of modeling components.*

In the case of the Gridspec project, one solution for describing the scientific aspects of models is the annotation. However, the Curator project does not present a domain ontology with the vocabulary that could be used by professionals to describe the models. But since Gridspec is a metadata for climate simulation, they could easily apply the SWEET ontology (Raskin and Pan, 2005), defined by NASA. The SWEET ontology define concepts used in Earth and Environmental sciences in a modular design. SWEET ontologies describe fields such as Ocean, Atmosphere, Substances, and so on. We try here to make up an example of how a Gridspec dataset could be annotated with a concept of the SWEET ontology. This is illustrated on Figure 4.8.

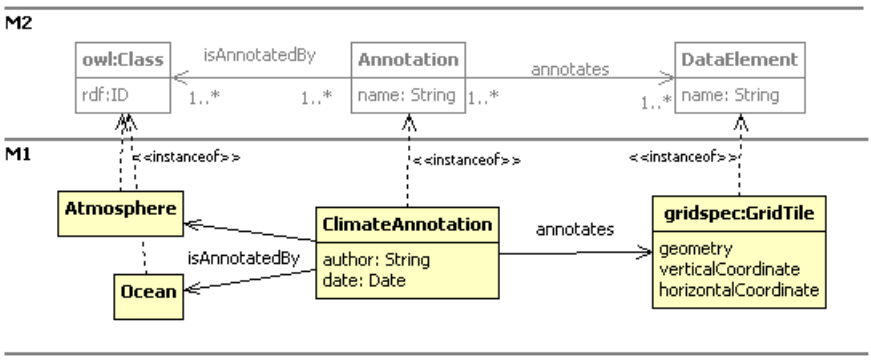


Figure 4.8: The Gridspec metadata represented as an instance of the Annotation Metamodel

Explanation. Figure 4.8 shows the Metamodel for Annotation as the M2 layer of the framework (nomenclature of layers inspired from the MOF metamodel - *cf.* Section 4 of Chapter 2), together with the OWL construct for ontology concepts (*owl:Class*). In the M1 layer we have: the ontology concepts *Atmosphere* and *Ocean*, which are built using the *owl:Class* construct. The Gridspec metadata can be seen as

[27] http://www.gfdl.noaa.gov/ vb/gridstd/gridstd.html

an instance of the *Engineering Metamodel*. Thus, the *GridTile* class from the Gridspec metadata is represented as an instance of the *DataElement* entity, in the *Engineering Metamodel*. Finally, an annotation can be made over the *GridTile* class, which links it to the *Atmosphere* or *Ocean* concepts, according to the interpretation that is made about the data inside the grid. The annotation is created as an instance of the *Annotation* entity, and the properties *author* and *date* are built with the *AnnotationProperty* entity. The instances of the M1 elements should be placed in the M0 layer, where we would have: (i) the actual grid data, (ii) the instances of the ontology and (iii) the actual annotations made by the professionals. An annotation instance would describe the author who performed the annotation and the date when it was performed. At this point, we would be able to create multiple annotations for one same grid object, each author giving its own opinion about the grid data.

The example above shows how to apply metamodels for annotation to a real domain. The *Annotation* construct can be used to define *typed-annotations* in the M1 layer: we define that some data element *D* is annotated by an ontology concept *C*. This is useful because within a specific task, independently of the tool or application that is used to perform this task, some types of datasets are interpreted always using the same ontology concepts (*C annotates D*). A grid dataset can be interpreted as atmosphere data or as ocean data, depending on the activity that is employing the grid. In each of the activities it is possible to define one typed-annotation that links the grid dataset to the concept.

1.2.5 Attributes of an annotation

There is no agreed-upon convention for the structure of annotations (*cf.* discussion in Section 2.3.2 of Chapter 2). The advantage of having an abstract metamodel for annotation is that we can create various Annotation data models, for the various domains.

Using the construct *AnnotationProperty* it is also possible to define as many properties as needed when defining a typed-annotation. In the above described examples, the properties *author* and *date* were created, but other properties could also have been defined, such as properties for storing the annotation version, its status, the activity in which the annotation was generated, the tools used during the activity, and so on. An expanded set of annotation properties could reuse terms from core vocabularies. Some examples are (1) the Dublin Core initiative (Core, 2000), which standardizes terms that define the characteristics of published resources, such as its title, creator, coverage, or publication date, with extensions for rights, permissions, digital right management; and (2) the FOAF vocabulary (Brickley and Miller, 2000), which covers terms needed for describing professional collaboration such as people and their relationships, organizations and projects. The FOAF vocabulary can be used for formally describing people involved in engineering activities. The annotation would then make a reference to the unique identifier of the person that interpreted the dataset. Figure 4.9 illustrates a possible configuration for some typed-annotation enriched with properties from the Dublin Core and FOAF vocabularies.

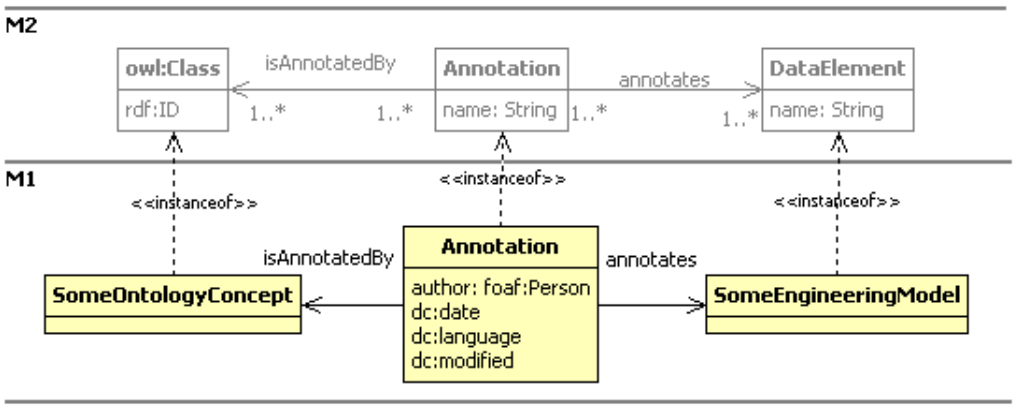


Figure 4.9: Annotation enriched with properties from the Dublin Core and FOAF vocabularies

2 Design of the approach based on model annotation in OntoDB

The environment for the implementation of the model annotation-based approach is the OntoDB ontology-based database (*cf.* Section 3 of Chapter 3). The meta-schema of the OntoDB system has been extended with the constructs proposed in the Engineering Metamodel and in the Annotation Metamodel. OntoQL was used as the exploitation language for handling the OntoDB meta-schema.

2.1 Extension of the OntoDB meta-schema

We recall, on Figure 4.10 the original architecture of OntoDB, with its four related parts: (1) metabase, (2) meta-schema, (3) ontology and (4) instance.

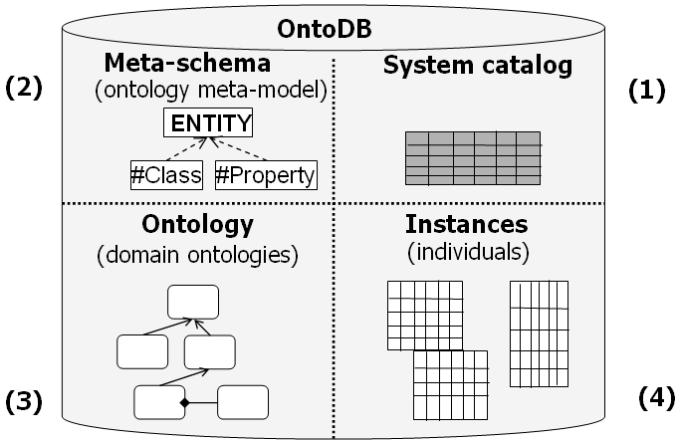


Figure 4.10: OntoDB architecture

2.1.1 The Engineering Metamodel

The first step is to modify the meta-schema of OntoDB in order to add the entities representing engineering models. As it was explained in Section 3.2 of Chapter 3, the OntoQL language is able to handle the set of primitives defined in the meta-schema of OntoDB. The OntoQL expressions make use of the *CREATE ENTITY* clause in order to create the primitives of the Engineering Metamodel in OntoDB's meta-schema. We explain as follows the definition of the entities of the Engineering Metamodel (illustrated on Figure 4.1 on page 84).

The entity ***#DataElement*** is created, independently of the ontology entity *#Class*, having an attribute *#name* of type String.

```
CREATE ENTITY #DataElement (
  #name STRING )
```

The entity ***#DataClass*** is created as a sub-entity of *#DataElement*, thus it inherits the attribute *#name*. The attribute *#subtype_of* makes a reference to another *#DataClass* and the attribute *#formatReader* is of type String.

```
CREATE ENTITY #DataClass UNDER #DataElement (
  #subtype_of REF (#DataClass)),
  #formatReader STRING
```

The entity ***#DataAttribute*** is created as a sub-entity of *#DataElement*. The attribute *#range* makes a reference to the primitive *#DataType*, which already exists in the meta-schema of OntoDB. The attribute *#scope* makes a reference to some *#DataClass*, the attribute *#min* is of type Integer, and the attribute *#max* is of type String (because the maximum cardinality is expressed by a character, such as 'N').

```
CREATE ENTITY #DataAttribute UNDER #DataElement (
  #range REF(#DataType),
  #min INT,
  #max STRING,
  #scope REF(#DataClass))
```

The entity ***#DataAssociationEnd*** is created as a sub-entity of *#DataElement*. The attribute *#type* makes a reference to some *#DataClass*, the attribute *#min* is of type Integer, the attribute *#max* is of type String (because the maximum cardinality is expressed by a character, such as 'N'), and the attribute *#aggregationType* is of type String (because the type of aggregation is defined by keywords, such as "composite" or "none").

```
CREATE ENTITY #DataAssociationEnd UNDER #DataElement (
  #aggregationType STRING,
  #min INT,
  #max STRING,
  #type REF (#DataClass))
```

The entity ***#DataAssociation*** is created as a sub-entity of *#DataElement*. The attributes *#connection1* and *#connection2* make references to some *#DataAssociationEnd*.

```
CREATE ENTITY #DataAssociation UNDER #DataElement (
  #connection1 REF(#DataAssociationEnd),
  #connection2 REF(#DataAssociationEnd))
```

2.1.2 The Annotation Metamodel

The next step consists in modifying the OntoDB's meta-schema in order to add the primitives from the Annotation Metamodel.

The entity ***#Annotation*** is created, independently of the ontology entity *#Class*, having an attribute *#name* of type String. The attribute *#annotates* makes a reference to the engineering metametamodel primitive *#DataElement*, the attribute *#isAnnotatedBy* makes a reference to the ontology primitive *#Class*. This establishes the link between an entity from the engineering metamodel and the ontology metamodel. The attribute *#property* makes a reference to *#AnnotationProperty*, which enables to add more contextual information to the annotation.

```
CREATE ENTITY #Annotation (
  #name STRING,
  #annotates REF (#DataElement),
  #isAnnotatedBy REF (#Class),
  #property REF (#AnnotationProperty))
```

The entity ***#AnnotationProperty*** is created having an attribute *#label* of type String. The attribute *#range* makes a reference to *#PrimitiveType*, because the properties of annotations are not supposed to be of Reference type.

```
CREATE ENTITY #AnnotationProperty (
  #label STRING,
  #range REF(#PrimitiveType))
```

The entity ***#RelationAnnotation*** is created, having an attribute *#name* of type String. The attribute *#annotates* makes a reference to the ontology primitive *#Property*, the attributes *#from* and *#to* make references to the ontology primitive *#Class*. The attribute *#property* is a reference to some *#AnnotationProperty*. This creates a means of adding contextual information to the creation of an ontology property.

```
CREATE ENTITY #RelationAnnotation (
  #name STRING,
  #annotates REF (#Property),
  #from REF (#Class),
  #to REF (#Class)),
  #property REF (#AnnotationProperty)
```

2.1.3 The extended architecture of OntoDB

Figure 4.11 illustrates the architecture of OntoDB after the extension of the meta-schema with the new primitives.

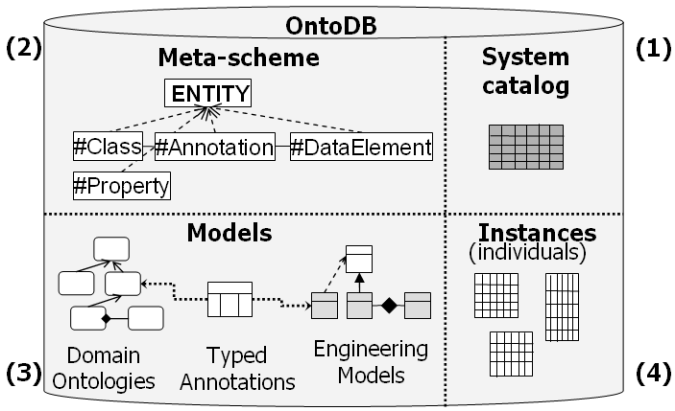


Figure 4.11: OntoDB architecture extended with Engineering and Model Annotation primitives

The OntoQL scripts, when executed, modify the OntoDB meta-schema (part 2 of Figure 4.11), which is equivalent to the *metamodel* layer of the MOF, adding new entities and attributes as instances of *#ENTITY* and *#ATTRIBUTE* respectively. In part 3 of Figure 4.11, which is equivalent to the *model* layer of the MOF, the Engineering Metamodel primitives allow one to define *engineering models*, which are not ontologies; and the Annotation Metamodel primitives enable the definition of *annotations* that link the engineering models to the domain ontologies, and also annotations over the ontology properties.

Thanks to the new entities added to its meta-schema, OntoDB will be able to store within a single data base, the engineering models data and their ontology-based annotations. This extended meta-schema will be employed to create the engineering models and annotations for the real case studied in this thesis. The implementation of the case study is described in the Chapter 8.

2.2 Example of use of the new OntoDB primitives

2.2.1 Creation of engineering model

The metadata of the SP3 model illustrated on Figure 4.5 on page 87 can now be included as an engineering model into OntoDB using the recently-added primitives. We present some examples of OntoQL scripts that show how to create the SP3 model elements as instances of Engineering Metamodel primitives.

In the following expression, the *INSERT INTO* clause is used for creating the SP3File type as an instance of the *#DataClass* entity. Notice that the clause *CREATE* could also be used for creating the SP3 model (e.g. *CREATE #DataClass SP3File*). In that case, the creation of a data element would be analogous to the creation of an ontology concept, using the *CREATE #Class* clause. However, the semantic processing of the OntoQL language would have to be changed in order to take into consideration the fact that the new clause *CREATE #DataClass*, when executed, creates an instance in the table *#DataClass*. Using the *INSERT INTO* clause, this operation is directly executed, without needing to change the language semantics.

```
INSERT INTO #DataClass (#name, #subtype_of)
```

```
VALUES ('SP3File',
  (select #oid from #DataClass
    WHERE #name = 'DataFile'))
```

For the same reason as the one cited above, the *attributes* of a SP3File are created as instances of the *#DataAttribute* element with the *INSERT INTO* clause. In the following expression, the attribute *numberOfSatellites* has as range a *#DataType* whose name is 'Integer'; has minimum and maximum cardinality of 1; and has as scope the instance of *#DataClass* whose name is 'SP3File'.

```
INSERT INTO #DataAttribute (#name, #range, #min, #max, #scope)
VALUES ('numberOfSatellites',
  (select #oid from #DataType
    WHERE #name = 'Integer'),
    1, '1',
  (select #oid from #DataClass
    WHERE #name = 'SP3File'))
```

An association between two data classes is created as an instance of the entity *#DataAssociation*. In the following expression, the association *hasCoordinates* gathers the two data association ends that are respectively connected to the data classes named *SP3File* and *Coordinate*.

```
INSERT INTO #DataAssociation (#name, #connection1, #connection2)
VALUES ('hasCoordinates',
  (select end.#oid from #DataAssociationEnd AS end
    WHERE end.#name = 'end1'
    AND end.#type= (select #oid from #DataClass
                  WHERE #name = 'SP3File')),
  (select end.#oid from #DataAssociationEnd AS end
    WHERE end.#name = 'end2'
    AND end.#type= (select #oid from #DataClass
                  WHERE #name = 'Coordinate')))
```

2.2.2 Creation of annotation types

The Annotation metamodel enables us to create *annotation types* in part 3 of the OntoDB architecture (Figure 4.11), which is equivalent to the *model* layer of the MOF. An annotation type is an instance of *#Annotation* that links specific ontology concepts to specific engineering models. To illustrate this, we reuse the example shown on Figure 4.8 on page 91, in which the Gridspec dataset is annotated with a concept of the SWEET ontology.

The following expression shows the annotation type *ClimateAnnotation*, created as an instance of the entity *#Annotation*. It annotates the engineering model *GridTile* with the ontology concepts *Atmosphere* or *Ocean*. The annotation properties are *author* and *date*.

```
INSERT INTO #Annotation (#name, #annotates, #isAnnotatedBy, #property)
VALUES ('ClimateAnnotation',
  SELECT #oid from #DataClass WHERE #name = 'GridTile',
  SELECT #oid from #Class WHERE #name = 'Atmosphere' OR 'Ocean',
```

```
SELECT #oid from #AnnotationProperty WHERE #name = 'author' OR 'date')
```

The following expression illustrates the creation of the annotation property *author*.

```
INSERT INTO #AnnotationProperty (#label, #range)
VALUES ('author',
  select #oid from #DataType where #name = 'String')
```

The instances of the annotation type *ClimateAnnotation* are restricted to annotate instances of the *GridTile* model using one of the concepts *Atmosphere* or *Ocean*. Defining annotation types is useful for restricting the domain and range of annotations, and for defining categories of annotations that are always performed in some specific engineering activity.

2.2.3 Querying the model annotations

The users are able to ask queries about an engineering model (e.g. all files that are part of a model) and also about the annotations, as we show with the following OntoQL queries.

Query: select all ontology concepts that annotate the data element *GridTile*.
Answer: *Atmosphere*, *Ocean*.

```
SELECT #Annotation.#isAnnotatedBy.#name FROM #Annotation
WHERE #Annotation.#annotates.#name = 'GridTile'
```

Query: select the name of the annotation types that uses the ontology concept *Ocean* to annotate models.
Answer: *ClimateAnnotation*.

```
SELECT #Annotation.#name FROM #Annotation
WHERE #isAnnotatedBy.#name = 'Ocean'
```

3 Conclusion

In this chapter we described a first step for ensuring semantic compatibility between engineering models. We presented a means of linking objects in engineering models to their ontological meaning. We proposed that the expert's interpretation about some data set be represented as an *annotation*. The annotation is an element that makes the link between the *data elements* and the *ontology concepts* that give meaning to the data.

In order to keep it a general approach, that can be applied to different domains, we used meta-modeling techniques and proposed a *metamodel for the engineering models* and a *metamodel for the annotation*. The Engineering Metamodel allows creating explicit and formal representation of the data models and of the actual files used by engineering models. It can be used as a common metamodel for operating transformations between different file formats.

The Annotation Metamodel transforms the *annotation* into a separate, explicit entity which is placed in the same abstraction level than the constructs for ontologies and data elements. In typical semantic-annotation approaches, annotation is defined as an instance of the ontology concept that is attached on the

resource and consequently it does not exist separately from the ontology. In our approach, annotation is a top-level construct separated from the ontological concepts. This approach allows us adding attributes that characterize the annotation (such as author, timestamp, version) while physically separating it from the annotated resources. The aim of the proposed conceptualization is showing that annotation should be made an explicit construct in domains where we can have many different opinions about a same set of objects. It is then possible to externalize these opinions by means of the annotation. As a result, the knowledge base maintains all the possible different interpretation made by experts about the domain objects.

We showed that this approach can be naturally implemented in the OntoDB ontology-based database. OntoDB is built on different layers, from the meta-scheme to the instance layer. The meta-schema layer of OntoDB was extended with the constructs proposed in the Engineering and Annotation Metamodels. The *DataElement* and *Annotation* entities were created in the same level as the *Class* entity, which is used for building ontologies. The *Annotation* entity makes the link between *DataElement* and *Class* by means of the relationships *isAnnotatedBy* and *annotates*. We demonstrated with some tool examples that with this approach, the user is able to ask queries about the engineering models structure, about the files used by engineering models, and also about the annotations.

We will show in later chapters that this approach can be also implemented in a *Semantic Web* paradigm, for example, when using OWL language. In that case, the *DataElement* and *Annotation* entities would become OWL classes, that is, *instances* of the *owl:Class* primitive. Consequently, the engineering model and annotation constructs would not be at the same level of the ontology construct. The consequence is that they would be part of the ontology, and would be included in all reasoning and inference operations. However, since they are *annotation information*, they would not provide useful information for these inference methods.

An RDF-based approach would consequently lack the *metamodel level*, which makes the methodology generic and thus applicable to other domains. This is due to the fact that RDF-based ontologies cannot have their metaclasses modified, unless they become OWL Full ontologies (*cf.* Section 4.3.1 of Chapter 2). However, OWL Full ontologies loose some guarantees (such as the decidability of basic inference) which OWL DL and OWL Lite provide for reasoning systems and which constitute an advantage when developing OWL ontologies.

Chapter 5

Ontology-Based Integration of Engineering Models: an *A posteriori* Approach

1 The problem of aligning different expertise fields to a federating domain

In the workflow of petroleum reservoir characterization (detailed later on in Section 1 of Chapter 6), one of the main issues is to determine how to create correspondences among each of the specific fields of the Geosciences involved in the task. Scientists involved in the construction of a final reservoir model need to be able to integrate data sets across disciplines related to earth sciences such as geophysics (seismics, well log analysis), geology (stratigraphy, sedimentology, structural geology), petrology, petrophysics or to various other fields (for instance solid modeling or geostatistics). The users expect to be able, within any stage of the workflow, to answer questions that relate geological objects issued from different domains each with the others. Currently, it is not possible to answer this type of question, since the relationship among the objects identified in the various phases of the workflow is not explicit.

It should be noticed that other engineering development processes also involve a set of activities that rely on different expertise fields. This is the case, for example, of Mechatronics. The mechatronics system design activity usually depends on various applications of different specialties. Bi et al. (2008) present an ontology for the domain of mechatronics that stands for an *integrated view of sub-domain ontologies* such as mechanical ontology, hydraulic ontology, control ontology and electronic ontology. There is no obvious integration schema for such different domains. The Mechatronics System Ontology (MSO) acts as a standard representation to which all other domains are linked.

This issue is similar to the multi-disciplinary problem in the reservoir modeling workflow, in which *geology* is seen as the red thread that should guide the modeling process (Rainaud et al., 2005). Mechatronics and Geology domains can be seen as macroscale versions of more specific domains, they act as pivot among the other expertise fields. This configuration can be seen as a problem of *perspectival heterogeneity*, as explained in Section 3 of Chapter 2, in which various models represent various points of view in relation to the same domain of interest. Figure 5.1 illustrates this intuition.

In this work, inspired from real world cases of multi-disciplinary domains, we chose one of the fields to be the *federating* domain that gathers the vocabulary that is shared by the professionals of all the spe-

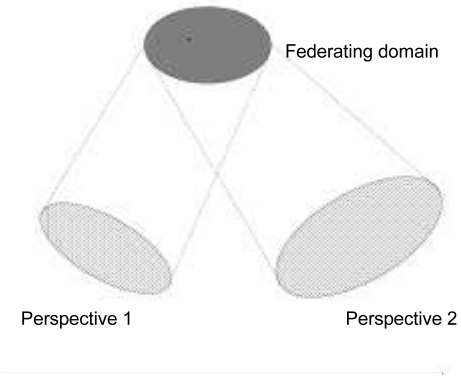


Figure 5.1: Different perspectives about one same domain

cialized domains. The *federating domain* characterizes the engineering domain (like the Mechatronics domain in the example cited above) and makes possible to integrate the data from these various disciplines. When integrated, these data allow the emergence of new knowledge, which is relevant for engineers and essential for timely and correct decision making. In several areas, this integration is still made by engineers themselves and is neither formalized nor computerized.

It is important to notice that users from these domains generally do not expect data to be put together in one same repository. Integration, in this context, means *finding correspondence between entities* from different fields, without merging the corresponding instances. The experts from these domains, in particular those from the petroleum industry, need data to remain where they are, and to keep their original format, which can differ completely from one field to the other. What they aim is to be able to have an integrated vision of the data issued from all the different fields. They need a *semantic-based* integrated vision of the data. The semantic-based integration approach must take into consideration the meaning of the various data for the experts, in order to define mapping rules. Clearly, there is the need of an non-conventional semantic integration approach.

In this context, techniques that provide an automatic matching of concepts issued from different fields may not be the best solution. We are not dealing with structure heterogeneity in ontologies that represent the same domain, but we must, on the contrary, bridge ontologies from fields that are completely different from each others. In this case, classical approaches of automatic recognition of equivalence between concepts by comparison of attributes or lexical analysis are unhelpful. Ontology matching becomes an engineering task, performed by the experts themselves.

One final matter is that ontologies that describe specific domains of expertise might be *reused* instead of being developed from scratch. In this case, bridges between ontologies should not change the structure of local ontologies that are just locally imported. Consequently, it is essential to provide a means of mapping the various ontologies in an *a posteriori* fashion. This means considering that the structure of the ontologies are already set up by the time experts suggest the correspondences between them. The objective is thus to keep ontology development independent from ontology matching.

The issue of data integration is still a challenge when considering activities that rely on heterogeneous fields. Our goal is to provide engineers the infrastructure they need to integrate data from multiple

expertise fields.

1.1 Integration of engineering domains

Domains such as engineering activities and scientific applications depend on very diversified fields, that are not obvious to integrate. These domains are called *complex multiple-world scenarios*, as it was explained in Section 1.2 of Chapter 1. As a consequence, when considering semantic integration of engineering models, we are not trying to integrate different schemas that refer to the same world entities (like the book information ontology). We are rather integrating different kinds of worlds that are completely independent from the others (like mechanics and electronic or solid modeling and geology) and that are each described by their own domain ontology.

1.1.1 Integration structure

In this multi-disciplinary context, two ontology integration structures are possible.

1. a **multi-ontology structure**, in which the correspondence between two ontologies is established directly from one to the other. If there are n ontologies, we need to create $[n * (n-1)/2]$ mappings. There is no upper ontology, therefore, no common access interface to the various ontologies.
2. An **hybrid structure**, in which the correspondence between two ontologies is established indirectly through a reference ontology. If there are n ontologies, we need to create $[n]$ mappings. The reference ontology is either one of the ontologies that need to be integrated, or a normalized ontology independent of the system. The interface of integration can be based on the concepts of the reference ontology, since it represents the global view of all the sources.

In our case, a multi-ontology structure would punctually solve the issue of mapping one domain to some other, but it cannot be a final solution, since engineers need using a unified vision of the system. For this reason, we should rather choose an hybrid structure of local and global ontologies.

In most of the works that aim to integrate ontologies, one upper ontology is chosen and the specific ontologies are mapped to its concepts. Among works that propose independent upper ontologies for these needs, we can cite CyC (Matuszek et al., 2006), DOLCE (Gangemi et al., 2002), and BFO (Grenon et al., 2004) upper ontologies. Upper ontologies describe very general concepts that are the same across different domains (such as Entity, Function, Spatial Region.).

Therefore, these ontologies seem to be the most appropriate choice for integrating the semantically unrelated ontologies of engineering domains. However, if such an upper ontology is used in the integration level, this obliges final users to use very general concepts for defining their queries. In our case, on the contrary, we aim to provide the users a way of defining queries that use *a common vocabulary shared among all specific domains*, but that is still *enough meaningful* to the context.

For this, we propose that the ontology of the *federating domain* be used as the reference ontology, or *global ontology* (GO), and that the other domains be described by independent *local ontologies* (LO). Consequently, the concepts of the various local ontologies will be aligned to the concepts of the global

ontology (*cf.* Figure 5.2).

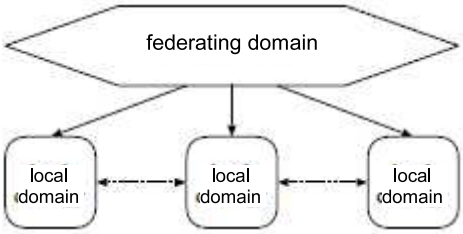


Figure 5.2: Hybrid ontology integration approach

If the user needs to query the sources in a integrated way, he will use the federating domain ontology as the query model. This approach allows any agent participating in an engineering activity to query the local ontologies even though this agent is not an expert of the local domain and does not know the related keywords.

1.1.2 Connecting sources to ontologies

We need to define how to connect engineering models to the ontology-based infrastructure. The proprietary data formats of engineering models need to be translated into semantically comparable intermediate representations. Following the approach proposed and detailed in Chapter 4, each local engineering data set is *wrapped* using the Engineering Metamodel, i.e. the formal and common metamodel for describing data elements.

Wache et al. (2001) say that "In order to achieve semantic interoperability in a heterogeneous information system, the *meaning* of the information that is interchanged has to be understood across the systems". Ontologies have been used for *content explication*, that is, for giving explicit description of the information source semantics. In this work, the various domains involved in the engineering activity are described by domain ontologies. Among these ontologies, those referring to specialized fields are designed as *local ontologies*, and the one referring to the federating domain as the *global ontology*.

In this work, we are applying an *annotation-based approach* for making explicit the semantics of the engineering models. As proposed and detailed in Chapter 4, the Annotation Metamodel describes the connection between local data and local ontologies. The annotation process will be performed manually by the experts when interpreting data from local fields, or semi-automatically by systems that accept to have an integrated annotation tool. Annotation will thus establish a connection between actual data sources and ontologies so that the user will be able to identify the data corresponding to the ontology concepts that he/she has queried.

1.1.3 Architecture for ontology-based integration of engineering domains

Motivated by the issues discussed in the previous section, we describe here an *architecture for ontology-based integration of engineering models*. We characterize actual engineering models as *local data sources*; their representation as instances of the Engineering Metamodel are called *local views*; the con-

nection of the views to their ontological meaning is given by the *annotations*; the specialized expertise domains are described by *local ontologies* and the federating domain by a *global ontology*; the mapping between the global and the local ontologies are called *local-to-global alignments*, and, finally, queries are defined over the local ontologies and over the global ontology. The architecture, illustrated on Figure 5.3, is described as follows.

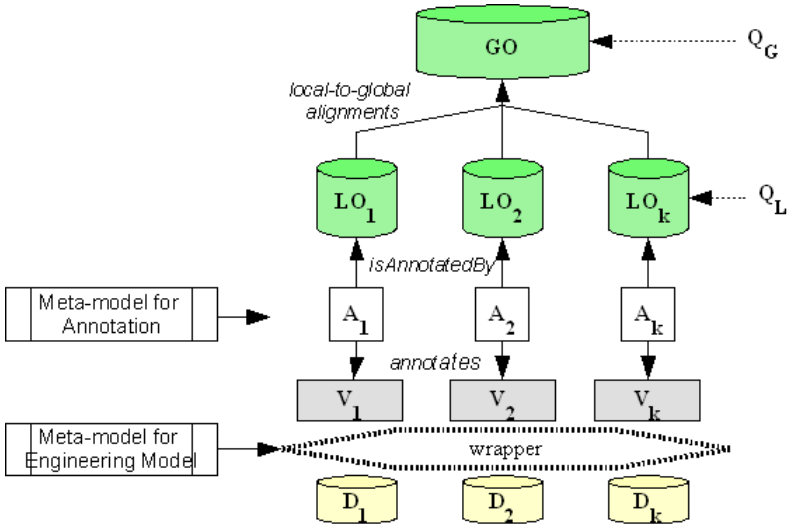


Figure 5.3: The architecture for ontology-based model integration

1. Local data sources (D_i) are *wrapped* in a unified language, the Engineering Metamodel. The result of the wrapping (formal representations of engineering models are local views (V_i) which are accessible to the integration system.
2. *Local ontologies* (LO_i) are set up. Each LO describes the vocabulary used in one specialized field. The concepts expressed in the LOs can be used as query model for the data sets associated to this field.
3. The wrapped local sources are connected to their respective LO by means of the *semantic annotations* links (A_i). Annotation can be manual or semi-automatic. The wrapped local sources and the local ontologies are not modified in this process.
4. A global ontology is set up (*GO*). The vocabulary described in the GO is shared through the various local fields.
5. A set of *mappings* is defined between the concepts in LOs and the concepts in the GO. The ontology matching must be manual, since the various ontologies do not refer to the same area of expertise (this issue is discussed in Section 1.1.4). The structures of the GO and of the LOs are not modified in this process.
6. When a new local data source needs to be integrated, steps 1 to 5 are repeated. New annotations (links from data to ontologies) and mappings (local-to-global links) are added to the knowledge base, but the original structure of the data and ontologies is not changed.
7. Whenever a user formulates queries in terms of the GO concepts (Q_G), the query is propagated to the local ontologies by applying the local-to-global alignments. The user can also formulate

queries in terms of LO concepts (Q_L), since LO structures are not hidden from the user. The results from the queries are joined by the system and presented to the user. The query processing is detailed in Section 1.2.

It is important to emphasize that no attempt is made to integrate all local ontologies against the global ontology. The objective is to establish mappings between local and global concepts. The set up of the architecture requires manual work from the experts, for defining semantic links, and from the database administrators, for wrapping the local data sources. This can generate a bottleneck at the time of the installation of the architecture. Even so, it is essential, in the present approach, to allow experts to provide manual mappings between the ontologies, as explained in the next section.

1.1.4 Manual ontology matching

Most works propose matching approaches that are based on the characteristics of the schema, such as the similarity between features (name, description, type, structure). These techniques aim to infer the semantics of the involved elements from the structure of the information sources. Such techniques are unhelpful when we deal with concepts whose meaning can be orthogonal. In the complex conditions described in this chapter, automated schema matching techniques are not probable to be successful in extracting mapping elements between the LOs and the GO, since they are ontologies developed for multiple-world scenarios. In this context, we consider that only the experts of the specialized fields are capable of providing the correspondence between concepts. The ontology matching is then *manual*. This has significant limitations, but guarantees that the mappings will be meaningful to most of the experts.

Lastly, the set of ontology alignments defined by the experts will not be used to perform any ontology merging. The objective is to integrate the concepts of the local ontologies inside the global ontology, but *not in the sense of merging* (which would create a single ontology with merged concepts, and make the original concepts disappear). Ontology alignments will be used for the sake of allowing users to query information across domains. It means that the GO acts as a *virtual middleware*, it does not materialize the integration of the data.

As a final reflection, we should take into consideration the fact that the matching of two or more ontologies is often subjective, depending on the application. Two different experts from a same domain can provide different mappings between ontologies, depending on the objective of the matching. As presented in Section 2.2.1 of Chapter 2, the term “annotated mapping” defines a mapping element that is annotated with usage-related characteristics. We believe that in the context of semantic integration of engineering domains, the mapping relations between the various ontologies may vary depending on the expert that performs it. In the context of defining manual mappings over ontologies, we should investigate the possibility of mapping relations to be annotated. This study is going to be a future work about the integration framework.

1.1.5 Mapping LO onto GO

In the context described previously, we consider that the general domain and the specific domain express *different perspectives* regarding one same domain of study (the considered engineering domain). The

concepts of these domains are closely related but are not exactly equivalent. It often happens that one concept is actually a specific case of the other, sharing some but not all properties. The concepts from the local ontologies are, thus, considered to be *more specific than* or *included in* or *subsumed by*[28] the concepts of the global ontology.

We believe that in order to map the local and global domains, it is necessary to define a set of *subsumption relations* from the LOs' concepts to the GO's concepts, rather than *equivalence relations*. This is due to the nature of the ontologies involved: they generally describe different perspectives of domains that are very similar. The typical subsumption relation is the *is-a* relation, which presumes total inheritance of properties, while the *is-case-of* relation is a subsumption relation that is not associated to an inheritance mechanism. We believe that, besides the classical alignment relations expressed between concepts of different ontologies, we should define alignments based on subsumption relations, more specifically, the *is-case-of* relation.

1.1.6 The *is-case-of* relation

As explained in Section 1.2.2.1 of Chapter 2 the *is-case-of* relation does not explicitly exist in most of ontology models. In OWL it is possible to simulate partial inheritance, because properties do not have a restricted domain. The knowledge engineer can choose that one class have some of the properties that another class already has. But it is not possible to represent the fact that one class is partially included into the other. The *is-case-of* relation is proposed as an explicit subsumption relation in the *PLIB ontology model* (Pierra, 2004). It is not associated to an automatic inheritance mechanism, which gives a higher level of independence to the classes that are supposed to be defined by different sources and that have different life cycles.

In multi-world domains such as engineering domains, the *is-case-of* relation between two concepts, such as *(B is-case-of A)*, expresses the situation where the expert interprets B as being a specific case of the concept A. It means that instances of the concept B are also considered to be instances of the concept A, even if they do not share the same properties. In ontologies that implement the PLIB model, the *is-case-of* relation is a *built-in primitive*. As explained in Section 3.3.2.1 of Chapter 2, languages based on the OWL ontology model have just the equivalent primitive for handling correspondences. We propose that the *is-case-of* relation be used as a mapping relation in engineering domains.

Using a relationship embedded in the ontology language for establishing mappings has pros and cons. As it was already demonstrated in Section 3.3.2.1 of Chapter 2, the drawback of using an embedded primitive for expressing mappings is that it forces the use of a particular language for representing ontologies. In the present case, only ontologies that implement the PLIB model are able to connect their concepts using the *is-case-of* relation.

On the other hand, it enables the exploitation of a mechanism which can directly navigate through the hierarchy of aligned concepts, in the same way we can navigate through an *is-a* hierarchy. This aspect will favor the translation of queries from one domain to the other, as it will be explained further on. Another advantage of using the *is-case-of* relation for establishing the correspondence between different

[28] A subsumption relationship is an implication relation that links more specific to more general concepts (*cf.* Section 3 of Chapter 2).

ontologies is that it does not need to be defined at the design time, like the *is-a* relation. The user is able to create mappings at run-time. This corresponds to an *a posteriori* approach.

1.1.7 *A posteriori* approach of integration

We aim to integrate heterogeneous sources maintaining, however, their inter-independence. With this objective, the matching process between ontologies must be performed *a posteriori* .[29]

An *a posteriori* integration approach is characterized by the following principles (Xuan et al., 2006):

- the ontologies to be integrated are independent from each other and are previously defined;
- the concepts of the ontologies are put into correspondence by means of external bridges, which do not change their internal structure.

It means that the structure of the ontologies is already set up when the experts suggest the correspondence between them. That differs from an *a priori* approach, in which the mappings are defined in the *ontology design time* and are *embedded in the ontology definition* (Bellatreche et al., 2004). The *is-case-of* relation is an asset when setting up an *a posteriori* approach, since it is a built-in primitive that can be defined subsequently to the design of the ontology. It is possible to keep the ontology development independent of the ontology matching.

1.2 Query processing

In data integration systems, a mediator code written by the system designer is responsible for performing the query-rewriting mechanism, which is responsible for evaluating mappings, executing queries in various local ontologies and joining results.

In the architecture set up in Section 1.1.3, local ontologies are mapped to the global ontology, which describes a federating domain, the global ontology. In this situation, the global ontology can assume the role of *query model*. The users will be able to chose concepts from the GO to formulate queries about specialized fields or of the LO if they have more knowledge about the specific concepts. The structure of the query will be more intuitive for *users that are not experts of some specific field*.

We propose to use a *subsumption* relationship to map the concepts of the GO and LOs. When concepts of the local ontologies will be represented as being *is-a* or *is-case-of* of concepts of the global ontology, it will be possible to query the local source in terms of the referenced global ontology.

We describe in the next section the extension of an ontology query language in order to support the navigation on hierarchies produced by the *is-case-of* relation.

[29] In this work we use the expression *a posteriori* meaning *what comes after*.

2 Design of knowledge integration operators in OntoQL

The OntoQL language is composed of a syntactic part and a semantic part. The syntactic part is the grammar of the language, which refers to the logical and structural rules that govern the composition of expressions in OntoQL. The semantic part corresponds to the interpretation of the OntoQL expression as operations to be executed on the database. The complete syntax of OntoQL language, its lexical elements and grammar rules, and the algebra that defines the semantics of OntoQL expressions are detailed in Jean (2007).

We describe in this section how the grammar and semantics of the OntoQL data definition language were modified in order to include the *is-case-of* operator, and also how the OntoQL query language was changed to take into account the hierarchy of *is-case-of* concepts.

2.1 Extension of the OntoQL definition language with *is-case-of* operators

As explained in Section 1.1.6, the *is-case-of* operator, available in PLIB model to articulate different concepts or concepts of different ontologies, defines a subsumption relation that is not associated to inheritance of properties. The class declaring itself *a case of* another, must explicitly import the needed properties from the class of which it is a case. Moreover, we understand that there are two approaches for setting up *is-case-of* relation: the *a priori* approach, in which the relation is created in the *ontology design time*, and the *a posteriori* approach, in which the relation is created subsequently to the design of the ontology. Considering a concept which is case of another concept, we call the first one as *subsumed concept* and the second one *subsuming concept*. Using these definitions, we propose two types of *is-case-of* operations for the OntoQL language:

1. ***a priori case-of*** : an ontology concept (subsumed concept) is created as being case of another concept (subsuming concept). The subsumed concept explicitly defines the properties to be imported from the subsuming concept.
2. ***a posteriori case-of*** : after the creation of the ontology concepts, an *is-case-of* relation is created between two concepts. Some properties of the subsumed concept (which already exist) are *mapped* to the chosen properties of the subsuming concept.

The OntoQL grammar is defined using production rules, that is, the left hand side statement produces the right hand side statements. Statements in square brackets are optionally produced. The statement ⟨*element list*⟩ represents a list of ⟨*element*⟩, and its definition is ⟨*element*⟩ { , ⟨*element*⟩}. The lexical elements in the right hand side can be statements, expressions, identifiers, lists, terminal tokens, symbols. An example of production rule is given as follows:

⟨*statement*⟩ ::= `KEYWORD` ⟨*other statement*⟩ [⟨*ident*⟩ = ⟨*expr*⟩] [⟨*statement list*⟩]

We detail in the next sections how each one of the proposed operations are introduced in the OntoQL grammar.

2.1.1 Extension of the OntoQL grammar to support *a priori case-of* relation

The *a priori case-of* operator is intended to be used in the moment of the creation of an ontology concept. As explained in Chapter 3, the OntoQL language works on top of the OntoDB architecture, which is mainly based on PLIB model. The PLIB model contains primitives that support the creation of a *is-case-of* relation. Figure 2.4 on page 44 illustrates the PLIB model, and shows the primitive *Item_class_case_of* which represents *a class that is case of* another class. Consequently, the only work to be done is to include the *is-case-of* relation as an operator in the OntoQL language. For this, we must modify the original grammar of OntoQL. In the OntoQL language, the syntax for the creation of an ontology concept is defined as follows (Jean, 2007):

⟨*class definition*⟩ ::= `CREATE` ⟨*entity id*⟩ ⟨*class id*⟩ [⟨*under clause*⟩]
[⟨*descriptor clause*⟩] [⟨*properties clause list*⟩]

⟨*under clause*⟩ ::= `UNDER` ⟨*class id list*⟩

⟨*descriptor clause*⟩ ::= `DESCRIPTOR (` ⟨*attribute value list*⟩ `)`

⟨*attribute value*⟩ ::= ⟨*attribute id*⟩ = ⟨*value expression*⟩

⟨*properties clause*⟩ ::= ⟨*entity id*⟩ `(` ⟨*property definition list*⟩ `)`

⟨*property definition*⟩ ::= ⟨*prop id*⟩ ⟨*datatype*⟩ [⟨*descriptor clause*⟩]

Grammar 5.1: Original grammar for CREATE statement

The ⟨*class definition*⟩ element produces the *CREATE #Class* instruction, followed, optionally, of the clauses ⟨*under clause*⟩, ⟨*descriptor clause*⟩ and of a list of ⟨*properties clause*⟩. The ⟨*class definition*⟩ element was modified as follows in order to include a new clause in the right hand side: the clause for defining the *is-case-of* operator.

⟨*class definition*⟩ ::= `CREATE` ⟨*entity id*⟩ ⟨*class id*⟩ [⟨*under clause*⟩]
[⟨*caseof clause*⟩]
[⟨*descriptor clause*⟩] [⟨*properties clause list*⟩]

⟨*under clause*⟩ ::= `UNDER` ⟨*class id list*⟩

⟨*descriptor clause*⟩ ::= `DESCRIPTOR (` ⟨*attribute value list*⟩ `)`

⟨*attribute value*⟩ ::= ⟨*attribute id*⟩ = ⟨*value expression*⟩

⟨*properties clause*⟩ ::= ⟨*entity id*⟩ `(` ⟨*property definition list*⟩ `)`

⟨*property definition*⟩ ::= ⟨*prop id*⟩ ⟨*datatype*⟩ [⟨*descriptor clause*⟩]

⟨*caseof clause*⟩ ::= ⟨*caseof class clause*⟩ [⟨*import clause*⟩]

⟨*caseof class clause*⟩ ::= `ISCASEOF` ⟨*class id list*⟩

⟨*import clause*⟩ ::= `IMPORTS` ⟨*prop id list*⟩

Grammar 5.2: Modified grammar for CREATE statement: ⟨*caseof clause*⟩

The heading of this instruction begins with the '`CREATE`' literal, followed by the name of the entity, e.g.

#Class and by the class identifier (the name of the class being created). The ⟨*caseof clause*⟩ is optional for the creation of a class. For producing the ⟨*caseof clause*⟩, one must make use of the 'ISCASEOF' literal, followed of one or many class identifiers (the names of the subsuming classes). This constitutes the ⟨*caseof class clause*⟩. The ⟨*import clause*⟩ begins with the 'IMPORTS' literal, followed by one or many property identifiers; which are the names of the properties that the subsumed class wants to import from the subsuming classes.

Let us now consider concepts *A* and *B* already exist, with their respective properties *a1* and *b1*, as pictured in Figure 5.4. Concept *C* is created as a case of both concepts *A* and *B*, importing respectively properties *a1* from *A* and *b1* from *B*.

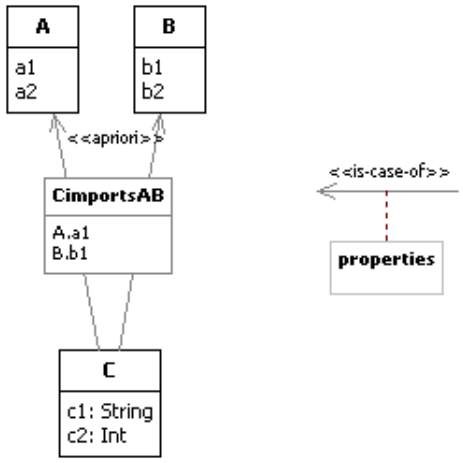


Figure 5.4: Example of *a priori case-of*

An example of valid OntoQL expression for creating the concept *C* as case of *A* and *B* can be given as follows.

```
CREATE #Class C ISCASEOF (A, B) (
  IMPORTS(A.a1, B.b1)
  PROPERTIES (c1 STRING, c2 INT))
```

This expression creates an ontology concept *C* which is a case of the concepts *A* and *B*, and imports property *a1* from *A* and property *b1* from *B*. The concept *C* specifies in addition its own properties, *c1* and *c2*. Alternatively, a *is-case-of* class could be created without importing properties from the subsuming class(es).

2.1.2 Extension of the OntoQL grammar to support *a posteriori case-of* relation

The *a posteriori case-of* relation has a particular behavior: it is intended to be created *after* the creation of the concepts of interest, thus, the original definition of the used concepts cannot be changed. The consequence is that the subsumed concept cannot import properties from the subsuming concept(s). The solution proposed by the PLIB model is to define an independent entity between the classes that are

related through the *is-case-of* relation; and to create *mappings* between their properties. However, the *a posteriori case-of* relation is not originally defined in the OntoQL model. It means that besides changing the OntoQL grammar to include an *a posteriori case-of* operator, we must create a new entity in the same level of the original entities *Class* and *Property*: the *AposterioriCaseOf* entity. The sucessive steps followed in order to support the *a posteriori case-of* relation are as follows:

- analysis of the OntoQL grammar and of the OntoDB entities;
- definition of the *a posteriori case-of* entity and the pair of map properties entity in OntoDB;
- extension of the OntoQL grammar adding the *a posteriori case-of* operator.

2.1.2.1 Creation of the *AposterioriCaseOf* entity The *a posteriori case-of* relation needs to be created as an *association* between concepts, but it cannot be represented as an ordinary binary relation. The *a posteriori case-of* relation has to be an independent entity, that relates one concept to a list of other concepts (the subsuming concepts), and gathers the list of properties that are mapped among concepts. The mapping operation will create a correspondence between a property of the subsumed concept and a property of one of the subsuming concepts. The conception of the new entity *AposterioriCaseOf* to be created is given by the UML diagram of Figure 5.5.

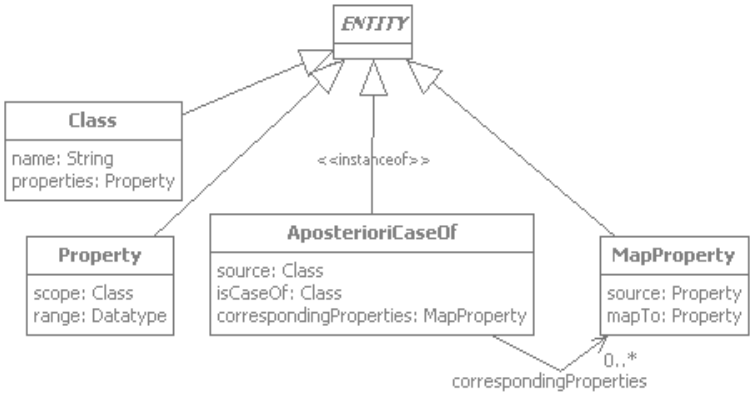


Figure 5.5: The new OntoQL entity *AposterioriCaseOf*

The *AposterioriCaseOf* is an instance of the *ENTITY* construct. The attribute *source* represents the source (subsumed) *concept*, while the attribute *isCaseOf* makes a reference to the concept that is subsuming. The link *correspondingProperties* makes a reference to the entity *MapProperty*, which represents a pair of properties: the subsumed property is represented by the attribute *source*, whose range is the subsumed concept, and the subsuming property is represented by the attribute *mapTo*, whose range is the subsuming concept.

The *MapProperty* and *AposterioriCaseOf* entities were created respectively with the following OntoQL expressions, which make use of the *CREATE ENTITY* clause to create top level entities.

```
CREATE ENTITY #MapProperty (
  #source REF(#Property),
```

```
  #mapTo REF(#Property))

CREATE ENTITY #AposterioriCaseof (
  #source REF(#Class),
  #isCaseOf REF(#Class),
  #correspondingProperties REF(#MapProperty))
```

The attributes *#source* and *#mapTo* of *#MapProperty* make reference to the *#Property* entity. The attributes *#source* and *#isCaseOf* of *#AposterioriCaseof* make reference to the *#Class* entity and the attribute *#correspondingProperties* references the *#MapProperty* entity.

Thanks to the new entities included in the OntoQL metamodel, the *a posteriori case-of* relation can be created between two ontology concepts. Let us consider, for example, a domain ontology comprising a concept A, which has a property *propA*, and a concept B, which has a property *propB*, as pictured in Figure 5.6.

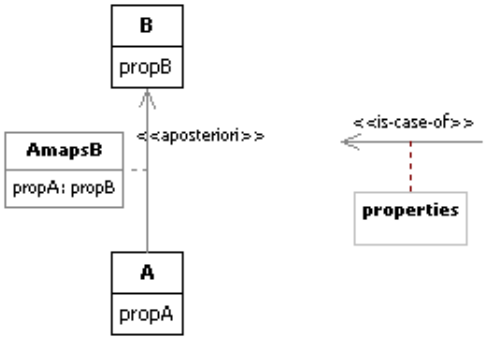


Figure 5.6: Example of *a posteriori case-of*

In order to make A a case of B, and to map property *propA* to property *propB*, one first needs to create the pair of properties, as follows.

```
INSERT INTO #MapProperty (#source, #mapTo)
VALUES (
  (select #oid from #Property where #name = 'propA1'),
  (select #oid from #Property where #name = 'propB1'))
```

The above expression creates a correspondence between the properties *propA* and *propB*. The next step is to create the relation between the two classes, by means of the *AposterioriCaseOf* entity.

```
INSERT INTO #AposterioriCaseof (#source, #isCaseOf, #correspondingProperties)
VALUES (
  (SELECT #oid FROM #class where #name = 'A'),
  (SELECT #oid FROM #class where #name = 'B'),
  ARRAY(SELECT #oid FROM #MapProperty
        where #source.#scope = 'A' and #mapTo.#scope = 'B'))
```

The above expression creates an instance of the *#AposterioriCaseof* entity defining the concept A (*#source*)

as case of the concept *B* (*#isCaseOf*). The pairs of properties are retrieved from the *#MapProperty* entity, by searching for the scope of the property.[30]

The creation of the *a posteriori case-of* relation using the *INSERT INTO* clause was necessary at this point because the OntoQL grammar was not yet changed to support the creation of this new type of relation. The grammar modification is explained in the next section.

2.1.2.2 Modification of the OntoQL grammar The syntax for creating the *a posteriori case-of* relation in OntoQL is defined by the same grammar used for creating classes (*cf.* Section 2.1.1). This grammar has been modified in order to include the clause for the *a priori case-of* relation. It must now be extended again in order to include a clause for creating the *a posteriori case-of* relation. The resulting grammar is as follows.

⟨*class definition*⟩ ::= `CREATE` ⟨*entity id*⟩ ⟨*class id*⟩ [⟨*under clause*⟩]
[⟨*caseof clause*⟩] [⟨*apostcaseof clause*⟩]
[⟨*descriptor clause*⟩] [⟨*properties clause list*⟩]

⟨*under clause*⟩ ::= `UNDER` ⟨*class id list*⟩

⟨*caseof clause*⟩ ::= ⟨*caseof class clause*⟩ [⟨*import clause*⟩]

⟨*caseof class clause*⟩ ::= `ISCASEOF` ⟨*class id list*⟩

⟨*import clause*⟩ ::= `IMPORTS` ⟨*prop id list*⟩

⟨*descriptor clause*⟩ ::= `DESCRIPTOR (` ⟨*attribute value list*⟩ `)`

⟨*attribute value*⟩ ::= ⟨*attribute id*⟩ = ⟨*value expression*⟩

⟨*properties clause*⟩ ::= ⟨*entity id*⟩ `(` ⟨*property definition list*⟩ `)`

⟨*property definition*⟩ ::= ⟨*prop id*⟩ ⟨*datatype*⟩ [⟨*descriptor clause*⟩]

⟨*apostcaseof clause*⟩ ::= ⟨*apostcaseof class clause*⟩ [⟨*map clause*⟩]

⟨*apostcaseof class clause*⟩ ::= `CASEOF` ⟨*class id list*⟩

⟨*map clause*⟩ ::= `WITH` ⟨*map definition list*⟩

⟨*map definition*⟩ ::= ⟨*prop id*⟩ `MAP` ⟨*prop id*⟩

Grammar 5.3: Modified grammar for CREATE statement: ⟨*apostcaseof clause*⟩

The heading of this instruction begins with the '`CREATE`' literal, followed by the name of the entity, in this case, the *#AposterioriCaseOf* entity, and by the class identifier (the name of the subsumed class). For producing the ⟨*apostcaseof class clause*⟩, one must make use of the '`CASEOF`' literal, followed by one or many class identifiers (the names of the subsuming classes). The '`WITH`' literal begins the ⟨*map clause*⟩, followed by a list of ⟨*map definitions*⟩, which is a pair of property identifiers linked by the '`MAP`' literal. The first ⟨*prop id*⟩ is the name of the property in the subsumed class and the second ⟨*prop id*⟩ is the name of the property in the subsuming class.

[30] *Scope*, in computer programming, is the context where values and expressions can be evaluated in a program. The term "scope" is being used here instead of the term *domain*, meaning the context of some property, as explained in page 40 of Chapter 2.

Let us now consider the case when concepts *A*, *B* and *C* already exist, with their respective properties *a1*, *b1*, and *c1* and *c2*. An example of valid OntoQL expression for creating an *a posteriori case-of* relation between the concept *C* and the concepts *A* and *B* can be given as follows.

```
CREATE #AposterioriCaseOf C CASEOF (A, B)
WITH (C.c1 MAP A.a1, C.c2 MAP B.b1)
```

This expression creates an instance of the entity *#AposterioriCaseOf* that relates concept *C* to concepts *A* and *B*. It maps the property *c1* from *C* to the property *a1* from *A* and also the property *c2* from *C* to the property *b1* from *B*. Alternatively, an *a posteriori case-of* relation could be created without mapping properties. Notice that it is not necessary to map all properties of the subsumed class, nor to map to all properties of the subsuming classes.

2.1.3 Interpretation of the extended OntoQL definition language

The OntoQL language was implemented in Java[31] language using the parser generator ANTLR[32] to carry out the lexical and syntactic analysis of the language. The approach applied for handling the interpretation of an OntoQL expression consists in translating it into a SQL expression that is specific to the OBDB in use. As a consequence of the extension of the OntoQL grammar with new operators, we had to extend, also, the OntoQL *semantics*, that is, the mechanism that interprets the expressions and actually carries out the required database operations. The OntoQL semantics was extended to be able to interpret the new expressions that can be formed with the extended grammar.

The method that handles *class creation* was modified to be able to **create an *is-case-of* class**. If the expression identified by the parser is the *creation of an an a priori case-of class*, such as
`CREATE C ISCASEOF (A,B) (IMPORTS (a1,b1))`, the interpreter translates it in a SQL expression proceeding as follows: (i) it inserts the new class *C* in the *Item_class_case_of* table of OntoDB, instead of using the *Item_class* table. (ii) It adds in the column *is_case_of* the identifiers of the classes *A* and *B*, of which *C* is a case. (iii) It finally adds the identifiers of the properties *a1* and *b1*, imported by *C*, in the column *imported_properties*.

```
INSERT INTO item_class_case_of (name, is_case_of, imported_properties)
VALUES ('C', ARRAY[A, B], ARRAY[a1, b1])
```

A simplified representation of the table resulting of this operation is presented on Figure 5.7.

Item_class_case_of			
rid	name	is_case_of	imported_properties
1	C	{A, B}	{a1,b1}

Figure 5.7: Table *Item_class_case_of* after creation of new *a priori case-of* class

In order to be able to **create *a posteriori case-of* relations**, the method that handles *entities creation* had to be modified. In addition the creation of an entity *#Class*, the method had to take into consideration the particularities of the creation of an entity *#AposterioriCaseOf*. If the expression identified by the parser

[31] http://java.com/
[32] http://www.antlr.org/

is the *creation of an a posteriori case-of relation*, such as
`CREATE #AposterioriCaseOf C CASEOF A (WITH (c1 MAP a1, c2 MAP a2))`, the interpreter proceeds as follows: (i) for each pair of mapped properties in the expression, it creates an *INSERT INTO #MapProperty* expression that adds a new row in the *MapProperty* table. For example, for the first pair *(c1 MAP a1)*, it creates the expression:

```
INSERT INTO #MapProperty (#source, #mapTo)
 VALUES ((SELECT #oid FROM #Property where #name='c1'),
        (SELECT #oid FROM #Property where #name='a1'))
```

(ii) The interpreter creates an *INSERT INTO #AposterioriCaseOf* expression that adds a new row in the *AposterioriCaseOf* table. The expression specifies the source class *C* and the classes of which *C* is a case (class *A*). In order to fill the *#correspondingProperties* column, the identifiers of the pairs of properties must be retrieved from the *MapProperty* table.

```
INSERT INTO #AposterioriCaseof (#source, #isCaseOf, #correspondingProperties)
  VALUES ((SELECT #oid FROM #Class where #name='C'),
         (SELECT #oid FROM #Class where #name='A'),
         ARRAY(SELECT #oid FROM #MapProperty
              WHERE #source.#name='C' AND #mapTo.#name='A'))
```

A simplified representation of the tables resulting of this operation is presented on Figure 5.8b.

MapProperty		
rid	source	mapTo
1	c1	a1
2	c2	a2

A_posteriori_case_of			
rid	source	is_case_of	corresponding_properties
3	C	{A}	{1,2}

Figure 5.8: Tables *MapProperty* and *AposterioriCaseOf* after creation of new *a posteriori case-of* class

2.1.4 Constraints on the *is-case-of* operator

In the definition of a language, respecting grammar rules does not fully prevent from creating expressions syntactically valid but semantically invalid. In order to restrict fallible expressions concerning the *is-case-of* operator, some semantic constraints were defined. Some of these constraints were implemented directly inside the Java methods that implement the semantics of the OntoQL operators (*cf.* Section 2.1.3). Other constraints were implemented as triggers directly in the RDBMS.

Constraint 1 (*Domain of subsuming properties*)

***Definition**: The domain of the* subsuming *properties must be one of the classes of the list of subsuming classes or some super-class of one of them.*
***Example**: If the domain of stageName is not Student or Musician or some superclass of Student or Musician, the creation of both is-case-of relations will return an error.*

```
CREATE #AposterioriCaseOf StudentMusician CASEOF (Student, Musician)
  WITH (name MAP stageName)

CREATE StudentMusician ISCASEOF (Student, Musician)
```

```
IMPORTS (stageName)
```

2.1.4.1 Specific constraints of the *a priori case-of* operator .

Constraint 2 (*Properties with same name*)

***Definition**: The subsumed class cannot import more than one* subsuming *properties that have the same name, even if their domains are different classes.*
***Example**: Property birthdate is defined both in Student and Musician classes. If the class StudentMusician imports both of them, the property birthdate will be duplicated, and the class creation will return an error.*

```
CREATE StudentMusician ISCASEOF (Student, Musician)
  IMPORTS (Student.birthdate, Musician.birthdate)
```

2.1.4.2 Specific constraints of the *a posteriori case-of* operator .

Constraint 3 (*Double subsumption*)

***Definition**: It is not possible to create an is-case-of relation between two classes that already have a* subsumption relation *between them, in any direction.*
***Example**: The creation of an a posteriori case-of relation between Student and Person is not possible, since there already exists a subsumption between them (the is-a relation).*

```
CREATE #Class Student UNDER (Person)
CREATE #AposterioriCaseOf Student CASEOF (Person)
```

Constraint 4 (*Domain of subsumed properties*)

***Definition**: The domain of the* subsumed *properties must be the subsumed class or some super-class of the subsumed class.*
***Example**: If the domain of property name is not Student or some superclass of Student, the creation of the a posteriori case-of relation between Student and Person will return an error.*

```
CREATE #AposterioriCaseOf Student CASEOF (Person)
  WITH (name MAP fullname)
```

Constraint 5 (*Properties with same name*)

***Definition**: If two or more* subsuming *properties, which have different domains, have the same name, the user must explicitly specify the domain of the properties.*
***Example**: Property birthdate is defined both in Student and Musician classes. If the domain of birthdate is not made explicit with a path expression, the creation of the a posteriori case-of relation between StudentMusician and Student, Musician will return an error.*

```
CREATE #AposterioriCaseOf StudentMusician CASEOF (Student, Musician)
```

```
  WITH (dateOfBirth MAP Student.birthdate, dateOfBirth MAP Musician.birthdate)
```

Constraint 6 (*Range of properties (a)*)

***Definition**: The* range *of two properties that are being mapped one to the other must be of the same type, that is, either both are the same primitive type, or both are a reference type or both are collection type.*
***Example**: The ranges of the properties hasSupervisor and tutorName are not of the same type (reference type and primitive type). The creation of the a posteriori case-of relation between StudentMusician and Student will return an error.*

```
CREATE #Class Student (PROPERTIES (hasSupervisor REF(Person)))
CREATE #Class StudentMusician (PROPERTIES (tutorName String))

CREATE #AposterioriCaseOf StudentMusician CASEOF (Student)
  WITH (tutorName MAP hasSupervisor)
```

Constraint 7 (*Range of properties (b)*)

***Definition**: When the ranges of both properties are* reference type*, the class referenced by the subsumed property must be the* same *as or a* super-class *of the class referenced by the subsuming property.*
***Example**: The ranges of the properties hasSupervisor and tutorName are of the same type (reference type) but make reference to different classes. Moreover, Musician is not a super-class of Person. The creation of the a posteriori case-of relation between StudentMusician and Student will return an error.*

```
CREATE #Class Student (PROPERTIES (hasSupervisor REF(Person)))
CREATE #Class StudentMusician (PROPERTIES (tutorName REF(Musician)))

CREATE #AposterioriCaseOf StudentMusician CASEOF (Student)
  WITH (tutorName MAP hasSupervisor)
```

Constraint 8 (*Range of properties (c)*)

***Definition**: When the ranges of both properties are* reference type*, the range of the* subsuming *property can be a class which is a* case of *the range of the subsumed property.*
***Example**: The range of tutorName is MusicProfessor, which is a case of Professor. Thus, tutorName can be mapped to the property hasSupervisor, since the range of the latter is Professor.*

```
CREATE #Class Professor
CREATE #Class MusicProfessor ISCASEOF Professor

CREATE #Class Student (PROPERTIES (hasSupervisor REF(Professor)))
CREATE #Class StudentMusician (PROPERTIES (tutorName REF(MusicProfessor)))

CREATE #AposterioriCaseOf StudentMusician CASEOF (Student)
 WITH (tutorName MAP hasSupervisor)
```

Constraint 9 (*Range of properties (d)*)

***Definition**: When the ranges of both properties are* collection type*, the type of the internal elements of both collections should follow the constraints 6 and 7 above.*
***Example**: The ranges of the properties courses and specialities are collection types (defined by the keyword ARRAY), and the internal type of both of them is STRING. Otherwise, the creation of the a posteriori case-of relation between StudentMusician and Student would return an error.*

```
CREATE #Class Student (PROPERTIES (courses STRING ARRAY))
CREATE #Class StudentMusician (PROPERTIES (specialities STRING ARRAY))

CREATE #AposterioriCaseOf StudentMusician CASEOF (Student)
  WITH (specialities MAP courses)
```

Constraint 10 (*Repeated subsumed properties in the same expression*)

***Definition**: When the name of a subsumed property appears in more than one list of properties mapping, the subsuming classes that are referred in these lists must not be part of the same subsumption hierarchy. This will be also verified when some property that was already mapped to some property, is subsequently mapped to a different property.*
***Example**: The property dateOfBirth is mapped to two properties of two different classes. If the classes Student and Musician are related by a subsumption relation (such as is-a or is-case-of), the creation of the a posteriori case-of relation between StudentMusician and Student will return an error.*

```
CREATE #AposterioriCaseOf StudentMusician CASEOF (Student, Musician)
  WITH (dateOfBirth MAP Student.birthdate, dateOfBirth MAP Musician.birthdate)
```

2.2 Handling the *is-case-of* subsumption relation in a OntoQL query

At the origin, an OntoQL query only takes into account the inheritance hierarchy of classes (*is-a* relation), as explained in Section 3.1.3 of Chapter 3. The *SELECT* clause had to be extended in order to also include the *is-case-of* hierarchy. We decided to propose to the user two special operators that allow to explicitly request for *is-case-of* classes. These operators, when added to a classical OntoQL query, modify the SELECT operation as follows.

- *WITH APRIORI* : activates the search for *a priori case-of* classes.
- *WITH APOSTERIORI* : activates the search for *a posteriori case-of* classes.

The user may want to explore three types of class hierarchies — *is-a* , *a priori case-of* and *a posteriori case-of* hierarchies — and the query can be polymorphic or non-polymorphic[33]. Considering these possibilities, there are eight types of queries that can be formulated by the user which are detailed in Table 5.1. The two first rows constitute the original syntax of the clause, while the six next rows present the possible variations of the *FROM* clause when including *a priori case-of* and *a posteriori case-of* operators. For each clause the table indicates from which classes instances are retrieved and which are the

[33]When a select operation is applied in cascade over the subclasses of a concept, we call this a *polymorphic* select. When the select is restricted to the instances of the declared concept, it is called a *non-polymorphic* select.

considered properties. For example, the clause *SELECT * FROM C WITH APRIORI* selects instances from class C, plus instances of subclasses of C, plus instances of classes that are *a priori case-of* C. Only properties defined in or inherited or imported from C are considered.

Table 5.1: Types of class hierarchies explored by the OntoQL SELECT clause

Query	**Imports instances from**				**Considered properties**			
SELECT *	C	subclass of C	a priori case of C	a posteriori case of C	defined in C	inherited from C	imported from C	mapped to C
FROM ONLY (C)	X				X			
FROM C	X	X			X	X		
FROM ONLY (C) WITH APRIORI	X		X		X		X	
FROM ONLY (C) WITH APOSTERIORI	X			X	X			X
FROM ONLY (C) WITH APRIORI APOSTERIORI	X		X	X	X		X	X
FROM C WITH APRIORI	X	X	X		X	X	X	
FROM C WITH APOSTERIORI	X	X		X	X	X		X
FROM C WITH APRIORI APOSTERIORI	X	X	X	X	X	X	X	X

The *WITH APRIORI* and *WITH APOSTERIORI* operators modify the *behavior* of the query. They include other classes within the union of classes considered, and they change the set of projected properties. In Logics, an operator that limits the variables of a proposition, as *some* or *all*, is called a *quantifier* operator. For this reason, the new proposed operators are called *is-case-of quantifiers*.

2.2.1 Modification of the OntoQL SELECT clause to handle the *is-case-of* quantifiers

Differently from the OntoQL extension for *is-case-of* relations (Section 2.1), the addition of query quantifiers to OntoQL does not require the core entities to be changed. This is due to the fact that we are not *writing* new information in the database, but just *reading* information that was already stored in existing tables. Consequently, it was just necessary to modify the OntoQL grammar in order to include the *is-case-of* quantifiers.

The general syntax of an OntoQL query is the following (Jean, 2007):

⟨*query specification*⟩ ::= ⟨*select clause*⟩ ⟨*from clause*⟩ [⟨*where clause*⟩]
[⟨*group by clause*⟩] [⟨*having clause*⟩] [⟨*order by clause*⟩]
[⟨*namespace clause*⟩] [⟨*language clause*⟩]

Grammar 5.4: Original grammar for SELECT statement

The goal was to include a new clause, independent of the others, that allowed to make use of the proposed

quantifiers in combination with all the other original clauses. To this end, the grammar was then modified as follows:

⟨*query specification*⟩ ::= ⟨*select clause*⟩ ⟨*from clause*⟩ [⟨*where clause*⟩]
[⟨*group by clause*⟩] [⟨*having clause*⟩] [⟨*order by clause*⟩]
[⟨*namespace clause*⟩] [⟨*language clause*⟩] [⟨*withcaseof clause*⟩]

⟨*withcaseof clause*⟩ ::= `WITH` ⟨*caseofquantifier*⟩

⟨*caseofquantifier*⟩ ::= `APRIORI` | `APOSTERIORI` | ⟨*doublecaseofquantifier*⟩

⟨*doublecaseofquantifier*⟩ ::= `APRIORI APOSTERIORI` | `APOSTERIORI APRIORI`

Grammar 5.5: Modified grammar for SELECT statement: ⟨*withcaseof clause*⟩

The grammar was modified by adding the optional clause ⟨*withcaseof clause*⟩ to the end of the statement ⟨*query specification*⟩. The withcaseof clause produces a statement that begins with the literal '`WITH`' and is followed of all the possible combinations of using the quantifiers '`APRIORI`' and '`APOSTERIORI`': each one alone, or both together, in any order.

2.2.2 Executing the extended OntoQL SELECT clause

Subsequently to the extension of the OntoQL query with new quantifiers, we had to adapt the *semantics* of the OntoQL query.

The semantics of any relational database query language must be defined over a mathematical basis. Operators such as *UNION* are based on traditional set theory, others (such as *AND*, *OR*) are strongly dependent of the use of logical expressions. The formal definitions of the operators of a relational query language is referred as its *algebra*.

The OntoQL query language is based on *OntoAlgebra*, an adaptation of *Encore* object-oriented algebra (Zdonik and Mitchell, 1991) for the model of an OBDB. The *Encore* model was extended in order to take into consideration operations over the *ontology model*, in addition to the queries over the data model (*cf.* Section 3.2 of Chapter 3). The *OntoAlgebra* was proposed by Jean (2007), and proposes operators such as *OntoProject*, *OntoSelect*, *OntoOJoin*, *OntoNest*. The *OntoAlgebra* specifies transformation rules that are applied to build the *algebraic expression* of an OntoQL query. As an example of the application of those trasformation rules, we present in Table 5.2 (adapted from (Jean, 2007)) the algebraic expressions derived from the application of *OntoAlgebra* rules to the following OntoQL query.

```
SELECT l.title, p.lastname FROM Laboratory AS l, Person AS p
```

The first two lines use *OntoProject* for building the set of classes that will be queried by projecting all properties of a class over the class itself and over all its *subclasses*. In line 3 *OntoOJoin* creates the Cartesian product of the tuples resulting from the projections. Finally, in line 4, *OntoProject* projects the listed properties *title* and *lastname* over the set of classes.

In order to enable the evaluation of *is-case-of* classes within an OntoQL query, the following *OntoAlgebra* operators had to be changed.

Table 5.2: Algebraic expression derived from OntoQL query

exp_i	*Clause*(exp_i)	*Result*(exp_i)
exp_1	`Person AS p`	`OntoProject(Person, subClasses(Person), allProperties(Person))`
exp_2	`Laboratory AS l`	`OntoProject(Laboratory, subClasses(Laboratory), allProperties(Laboratory))`
exp_3	`FROM Clause(`exp_1`), Clause(`exp_2`)`	`OntoOJoin(Result(`exp_1`), Result(`exp_2`))`
exp_4	`SELECT l.title, p.lastname`	`OntoProject(Result(`exp_3`),title,lastname)`

1. ***OntoOJoin***: If an *is-case-of* quantifier is used in the query, the product of classes must include *is-case-of* classes.
2. ***OntoProject***: Regarding *is-case-of* classes, the only properties that must be projected are those that were imported/mapped.

We illustrate the modification of the *OntoAlgebra* operators as follows. Considering the database configuration below, in which the class *Student* is a case of the class *Person*.

```
CREATE #Class Person (
  PROPERTIES (name STRING, age INTEGER, profession STRING, email STRING))

CREATE #Class Student ISCASEOF Person (
  IMPORTS (name, age)
  PROPERTIES (registrationID INTEGER))
```

The following OntoQL query makes use of the *is-case-of* quantifier *WITH APRIORI* in order to select instances of the class *Person*, of subclasses of *Person* and of classes that are case of *Person* .

```
SELECT name, profession FROM Person WITH APRIORI
```

For the previous OntoQL query the modified *OntoAlgebra* operators will generate the algebraic expressions described in the Table 5.3.

Table 5.3: Algebraic expression derived from OntoQL query with *is-case-of* quantifiers

exp_i	*Clause*(exp_i)	*Result*(exp_i)
exp_1	`Person`	`OntoProject(Person, subClasses(Person), {name, age, profession, email})`
exp_2	`Person WITH APRIORI`	`OntoProject(Student, {name, age})`
exp_3	`FROM Clause(`exp_1`), Clause(`exp_2`)`	`OntoOJoin(Result(`exp_1`), Result(`exp_2`))`
exp_4	`SELECT name, profession`	`OntoProject(Result(`exp_3`),{name,profession})`

Line 2 shows the result of the use of the *is-case-of* quantifier *WITH APRIORI*. The operator *OntoProject* is used for retrieving instances of the *is-case-of* class *Student*, projecting only properties that are imported from the class *Person*: *name* and *age*. In line 3, operator *OntoOJoin* creates the Cartesian product of the resulting set of classes. In order to perform a Cartesian product, two classes need to have the

same number of columns. To this end, the *OntoOJoin* operation adds *NULL* columns to class *Student*, until the number of columns of class *Person* is reached. Finally, in line 4, *OntoProject* projects the listed properties *name* and *profession* over the set of classes.

2.2.3 Toy example of the *is-case-of* relation in OntoQL

In order to show the use of the *is-case-of* relation for building ontologies, we next describe a toy example of an ontology whose concepts are related through *is-case-of* relations. The example ontology is created by the following OntoQL expressions.

```
CREATE #Class A1 (
 PROPERTIES (prop_a1 STRING, prop_a2 INT))

CREATE #Class B1 (
 PROPERTIES (prop_b1 STRING, prop_b2 INT))

CREATE #Class C (
 PROPERTIES (prop_c1 STRING, prop_c2 INT))

CREATE #Class D
 ISCASEOF A1 (IMPORTS(A.prop_a1, prop_a2)
 PROPERTIES (prop_d1 INT))

CREATE #Class E1
 ISCASEOF A1 (IMPORTS(A1.prop_a1)
 PROPERTIES (prop_e1 STRING, prop_e2 INT))

CREATE #Class A2 UNDER A1

CREATE #Class B2 UNDER B1

CREATE #Class E2 UNDER E1

CREATE #AposterioriCaseof C CASEOF A1
 WITH (C.prop_c2 MAP A1.prop_a2)
```

The resulting structure of classes is illustrated by Figure 5.9.

After the creation of table extents (using OntoQL expressions such as
`CREATE EXTENT OF A1 (prop_a1, prop_a2)`) and after the insertion of the instances of each class (with OntoQL expressions such as
`INSERT INTO A1 (prop_a1, prop_a2) VALUES ('val_A1a1', 1)`), the logical schema of the considered ontology appear as shown on Figure 5.10.

Finally, we are able to formulate queries that take into account the *is-case-of* relations between classes. A complete example of the *is-case-of* operators behavior is given in the next query, whose result is illustrated on Figure 5.11.

```
SELECT prop_a1, prop_a2 FROM ONLY(A1) WITH APRIORI APOSTERIORI
```

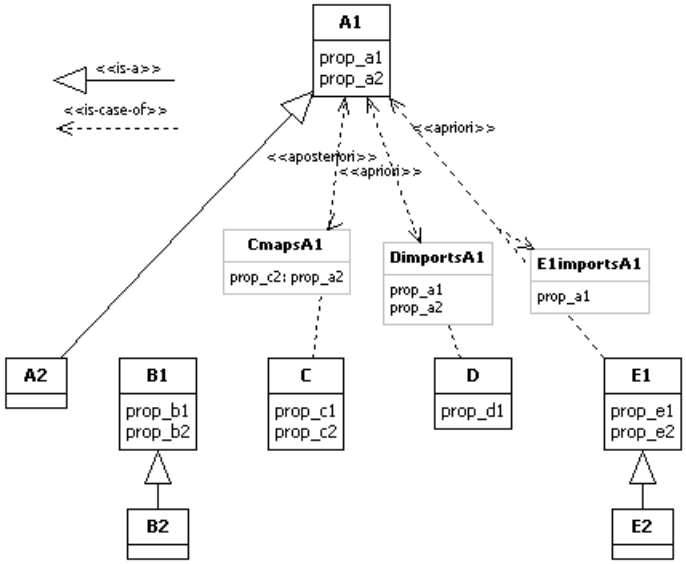


Figure 5.9: Ontology that uses *is-case-of* relation

The above query result in instances selected from class *A1* only, plus instances from class *C* and from class *E1* (which are *a posteriori case-of A1*), plus instances of class *D* (which is an *a priori case-of A1*). Notice that for class *C*, the column *prop_a1* has value *NULL*, since this property of class *A1* is not mapped by the class *C*. The column *prop_a2* for class *E1* has also value *NULL*, since this property was not imported from *A1* to *E1*. This query is non-polymorphic, i.e. it does not concern instances of subclasses of *A1*. That is why the instances of class *A2* are not in the result table.

2.2.4 Using the *is-case-of* relation for mapping concepts of pre-existing ontologies

When we have data sources that are described by several reference ontologies, a class of a local ontology may be described as *subsumed* by one or several other class(es) defined in other ontologies. This is a typical example of the use of the *is-case-of* relation for mapping concepts in various ontologies.

Example. Figure 5.12 present a user-defined ontology *O2* mapped on a reference ontology *O1*.

The user has the option of specifying the *is-case-of* relation in the ontology engineering time (*a priori case-of*). In this circumstance, the following OntoQL expressions will be used to create the concepts of the ontology *O2*. The concepts will be created as a case of *O1*'s concepts, and *O1*'s properties may also be imported.

```
CREATE #Class Items ISCASEOF Resources;

CREATE #Class Products ISCASEOF Resources (
  IMPORTS(Resources.mass));

CREATE #Class Computer_Hardware ISCASEOF Hardware;
```

A1	
prop_a1	prop_a2
'val_A1a1'	1

A2	
prop_a1	prop_a2
'val_A2a1'	6

B1	
prop_b1	prop_b2
'val_B1b1'	2

B2	
prop_b1	prop_b2
'val_B2b1'	7

C	
prop_c1	prop_c2
'val_Cc1'	3

D		
prop_a1	prop_a2	prop_d1
'val_Da1'	4	100

E1		
prop_a1	prop_e1	prop_e2
'val_E1a1'	'val_E1e1'	5

E2		
prop_a1	prop_e1	prop_e2
'val_E2a1'	'val_E2e1'	8

Figure 5.10: Logical schema of the ontology of Figure 5.9

```
CREATE #Class Electronic_Components ISCASEOF Components;

CREATE #Class Software ISCASEOF Software;
```

The user can also choose to create the domain ontology independently of the reference ontology, and to define afterwards *is-case-of* relations between their respective concepts (*a posteriori case-of*). Considering that both ontologies are already created, the OntoQL expressions that will be used to define

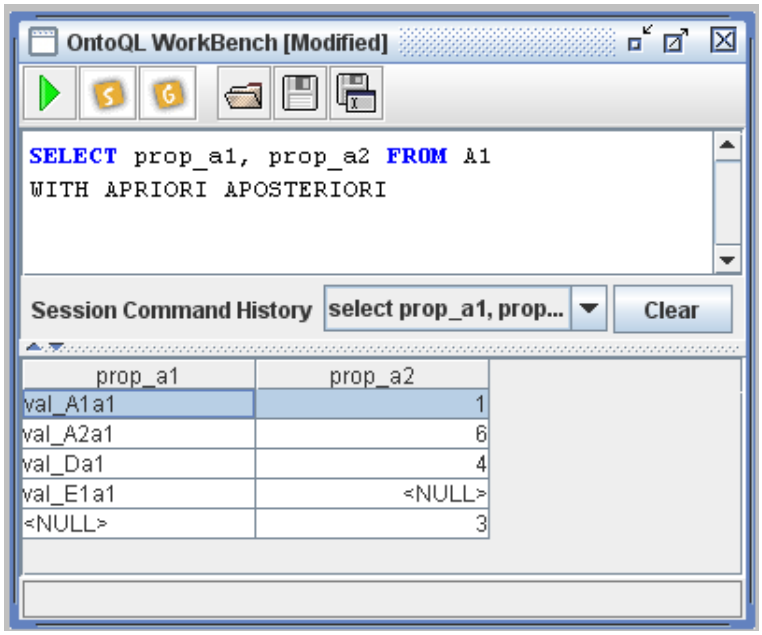


Figure 5.11: Result from the SELECT query over the *A1* class

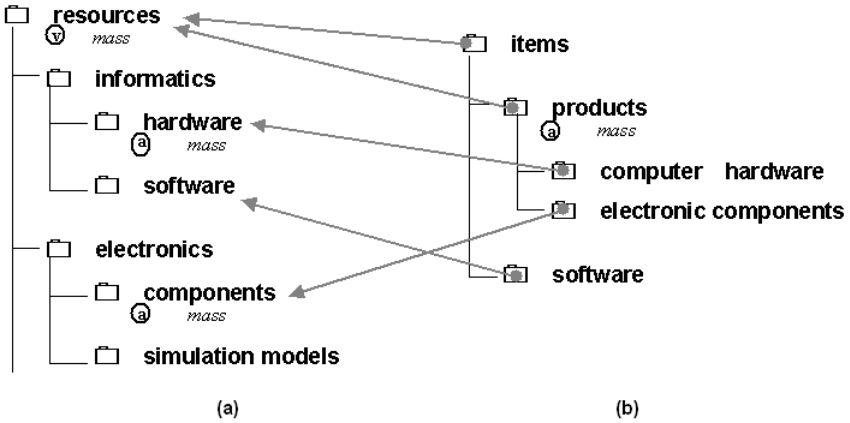


Figure 5.12: An example of a reference ontology (a) and of an user defined ontology (b)

mappings between concepts are as follows:

```
CREATE #AposterioriCaseOf Items CASEOF Resources;

CREATE #AposterioriCaseOf Products CASEOF Resources (
  WITH(Products.mass MAP Resources.mass));

CREATE #AposterioriCaseOf Computer_Hardware CASEOF Hardware;

CREATE #AposterioriCaseOf Electronic_Components CASEOF Components;

CREATE #AposterioriCaseOf Software CASEOF Software;
```

By asking a query over the *O1* concept *Resources*, the user is now able to also retrieve instances from concepts of the mapped ontologies (such as *O2*). The following OntoQL query searches instances of *Resources* and from classes that are case of *Resources*.

```
SELECT * FROM Resources WITH APRIORI APOSTERIORI;
```

The result will return instances of the classes *Resources*, *Items* and *Products*. The *mass* property will be evaluated for the classes *Resources* and *Products*, but the corresponding column will present no values for the class *Items*.

The structure of a user ontology may be quite different from the one of a reference ontology. Nevertheless, a system storing the user ontology along with mappings to other ontologies may automatically answer queries against the reference ontology(ies) on which the user-defined ontology is mapped. The mappings will be used to automatically translate the results of the queries over the local ontologies. Instead of having to design a mediator to process the translation, the work of query processing is let to the query system coupled with the ontology language. The query system is acquainted to the built-in primitives, such as the *is-case-of* relation.

In the next chapter, we will describe the ontologies actually built for the domain of application of this thesis, the geoscience ontologies. We will be able, then, to show the mappings that can be created

between global and local ontologies, and we describe how to formulate queries that interrogate the GO and return instances from different LOs.

3 Conclusion

We described in this chapter an architecture for *ontology-based model integration*. This architecture proposes a solution for semantically integrating information issued from models that are described by different domain ontologies.

Each type of model makes reference to one specific domain of expertise formalized by means of a *local ontology*. We decided to integrate these local ontologies applying a hybrid structure of integration. For this, it is necessary to choose a *reference ontology* to whose concepts all local concepts are mapped. The reference ontology should be an ontology that is general enough to include all the local concepts, and at the same time, specific enough to be used as a query interface that provides the user with meaningful vocabulary. We introduced the idea of a *federating domain* as an expertise domain covering all the local domains related to a definite field, for instance, Mechatronics, which covers both Electronics and Mechanics or Geology which covers a large bunch of geoscience domains. We proposed that the *federating domain* should be used as one *Global Ontology* (GO) to which all local ontologies (LO) are mapped. We are not using an external upper level ontology (such as SUMO or DOLCE) as the global ontology, since we aim to provide to the user a global ontology that is part of the engineering activity and leave the possibility for the user to query the LO as well.

The understanding of the semantic structure that integrates the considered ontologies is left to the domain expert, who is responsible for manually defining mappings between the various ontologies. This is due to the fact that we are dealing with *complex multiple-world scenarios*, as it was explained in Section 1.2 of Chapter 1, which are not obvious to integrate.

For operating between global and local ontologies, we proposed using a *subsumption relationship*. The typical subsumption relationship is represented by the *is-a* relation, which defines an inheritance hierarchy between the concepts put in relation. In our case, we proposed to use the *is-case-of* relation, which is a subsumption relation similar to the *is-a* relation but which only creates *partial inheritance* hierarchies between concepts. Using a *is-case-of* relation allows to import/map not all the properties of the subsuming (more general) concept, but just those that are adequate for the subsumed (more specific) concept. The properties to be imported/mapped are chosen by the user who defines the correspondence between concepts.

We needed the *is-case-of* relation to be embedded in the ontology definition language in the same way as the *is-a* relationship. Therefore, the *is-case-of* relation was implemented as an operator of the OntoQL language, declined in two operations: the *a priori* fashion, which enables to align concepts from different ontologies in design time (i.e. when the ontology concepts are being defined), and conversely the *a posteriori* fashion, which enables the correlation of concepts in the case when the ontologies were already defined by the time when one decided to correlate them.

We also performed an extension of the OntoQL *SELECT* clause so as to include an *is-case-of* quantifier, that allows to navigates through all the hierarchies of the subsumed classes (*is-a* classes, *a posteriori*

case-of classes and *a priori case-of* classes, altogether or alternatively). When the user asks to retrieve all the instance of a concept that is the most general in a *is-case-of* hierarchy, the result will be the instances of all concepts that are a case of the general concept, not considering which ontologies these concepts belong to.

Thanks to the *is-case-of* relation, it became possible to address in one same query instances of concepts that had not been originally defined to be in a hierarchy. The *is-case-of* relation also allows to define concepts without making use of *multi-inheritance*. One concept may be only related to one super concept and be a case of several other concepts. This is an important aspect for ontologies that must be "strongly typed", such as those implemented in relational databases.

This part concludes the contribution of this thesis for semantic-based integration of engineering models. We presented an annotation model that enables engineering models to be enriched in semantics and an alignment relation that allows to integrate ontologies of *semantically unrelated* domains. In Part III we present the application of these proposals to the activity of earth modeling in the domain of petroleum exploration.

Part III

Contribution: Application to Geology and to Petroleum Engineering

Chapter 6

Example of an engineering domain: petroleum reservoir studies and modeling

1 Introduction

Petroleum exploration is an activity in which acquisition, distribution and use of expert knowledge are more critical for the decision-making. According to (Rainaud, 2005), 43% from the total budget of petroleum is currently dedicated to information integration.

Petroleum industry depends on computer models related to several processes: 3D seismic interpretation, well bore drilling, reservoir modeling and monitoring and also plant/facility modeling or monitoring capabilities. Each of these processes can generate massive volumes of data, from which interpretation are derived, scenarios are developed, interdependent models are created, and decisions are taken. Other kinds of models such as decision models, investment models, and facility models are also used for reducing decision-making uncertainty and risk.

Earth models are key tools for identifying and characterizing potential hydrocarbon reservoirs. Earth models are three- (3D) or four-dimension (4D) representations of data and interpretation concerning subsurface resources (i.e. resources that are found below the surface of the earth or below the seabed). Earth models are developed by *geoscientists* who are responsible for evolving a hydrocarbon prospect through various stages of modeling. Their final goal is the building of a *reservoir model*, which will be used for simulating oil accumulation in the underground. Currently, this final model is connected to the original raw data by a long chain of successive interpretation. This chain of activities, which starts with data acquisition and proceeds with several different steps of data analysis and interpretation is known as the *Earth Modeling Workflow* or *Reservoir Modeling Workflow*. Figure 6.1 illustrates the most important steps of this workflow, that we will examine in more details in the next section.

1.1 Overview of the earth modeling activity for petroleum exploration

The earth modeling workflow starts with the definition of a *prospect*, which corresponds to a spatial 3D area of interest. Initial data acquisition concerning a prospect is basically carried out by means of

seismic reflection (Figure 6.1(1)) and well bore drilling (Figure 6.1(2)). Geoscientists also take into account former studies about the prospect, such as documents concerning regional geology, geological maps or cross-sections (Figure 6.1(3)).

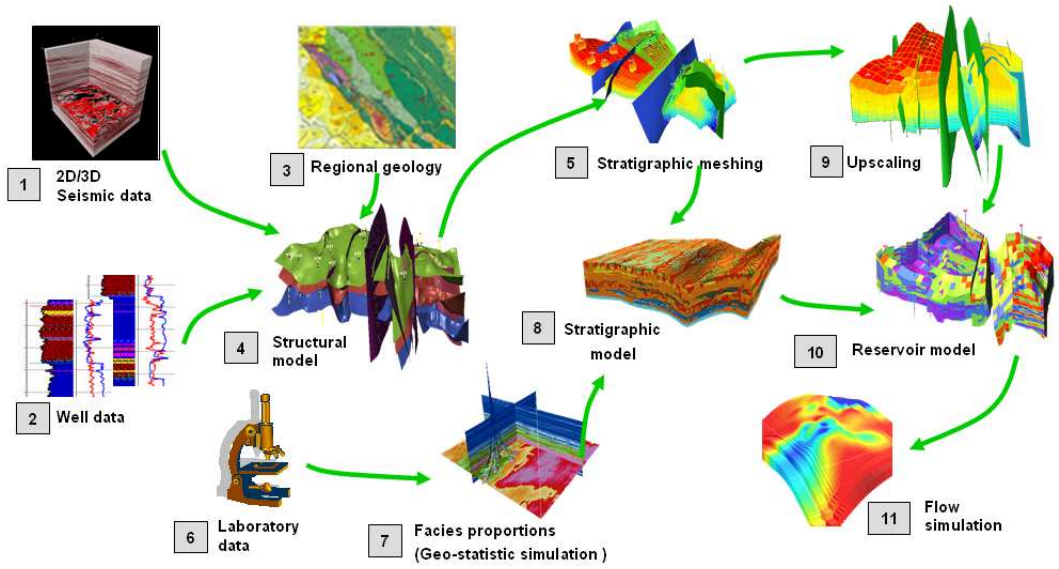


Figure 6.1: The Earth Modeling Workflow

The **seismic reflection** technique consists in recording, with the help of sensors, the echoes resulting from the propagation of an artificially acoustic wave produced on the sea or on the earth surfaces.

In stratified terrains notable changes of rock physical properties are generally observed when crossing a sedimentary boundary. Such change in the properties of the physical middle causes a reflection of acoustic waves. The reflected waves reemitted by the various sedimentary boundaries towards the sea or earth surface are registered and constitute in all a seismic image. Such a seismic image shows lines of different colors and widths corresponding to variations of wave amplitudes. Each line corresponds to a given reflector i.e. to a stratigraphic boundary portion. With the aid of computer tools, geoscientists perform a task of **seismic interpretation** over the seismic image, which consists in identifying patterns that will be recognized as surfaces (such as horizons and faults) or assemblages of surfaces corresponding to specific sedimentary objects (for instance, channel, salt dome). After having identified some object, the interpreter manually picks up points for specifying its geometry. These points are recorded as corresponding to one or several interpreted surfaces. The apparent depths of these various surfaces on the seismic image do not strictly correspond to actual depths since seismics recordings only register travel times (i.e. the delay in which a given wave reaches the sea or earth surface).

In parallel to the seismic data, the interpreter generally examines a collection of well logs recording changes of physical properties in the rocks crossed by drilling trajectories. Each well log can be interpreted as a succession of well markers, each corresponding to some lithology discontinuity. A depth within the borehole is associated to each of these markers. In consequence, well logs can provide true geometric information about the position of sedimentary horizons in the prospect area under study. Thanks to the **well correlation** task, markers are used for adjusting the vertical position of each of the surfaces identified through the seismic interpretation. Geoscientists currently perform correlations between sev-

eral well logs, seismic cross sections, and core properties. This identification and correlation work allows them identifying relations between the structures identified on the seismic image, which will be useful for constructing the structural model later on.

With the aid of computer modeling tools (called geomodelers), the surfaces identified by seismic interpretation are loaded. Taking into account in addition all available data issued from regional geology studies (reports, research papers, maps etc.), geoscientists then proceed a **structural interpretation**. This allows specifying the spatial and chronological relationships between the identified objects. The topology of the object assemblage is of paramount importance since it strictly depends from geological interpretation (Perrin, 1998). This is a crucial step in the workflow because the **structural model** (Figure 6.1(4)) is the "skeleton" on which other earth models will be built. It basically consists in an assemblage of geological surfaces that mark the boundaries of individual geological blocks.

In the **stratigraphic modeling** activity, stratigraphic meshes are built inside each of the blocks of the structural model (Figure 6.1(5)). Petrophysical properties must them be affected to each cell of each mesh. For this, geomodelers first consider the properties acquired from isolated points corresponding to samplings and to laboratory studies (Figure 6.1(6)). These properties are then propagated to the whole volume using geostatistic simulation (Figure 6.1(7)). The resulting model, where the stratigraphic mesh cells are filled with rock properties, is called **stratigraphic model** (Figure 6.1(8)).

In order to use this model for simulations, it is necessary to transform the geometry of the stratigraphic model mesh in order to obtain a coarser reservoir mesh and then to upscale the property values (Figure 6.1(9)). There results a **reservoir model** (Figure 6.1(10)), which provides a complete set of continuous reservoir parameters (i.e. porosity, permeability, water saturation) for each cell of the 3D grid. It will be used by reservoir engineers to compute realistic hydrocarbon fluid migration **simulation** (Figure 6.1(11)), and to estimate the amount of exploitable hydrocarbon reserves present within the reservoir and the quality of these reserves (heavy or light oil, gas etc.).

1.2 Knowledge management in the earth modeling activity

Various skills are required along the modeling chain, corresponding to the expertise of geophysicists, structural geologists, stratigraphers, sedimentologists, petrologists and petrophysicists, reservoir engineers, computer graphics and volume modeling professionals, drilling engineers, project managers, etc. These actors use heterogeneous data management environments which use various data representations and encoding conventions for dealing with the same information in different parts of the workflow. It would be desirable that this workflow be replayed several times, considering different interpretation hypotheses, possibly introduced at various stages of the modeling chain.

Earth modeling activities deal with a loose federation of autonomous, heterogeneous data repositories where the semantics of information is embedded in applications and databases. Because of this, there are practical data management questions that cannot be answered, such as "What data do we have?", "Where are they located?", "What do they mean?" "From which interpretation are they issued?". Furthermore, the earth modeling domain is strongly based on *interpretative* tasks. Complex activities involving interpretation are a "mixture" of raw and interpreted data. Earth modeling as well as some other activities, like medical diagnosis for instance, rest for a good part on interpretation.

The various data considered throughout workflows attached to such activities maybe considered either as “raw” data or as “interpreted” data. In most cases, data depend on interpretation in some way. For instance, the raw seismic data that are considered at the early stages of geological modeling workflows, themselves result from more or less complicated signal processing procedures. However, considering data as raw or interpreted depends from the modeler’s decision. In all cases, when the interpreter wants to keep the memory of the interpretation that lies behind some data, data should be considered as interpreted. 3D geological models are highly dependent from interpretation operated by the geologists or geophysicists using their expert knowledge. Geological interpretation operates at the various stages of the workflow for deciding which surfaces should be modeled, how they should be assembled and which relations they should have with the internal stratification within each block of the model (Perrin et al., 2005).

The interpretation of the user about raw data is what gives it a meaning in the context where data are used (relevance) and for a specific objective (purpose). When we take an interest in the user perspective, we acknowledge that the user wraps the data in study in an interpretative envelope, giving the information a subjective meaning. It is argued that this combination of content and interpretation is what the user finds valuable (Stenmark et al., 2002).

Example. Lets take a naïve example, in the classical approach. Figure 6.2a can be interpreted by some geologist as a topological assemblage consisting in a lower surface A interrupted by an upper and older than surface B Figure 6.2b.

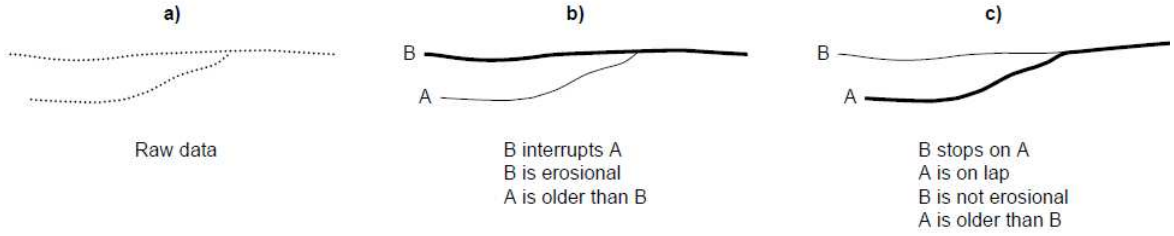


Figure 6.2: Changes in the geological hypotheses induce changes in the model

However, another geologist may want to consider that surface B is not erosional and that surface A is an onlap surface. It would consequently be desirable to have B stopping on A rather that the contrary (Figure 6.2c). Currently, this interpretation change is difficult to operate because the interpretation is not stored independently of the model.

This difficulty can easily be overcome if the geometrical/topological relationships between surfaces A and B are no longer considered as an intrinsic feature of the model. Surfaces A and B can then be considered as a two geological objects linked by specific geological relationships. Topology can then be simply deduced from *geological interpretation*, which then becomes an added value brought by the geologist to the raw data.

This can be clearly understood considering the portions of surfaces which lies at the right end side in the intersection between A and B on Figure 6.2a. This portion of surface belongs to B in the first interpretation, and to A in the second.

For the moment, considering the state of the art, it is not possible to both integrate data and interpretation, since the modeling tools used in the industry are data-driven tools. We present in the next section some approaches proposed and/or adopted by the petroleum industry for enhancing a knowledge-driven modeling workflow.

1.3 Solutions adopted by the petroleum companies

A brief history of semantics in the oil and gas industry was presented in the European Semantic Web Conference in 2005 (Braunschweig and Rainaud, 2005). The early 80's saw the advent of expert systems and knowledge-based systems (such as Drilling Advisor, Regent, Picon/G2). In the 90's, industry started considering *shared data models*, as well as ontologies and software interoperability, through projects such as POSC,[34] CLIP, OPC, CAPE-OPEN and Open Spirit.

The creation of a common working platform has always been the major concern of petroleum software vendors. However, from a user viewpoint, it is still very difficult to transfer data from one platform to another, without having to reformat the relevant files. Two tendencies have been emerging in oil companies (Cosentino, 2001): choosing to use only one vendor's platform, offering the whole chain of reservoir applications, or choosing to work with the best applications in the market and establish a loose, manual interoperability among different systems.

Considering the need of sharing modeling procedures between the various experts acting along the modeling chain, petroleum industry has been promoting a **Shared Earth Modeling** approach (SEM, (Perrin et al., 2003, 2005; Rainaud et al., 2005; Cosentino, 2001)). In the ideal view, SEM propose manners of integrating all the information acquired from different studies and results related to a reservoir and of sharing them among all the professionals and users involved. It would organize the work of multidisciplinary teams in a community of practice around the construction of common earth models (Fanchi, 2002). This is not a simple task since, in order to aggregate different models in a common environment, it is needed a common understanding about how the scientists and engineers of the various disciplines make use of their respective knowledge. Some multi-company initiatives envisaged integration platforms such as Open Spirit (OpenSpirit, 2000). In this case, application adapters are proposed, which enable applications to connect to OpenSpirit and access data from any data store within other plugged applications. However, this solution only works for OpenSpirit-enabled applications. This issue was addressed by another multi-company collaboration, the Epicentre Shared Earth Model (EpiSEM) project (Posc, 2001). EPISEM is an open data model that aims to be a standard for applications in petroleum exploration life cycle. Common terminology is important for information sharing. Mapping data objects to Epicentre gives a basis for sharing common concepts between data objects, applications, systems and users. A successful implementation of the Epicentre logical model was the RESCUE exchange format (REServoir Characterization Using Epicentre). RESCUE provided libraries with documentation to read and write all the information in binary files. However, RESCUE covered only the stage between petrophysical modelling and fluid flow simulation. Oil companies decided then to using XML technology as exchange format and extend its domain of coverage. The RESQML Special Interest Group was launched then to propose the RESQML technology. RESQML intends to provide documentation, XML schemas

[34] POSC now becomes Energistics, the Energy Standards Resource Centre (http://www.energistics.org/)

and ways to efficiently handle binary datasets with an open source library.

Oil and gas industry has taken advantage of **knowledge management** (KM) approaches for more than a decade. The goal of knowledge management is to capture explicit and tacit knowledge of an organization in order to facilitate the access, share, and reuse of that information (Dieng-Kuntz and Matta, 2002). One of the main focuses of petroleum companies concerns KM approaches for *processes and workflows*. This is due to the fact that the modeling chain for reservoir characterization is very complex. It involves several specialists from different disciplines, who perform activities that can be replayed several times, considering different hypothesis of interpretation, which can be introduced at several stages of the chain. Some declarations of petroleum industry leaders show that KM has been embraced in petroleum industry:

> "We must become experts in capturing knowledge, integrating and preserving it, and then making what has been learned quickly and easily available to anyone who will be involved in the next business decision." D.E. Baird, *Schlumberger*

However, after having been considered as the holy grail for some years, knowledge management practices still lack actual technologies to explicit knowledge about companies. In order to produce an actual system that manages the knowledge of the company, techniques that are original from various other areas must be incorporated: knowledge engineering, artificial intelligence, databases, computer networks, and so on (Boff and Abel, 2005).

At that point, petroleum companies started to search for concrete approaches for representing knowledge. Among the various approaches for handling domain knowledge, **ontology engineering** is one of the most cited ones regarding oil industry. There exist works which proposed formal conceptualization of geological domains in order to provide standard controlled vocabulary, and centralized conceptual models. We will review them in Chapter 7. However, one should clearly notice that creating "yet another ontology for the semantic integration issue" does not solve the problem, but may, on the contrary, increase heterogeneity. In the opinion of many, we should start considering possible reuse of formally developed ontologies. And that is where the **Semantic Web** comes.

Semantic Web techniques have been then adopted for addressing the semantic interoperability issues in petroleum industry domain. However, for cultural reasons, the petroleum industry has not yet completely embraced the distributed and shared solutions suggested by SW initiatives. Petroleum industry applications are developed in a context in which little semantic information is available on the Web. As a consequence, these applications are based on local, proprietary knowledge repositories, and produce and consume their own data, much like traditional knowledge-based applications.

Even though, some Semantic Web projects have been coming into sight. The W3C site for Semantic Web Case Studies and Use Cases (Herman and Stephens, 2007) describes testimonials on how Semantic Web technologies are used by companies and institutions. From the forty cases described, two were submitted by oil & gas industry:

- Chevron company (Chum, 2007), claims that still a large amount of heterogeneous data is generated every day from multiple sources such as seismic data, well data, drilling data, transportation data, and marketing data. In order to deal with the flood of information, as well as the heterogeneous data formats of the data, we need a new approach for information search and access.

The Chevron use case enumerates the main possible applications of Semantic Web technologies within the oil and gas industry. However, it remains for the moment a proposition that is not yet functional.

- The Active Knowledge Systems for Integrated Operations (AKSIO) project (Fjellheim and Norheim, 2005; Norheim and Fjellheim, 2006) is developing an integrated system in knowledge management to support drilling operations in offshore oilfields. This requires that data be linked together from databases, applications, and specialized knowledge networks. This needs to be combined with real-time data from the field to provide timely and contextual knowledge for collaborative work processes. Core functionality of the AKSIO system is provided by application of Semantic Web technology, including a drilling ontology defined in OWL language, semantic annotation of experience reports by experts, and integration of the knowledge base with work process.

The Integrated Information Platform (IIP) project (Omdal, 2006; Sandsmark and Mehta, 2004) is not cited in the W3C site, but is an important project based on Semantic Web techniques. IIP aims to create an information platform for industry by integrating ontologies from several industrial data and technology standards and also by creating new ontologies. This project integrates data and information for subsea seismic equipment, drilling, production, onshore operations and maintenance for vendors and operators, and expert centers with taxonomies and ontologies in a semantic markup language

The Semantic Days[35] is an annual conference that has become an important meeting place for industrial use of semantic technologies with contribution from industry, vendors and academia. The conference is located in Stavanger, which is the oil capital of Norway and, as such, has a preeminent oil-related research community.

The research community agree in one point: there is still little integration across phases and disciplines in the petroleum industry. Moreover, for the moment the few developments that lead to actual software products do not concern geology.

2 Requirements for a knowledge-driven solution for earth modeling

According to Rainaud et al. (2005), in order to operate knowledge-driven earth modeling, there is the need of:

- identifying entities belonging to different categories (raw data, interpretation and visual representations);
- keeping the memory of the data/interpretation/models attached to a prospect;
- keeping the context about each model (who, when, where, with what);
- checking the consistency and completion of the tasks undertaken and guaranteeing their quality of service inside the workflows that are operated; and finally,
- making eventually suggestions for improving these workflows.

[35] https://www.posccaesar.org/wiki/PCA/SemanticDays2009

We describe hereunder the items identified as current expectations for knowledge-driven earth modeling, why they are important to be taken into consideration and how the present work proposes to operate them.

2.1 Explicit representation of geological objects

The most frequent complain of modelers is that current models provide no explicit representation for geological objects. Rainaud et al. (2005) write that "... the geological surfaces included in a earth model should not be considered as geometrical or graphic items but as true geological objects." It means that geological objects such as horizons or faults are, nowadays, being expressed just by their visual representations within models and have no symbolic representation, not even as classes inside the code.

The main goal of defining explicit representations is allowing to keep information about geological objects and about their properties.

2.2 Explicit representation of chronological and topological relationships between geological objects

As we have shown through the naïve example depicted on Figure 6.2, there exists in geology a strong correlation between the age relationships and properties of the various objects, which enter into model and which are described by means of a geological interpretation on one part and the all over topology of this model on the other part. The topology of a structural model thus strictly depends on geological interpretation (Rainaud et al., 2005).

The interactive tools, which are available in most of the modelers presently in use in industry, allow the geologist to manually modify on the screen the spatial organization of a structural model. Doing so, the geologist implicitly changes the interpretation. Some surface will no longer appear as "erosional" but will become "on-lap" or some fault, which was originally older than a given horizon will eventually become younger. However, since no record is kept about interpretations, the geological consequences of the spatial organization changes that were made are just impossible to evaluate. One will eventually be able to spot topological differences between two versions of a given structural model but surely not to evaluate the consequences of such differences on the identities and properties of the various geological objects entering into the model and on their mutual age relationships. In other words, geology will remain so to say "hidden behind the model".

Claiming for a knowledge based Shared Earth Modeling Approach, Rainaud et al. (2005) stipulate that the identity of geological objects and relationships should "be preserved throughout the modeling chain". This approach supposes that geological interpretations should be made explicit and be duly recorded at each stage of the modeling process. As explained in the comment of the example of Figure 6.2, the structural model topology will then just appear as the result of geological interpretation and will consequently always be geologically consistent. Moreover, geological models will then be easier to compare (by just considering the geological interpretations on which they rest) and easier revise (by just changing some interpretation hypotheses).

2.3 Correlation about objects from different models

One last expectation of professionals is to be able, within any phase of the workflow, to ask questions related to the geological objects and also to data and project management. More specifically, users need to correlate geological objects issued from different fields. An example would be to determine "Which are the reflectors intersected by a well X". The answer to this query requires information from objects studied on two different activities to be crossed: reflectors from the *seismic interpretation* activity and wells from the *well correlation* activity. Currently it is not possible to answer these types of questions, since we have no way of correlating the data associated with the various models produced in the workflow.

This issue is important because, even though the meaning of geological objects and relationships is made explicit, this does not guarantee that the interpretation about these objects can be integrated interpretation generated on other fields.

2.4 Approaches employed in the present work for addressing the requirements

Ontologies are the approach that we have chosen for explicitly representing geological objects. We will consider in Chapter 7 which ontologies can be built for meeting earth modeling requirements. The developed ontologies constitute the vocabulary that can be used for describing geological interpretations in a formal way. Considering a practical use case related to the first stages of the modeling workflow depicted on Figure 6.1(1) and (4) (Seismic interpretation → Structural model), we will show in Chapter 8, how semantic annotation and ontology integration can be practically used for recording and retrieving information about interpretations, which eventually crosses the various considered domains

Chapter 7

Building Ontologies for Geosciences

Ontology building has not been considered in the first part of this work. The reason for this is the fact that it strictly depends on the application field that is considered. It is however an important issue and we will take the opportunity of our considering the particular field of geology and earth modeling for illustrating some of the issues that are currently met when building ontologies for a complex scientific and technological field.

A significant part of the activity developed during this thesis has been dedicated to the definition of ontologies for describing the knowledge related to the disciplines included in the case study. However, ontologies are not a goal in themselves. Developing ontologies corresponds to defining a set of data and their structure for other programs to use.

As explained before, the characterization of oil and gas reservoirs is based on the expertise of professionals from various fields in Geosciences. In these domains there exist several communication standards for exchanging measured data representation. However, no standards are available for exchanging and sharing *knowledge*. Thus the need has appeared for companies to agree on a common way of representing *knowledge about geological objects*. The type of knowledge about the categories of objects that exist in a domain, and about the manner in which these objects are organized, is called *static knowledge*, by the knowledge engineering community. On the other hand, the problem-solving behavior of domain experts is often called *operational* or *procedural* knowledge.

There are various types of formalisms for representing various types of knowledge about a domain. For example, *inference rules* are generally used for describing *operational* knowledge. Ontologies are agreed to be a formalism that captures the static knowledge required to a knowledge-based system. As a natural consequence, ontologies have been chosen as the knowledge-representation formalism for making explicit and sharing the common understanding about earth sciences domains.

1 Ontologies formerly developed for Geosciences

Several authors claim that an ontology is built in reference to a practical goal. This means that the aspects of reality that are chosen for encoding some ontology depend on the task. Noy and McGuinness (2001)

more precisely indicates that “the best solution almost always depends on the application that you have in mind and the extensions that you anticipate”. This point is a key for understanding the choices that were made by the various categories of geoscientists, who have already proposed solutions for geological knowledge formalization.

During the last years intense efforts have been developed by various organizations (geological surveys, geoscience consortia, oil companies) for issuing codifications and formalizations of geological knowledge. These can be classified, we think, in various categories according to the specific domains or activities that they address.

1.1 Geological surveys

Geological surveys are national or regional institutions, which are notably in charge of issuing geological maps. Their needs regarding knowledge formalization are those required for exchanging the information contained in field or laboratory observations and for linking it with objects represented on a *geological map*. The related knowledge thus concerns geological objects and geological observations in atomic scale, notably rock sample descriptions.

In 2005 the Arizona Geological Survey developed a conceptual model for *geoscience features* (Richard, 2006). Taking advantage of this pioneer work, a working group involving geological surveys from various countries including France (BRGM) further developed this GeoSciML model. GeoSciML formalization is based on the normative Geography Markup Language (GML) for the representation of *geographic features* and *geometry*.

The US Geological Survey designed the NADM model (North American Geologic Map Data Model, (Nadm, 2004)) as an ontology for developing interoperable geologic map-centered databases. The NADM model is designed to be a technology-neutral conceptual model and an interchange format using evolving information technology (e.g. XML, RDF, OWL) to allow geologic information sharing between geologic map data providers and users, independently from local information system implementation.

The Geosciences Network (GEON, (Lin and Ludaescher, 2004))[36] project is a collaboration among institutions and agencies to develop cyber-infrastructure in support of an environment for integrative geosciences research. The GEON project is interested in the problem of integrating geologic maps, whose source files contain geologic age or rock type information in the tables with different schemas and vocabularies. They are proposing an interoperability system which loads spatial data sets as OWL files and saves them into an ontology repository. The user is then allowed to define an articulation between the ontologies, which enables him to perform queries over multiple ontologies.

The geologic time scale has been also a conceptualization target of geological surveys, since the IUGS International Commission on Stratigraphy guidelines recommends a very precise usage of the time relationships in order to establish a standard time scale for use in global correlations. Cox and Richard (2005) presented a formal representation of the geological time scale using formal notation to enforce the precise definition of the relationships between the time components.

[36] http://www.geongrid.org/

1.2 Specific geoscience domains

Specialized ontologies were defined for specific geoscience domains. The work of Babaie, Oldow, Babaei, Lallemant, and Watkinson (2006) describes the major steps in defining a preliminary conceptual model of parts of a Structural Geology ontology. They claim that developing ontologies for geosciences is likely to become a complex task if all the concepts and relationships in the domain are included in a single, large ontology. So they propose a *component-based ontology*, that merges several homogeneous sub-ontologies describing sub-disciplines of structural geology (such as FaultGeoOntology and FoldGeoOntology).

Specific ontologies were also developed by Malik et al. (2007) for igneous rocks, a field, which is of little concern for us, and by Tripathi and Babaie (2008),for hydrogeology. In the field of petrology, a remarkable work in this category is the ontology developed at UFRGS (Brazil) and by the Endeeper company, for supporting the knowledge-based system *Petroledge*, in which expert knowledge is attached to observations of rock samples under the microscope and used by geologists for inducing palaeoenvirommental interpretations (De Ros et al., 2007). The Petroledge project notably investigates the cognitive mechanisms involved in the rock interpretation process.

The SWEET (Semantic Web for Earth and Environmental Terminology) project provides an upper-level ontology developed at NASA with coverage of the entire Earth system (Raskin and Pan, 2005). The SWEET ontologies include several thousand terms, spanning a broad extent of concepts from Earth system sciences (such as Earth Realm, Space, Time, Natural Phenomena) and related concepts (such as data characteristics) using the OWL language.

1.3 Petroleum industry

The Integrated Information Platform (IIP, (Omdal, 2006; Sandsmark and Mehta, 2004)) project completed one of the largest ontologies ever developed for an industrial field for formalizing the terminology used in petroleum production. The project address many domains, such as subsea production equipment, seismics, drilling and logging, reservoir characterization, well production, operation and maintenance but does not include earth sciences. Parts of the ontology are based on ISO 15926 standard, for oil and gas *production* life-cycle data (which considerably differs from the oil and gas *exploration* life-cycle), but they also include concepts issued from other terminologies. Within IIP, more than 40000 concepts have now been defined and modeled in hierarchical conceptual structures.

At present, some online ontologies repositories are being developed, such as the future Open Oilfield Ontology Repository (O3R)[37] project backed by the Chevron, Exxon and Total companies, which sets out to "collect public oil and gas ontologies and make them freely available to the industry at large". This portal will provide search, navigation and delivery via a process called "ontology-driven information retrieval". A similar proposal is the Open Oilfield Ontology Organization (O4OIL)[38], a repository of open oilfield ontologies launched by Schlumberger. However, both proposals are not yet operational for the time being.

[37] http://www.oilit.com/2journal/2article/0706_5.htm
[38] http://www.o4oil.org/o4oil.html

From 2001 on, the IFP/ENSMP team for Geo-modeling has developed a new knowledge-driven paradigm for reservoir studies based on the belief that geo-model building should not be directly dependant from data (data-driven) but rather from geoscientists' interpretations (knowledge-driven) (Rainaud et al., 2005). Perrin (1998) showed that geoscientists' interpretations related to structural modeling can be described with the help of an adequate "geological syntax". This work distinguishes different types of geological surfaces either polarized (corresponding for instance to stratigraphic horizons) or non polarized (corresponding to faults or thrust surfaces). It also defines properties that allow to assemble these surfaces so that the topology of their assemblage is consistent with their geological properties. In 2005, the IFP/ENSMP team issued the first version of a *Geo-ontology* describing the geological objects to be modeled and defining the syntax rules to which they must obey. This *Geo-ontology* , described in (Perrin et al., 2005), is the precursor of the ontologies that have been developed for this thesis.

As a result of our review of the ontologies developed till now for geosciences or for petroleum production industry, it appears to us that none of them is perfectly suitable for representing knowledge about 3D geological modeling. First, our needs for reservoir studies are not the same as those of geological map editors. For example, the choice made in the NADM model is to carefully store field and sample observations attached to the objects described in geological maps, taking little or no account of genetic considerations. This choice can hardly be ours, since reservoir models first intend to describe the geological history of a prospect with the final goal of quantifying the amount of hydrocarbons produced as a result of this history. Moreover, the few ontologies which concern petroleum industry were built for other workflows then that of reservoir characterization.

For this reason, a significant part of the present work has been dedicated to definition of ontologies describing the objects that are manipulated within earth modeling workflows.

2 Ontologies for 3D earth modeling

For defining *geoscience ontologies* (*Geo-ontologies*), I have taken advantage of the participation of École des Mines and of my participation to an ANR project entitled e-Wok Hub (Environmental Web Ontology Knowledge Hub).[39] The e-Wok Hub project extended over 3 years from mid 2006 to mid 2009. It associated professionals and researchers concerned by fields such as computer science, knowledge engineering, reservoir engineering, geosciences, belonging to seven French institutions (INRIA, BRGM, IFP, ENSMP, ENSMA, EADS). This multi-disciplinary research group studied *Semantic Web* solutions for extracting and managing interpretations obtained from documents concerning potential CO_2 storage sites. Two use cases were defined which respectively concerned *reservoir modeling* and *CO_2 storage site identification* and, in both cases, the main issue was managing the interpretations produced or used by geologists while performing their tasks.

Within e-Wok Hub , *Geo-ontologies* were defined by a group comprising geoscience experts and ontology engineers.[40]

[39] Public website of e-Wok Hub project: http://www.inria.fr/sophia/edelweiss/projects/ewok/

[40] The e-Wok Hub ontology group comprised two Geoscience experts, one Petroleum Exploration engineer, and three Knowledge engineers, including myself. I was the only one among Knowledge engineers to have some experience in the field of Earth science. For this reason among others, within the group I was specifically in charge of finnalizing the various conceptual

At the initial stage of the project, domain experts manually extracted vocabulary relevant to CO_2 storage from a set of representative text documents. Afterwards, considering the extracted vocabulary and additional key concepts proposed by the geoscience experts, the group started to define prior ontology conceptualization (concepts + relations) for various sub-domains of geosciences that had been defined as relevant for the project. This was done using a graphic knowledge-acquisition tool, the *IHMC CMap Tools* software[41]. This methodology well agrees with the recommendation by (Gómez-Pérez et al., 2000) who claim that the conceptual model should represent the problem-solver view of the problem and that, consequently, the phase of *conceptualization* is the most important before the development of any computer-based solution. The knowledge conceptualization should thus be addressed by first establishing the domain structure and its components by means of conceptual maps.

The results of this conceptualization work are presented in this chapter, in the graphical format of UML diagrams. Further on, in order to create a formal ontology from the conceptualization, e-Wok Hub ontology group chose to use the OWL language, which is a W3C standard for ontology development. Since the elements required to express the relationships in the *Geo-ontologies* fell within the restrictions of OWL DL, the group decided to represent the *Geo-ontologies* in that sublanguage. Conceptual maps were accordingly encoded into OWL-DL ontologies. The *Protégé* Ontology Editor[42] was used in order to modify the created ontologies, and to add missing properties and semantic restrictions when necessary. The RDFS/OWL version of the ontologies related to the e-Wok Hub project can be downloaded from the internet site of the project.[43]

2.1 Geoscience ontologies

In the course of earth modeling workflows, geology science can be seen as the red thread to which all local objects should be attached. We have thus defined a global ontology, the Basic Geology ontology, which refers to the geological objects used both in earth modeling workflows and in CO_2 storage site identification. Babaie et al. (2006) rightly claim that developing ontologies for geosciences becomes a complex task if all the concepts and their relationships in the field are included in a single, large ontology. Consequently, the e-Wok Hub ontology group decided to separate the Basic Geology ontology into **sub-ontologies**, which provide more detail to the main top-level concepts, and into other **domain ontologies**, which represent fields that are independent of the Basic Geology ontology, but whose concepts are used by the Basic Geology concepts. Therefore, the Basic Geology ontology describes and interconnects all the geological entities that must be considered for reservoir modeling. The other domain ontologies linked to the Basic Geology ontology are:

- the **GeoLocation ontology**: an ontology of geographical terms, which both rests on administrative nomenclature and on spatial (polygonal) area definition;
- ontologies for the disciplines of **Palaeogeography**, **Lithology** and **Hydrogeology**;

schemas that were elaborated as the first step of the development of the various ontologies that were produced. I notably dedicated much work to building the Geological Time and Geological Dating ontologies (*cf.* Section 2.1.2).

[41] http://cmap.ihmc.us/

[42] http://protege.stanford.edu/

[43] http://www-sop.inria.fr/edelweiss/projects/ewok/ontologyview/ontologies.html

- ontologies for defining and managing geological ages: **Geological Time** and **Geological Dating** ontologies.

We will first describe hereafter in Section 2.1.1 the Basic Geology ontology, which will play the key role of Global ontology (GO) in the use case that will be developed in Chapter 8 for illustrating the methodology proposed in our work for ontology integration.

We will then describe with some detail in Section 2.1.2 the Geological Time and Geological Dating ontologies. Considering these ontologies allows to catch some of the major difficulties that knowledge engineers are likely to meet when they intend to formalize a complex and mature scientific domain. In such a case, it is necessary that they develop an extensive discussion with the domain experts in order to capture not only some key concepts but also their relative importance, the way in which are mutually organized and the various constraints to which they are submitted. All this will be concretely illustrated when describing the Geological Time and Geological Dating ontologies.

Finally, we will present in Section 2.2 the ontologies that were developed for describing the concepts of the activities that precede the structural earth modeling: the Seismic interpretation and Well definition ontologies.

2.1.1 Basic Geology ontology

The Basic Geology ontology was built around the concept *GeologicalObject*. Geological objects are very diversified (examples among many others are: a stratified sedimentary unit, a reef, a diapir, a fault network etc.) and can be simple or complex. It is possible however to consider that *complex* geological objects are made of a various number of *atomic* geological objects. There are two kinds of elementary geological objects:

- 2D objects, that is, the **Geological Boundaries**, such as the erosion surface E, the fault F and the upper and lower boundaries b_u and b_l on Figure 7.1;
- 3D objects, that is, the **Geological Units**, such as the sedimentary unit U limited by the boundaries b_u and b_l on Figure 7.1. Geological Unit is a volume of continuous geological matter limited by one or several Geological Boundaries.

The sub-concepts of *GeologicalUnit* (e.g. Sedimentary Unit, Metamorphic Unit, etc.) and *GeologicalBoundary* (e.g. Genetic Boundary, Tectonic Boundary, etc.) are detailed in two homonymous sub-ontologies. A geological unit is filled by some substance, whose nature is detailed in the *Substance* ontology. The description of some types of substances (such as *Rocks*) uses terms imported from the *Lithology ontology*.

The various types of geological objects, such as Diapir, Reef and many others are detailed in the *Geological Objects* ontology. A given geological object is the result of some geological event (represented by the concept *GeologicalEvent*). A geological event may consist of a single geological process (e.g. the deposition of a sedimentary unit) or be composed of multiple geological processes (e.g. a metamorphic formation deformed by late tectonics). The various specializations of *GeologicalProcess*, corresponding to creation, destruction or transformation of geological matter, are detailed in the Geological Process

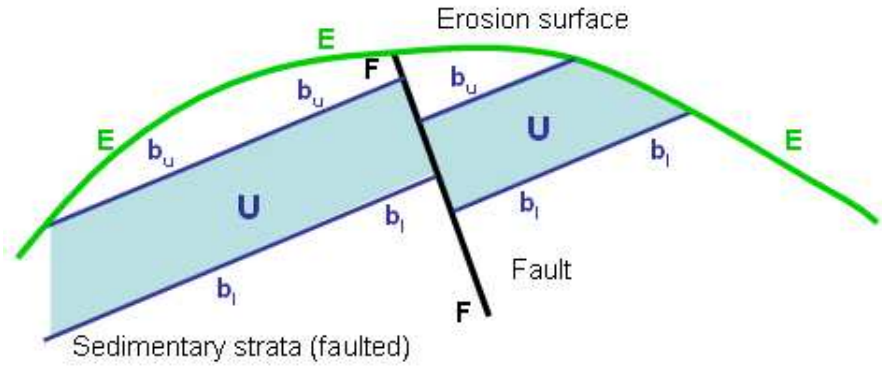


Figure 7.1: Geological objects: Erosion surface E, fault F and Sedimentary strata unit U, constituted of volume and boundaries b_u and b_l

sub-ontology. A geological event is responsible for geological object structures (e.g. Synform Fold, Reverse Fault). These structures are detailed in a *Geological Structure* ontology, which describes various geological items simple or complex (folds, faults, diapir for instance) considering their particular geometry or topology.

A geological object is precisely defined by its location and age. The object's geographic location is described by concepts of the *GeoLocation ontology*, while its age is described using the *Geological Dating* and the *Geological Time ontologies*. The top-level part of the Basic Geology ontology is shown on Figure 7.2.

The concepts *SedimentaryFormation* and *Reservoir* from the Geological Objects ontology are presented on Figure 7.2 so as to show the link to the *Palaeogeography* and *Hydrogeology* ontologies respectively. These ontologies are described in detail in Perrin et al. (2008). From the ontologies cited above, only the ontologies for geological time will be described in more details.

2.1.2 Geological time formalization

Considering geological time is essential from many points of view. As a historical science being made of a *succession of events*, geology can be described by specifying the chronological order in which geological events occurred. But since most of these processes consist in a combination or in a succession of possibly long-lasting events, they also need to be studied considering *time durations* at geological scale.

As it will be explained hereafter, the main interest of geological chronologies (Geological Time scales or absolute age determination) is their use for dating the geological objects that are described at the regional scale by geological regional maps or written documents and which also eventually enter into geological models.

After having briefly examined how geological time is currently described by geologists and which requirements this poses for geological time and geological dating formalization, we will present the two ontologies that were developed within the e-Wok Hub project respectively for describing geological time

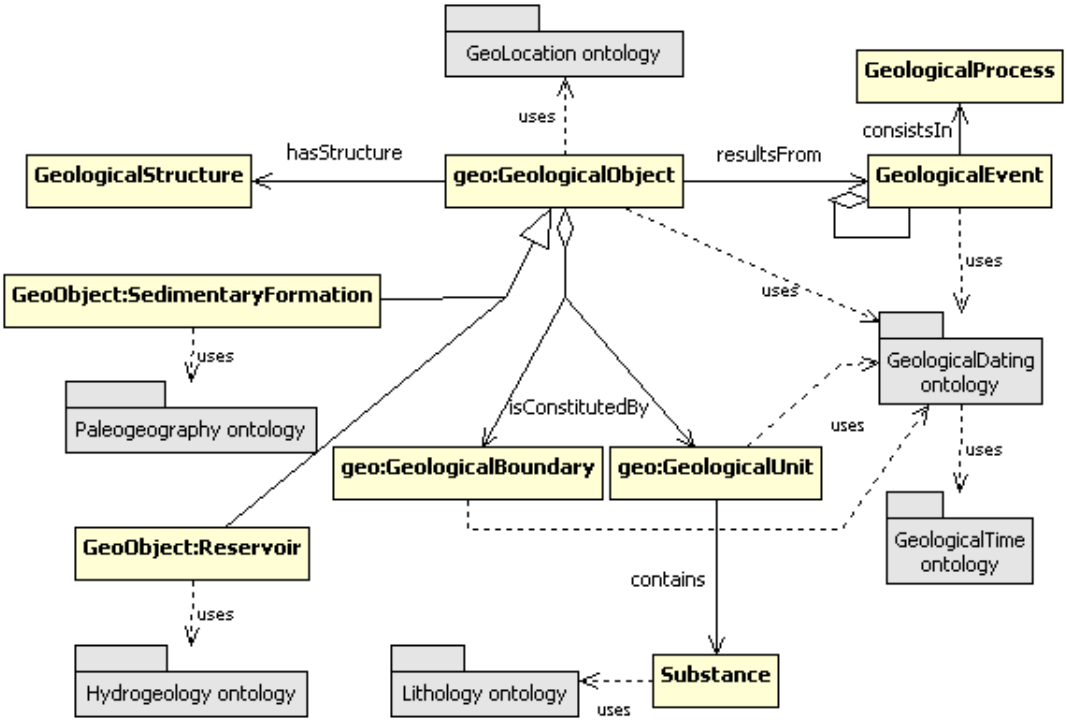


Figure 7.2: The top level Basic Geology ontology

scales (**Geological Time ontology**) and for associating time units and boundaries to actual geological objects (**Geological Dating ontology**) (Mastella et al., 2008c). This formalization well compares with those formerly proposed and notably with that of Cox and Richard (2005) used in NADM. It not only allows to affect ages to geological data and to chronologically order them but, in contrast to the above cited formerly developed ontologies, it is also a tool for easily operating stratigraphic correlations between time scales and/or between stratigraphic successions of any type.

2.1.2.1 Geological chronologies For developing an ontology describing geological time and its relations to geological objects through dating, it is necessary to consider some peculiarities of the methods that are currently used for attributing ages to rocks. The possibility of quantifying geological time and geological durations by using figures expressed in million years (My) only appeared a few decades ago. Such dating known as “absolute dating” is based on radioactivity and goes through evaluations of small quantities of radioactive and radiogenic matter trapped inside minerals. The corresponding laboratory measures are most often difficult and expensive to operate and their results are affected by significant numerical uncertainties.

For these reasons, absolute ages are not the tool most commonly used for geological dating. In most cases, geologists prefer using classical Geological Time Scales that were historically built considering particular geological events and notably fossil species appearances and disappearances. Geological time scales such as the *International Stratigraphical Scale* use an *event-based chronology*, which rests on the

stratigraphic model. This model was elaborated at an early stage of geological science development, when geologists observed that, in many places, underground is constituted by a succession of sedimentary layers, which each lie over the other and which generally have a great lateral continuity. Geological history is then seen as a succession of elementary events, each constituted by the deposition of an individual layer over older ones. In this model, sedimentary successions observed on the field materialize a slice of geological time and time correlations can then be operated by laterally following individual layers.

Since individual geological strata can hardly be followed in geometric continuity over distances overcoming a few kilometers, other criteria had to be considered for operating stratigraphical correlation at long distances and possibly around the entire world. For this, geologists took advantage of the quick evolution through geological times of some animal or vegetal fossil species. Considering that appearance and disapperance of particular fossil species were world scale events, they built a full event based chronology, which resulted in the *International Stratigraphic Scale* established by the International Commission on Stratigraphy (ICS) of the International Union of Geological Sciences (IUGS), described in (Callec et al., 2006).[44] It still constitutes the tool that is most widely used at present for dating geological objects.

The ISS divides geological time in *Geochronologic Units* of various *ranks*, each unit of rank n being included in a unit of rank $n-1$ as shown on Table 7.1. In a given rank, each geochronologic unit corresponds to a given slice of geological time, which is separated from its two neighbors respectively by a bottom and by a top boundary. These boundaries correspond to two punctual *instants* at the scale of geological time and have no duration. Moreover each elementary time slice thus defined is supposed to correspond to a reference stratigraphic succession (*stratotype*), which contains the fossil species that were used for defining it.[45]

At world scale, bio-stratigraphic correlation is not always possible. For instance, geological layers deposited in a continental environment only contain continental fossils and cannot be easily correlated with marine deposits. Moreover, coeval sediments deposited in sedimentary basins very distant from each others (for instance possibly belonging to different oceans at the world scale) may contain very different fossil associations. For these reasons, regional time scales were eventually defined in various parts of the world for describing stratigraphic successions, which could not be easily correlated to the ISS. For instance, in a significant part of Europe, most sediments deposited during the Triassic period are continental so that no direct correlation can be made with the ISS stratotypes, which all correspond to marine environment. Accordingly, there exist two geological time scales for Triassic in Europe: one (ISS) for describing marine sediments and another (a regional scale known as *Continental Triassic Scale*) for describing continental deposits (*cf.* Figure 7.3).

[44] The 2009 version of the International Stratigraphic Chart is presented in Appendix C.

[45] For instance, some geological formations exposed in quarries in the region of Oxford (UK) are considered as a reference for defining the Oxfordian stage. They constitute the stratotype of Oxfordian. Progress achieved in fossil studies and in defining new rock dating methods have lead the ICS to proposing in the seventies a new type of geological time standardization no longer based on stratotypes but on GSSP (Global Stratotype Section and Point) each representing the point in time at which a particular stage is starting (Gradstein et al., 2004).

2.1.2.2 Requirements for geological time formalization It results from what has just been exposed that geological time formalization should necessarily describe both the quantitative chronology corresponding to *absolute ages* and the *event-based* chronology expressed by stratigraphical time scales. Describing a chronology expressed by definite time instants is a simple issue, but dealing with event-based time scales is a more difficult problem that has not yet been object of much research work. This is the reason why we must consider with some detail the requirements connected with this last issue.

Geological time scales are globally organized as partonomies since each geochronologic unit of rank n corresponds to a fraction of the time span of the related unit of rank $n-1$. However they are far more than partonomies. The first reason for this is that the various units of rank n attached to a geochronologic unit of rank $n-1$ are chronologically ordered with respect with each others, each of them being either older or younger than any other one.

A second reason is that, within any time scale, there exists, attached to the set of geochronologic units, a dual set comprising the associated *time boundaries*. These various boundaries must be described and classified by specifying their links with the limiting geochronologic units. This supposes to take into account many synonymies since the top boundary of a given unit is equivalent to the bottom boundary of the unit which directly overlies it and since a given limit is likely to have different names depending from the rank of the units to which it is attached. An example is shown in Table 7.1, where all the listed terms correspond to one same geochronologic boundary: the base of the Triassic period.

Table 7.1: Synonym for the base of Triassic

Base of	Mesozoic	***Triassic***	Lower Triassic	Indusian	Buntsandstein	Lower Buntsanstein
Top of	Palaeozoic	Permian	Lopingian	Changhshingian	Upper Permian (Thuringian)	Tatarian
	*Rk*2	*Rk*3	*Rk*4	*Rk*5	*Rk*4	*Rk*5
	International Stratigraphic Scale				Continental Facies Scale (Europe)	

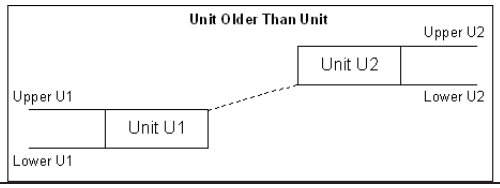
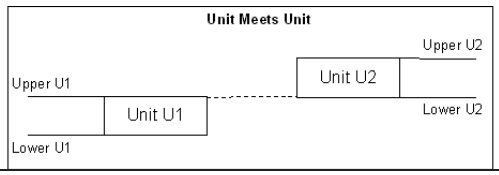
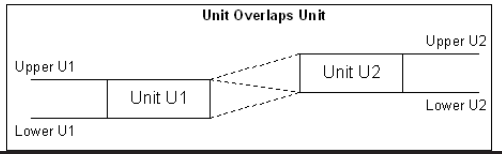
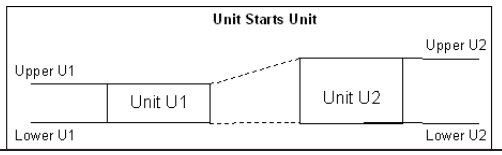
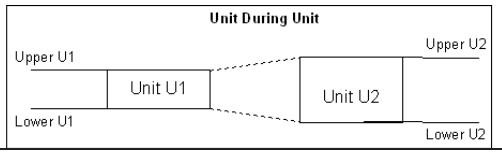
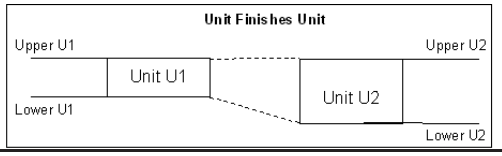
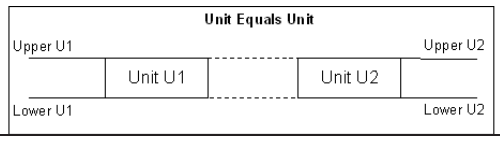
Moreover since both geochronological units and boundaries are *chronologically ordered*, it is necessary to specify the temporal relationships that they have each with the others. Possible relationships between boundaries and units were defined by Allen (1983), who proposed thirteen basic relations between time intervals, the *Allen's interval algebra*. An ontology describing geological time should of course offer the possibility of describing such possible relationships between geological units. Since the time units within a given geological time scale can be seen as *time intervals*, we decided to formalize geological time relations using Allen's interval algebra.

We depict in the first column of Table 7.2 some configurations of possible relations between units and boundaries in some GTSs. Next to it, we present the formalized interval relation that describe the configuration, and also the derived relationships, where:

- Unit 1 (U_1) and Unit 2 (U_2) are units from some Geological Time Scale (GTS);
- Upper unit 1 (UU_1) and Upper unit 2 (UU_2) are top boundaries of GTS units, Lower unit 1 (LU_1), Lower unit 2 (LU_2) are botton boundaries of GTS units

Moreover, within the framework of the e-Wok Hub project, the presented temporal relations were for-

Table 7.2: Temporal relationships for Geological Time Scales

Correlation	Unit to Unit relationships	Derived relationships
	U_2 older than U_1	UU_2 older than LU_1
	U_2 meets U_1	UU_2 equivalent to LU_1
	U_2 overlaps U_1	LU_2 older than LU_1 AND UU_2 younger than LU_1 AND UU_2 older than UU_1
	U_2 starts U_1	LU_2 equivalent to LU_1 AND UU_2 older than UU_1
	U_2 during U_1	LU_2 younger than LU_1 AND UU_2 older than UU_1
	U_2 finishes U_1	LU_2 younger than LU_1 AND UU_2 equivalent to UU_1
	U_2 equals U_1	LU_2 equivalent to LU_1 AND UU_2 equivalent to UU_1

malized as *inference rules* in the Corese language.[46] For example, the rule that infers that a given unit happens *during* another, from the correlation of their boundaries, is encoded as follows.

```
<cos:if>
{ ?unit1 :hasTop ?upperboundary1 .
  ?unit1 :hasBase ?lowerboundary1 .
  ?unit2 :hasBase ?lowerboundary2 .
  ?unit2 :hasTop ?upperboundary2 . }
</cos:if>
```

[46]Corese is an RDF engine based on Conceptual Graphs (see: http://www-sop.inria.fr/edelweiss/software/corese/).

```
<cos:then>
  { ?unit1 :during ?unit2. }
</cos:then>
```

These inference rules are applied whenever the user wants to discover, from the fournished relations, new possible correlations between time scale elements. The application of these rules is illustrated with an example in Section 2.1.2.5.

The main interest of geological time scales and of absolute age determinations is their use for dating geological objects. Formally dating consists in establishing a link between a geological age (stratigraphical or absolute) and some geological objects. Specifying what is the age of a definite object is not always a simple issue since some objects were eventually the result of a complicated succession of geological events. For instance, some sedimentary unit was possibly further transformed into a metamorphic rock, in which case it will have two ages: one corresponding to that of the original sedimentary deposit and another one to the age of the metamorphic event that later modified the lithology. Another example would be that of a complex fault along which various differential movements took place at various geological periods. In such cases and in all others, it is necessary that geologists specify to which event(s) should the age of a given object be attached in order to avoid all possible ambiguities. This also means that the age of a given object may correspond either to a given geological date related to some event considered as "instantaneous" at geological time scale or to a geological time span that begun at a given geological date and ended at another one and that had thus a significant duration. Formalization should take into account the above mentioned peculiarities and allow age attribution according to two different time formats, a chronological one referring to absolute ages and an event based one referring to some geological time scale.

Finally, another point should be mentioned concerning geological object dating, which is the fact that this dating is always the result of some interpretation. Several different ages resulting from several interpretations may thus be eventually attributed to one same geological object. In the Shared Earth Modeling approach that we are considering in the present work, there is thus the need that the dating versions attached to some geological object and their various characteristics (such as their author or the dates at which they were performed) be duly recorded. In accordance to what was exposed in then first part of this work, we recommend that this should be preferably done by using semantic annotations. For this reason, we consider here that the interpretative aspect of geological dating has no particular incidence on the dating ontology to be built.

2.1.2.3 Geological Time ontology The hierarchy of the geological periods of time as it appears in stratigraphical time scales is described using the *Geological Time ontology* that we have developed. Moreover, this ontology allows to establish correspondences between the ISS, which is the international standard scale for geological time (Callec et al., 2006)) and any other time scale based either on the use of the fossils or on absolute ages. An extract of the international standard scale for the geological time scale (ISS) is shown on Figure 7.3.

As we already mentioned, this stratigraphic time is composed, as most others, of geological *time intervals*, such as Quaternary, Pliocene, Middle Pleistocene, etc., which are chronologically ordered and are

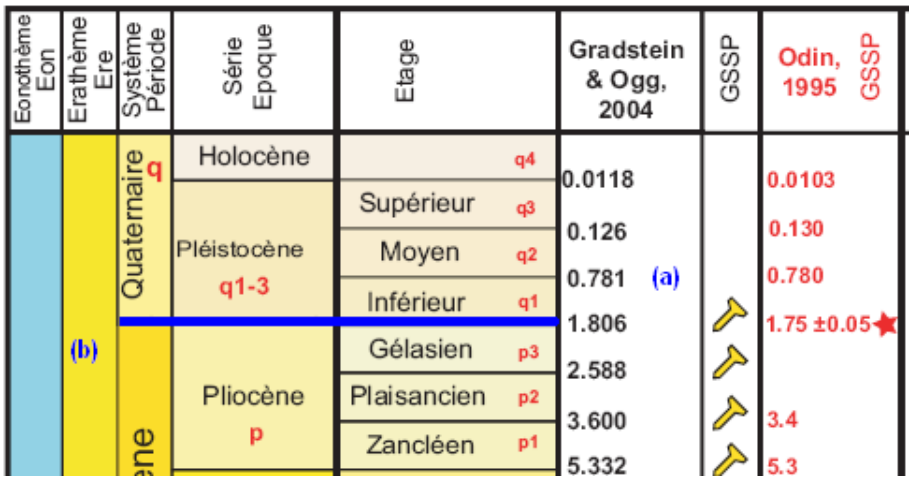


Figure 7.3: Extract of the International Geological Time Scale

contained one inside the others, and thus constitute a partonomy.

In our case, the concept considered for defining the relation between time intervals is the *boundary* between geological ages, one same boundary possibly limiting several time units of different ranks. For example, the boundary shown on Figure 7.3b is at the same time the lower boundary of the stage Lower Pleistocene, of the epoch Pleistocene, and of the period Quaternary. This information can derive several other relations about these three geological ages. For example, one can make statements about the partonomy of objects, such as *isPartOf(Lower Pleistocene, Pleistocene)* and *isPartOf(Pleistocene, Quaternary)*.

In this context, we have defined two main concepts to represent the main elements of the geological time scale: *GeochronologicUnit*, which represents geological time *intervals*, and *GeochronologicBoundary*, which represents geological time *boundaries*. These geological time boundaries represent the opposite idea of geological time intervals, since they are considered to correspond to *instants* having no temporal duration. Although these instants can be dated using absolute age measurements, it should be noticed that, in many cases, these absolute measurements are not precise enough to provide a non-ambiguous chronology. The Geological Time ontology is represented on Figure 7.4. The concepts and relationships related to this ontology can be described as follows.

- The abstract concept *GeochronologicElement* is the superclass of *GeochronologicUnit* and *GeochronologicBoundary*, and defines the relations *isYoungerThan* and *isOlderThan*, which eventually specifies the order of occurrence of the objects in the course of geological times.
- *GeochronologicUnit* instances are organized in a partonomy (i.e. by a *part of* relation).
- Units such as Eon, Era, Period, etc. are sub-concepts of *GeochronologicUnit* and are organized in specific partonomies: an instance of Chron is part of some Age, which is part of some Epoch, and so on.
- A *GeochronologicUnit* is described by some Reference System, such as the ISS or the Continental Triassic Scale.

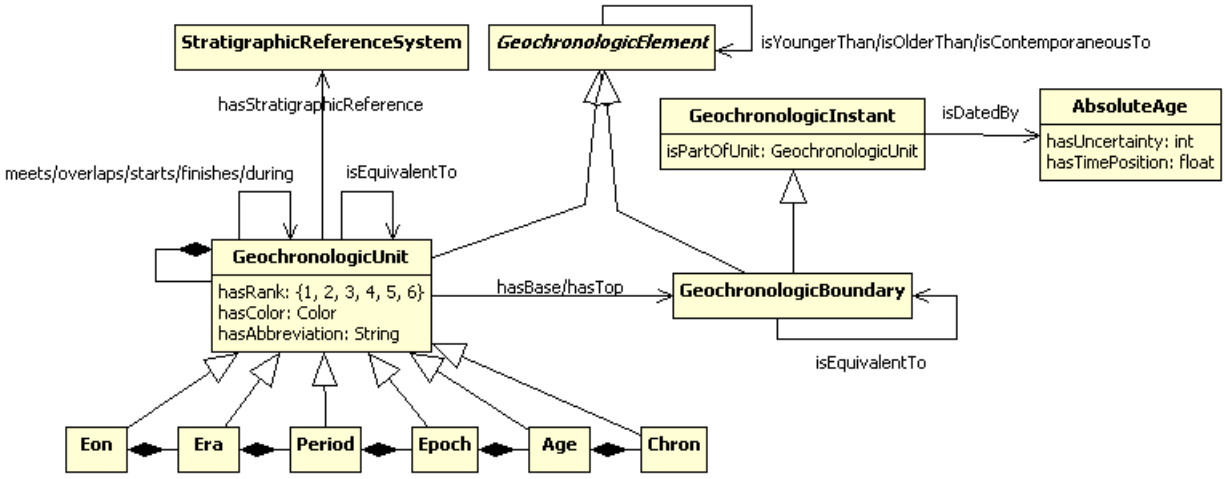


Figure 7.4: The Geological Time ontology

- The base and the top of some *GeochronologicUnit* is a *GeochronologicBoundary* (relations *hasBase* and *hasTop*).
- *GeochronologicInstant* is a generalization of *GeochronologicBoundary*, which represents one particular instant within a GTS which does not correspond to a boundary between units. For example, according to the absolute age figures mentioned on Figure 7.3 the date of 1.5 millions of years before present (1.5 My) does not correspond to any of the time boundaries represented on the ISS. Consequently such an instant, which does not correspond to the top or to the bottom of any unit just represents an instance of *GeochronologicInstant*. Its age may be expressed by an *AbsoluteAge* figure (1.5 My) or as a stratigraphical age (Lower Pleistocene).
- Actual GTS units, such as Triassic, Jurassic, and so on, are represented as *instances* of the concept *GeochronologicUnit* (in fact, as instances of the concepts *Eon*, *Era*, and so on). For example, Quaternary is an instance of the concept *Period*, and Holocene is an instance of the concept *Epoch* (see Figure 7.3).
- Actual boundaries between units are represented as instances of the concept *GeochronologicBoundary*.
- A *GeochronologicUnit* instance relates to other *GeochronologicUnit* instances by means of very detailed interval relations (e.g. *overlaps*, *meets*, *starts*, etc.) which enable to precisely describe their configuration. These relations are those defined by the Allen's interval algebra, as mentioned in Section 2.1.2.2.

One peculiarity of the Geological Time ontology that we have defined is that it can be applied for **geological time correlation**, that is, for creating a correspondence between elements of different GTSs, as shown in the example below.

Example. As we mentioned in Section 2.1.2.1, there exist two geological time scales for Triassic in Eu-

rope: one (ITS) describing marine sediments and the other (Continental Triassic Time Scale) describing continental deposits. These two scales are represented on Figure 7.5.

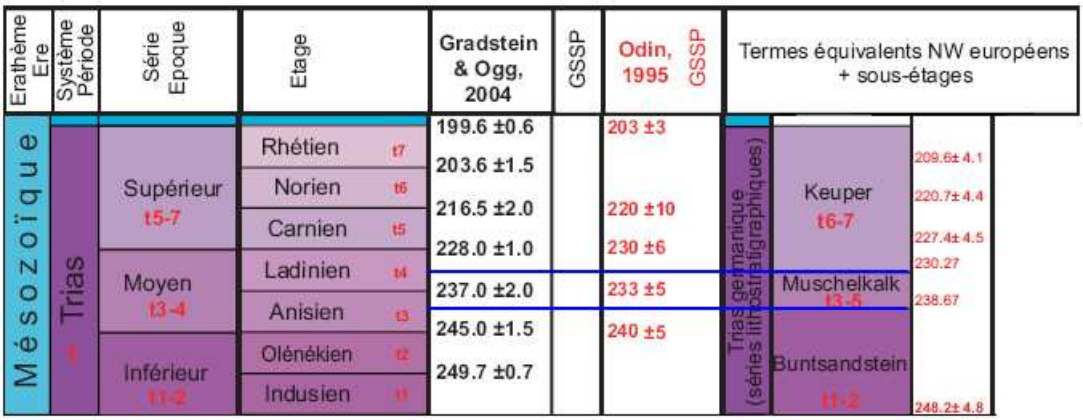


Figure 7.5: Correlation between units of different time scales

In this case, we could imagine establishing correlations between the two scales by referring to absolute ages. However, the absolute ages mentioned on Figure 7.5 respectively for marine and continental Triassic refer to two different absolute age scales. As a consequence of the differences existing between the two absolute age scales and of absolute age measure uncertainties, absolute ages cannot be used in this case for integrating the marine and continental Triassic time scales.

According to geologists' interpretations, correlations between units can be created by means of the *time interval relations*. One can assert that Buntsandstein begins at the same time than Triassic by stating the relation *starts(Buntsandstein, Triassic)*. A correlation can be made between the Muschelkalk and the Middle Triassic, since one occurs during the occurrence of the other (see the boundaries of the Muschelkalk which have been extended to show its relation to the ISS units): *during(Muschelkalk, MiddleTriassic)*. Another way of correlation is stating equivalences. For example, the German Triassic is equivalent to the International Triassic, even though their boundaries do not present the same absolute ages (*isEquivalentTo(GermanTriassic, Triassic)*).

2.1.2.4 Geological Dating ontology The *geological dating* procedure consists in affecting an "age" to any geological object. As we formerly mentioned, geological dating should take into account the two different ways of characterizing geological dates: *absolute* dating and *relative* dating.

In order to represent the particularities of geological dating, we defined the *Geological Dating ontology* (Figure 7.6), which introduces abstract concepts which make the link between concepts of the *Geological Time ontology* and of the *Basic Geology ontology*. The concepts imported from the Geological Time ontology have been given the prefix *GeoTime*, and those imported from the Basic Geology ontology, the prefix *BasicGeo*.

The concepts and relationships related to this ontology can be described as follows:

- *GeoTemporalEntity* is the abstract superclass that specifies the temporal relations *isOlderThan*,

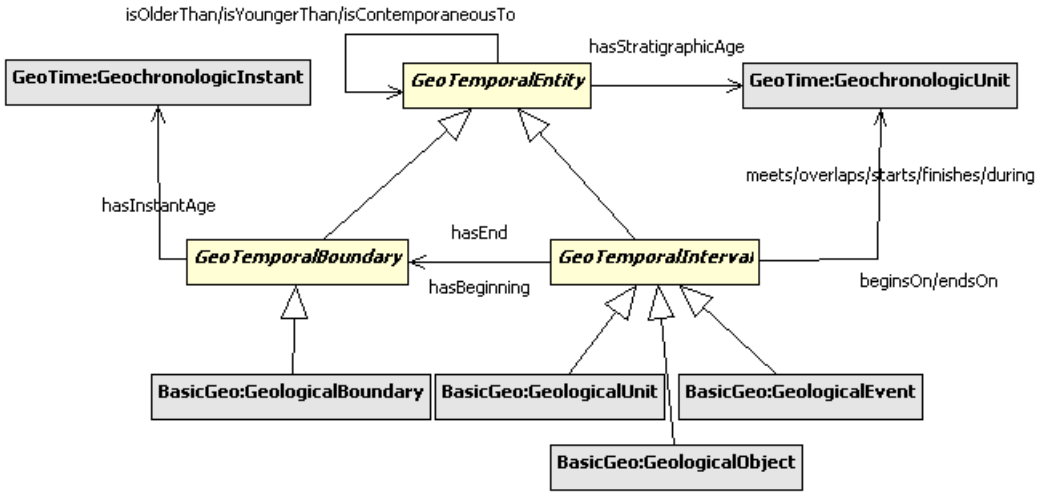


Figure 7.6: The Geological Dating ontology

isYoungerThan and *isContemporaneousTo*. A *GeoTemporalEntity* has a defined stratigraphic age, which is represented by the Geological Time ontology concept *GeoTime:GeochronologicUnit*.

- *GeoTemporalBoundary* and *GeoTemporalInterval* are abstract subclasses of *GeoTemporalEntity*. They inherit all relations defined to the superclass. Some *GeoTemporalInterval* begins and ends in some *GeoTemporalBoundary*.
- A *GeoTemporalInterval* may be dated by an interval relation with some *GeoTime:GeochronologicUnit* or by having a defined stratigraphic age, since it inherits the relation *hasStratigraphicAge*.
- *GeoTemporalBoundary* can be also dated by stating a stratigraphic age. Moreover, since it is a boundary, it may be dated by a specific instant in time, represented by the concept *GeoTime:GeochronologicInstant*. Notice that the instant can be either an absolute age or a GTS boundary, since *GeoTime:GeochronologicInstant* is a superclass of *GeoTime:GeochronologicBoundary*.
- The geological objects that can be dated are: *BasicGeo:GeologicalBoundary*, *BasicGeo:GeologicalUnit*, *BasicGeo:GeologicalEvent* and *BasicGeo:GeologicalObject*.
- *BasicGeo:GeologicalBoundary* is a subclass of *GeoTemporalBoundary*, inheriting its properties and those of *GeoTemporalEntity*. *BasicGeo:GeologicalUnit* and *BasicGeo:GeologicalEvent* are objects that have *duration*. So they are subclasses of *GeoTemporalInterval*, and can, thus, be dated by specifying a stratigraphic age, by defining an interval relation with some <GeoTime:GeochronologicUnit>, or by stating that it begins or ends on some <GeoTime:GeochronologicUnit> or at definite <GeoTime:GeochronologicInstant>.

The Geological Dating ontology provides to the geologist a very flexible tool for attributing a geological age to some object. One may either use all possible temporal relations or use just one relation and write inference rules that infer other ones. For the above example, inference rules can be written that infer the

date of some geological object by the date of the event that created it. A geological object can also be explicitly dated by means of the *beginsOn* and *endsOn* relations.

2.1.2.5 Example of practical use of Geological Dating A practical example of the application of the inference rules to Geological Dating can be showed from the results of the e-Wok Hub project.

The e-WOK client is a Web portal proposing the access to various services and applications, among which, the establishment of queries about the document bases. The users ask, in particular, to recover the documents which refer to a unit of the GTS. And the most important for the expert user is that the documents found refer to the unit explicitly chosen, but also, to the units that are equivalent to or included in the chosen unit. Here it lies the application of the *inference rules*.

Figure 7.7a shows the visual interface where the user is able to choose a geological time unit from the GTS, and then require to search for the related documents. The answer is a list of documents that refers to the *Jurassic* period or to a period that is equivalent or contained in the *Jurassic* period. We show in Figure 7.7b an extract of one of these documents, which was found because it makes reference to the *Dogger* period, which is an *equivalent* to the *Middle Jurassic* in the GTS. The *Middle Jurassic* occurs during the *Jurassic* unit, therefore, the *Dogger* also occurs during the *Jurassic*, and documents citing it should thus be retrieved.

This example is showed here to show how the inference rules work. The inclusion of inference rules in the OntoDB database is an issue that will be discussed in the Future Works section.

2.2 Ontologies describing specific earth modeling activities

Geo-modelers are interested in building 3D or 4D earth models for describing oil and gas reservoirs. In a Shared Earth Modeling, they wish to access to the knowledge attached to the modeled objects (geological units and boundaries, faults and fault network) and to their interpretation (identity and mutual age and topological relationships of the various objects). This supposes knowledge formalization, which not only consists in defining concepts and properties directly related to the Basic Geology ontology, but also in building local ontologies representing concepts attached to specific activities in the reservoir modeling chain, such as:

- an ontology for seismic interpretation, the **Seismic ontology**;
- an ontology for well description, the **Well ontology**.

The concepts of the Seismic Interpretation ontology (Figure 7.8a) were defined within the PhD work of Philippe Verney (Verney, 2009), since he is considered a user with expertise in the Geological Seismic domain. This ontology is detailed in Chapter 8.

The Well ontology (Figure 7.8b) was defined entirely based in a subset of concepts already elicited and formalized from the Drilling domain: the WITSML standard.[47] The only concepts that have been chosen

[47] WITSML (Wellsite information transfer standard markup language) is a XML standard, developed by Energistics, for data exchange between organisations in the petroleum industry (see http://www.witsml.org/).

(a) Chosing unit from GTS

having low permeability—have been suggested as potential long-term reservoirs for toxic materials such as nuclear or chemical waste. But information about the isolation properties of aquitard layers is essential to evaluate whether they can indeed be used safely as reservoirs. Here we investigate the long-term mobility of groundwaters between two aquifers surrounding an aquitard layer in the eastern recharge area of the Paris basin, France, using helium isotopes as a geochemical tracer. The deeper Trias sandstone aquifer, which lies above the crystalline basement, accumulates radiogenic 4He and primordial 3He from large regions of the crust and mantle at rates comparable to the degassing of the whole crust1 and of mid- ocean ridges2. W e show that the overlying carbonate Dogger aquifer, which is separated from the Trias aquifer by an aquitard layer consisting of a ,600 m succession of shales and clays, is stagnant and has been extremely well isolated from the Trias over the past several million years. This finding, together with pre- vious studies at the centre of the Paris basin3,4, shows that diffusive mass transfer across aquitards is negligible and that cross-formational flow in basins takes place preferentially in faulted areas. The long-term (105–107 yr) impact of human activity is a major environmental issue, in particular regarding confinement of toxic material in geological formations. Certain shale- and clay-rich successions which

(b) Extrait of document retrieved that cites the Dogger period

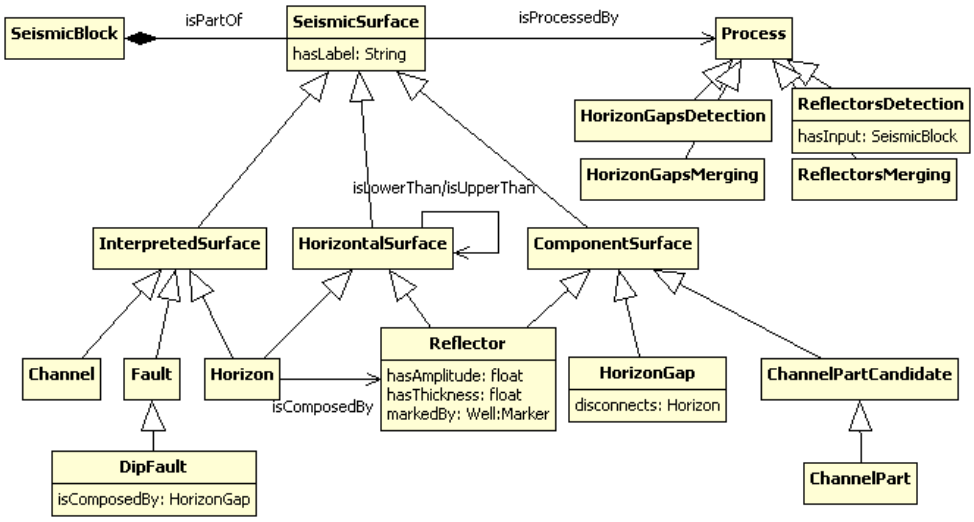
Figure 7.7: EWok Client: use of inference rules for searching documents

are those that related to the description of wells, which excludes considering the information regarding the entire drilling process itself. The most important concept of the Well ontology is *Marker*, which is part of a Wellbore, which is part of some Well. The *Reflector* concept in the Seismic ontology relates to the *Marker* concept through the *isMarkedBy* relationship.

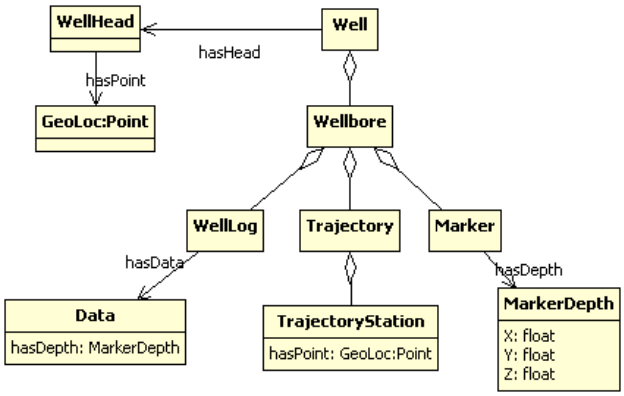
3 Persistence of ontologies in OntoDB

As the result of the work of ontology development that has just been presented, 5 domain ontologies in total were defined for the geoscience fields considered in this work, which approximately comprise 151 concepts and 137 properties. The repository used within our approach is the extended OntoDB ontology-based database (presented in Section 2 of Chapter 4), thanks to its support to model transformation and semantic annotation.

In the framework of the e-Wok Hub project, OntoDB was also chosen as an ontology repository, among others. Therefore, in order to provide a normalized access for *any type* of repository, a common access



(a) Seismic Interpretation ontology



(b) Well ontology

Figure 7.8: Local ontologies for Earth modeling

interface to ontologies was required. The e-Wok Hub partners chose to use SPARQL as the ontology exploitation language, since this language plays the role of a standard in the *Semantic Web* area. SPARQL was then implemented on top of OntoDB. More precisely, a sequence of OntoQL algebra operators (the *OntoAlgebra*, *cf.* Section 2.2.2 of Chapter 5) calls is executed after interpretation of a given SPARQL query. As a consequence, the e-Wok Hub project gets benefits from the OntoQL queries implementation and optimization (eWokHub, 2008).

The approach chosen in the present work is different, since we use just one repository, which is OntoDB. Therefore, the choice of OntoQL language as exploitation language is natural. Accordingly, the OWL ontologies resulting from the ontology engineering phase were stored in OntoDB using the OntoQL language as interface. Ontologies in OWL had to be thus transformed into *OntoQL descriptions*.

3.1 Mapping from OWL to OntoQL

As explained in Section 1.2.2 of Chapter 2 the relational model of a database is often not rich enough to represent most OWL concepts. For this reason, in order to transform OWL ontologies into OntoQL expressions to be executed over OntoDB, we had to consider only a suitable fragment of OWL. OWL constructs processable on OntoDB were translated on OntoQL by means of *mapping rules*. Table 7.3 shows some examples of OWL constructs that were mapped to equivalent OntoQL expressions.

Table 7.3: OWL to OntoQL mappings

	OWL	**OntoQL**
ontology name	<http://teste.fr> rdf:type owl:Ontology	SET NAMESPACE "http://teste.fr"
named class	:ClassA rdf:type owl:Class	CREATE #Class ClassA
labels	:ClassA rdf:type owl:Class; rdfs:label "ClassA-EN"@en, "ClassA-FR"@fr.	CREATE #Class ClassA (DESCRIPTOR (#name[en]='ClassA-EN', #name[fr]='ClassA-FR')
subclass	:ClassB rdf:type owl:Class; rdfs:subClassOf :ClassA.	CREATE #Class ClassB UNDER ClassA
single-valued object property	:propObj rdf:type owl:FunctionalProperty, owl:ObjectProperty; rdfs:range :ClassC.	PROPERTIES (propObj REF(ClassC))
single-valued datatype property	:propInt rdf:type owl:DatatypeProperty, owl:FunctionalProperty; rdfs:range xsd:int.	PROPERTIES (propInt INT)
multi-valued object property	:propMultObj rdf:type owl:ObjectProperty; rdfs:range :ClassC.	PROPERTIES (propMultObj REF(ClassC) ARRAY)
multi-valued datatype property	:propMultStr rdf:type owl:DatatypeProperty; rdfs:range xsd:string.	PROPERTIES (propMultStr STRING ARRAY)
instance	:someA rdf:type :ClassA, :propInt "2"^ xsd:int; :propObj :someC.	INSERTO INTO ClassA (URI, propInt, propObj) VALUES ("someA", 2, (SELECT oid from ClassC WHERE URI='someC'))

The table above only shows the mappings that can be directly made between constructs of both languages. The OWL language presents other types of constructors, that do not map directly to OntoQL constructs, but that had to be taken into consideration. Moreover, a subset of OWL constructs was simply not mapped

because they are not translatable to databases. We notably identified the following mapping constraints:

Named classes. The OWL-OntoQL mapping only identifies *named classes*, that is, classes explicitly defined in the ontology. Classes defined by *enumeration* or by *union*, *intersection* or *complement* are not considered, since databases have no support for this type of class construction. [48]

Cardinality. The maximum and minimum cardinality of properties in OWL can be defined in two manners. One can define a property as being a *owl:FunctionalProperty*, which states the maximum cardinality of the property to 1 and says nothing about its minimum cardinality. One can also create restrictions using the *owl:MaxCardinalityRestriction* and *owl:MinCardinalityRestriction* resources onto some property.

If the cardinality of the property has a maximum value of 1, the property is considered to be single-valued, and is mapped to OntoQL in the same way of single-valued properties on Table 7.3. Otherwise, it is considered to be a multi-valued property, and it is mapped to OntoQL a property using the ARRAY keyword, as shown on Table 7.3. The *owl:MinCardinalityRestriction*, on the other hand, is not translated, since the idea of required or optional property has no equivalent in OntoQL.

Domains and ranges of properties. The OWL-OntoQL mapping will treat the ontology in a frame-view. A property will thus be associated to the classes that are defined as the property ranges. Moreover the mapping also recognizes ranges that are defined by restriction on the property. In OWL one can notably define the range of some object property by specifying that, for that property, *all* values or *some* values come from a specific class (OWL restrictions *owl:allValuesFrom* and *owl:someValuesFrom*). In the example below, the range of property *propObj* (when applied to class *ClassA*) is the class *ClassC*.

```
:ClassA rdf:type owl:Class ;
  rdfs:subClassOf
   [ rdf:type owl:Restriction ;
     owl:onProperty :propObj ;
     owl:allValuesFrom :ClassC
   ].
```

The mapping recognizes properties that have more than one class in the domain (union classes). The property is then spammed over all classes when it is translated to OntoQL. However, the mapping does not recognize union classes in the range of a property, since multi-range properties are not handled by OntoQL. Missing domain/ranges are explained next.

Names of instances. In OWL, instances are differentiated by their names. An instance name in OWL is unique and is stored in the *rdf:ID* attribute. Instances in a database are rows within some table, and are generally differentiated by the table unique identification, which, in the case of OntoDB, is the numeric attribute *oid*. Therefore, in OntoDB one can have multiple instances that have the same name. Consequently, the OntoQL language does not propose a built-in attribute for storing an instance name. For this reason, we had to create an equivalent of the *rdf:ID* attribute that applied to *every* concept of all domain ontologies. We called this attribute *URI*. The value of the *rdf:ID* attribute of some OWL instance will be copied into the *URI* attribute in OntoQL, just like we show in line 6 of Table 7.3.

[48] A class is defined by *enumeration* when it is described by exhaustively enumerating its instances. A class can be also defined by the description of a list of classes in which one applies logical operators such as AND, OR, NOT (respectivelly, *intersection*, *union* and *complement*).

No Domain/Ranges of properties. In databases, a property will always be defined within the domain of some table and will point to some range. In OWL one can create properties independently of classes, that is, with no domain and/or no range. This causes a difficulty when translating an OWL ontology to databases.

In order to handle that issue, we executed the following actions:

- First, a top-level class was added as parent class of all concepts of the domain ontologies. We called this class *OWLRootClass*.
- When an OWL property was found having undefined domain/range, we assumed that the property might be applied to *any* of the concepts within the ontology. For this reason, we added the *OWLRootClass* concept to the missing domain/range. In this way, all concepts of the ontology will inherit that property from the super-class *OWLRootClass*.
- The creation of the super concept OWLRootClass also allowed us to handle the issue of the *URI* attribute described above. Since it needed to be created in every ontology concept, we defined *URI* as being an attribute of *OWLRootClass*. Then, *URI* will also be inherited by all concepts of the ontology.

Multiple inheritance. Multiple inheritance of classes (as in Figure 7.9a) is not handled by OntoDB. We simulated this by mapping the extra super classes to the *is-case-of* relation recently implemented in OntoQL (*cf.* Section 2 of Chapter 5). The *a priori case-of* operator is applied, since the *is-case-of* relation is being established in the moment when the class is being created. Figure 7.9b shows the result.

```
:ClassD rdf:type owl:Class ;
   rdfs:subClassOf :ClassA ,
                   :ClassC .
```

(a) Multi-inheritance in OWL

```
CREATE #Class ClassD
UNDER ClassA ISCASEOF (ClassC) (
IMPORTS(ClassC.propC))
```

(b) Multi-inheritance in OntoQL

Figure 7.9: Mapping multi-inheritance to OntoQL

Order of instance creation In OWL it is possible to have *cycles*, that is, instances that reciprocally make reference one to the others. In a database, an instance must exist before, in order to be referenced, thus, referencing cycles are not allowed. In order to overcome this problem, when translating the individuals of some OWL ontology, we first created all instances with their *URI* (as shown in code (a)) and with no property. This ensures that we will not create properties that make references to instances that were not yet created. When all instances are created, we update these instances adding their properties, as in code (b). The line *instance* of Table 7.3 is, in fact, separated in two operations: insert and update.

In order to implement the OWL-OntoQL mapping, an OWL-OntoQL translator was created using the JENA framework[49] which loads the objects from an OWL file to the work memory. The algorithm uses the Jena API methods to browse all entities of the given OWL ontology (classes, properties, instances, restrictions, and so on). It then applies the above defined rules in order to create an equivalent OntoQL expression. The result of the mapping is saved afterwards in a text file, in which each OntoQL expres-

[49] Jena is an open source Java framework that provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine (see http://jena.sourceforge.net/).

```
INSERT INTO ClassA(URI)
VALUES ('instanceA')
;
INSERT INTO ClassC(URI)
VALUES ('instanceC')
```

(a) Inserting instances

```
UPDATE ClassA SET propObj='instanceC'
WHERE URI = 'instanceA'
;
UPDATE ClassC SET propObj='instanceA'
WHERE URI = 'instanceC'
```

(b) Updating instances

Figure 7.10: Mapping instances from OWL to OntoQL

sion is separated from the other by the character ';'. This defines a OntoQL script that can be loaded and executed by the OntoQLPlus interface (*cf.* Section 3.3 of Chapter 3). The domain ontologies developed for reservoir modeling, that is, the Basic Geology, Geological Time, Geological Dating, Seismic Interpretation and Well ontologies were all processed by the OWL-OntoQL translator. The created OntoQL script was executed so as to store all domain ontologies in the OntoDB OBDB.

4 Conclusion

In order to acquire and formalize the knowledge about geoscience domains, ontologies were built for describing the fields of basic geology, geological time and dating, seismic interpretation and well identification. This appeared necessary since the previous formalizations of geological ontologies proposed by various organizations that we examined were oriented towards goals that were different from those related to earth modeling. So, the ontologies developed in this work specifically represent the view of the geoscientists involved in *earth modeling*.

The Basic Geology ontology describes concepts from the Geology science, which are shared through all the geosciences disciplines. The Basic Geology ontology was built around the concept Geological Object, which is declined in Geological Units and Geological Boundaries. Since Geology is a historical science, we also formalized the *temporal aspect* of geological descriptions. This formalization was exposed with some detail since it provides a good example for understanding some of the difficulties that are likely to be met when formalizing knowledge related to some complex scientific domain. Geologic ages may be expressed in different ways. One consists in expressing them as *absolute ages* (figures expressed in million of years). However, since absolute ages may be determined with variable precision, they are not the tool most commonly used by geoscientists for geological dating. Conversely, geologists most commonly use geological time scales (GTS's) that define an event-based chronology based on *events* such as apparition or disappearance of various fossil species.

We studied GTS's with some detail in order to acquire the main elements that were necessary for representing geological ages defined in relation with such event based chronologies. In GTS's, geological ages are based on the beginning and end of specific events. Accordingly, the two basic concepts that are to be considered when defining geological time ontologies are *Geochronologic Boundary* and *Geochronologic Unit*. The concept of Geochronologic Boundary is used for defining to a point within geological time. Geological ages are represented by Geochronologic Units, which can be of various ranks: Eon, Era, Period, and so on. The lower and upper Geochronologic Boundaries attached to a given Geochronologic Unit respectively identify the time points at which this unit begins or ends. Mutual relationships between

Geochronologic Units and/or Geochronologic Boundaries are commonly used for specifying age relationships between elements belonging to one given GTS. We showed however that by using inference rules deduced from a set of sophisticated temporal relationships based on the Allen's interval algebra, it is also possible to describe the relationships that exist between elements belonging to two different GTS's. Since there presently exists no tool for operating time between GTS's, the geological time formalization that we developed appears as a significant contribution.

A direct consequence of the definition of the Geological Time ontology is the need of a tool for affecting the ages to geological objects. We thus proposed a *Geological Dating* ontology, which introduces abstract concepts that make the link between concepts of the Geological Time ontology and of the Basic Geology ontology. We showed an example of a practical use of the Geological Time and Geological Dating ontologies using the results of the E-WOK HUB project.

We finally defined ontologies for the specific fields of geosciences used in earth modeling activities. The Seismic Interpretation and Well Identification ontologies that were defined re-use concepts described in the Philippe Verney's PhD Thesis. All the defined ontologies have been described in the W3C standard OWL language, and stored in the OntoDB ontology-based database. The problem of having to translate "semantic web ontologies" to a data-centric semantic repository such as OntoDB was handled with transformation rules from the OWL model to the OntoDB model. We described the mappings operated between OWL and OntoQL primitives, and we detailed some issues that prevent a direct mapping between the primitives of both models, such as the fact that OntoQL cannot represent logical operation among classes.

Significant new developments are presently undertaken using the Geological Time and Geological dating ontologies that we have defined. In complement to our ontology proposition, we are presently studying with Michel Perrin (École des Mines) a methodology for a *codification* of geological time units and boundaries applicable to any time scale or stratigraphic succession. We intend to define a full codification of ISS and of regional European time scales. It will allow the user to find easy answers to all questions related to identification and correlation of geological time units and boundaries, such as "Which are the units that are immediately next some given unit?"

In order to be fully operative, the above work will be completed by an interface allowing users to ask questions and to visualize answers in a suggestive way. The interface will allow the user to display any part of any time scale and to show local correlations between different time scales. The main time scales (International scale, European scale etc.) will be stored in the system knowledge-based and the user will be able to ask any questions related to these time successions or to their mutual relationships. The interface will also allow the user to enter any new chronostratigraphic succession manually or in an assisted way.

We believe that the Geological Time and Dating ontologies, completed by the geological time codification and implemented in a graphical interface will constitute a powerful tool for geoscientists. It not only allows to attribute ages to geological data and to chronologically order them but, in contrast to the formerly developed tools, it can also be used for easily operating stratigraphic correlations between time scales and/or between stratigraphic successions of any type.

Chapter 8

A use case for semantic annotation and ontology integration: workflow for seismic interpretation and structural modeling

In this chapter we describe a use case concerning the reservoir modeling chain for petroleum exploration. This will illustrate how the approaches proposed in Chapter 4 and Chapter 5 can be employed in an actual case study.

1 A scenario concerning the seismic interpretation activity

As we explained in Chapter 6, reservoir modeling workflows involve many different stages each consisting in many elementary activities. We have chosen to consider here the activities related to seismic interpretation, which usually correspond to the first stage of reservoir model building. One reason for this choice is that, thanks to the work recently performed at IFP and ENSMP by Verney (2009) the use case can be used for fully illustrating a "white-box" scenario of annotation (*cf.* Section 1.2.1 of Chapter 4) and validating the proposed approach.

For a full understanding of the use case, we will first briefly describe here two different approaches for seismic interpretation. The first one corresponds to a *data-centric approach* based on "manual" surface recognition and handpicking. It is the method most commonly used at present in the petroleum industry. The second one corresponds to the *knowledge-based approach* supported by P.Verney's approach. In contrast to "manual interpretation", this approach based on cognitive vision enables to perform seismic interpretation in a partly automated way and to keep the memory of the interpretation choices made by the interpreter.

1.1 Presentation of the Alwyn prospect

The data on which the two considered approaches were applied are those of the **Alwyn** exploration field operated by Total UK company in a portion of the North Sea located on the East of the Shetlands.[50] The

[50] These informations are presented with the authorisation of Total.

geology of the Alwyn field is pictured on Figure 8.1.

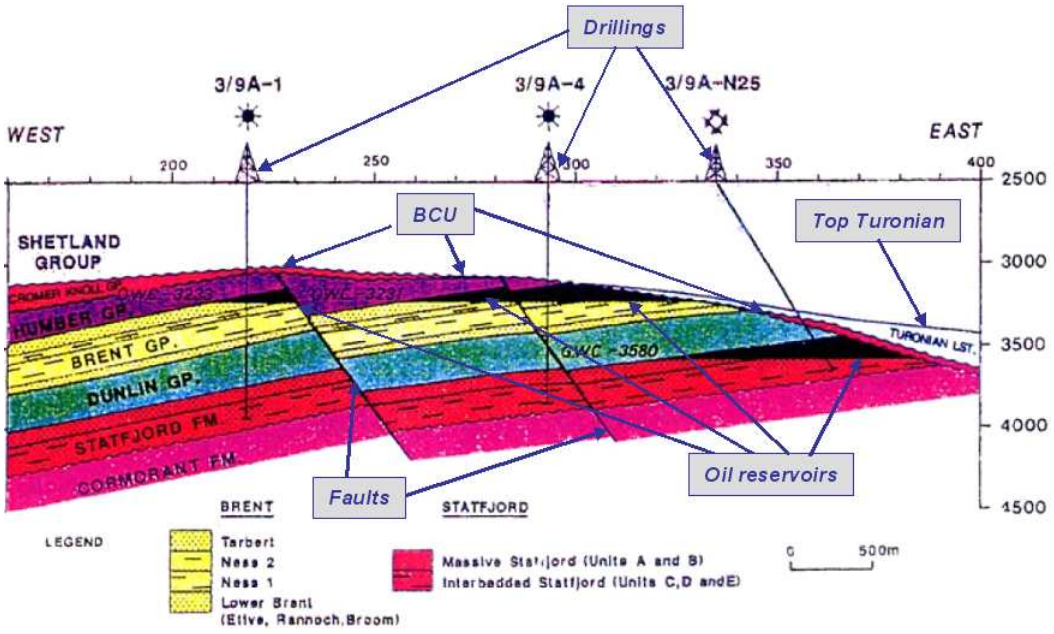


Figure 8.1: Geological section across the Alwyn exploration field

The zone of interest for oil prospecting is located under an erosion surface known as **BCU** (Base Cretaceous Unconformity), which corresponds to the upper limit of the geological formations (the colored part of Figure 8.1). This zone shows a succession of 6 sedimentary formations i.e.from bottom to top: *Cormorant and Statfjord Formations* (PermoTriassic) / *Dunlin Group* (Lower Jurassic) / *Brent Group* (Middle Jurassic) / *Humber Group and Cromer Formation* (Upper Jurassic). Among these formations, the Brent group is of particular interest since it hosts several oil reservoirs (pictured in black on Figure 8.1). It is itself divided into 4 units known from bottom to top as: *Lower Brent*, *Ness 1*, *Ness 2*, *Tarbert*.

During late Jurassic, due to tectonic efforts, all the above mentioned formations were *tilted towards West*, and *split by faults*. These events were followed by an intensive phase of erosion. The BCU surface is the resulting erosional surface. *Sedimentation* resumed during Cretaceous inducing the deposition of other formations, which are not represented on Figure 8.1. This figure only shows over BCU one particular surface corresponding to the top of the Turonian deposits.

The available data for interpreting the Alwyn prospect consist in a 3D seismic block illustrated in cross-section on Figure 8.2 and in well logs related to 7 vertical drillings.

1.2 "Manual" interpretation of Alwyn prospect

The Alwyn prospect is the support of a training session of the IFP School dedicated to seismic interpretation. In the course of this session, students perform manual interpretation of the seismic and well log data. Their interpretation is then validated by their supervisors, who possess an extensive knowledge of the Alwyn prospect.

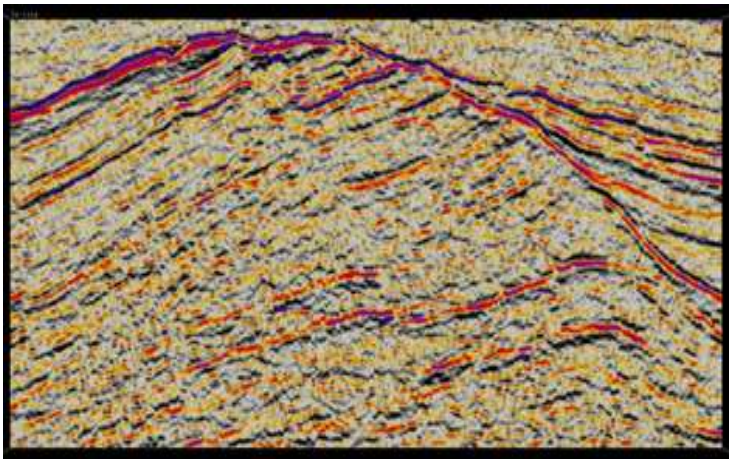


Figure 8.2: Seismic cross-section of the Alwyn block

The seismic data corresponding to the Alwyn block is provided in the **SEG-Y** format. The name of the file in analyze is *IL16601949.segy*. Figure 8.2 shows a cross section of the Alwyn block chosen by the interpreter, which can be interpreted by considering:

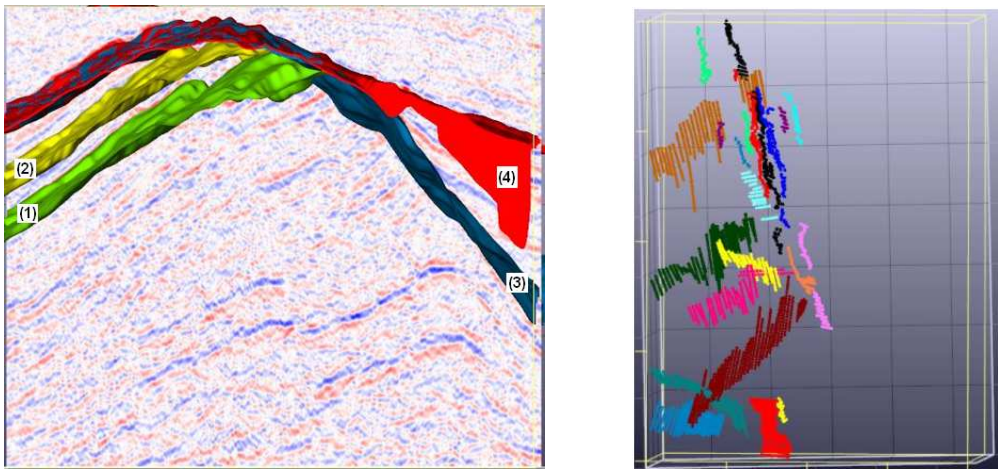
- some of the *reflectors*, which correspond to the various colored horizontal traces on the figure and which can be interpreted as parts of geological horizons;
- aligned *horizon gaps*, which can be interpreted as portions of fault surfaces (*fault mirrors*).

The interpreter, who operates a manual interpretation, selects points that are part of some reflector or that correspond to a horizon gap by manually *picking* these points over the cross section image. In a second phase, the set of reflectors and gaps are re-interpreted in order to reflect real geological objects. Reflectors that are aligned and visually similar are considered as corresponding to one same *horizon* and aligned horizon gaps corresponding portions of fault mirrors are used to identify full *fault surfaces*. The result of the manual picking operated by IFP School students for horizon identification is shown on Figure 8.3. The horizons identified are: Top Dunlin on Figure 8.3a(1), Top Brent on Figure 8.3a(2), BCU on Figure 8.3a(3) and Top Turonian on Figure 8.3a(4). The 28 faults identified are illustrated on Figure 8.3b.

Subsequently, the interpreter analyzes *well log files*. From the interpretation of the well logs, he/she produces for each well bore an ordered list of *markers* each corresponding to the intersection between the well trajectory and some geological surface. The most important reflectors identified from the seismic image are then manually correlated with the markers identified in well logs. This allows the interpreter to put labels on each of the identified horizons. It is thus possible to put these horizons in their right geometric position, which is not given by seismics but only by well log data.

As a result of their extraction from the seismic image by hand-picking, horizons or faults result from seismic interpretation are represented as “clouds of points” (in a format such as **XYZ** or **PLO**). At this stage, the user is not able to store any other information about the identified surfaces or about their mutual age relationships.

In order to build a structural earth model, files corresponding to interpreted seismics are imported into a



(a) Interpreted horizons (b) Interpreted faults

Figure 8.3: Manual seismic interpretation of the Alwyn block (IFP School)

modeling application. The geoscientist assembles the objects according to a definite topology in order to produce a geologically consistent 3D model. However, while this model is saved, the interpretation hypotheses carried out by the expert in order to produce it are commonly lost. As a consequence, there exists no possibility for later checking out the consistency of the model topology with the relative ages of the surfaces, deduced from well log information or from regional geology data. This is one of the dramatic consequences of this manual data-centric approach.

1.3 Semi-automatic interpretation according to a *white-box annotation* scenario

As a use case for illustrating the new method that he is proposing, P.Verney has provided in his doctoral work (Verney, 2009) results obtained on the Alwyn prospect data by operating a semi-automatic interpretation based on a cognitive vision approach.

In a first stage, P.Verney automatically extracts from the Alwyn cross-section shown on Figure 8.1 about 100 reflectors and almost 2000 horizon gaps (the exact number varies depending on the amplitude threshold chosen for the interpretation). Each reflector and horizon gap are represented by a unique cloud of points file (in the cloud of points format named **PLO**).

P.Verney performs the following phases of his interpretation in a largely automated way using a knowledge-oriented approach that we will shortly describe in the case of horizon identification. Verney's method for identifying geological horizons rests on the *Seismic Interpretation ontology*, which was presented in Chapter 7 and illustrated on Figure 7.8a on page 159. This ontology stipulates that each horizon is composed of various reflectors; that a fault is composed of gaps that disconnect horizons; a reflector is associated to a well's marker; horizontal surfaces may be lower or upper than other surfaces. All of these objects have been processed by some process (such as Reflector Gaps Merging, or Reflectors Detection), whose input is the seismic block.

Using this formal description, P.Verney is able to automatically operate the fusion of reflectors which have compatible visual features (amplitude, width and direction) and the same relationships with other

above or bottom positioned reflectors. It is thus possible to automatically identify various *horizons*, which can then be correlated with well markers using a horizon to well distance criterion. In a similar way, P.Verney also defines a knowledge based method for identifying *faults*.

For characterizing the horizons that have been found, it remains to adjust each of them to some of the markers identified along the various well bore trajectories. The well logs from the 7 wells that were drilled in the Alwyn propect are represented in the LAS format. Horizon adjustment is automatically operated, each horizon being adjusted to the markers that are located at a minimum distance inferior to a given threshold. At the end of the final operation, the user can save his interpretation by storing the identified surfaces in *cloud of points* files.

In the semi-automated approach proposed by P.Verney, the main task of the user actually consists in providing *threshold* values: the reflector *amplitude*, *width* and the tolerated angular uncertainty on directions; the tolerated uncertainty on neighbor reflector vertical distances; the minimal distance for connecting a given horizon to some well marker.

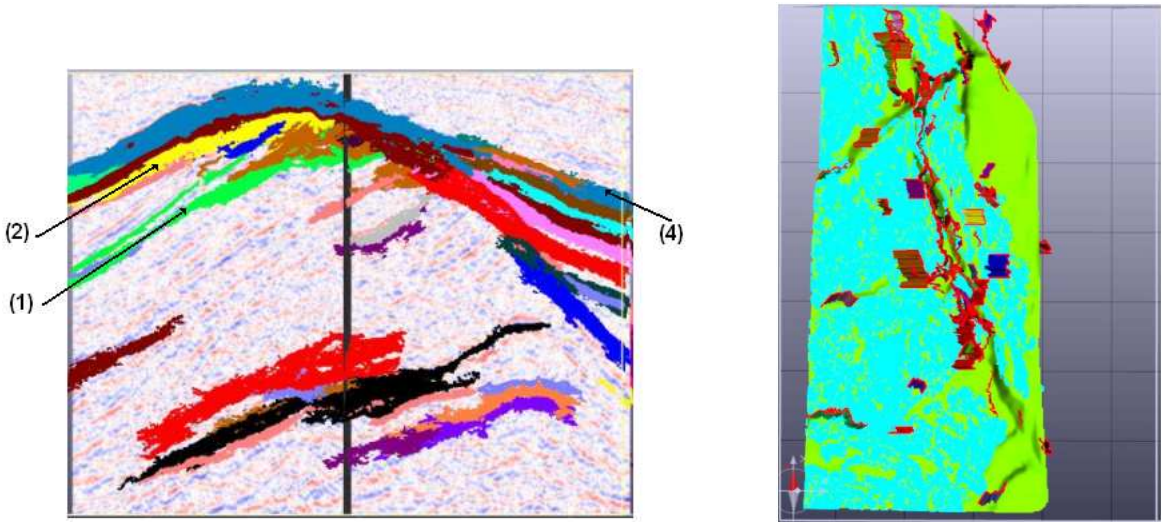
The major difference between this semi-automated method and the "manual" method described before is that semi-automated method *automatically stores in the course of the procedure* all the information concerning:

- the name of the seismic image from which the surface was interpreted;
- the parameters selected by the user for the interpretation procedure;
- the the identification, the calculated and the interpreted properties of the geological objects (such as mean amplitude and thickness);
- the relationships "is part of" and "is composed by" between reflectors and horizons;
- the age and topological relationships of each of these objects with all the others (such as relations "is upper than" and "is lower than");
- the date when the the seismic interpretation was carried out;
- name of the person that acted as interpreter;

All these information being stored as external metadata that is independent of the data itself. Applying the automatic method to the Alwyn prospect data, P.Verney identified after automatic reflector fusion and after correlation with well markers the following 6 main horizons:

- Top Dunlin (Figure 8.4a(1));
- Top Brent (Figure 8.4a(2));
- Top Turonian (Figure 8.4a(4));
- Top Ness 1;
- Top Etive.

The 3 main horizons (Top Dunlin, Top Brent and Top Turonian) figured on Figure 8.4a were object of a detailed comparison with those identified at IFP School. 101 faults were also identified for a seismic amplitude thereshold of 9000 (*cf.* Figure 8.4b).



(a) Interpreted horizons

(b) Interpreted faults

Figure 8.4: Semi-automated seismic interpretation of the Alwyn block (P.Verney)

1.4 Definition of the use case considered in the present chapter

For defining our use case, let us suppose that some geoscientist wants to *improve the structural models that were performed starting from the Alwyn prospect data*. For this, the user must first evaluate the previously obtained results. For operating this evaluation, the geoscientist will be willing to get for each previous interpretation answers to some questions such as those listed hereunder:

- $\mathbf{Q}_1$ - Which horizons where identified that are younger than Lias, and older than Cretaceous ?
- $\mathbf{Q}_2$ - From which seismic image comes the horizon BCU ?
- $\mathbf{Q}_3$ - With which amplitude threshold the horizons Top Etive and Top Brent were detected ?
- $\mathbf{Q}_4$ - Which reflectors are associated with the Top Etive horizon?
- $\mathbf{Q}_5$ - Which wells made possible the association of horizons extracted from the seismic image to the marker Top Etive?
- $\mathbf{Q}_6$ - When was made the interpretation which allowed the identification of the horizon Top Brent?
- $\mathbf{Q}_7$ - Who carried out this interpretation?
- $\mathbf{Q}_8$ - Among all horizons identified during seismic interpretation, specify those which are younger (or older) than the Top Dunlin horizon.
- $\mathbf{Q}_9$ - Which faults were identified with an amplitude threshold 10000 ?
- $\mathbf{Q}_{10}$ - Which faults were identified when using at the same time amplitude thresholds of 10000 and 9000 ?
- $\mathbf{Q}_{11}$ - Which faults interrupt the horizon Top Ness 1?

Q$_{12}$ - Retrieve the data files that represent the horizons that have been interpreted to be younger than the Top Dunlin horizon, and the author of this interpretation.

Q$_{13}$ - Which reflectors were interpreted by Philippe Verney as having an age younger than Lias and older than Cretaceous ?

In the case of the "manual" interpretation performed at ENSPM, no information is available that could allow to answer these questions without asking them directly to the operators of the interpretation themselves. In contrast, in the case of Verney's interpretation, all necessary information was stored to provide these answers in an automatic way by using the methodology that has been exposed in the present work.

We will describe in the next section how the user's interpretations about the geological objects were actually represented as *semantic annotations over the data*. We also show how the user's questions were transformed in semantic queries over the annotations.

2 Annotation-based approach applied to the use case of seismic interpretation

The objective of this implementation was to transform data and interpretation generated by geoscientists into formalized representations that are able to be stored in an ontology-based database (OBDB) and queried afterwards.

An engine for *generating annotation* over data artifacts was implemented. This **Engineering Annotation Generation engine** can be related with the **Data Interpretation module** in different ways: it can be *internally coupled* to the data interpretation engine, meaning that it knows the format in which data is organized, or it can be *external* to the process.

In the use case presented in this chapter, the Engineering Annotation Generation engine receives as input the data and experts interpretation issued from the seismic interpretation task. The engine "knows" the format in which the expert interpretation is represented (as instances of an OWL ontology[51]), and it "knows" the data structure of the data issued from seismics (e.g. PLO files). The Engineering Annotation Generation engine transforms the data files into instances of the *Engineering Metamodel* (*cf.* Chapter 4) and the interpretation into instances of the *Annotation Metamodel*, as we will present in Section 2.1.

According to the definition of the three types of annotation process that we gave in section Section 1.2.1 of Chapter 4, we classified various tasks related to the reservoir modeling workflow tasks according to the way in which annotation is generated.

- We consider that the task of seismic interpretation can be object of an **white box annotation**. The seismic interpretation module identifies the objects by automatic interpretation and generates metadata that is attached to the data files. The annotation generation engine is coupled to the

[51]The seismic interpretation module stores its results in OWL files. However, for each interpretation cycle, it generates a different OWL file. The consequence is that it is not possible to correlate objects interpreted in different cycles. That is why the output of the seismic interpretation module need to pass through the Annotation Generation engine, so as to the annotations to be made explicit and integrated in one only repository.

seismic interpretation module and is able to operate the format of the generated files. This is the case described in this chapter.

- The task of Structural Modeling Task currently realized in industry with modelers such as the Gocad®application (Earth Decisions) is a typical task that can be object of **black box annotation**. The GOCAD suite is a commercial software, whose data model is not open. Geologists' annotation activates the system that creates instances of a hand-made data model, which are linked to the ontology. These data model instances have no association with the original data file used in the application.
- Finally, a practical example of **intrusive annotation** is the utilization of well logs for various tasks. The set of files used to represent well information are described using an XML-based standard called WITSML (Wellsite Information Transfer Standard Markup Language) (Energistics, 2007). This standard defines XML tags that are specific to the well domain. A well data file created using WITSML is able to be processed by a *parser* its information being thus transformed in ontological instances in the knowledge base.[52] The link to the original data source is maintained by the annotation.

2.1 Formalization of seismic data and interpretation

We applied the approach proposed in Chapter 4 in order to make explicit experts' knowledge about seismic interpretation. The explicit representation of data and knowledge about the seismic interpretation task have been persisted as scripts in the OntoQL language, so as to be stored afterwards in OntoDB. All the scripts presented hereafter were automatically generated by the Engineering Annotation Generation engine.

2.1.1 Creating Engineering Models

The first step consists in representing all data formats used in the seismic interpretation task *as instances of the Engineering Metamodel*. The available metadata of seismic data models was reduced to the minimum necessary structure that allows a uniform description of those models (file name, identificator, main attributes). The considered use case uses files of formats: LAS (for well files), DAT (for markers files), SEG-Y (for seismic block files) and PLO (for surfaces files). These formats were represented as Engineering Models, as illustrated in Section 1.1.5 of Chapter 4.

The following OntoQL statements encode the seismic metadata in OntoDB using the Engineering Metamodel primitives added in OntoDB (*cf.* Section 2 of Chapter 4). They create *#DataClass* entities for representing the seismic data types.

```
CREATE #DataClass DataFile (
PROPERTIES (URI STRING, filename STRING, filepath STRING))
;
```

[52] In computer science, *parsing* is the process of analyzing a text to determine its grammatical structure with respect to a given formal grammar. In this case, the grammar considered is the WITSML syntax, which declares precisely which are the possible elements to be described in a WITSML document. The parser might convert WITSML elements into an OWL ontology.

```
CREATE #DataClass LASFile UNDER DataFile;
;
CREATE #DataClass PLOFile UNDER DataFile;
;
CREATE #DataClass DATFile UNDER DataFile;
;
CREATE #DataClass SEGYFile UNDER DataFile (
PROPERTIES (xmlFilename STRING))
```

A class named *DataFile* was created that defines the attributes URI, filename and filepath, which represent respectively the file unique identification, the file name and the relative path in which the file is stored. The classes *SEGYFile*, *LASFile*, *DATFile* and *PLOFile* are subclasses of *DataFile* and inherit the main attributes URI, filename and filepath.

2.1.2 Creating Instances of Engineering Models

The actual data files produced as the output of the Seismic Interpretation module are represented as instances of the seismic data models.

The following OntoQL statements encode the creation of some files as instances of their file type. For example, the well file whose file name is "A3.las" is created as an instance of *LASFile*, while the horizon gap file "Top_Dunlin.plo" is created as an instance of *PLOFile*.

```
INSERT INTO LASFile(URI, filepath, filename)
VALUES ('http://www.ifp.fr/SeismicInterp#A3', 'seismic/', 'A3.las')
;
INSERT INTO PLOFile(URI, filepath, filename)
VALUES ('http://www.ifp.fr/SeismicInterp#Top_Dunlin', 'seismic/9000/',
'Top_Dunlin.plo')
```

Figure 8.5 illustrates the resulting engineering models and instances in the metamodel structure.

2.1.3 Creating Instances of Ontologies

During the seismic interpretation task, the user identifies objects which correspond to geological interpretations of the data files produced by the Seismic Interpretation module. The objects identified and their properties are the most important result after the data files themselves. The geological objects are instances of concepts of the geosciences ontologies.

The Seismic Interpretation and Well ontologies were originally developed using the OWL language. As a direct consequence, the instances of the ontology concepts are represented within an OWL file. As it was explained in Section 3 of Chapter 7, the OWL ontologies and their instances can be mapped to OntoQL scripts by means of the OWL-OntoQL mapping engine. The following OntoQL statements encode the creation in OntoDB of the objects interpreted from the Alwyn seismic block.

```
INSERT INTO Well(URI)
VALUES ('http://www.ifp.fr/SeismicInterp#A3')
```

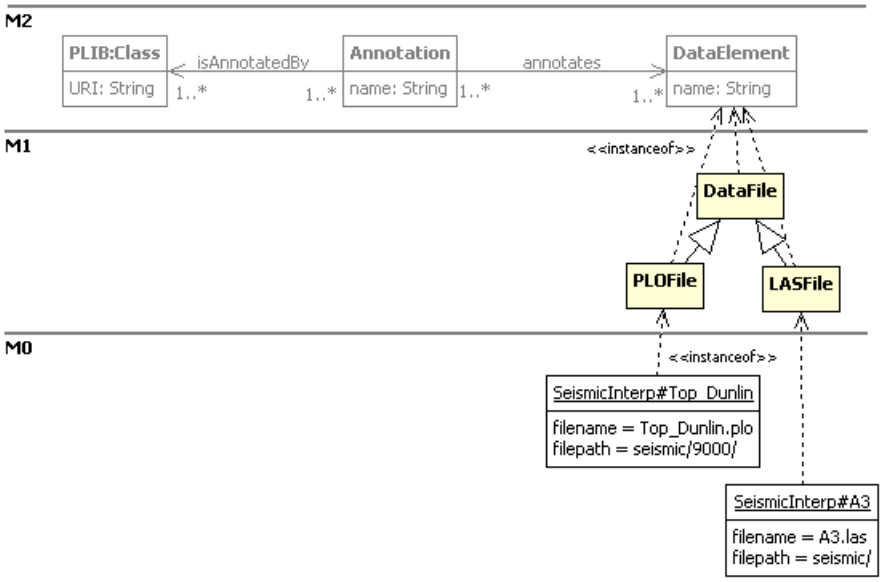


Figure 8.5: Engineering models and instances for Seismic Interpretation

```
;
UPDATE Well SET name = 'A3',
  isPartOfBlock = (SELECT SeismicBlock.oid
    FROM SeismicBlock WHERE SeismicBlock.URI =
      'http://www.ifp.fr/SeismicInterp#AlwynReservoir')
WHERE URI = 'http://www.ifp.fr/SeismicInterp#A3'
;
INSERT INTO Horizon(URI)
VALUES ('http://www.ifp.fr/SeismicInterp#Top_Dunlin_9000')
;
UPDATE Horizon SET hasMeanAmplitude = '8624.16',
 isUpperThan = ARRAY(SELECT Horizon.oid FROM Horizon
     WHERE Horizon.URI = 'http://www.ifp.fr/SeismicInterp#horizon_43_9000'
       OR Horizon.URI = 'http://www.ifp.fr/SeismicInterp#horizon_110_9000'
       OR Horizon.URI = 'http://www.ifp.fr/SeismicInterp#Top_Etive_9000'),
 hasLabel = 13, name = 'Top_Dunlin', isPartOfBlock = (SELECT SeismicBlock.oid
    FROM SeismicBlock WHERE SeismicBlock.URI =
      'http://www.ifp.fr/SeismicInterp#AlwynReservoir'),
 isLowerThan = ARRAY(SELECT Horizon.oid FROM Horizon
    WHERE Horizon.URI = 'http://www.ifp.fr/SeismicInterp#BCU_9000'
     OR Horizon.URI = 'http://www.ifp.fr/SeismicInterp#Top_Ness1_9000'),
 hasMeanThickness = '6'
WHERE URI = 'http://www.ifp.fr/SeismicInterp#Top_Dunlin_9000'
```

Figure 8.6 illustrates the resulting Seismic ontology concepts and instances.

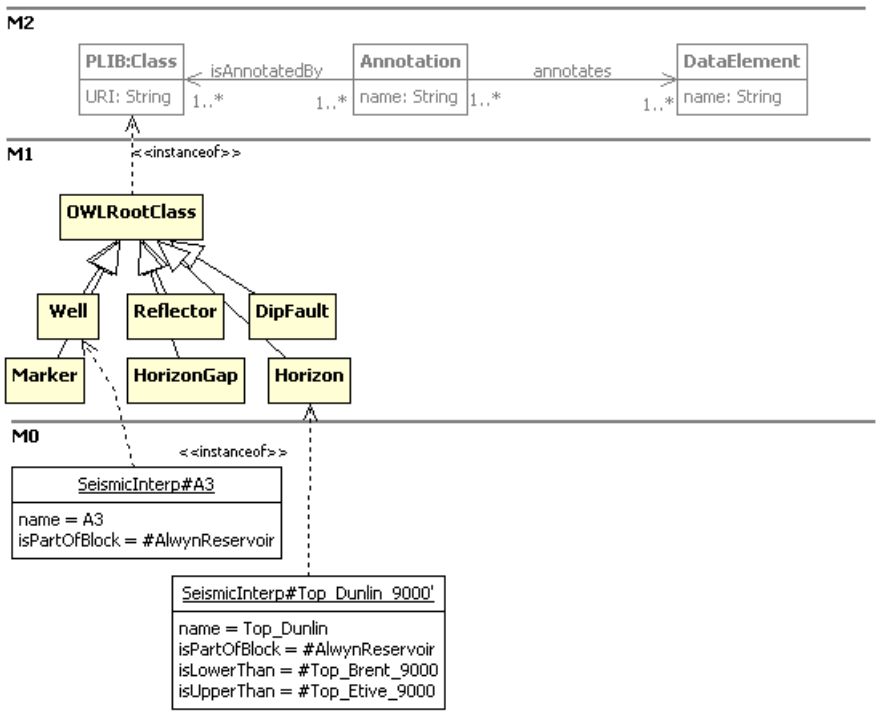


Figure 8.6: Seismic Interpretation ontology concepts and instances

2.1.4 Creating the Annotations of Engineering Models

The **annotation generation engine** receives the files and the metadata output by the seismic interpretation task and generates a specific annotation model. The annotation model defines all attributes and properties that are important for the semantic annotation on the Seismic Interpretation task.

The following OntoQL statements encode the seismic annotation model in OntoDB using the Annotation Metamodel primitives added in OntoDB (*cf.* Section 2 of Chapter 4). The seismic annotation is created as an *#Annotation* entity, and defines the properties *author*, *date*, *amplitudeThreshold*, which are specific for the seismic interpretation task; and the properties *name*, *isAnnotatedBy* and *annotates*, which are common for all annotation models. .

```
CREATE #Annotation SeismicAnnotation (
PROPERTIES (name STRING, author STRING, date STRING, amplitudeThreshold INT,
  isAnnotatedBy REF(OWLRootClass), annotates REF(DataFile)))
```

2.1.5 Creating the Annotation Instances

The annotation instances are the *link* between the data files and the interpreted geological objects. This link needs to be stored anyhow in order to be queried later on by the user that created it or by other users.

The seismic interpretation module outputs the interpretation links together with the ontology instances.

The **annotation generation engine** receives the interpretation information, organizes it according to the structure of Seismic Annotation model, and finally translates it into an OntoQL script. The following OntoQL statements encode the creation in OntoDB of the annotation links between data produced from and geological objects interpreted from the Alwyn seismic block.

```
INSERT INTO SeismicAnnotation(name, annotates, isAnnotatedBy,
  author, date, amplitudeThreshold)
VALUES ('seismicAnnotation3656',
  (SELECT PLOFile.oid FROM PLOFile
  WHERE PLOFile.URI = 'http://www.ifp.fr/SeismicInterp#Top_Dunlin'),
  (SELECT Horizon.oid FROM Horizon
  WHERE Horizon.URI = 'http://www.ifp.fr/SeismicInterp#Top_Dunlin_9000'),
  'Michel Perrin', 'Wed Oct 21 20:17:52 2009', 9000)
;
INSERT INTO SeismicAnnotation(name, annotates, isAnnotatedBy)
VALUES ('seismicAnnotation2578',
  (SELECT LASFile.oid FROM LASFile
  WHERE LASFile.URI = 'http://www.ifp.fr/SeismicInterp#A3'),
  (SELECT Well.oid FROM Well WHERE Well.URI = 'http://www.ifp.fr/SeismicInterp#A3'))
```

Figure 8.7 illustrates the final structure of engineering models, ontologies and annotations resulted from the seismic interpretation use case.

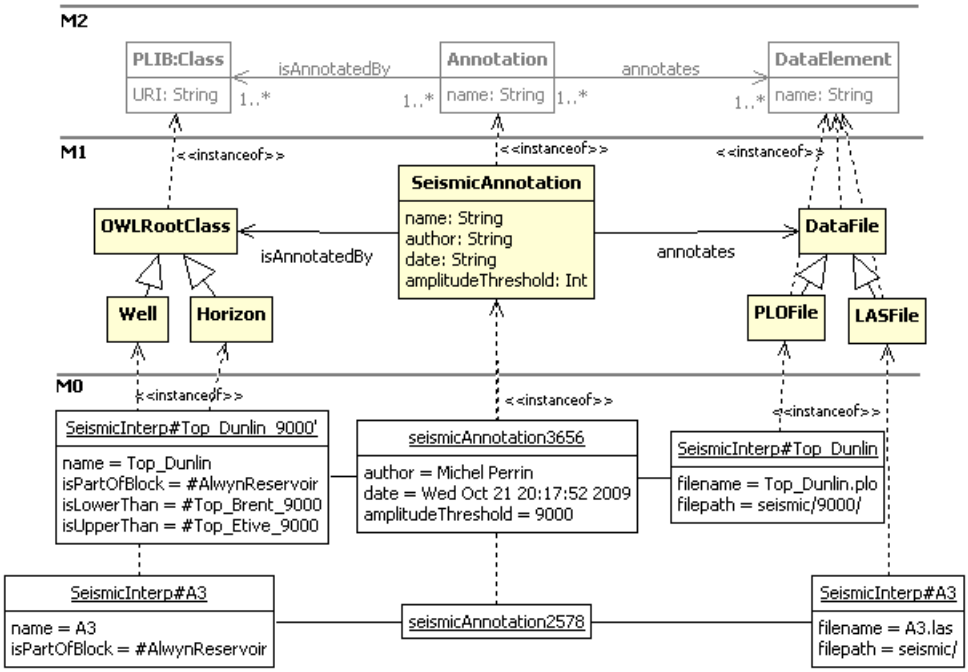


Figure 8.7: Seismic Interpretation use case: engineering models, ontologies and annotations

The generated scripts were executed on the OntoDB database with the support of the OntoQLPlus interface. The data issued from the seismics and the expert interpretation were stored in the same database

where we find the local and global ontologies.

The use case is thus implemented in the three levels of OntoDB: the *meta-schema* (M2), in which the Engineering and Annotation Metamodels are represented (*cf.* Section 2 of Chapter 4); the *model part* (M1), in which we have represented the geosciences ontologies (*cf.* Section 3 of Chapter 7), the data types (*cf.* Section 2.1.1), and the typed-annotations (*cf.* Section 2.1.4); and finally the *instances part* (M0), in which we stored data files representation annotated with ontology instances (*cf.* Section 2.1.2 and Section 2.1.5).

2.2 Exploitation of the formalized use case of seismic interpretation

We are now able to give answers to user's questions formulated in the use case. We present next the OntoQL queries corresponding to the user questions described in Section 1.4, and we show the values resulting from the execution of these queries over the OBDB, where we stored the geoscience ontologies and the seismic interpretation use case.

We will notice in this section that most of queries formulated for answering the user questions need to make use of multiple *joins* between various concepts.[53] The reason of so many jointures is that we must combine information issued from ontologies, annotations and engineering models in order to return a significant answer. We are aware that these kinds of queries are rather difficult for domains experts to write by their own. Not only they need to understand the syntax of the query language (either OntoQL or SPARQL, or yet another) in order to write input commands using correct expressions, but also they must identify the fields to be joined to each concept.

For this reason, the **PLIB Editor** was used for the design of user queries. The PLIB Editor is a graphical application implemented by Dehainsala (2007) over the ontology-based database (OBDB) management system. Among several functionalities over the OBDB, PLIB Editor proposes an *QBE interface*.[54] The QBE interface of PLIB hides the complexity of the OntoQL language syntax by allowing the user to choose the concepts and properties that he/she wants to search. A simple example is shown on Figure 8.8.

In part (1) of Figure 8.8 (the navigation tree), the user can choose the concept to be queried by simple selection. The concept chosen is *Marker*. In part (2), other concepts can be joined to the first entered concept by means of the pop-up menu. The concept chosen to be joined with *Marker* is *Well*. Below the properties names of each concept, it is possible to add query criteria. The user searches for instances of *Marker* which are part of some *Well* whose name is 'A2'. In part (3), the resulting OntoQL is designed. This query can be executed directly in the PLIB Editor, or can be copied to be executed in OntoQLPlus. The OntoQLPlus interface shows, together with the result table, the number of instances returned and the amount of time it took for them to be retrieved.

[53] A *join* operation is a means for combining fields from two tables by using values common to each. The query compares each row of both tables to find all pairs of rows which satisfy the join-predicate criteria (which is specified in the 'ON' clause).

[54] QBE stands for "Query By Example". A QBE interface provides a user-friendly interface to the design of queries by just filling in blanks or by selecting items.

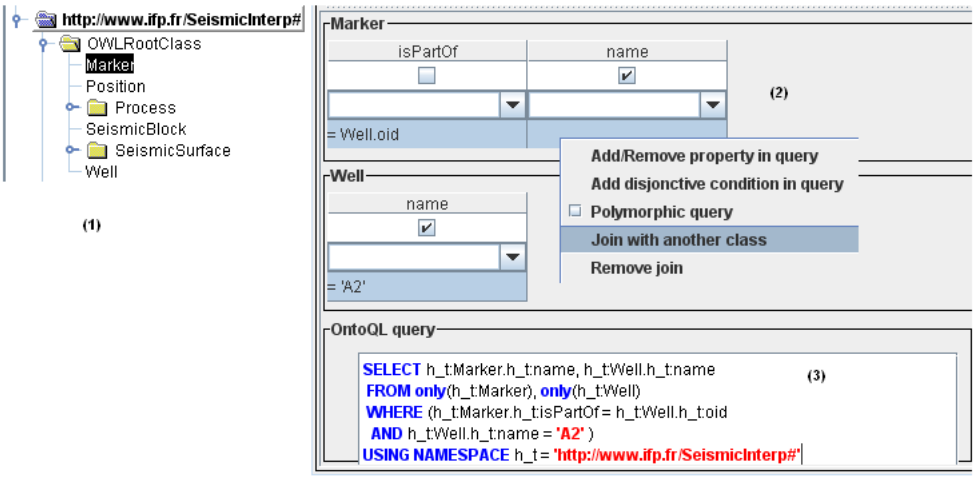


Figure 8.8: QBE interface for the design of OntoQL queries

2.2.1 Queries about geological ages

We show here an example of query that makes use of the geological dating relations. The user wants to retrieve geological horizons that can be interpreted as being younger than the Lias (Lower Jurassic) epoch and also older than the Cretaceous period.

2.2.2 Queries specific to the seismic interpretation objects

The objects considered in queries described in Table 8.2 to Table 8.10 are issued from the seismic interpretation. The user questions to which we answer in this section are entirely related to these objects, therefore, the "terms" (concepts and relations) employed for formulating the queries in OntoQL language is that from the local ontologies (Seismic and Well ontologies). So, all the horizons about which the questions are posed are *seismic horizons* (belonging to the Seismic Interpretation ontology), and not *stratigraphic horizons* (belonging to the Basic Geology ontology).

Some of the questions listed in the present section are formulated in the way in which users are likely to ask them, that is using pure geological vocabulary. However, for being answered by the system, these questions must be first reformulated as queries using a local ontology vocabulary. The reason for this is that the objects that will be queried are defined as instances of a given LO: the Seismic Interpretation ontology (or the Well ontology) and are thus pure seismic objects (or well objects). Accordingly in the corresponding queries, the term "Horizon" refers to the concept of *Seismic Horizon*, the term "Fault" to *Dip Fault* and relations such as *is younger/older than* are expressed as *is upper/lower than*.

2.2.3 Queries that integrate different local ontologies

In Chapter 5, we explained that it is the integration of different local ontologies that will allow the users (i.e. the geoscientists) to formulate questions in their *own* language (that is, the basic geology). We describe in the next sections how the local ontologies have been mapped to the global ontology that describes the geological concepts, and how this mapping allows the users to reformulate some of the previous queries in order to use a more significative vocabulary.

2.2.3.1 Creating correspondence between LO concepts and GO concepts using the *is-case-of* relation The last step for the validation of the use case if to allow users to formulate queries in their own language. For instance, geologists will be wanting to designate in their queries geological objects rather than seismic objects and geological properties (*isOlderThan*/*isYoungerThan*) rather than seismic properties (*isUpperThan*/*isLowerThan*) in the seismic image. All this supposes creating correspondence among the various domains related to the earth modeling workflow.

When developing ontologies for local domains of expertise, such as seismics and drilling, the main objective is allowing experts of these domains to make their knowledge explicit. One of the most important information that we want experts to describe is the relationships that exist between concepts in local earth science domains and concepts of the Basic Geology ontology. Establishing *subsumption relationships* between Local ontologies (LO) and the Global ontology (GO) is likely to enable the user to answer queries that cannot be addressed at present, because we cannot recover the relation among local objects identified in different phases of the geological modeling process (*cf.* Chapter 5).

A practical example is when the geologists refers to *seismic horizons* (which are objects identified by seismic interpretation) as being *geological horizons*. Despite the fact that they are almost homonymous, the two concepts represent different ideas and have different attributes. A seismic horizon has characteristics linked to the seismic processus, such as amplitude and thickness. A geological horizon has characteristics that are related to some geology, such as age and structure. The fact that the geoscientist may refer to some horizon interpreted from a seismic image as a geological horizon is an occurrence of a *subsumption* relation between a local and a global concept.

For aligning the concepts of LO and GO, we establish subsumption relationships, notably the *is-case-of* relation, i.e., LocalConcept *is-case-of* GlobalConcept. We depict in Figure 8.9 some subsumption relations established by domain experts, represented here as *is-case-of* relations.

The *is-case-of* relations established in this case are typically *a posteriori case-of* relations, since all classes involved in the relation had already been created. The correspondences between LOs and GO are thus created when all ontologies have already been stored in OntoDB. The *is-case-of* relations are manually defined by experts, directly in the form of OntoQL expressions.

In the following example, we show the creation of the concepts *geo:StratigraphicBoundary* (which is the "technical" name of geological horizons) and *geo:FaultBoundary*, from the GO, of the concepts *seismic:Horizon*, *seismic:Reflector*, *seismic:DipFault* and *seismic:HorizonGap* from the Seismic LO, and of the concept *well:Marker* from the Well LO.

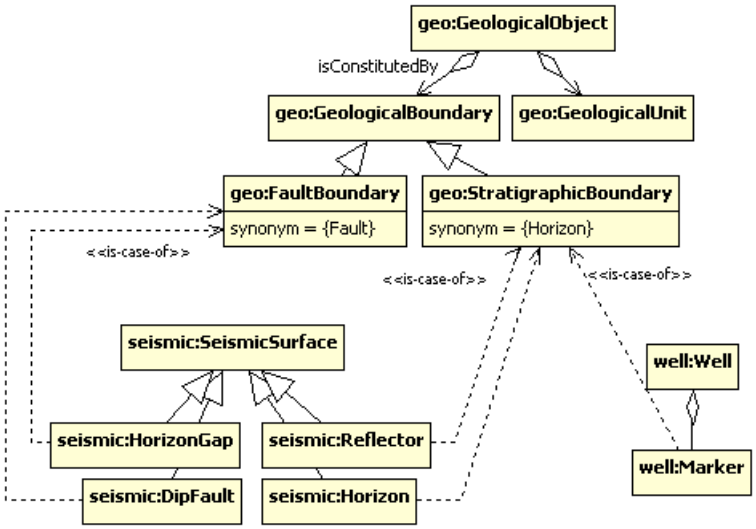


Figure 8.9: *is-case-of* relation between LO and GO concepts

```
CREATE #Class StratigraphicBoundary UNDER GeneticBoundary (
 DESCRIPTOR (#synonym = 'Horizon')
 PROPERTIES (URI String, isContemporaneousTo REF(GeoTemporalEntity) ARRAY,
    isOlderThan REF(GeoTemporalEntity) ARRAY, isYoungerThan REF(GeoTemporalEntity) ARRAY,
    hasStratigraphicAge REF(GeochronologicUnit), hasInstantAge REF(GeochronologicInstant),
    hasStructure REF(GeologicalStructure) ARRAY, pendage INT) )
;

CREATE #Class FaultBoundary UNDER MacroFractureBoundary (
 DESCRIPTOR (#synonym ='Fault')
 PROPERTIES (URI String, isContemporaneousTo REF(GeoTemporalEntity) ARRAY,
    isOlderThan REF(GeoTemporalEntity) ARRAY, hasStratigraphicAge REF(GeoTemporalEntity),
    isYoungerThan REF(GeoTemporalEntity) ARRAY, hasInstantAge REF(GeoTemporalEntity),
    hasStructure REF(GeologicalStructure) ARRAY, pendage int,
    interrupts REF(GeologicalBoundary) ARRAY, goesThrough REF(GeologicalBoundary) ARRAY,
    stopsOn REF(GeologicalBoundary)))
;

CREATE #Class Fault UNDER MacroFracture (
 DESCRIPTOR (#synonym ='Fault')
 PROPERTIES (URI String, isContemporaneousTo REF(GeoTemporalEntity) ARRAY,
    isOlderThan REF(GeoTemporalEntity) ARRAY, hasStratigraphicAge REF(GeoTemporalEntity),
    isYoungerThan REF(GeoTemporalEntity) ARRAY, resultsFrom REF(GeologicalEvent),
    isComposedOf REF(FaultBoundary) ARRAY, hasStructure REF(GeologicalStructure) ARRAY,
    depth INT, thickness INT)
;

CREATE #Class Horizon UNDER InterpretedSurface (
 DESCRIPTOR (#synonym ='SeismicHorizon')
```

```
 PROPERTIES (URI String, isUpperThan REF(Reflector) ARRAY,
    isComposedBy REF(Marker) ARRAY, isLowerThan REF(Reflector) ARRAY,
    hasMeanAmplitude STRING, hasMeanThickness INT, isPartOfBlock REF(SeismicBlock)))
;

CREATE #Class Reflector UNDER ComponentSurface (
 DESCRIPTOR (#synonym ='SeismicReflector')
 PROPERTIES (URI String, isUpperThan REF(Reflector) ARRAY,
    hasBeenMarkedBy REF(Marker) ARRAY, isLowerThan REF(Reflector) ARRAY,
    hasMeanAmplitude STRING, hasMeanThickness INT))
;

CREATE #Class DipFault UNDER SeismicFault (
 DESCRIPTOR (#synonym ='SeismicFault')
 PROPERTIES (URI String, isComposedBy REF(HorizonGap) ARRAY,
    isPartOfBlock REF(SeismicBlock)))
;

CREATE #Class HorizonGap UNDER ComponentSurface (
 PROPERTIES (URI String, disconnects REF(Horizon) ARRAY, isPartOfBlock REF(SeismicBlock)))
;

CREATE #Class Marker (
 PROPERTIES (URI String, isPartOf REF(Well), isLocatedOn REF(Position) ))
```

We must then represent the following alignment between the concepts.

- The concepts *seismic:Horizon, seismic:Reflector* and *well:Marker* are aposteriori case of the concept *geo:StratigraphicBoundary*.
- The concepts *seismic:DipFault* and *seismic:HorizonGap* are aposteriori case of the concept *geo:FaultBoundary*.

The respective OntoQL expressions are shown next:

```
 CREATE #AposterioriCaseOf Horizon CASEOF StratigraphicBoundary
   WITH (Horizon.name MAP StratigraphicBoundary.name,
         Horizon.URI MAP StratigraphicBoundary.URI,
         Horizon.isLowerThan MAP StratigraphicBoundary.isOlderThan,
         Horizon.isUpperThan MAP StratigraphicBoundary.isYoungerThan)
 ;

 CREATE #AposterioriCaseOf Reflector CASEOF StratigraphicBoundary
   WITH (Reflector.name MAP StratigraphicBoundary.name,
         Reflector.URI MAP StratigraphicBoundary.URI,
         Reflector.isLowerThan MAP StratigraphicBoundary.isOlderThan,
         Reflector.isUpperThan MAP StratigraphicBoundary.isYoungerThan)
 ;

 CREATE #AposterioriCaseOf Marker CASEOF StratigraphicBoundary
```

```
  WITH (Marker.URI MAP StratigraphicBoundary.URI,
        Marker.name MAP StratigraphicBoundary.name)
;

CREATE #AposterioriCaseOf DipFault CASEOF Fault
  WITH (DipFault.name MAP Fault.name,
        DipFault.URI MAP Fault.URI,
        DipFault.isComposedOf MAP Fault.isComposedOf)
;

CREATE #AposterioriCaseOf HorizonGap CASEOF FaultBoundary
  WITH (HorizonGap.name MAP FaultBoundary.name,
        HorizonGap.URI MAP FaultBoundary.URI,
        HorizonGap.disconnects MAP FaultBoundary.interrupts)
```

The *URI* and *name* properties are mapped so that the user can retrieve them within an *a posteriori case-of* query.

The properties *isLowerThan* and *isUpperThan* of the concepts *seismic:Horizon* and *seismic:Reflector* are respectively mapped to the properties *isOlderThan* and *isYoungerThan* of the concept *geo:StratigraphicBoundary*. The reason is that these pair of properties express the same idea: the *order of appearance* of the objects. But they are differently expressed, since in seismics there does not exist the notion of time relation between objects. This way, the user is able to retrieve one part of the topology of seismic objects, by means of *a posteriori case-of* queries.

Finally, the property *disconnects* of the concept *seismic:HorizonGap* is mapped to the property *interrupts* of the concept *geo:FaultBoundary*. Also, the property *isComposedOf* of the concept *seismic:DipFault* is mapped to the property *isComposedOf* of the concept *geo:Fault*.

2.2.3.2 Querying ontologies using the *is-case-of* operator By means of the *is-case-of* relation, instances of the concepts *seismic:Horizon*, *seismic:Reflector* and *well:Marker* are *considered to be instances also* of the concept *StratigraphicBoundary*. As a result, in the case when the user wants to retrieve *cases of StratigraphicBoundary* objects from all the specific domains of the workflow, he/she just need to search for instances of the concept *StratigraphicBoundary*. We show in Table 8.11 and Table 8.12 some queries that are interesting for the user to be performed over the subsumption hierarchies of the implemented use case. The first take back the query described in Table 8.8 and show that it can be performed using the vocabulary of the global ontology. The second retrieves faults and horizons from the seismics using relations and concepts of the global ontology.

The ontology integration approach (which maps ontologies using subsumption relation) can also be combined with the *semantic annotation of engineering models* approach (*cf.* Section 1.2 of Chapter 4). This is possible as we illustrate in the examples of Table 8.13 and Table 8.14. Those queries take back the questions answered in Table 8.1 and in Table 8.8 and show that it is possible to retrieve the data files that are associated to the interpreted objects, while keeping to use the global vocabulary.

In order to appropriately answer query Q_{13}, a "shortcut strategy" must be used. As indicated by the *a.isAnnotatedBy.name = S.name* line, a comparison was made between the name of the global object (*StratigraphicBoundary*) and the name of the local object (that annotated by Philippe Verney). The query retrieved as result only local objects whose name was the same as the global object found (those that are younger than Lias and older than Cretaceous).

The reason for this name comparison is that the instances that share the same name in the two different ontologies are, in reality *the same instance*. This is due to a subtle misunderstanding between experts and knowledge engineers concerning the process of seismic interpretation. In a first moment, the output of seismic interpretation was believed to be objects that only were instances of Seismic ontology concepts. The Seismic ontology being a local ontology (LO), those objects had no direct relation with objects from the Basic Geology ontology, i.e. with the global ontology (GO).

However, the detailed analysis of the query Q_{13} by the expert brought to light the following issue: the objects output by seismic interpretation have their *names* given by an object already existing in the Global ontology. The Top Brent object, for example, is an actual stratigraphic boundary that gave the name to the corresponding object issued from the seismics. They correspond to the same *real* object. Since this requirement had not been identified from the beginning, a structure for instance mapping was not defined.

The *implicit* mapping defined in *query time*, i.e. the name comparison, is effective while the instances have the exact same name. However, since this will not be always the case, an explicit *instance mapping* would be necessary in order to establish that the two objects are equivalent. This issue will be discussed in details in the Future Works section.

2.3 Summary of results

The above given execution times related to the queries presented in this chapter must be appreciated considering that these queries were processed on a computer equipped with 2 Gbytes of RAM memory and with an Intel Core 2 Duo processor with 2 GHz under the Windows operational system.

The OntoQLPlus interface was built under the Java Development Kit 6. The OntoDB OBDB was developed on top of the relational database system PostgreSQL, whose version used in this work is PostgreSQL Database Server 8.2.

After the implementation of the complete use case, the database logical schema contains 737 tables, among system tables (e.g. the table that represent the *a posteriori case-of* relations), meta-schema tables (e.g. the tables that represent the entities *Class*, *DataElement* and *Annotation*) and domain specific tables (e.g. the tables that represent the seismic ontology concepts). The database contains 21070 rows sizing up 22 Mbytes of data.

Due to the distinction made in the extended OntoDB about *data*, *annotation* and *ontology* elements, we are able to produce separate statistics for each of these elements. The queries described as follows address the *metamodel part of OntoDB*, enabling thus the user to retrieve information associated only to ontologies (*#Class*), only to annotations (*#Annotation*) or only to data elements (*#DataElement*).

```
SELECT DISTINCT COUNT(#oid) from #Class
```

```
SELECT DISTINCT COUNT(#oid) from #Annotation

SELECT DISTINCT COUNT(#oid) from #DataElement
```

The Table 8.15 sums up the number of elements inside the OBDB.

Regarding the Basic Geology ontology, the Geological Time ontologies and the Seismic and Well local ontologies, 151 ontology concepts and 4334 instances were created.

Considering finally the extended part of the OBDB, in relation to the Seismic interpretation task, we have described 5 data elements (instances of the *DataElement* entity) and 1 annotation element (instance of *Annotation* entity); and created 2344 instances of data elements and 3724 instances of the annotation element. A comparison of the OntoQL queries with SPARQL queries is presented in Appendix C.

3 Conclusion

One of the main contributions of this work is the formalization of a use case for the application of knowledge-based techniques which has a potential interest for industry. The set of activities that was chosen concerns earth modeling workflows and more specifically *seismic interpretation*. We presented all the data sets related to this activity, the way in which the expert's interpretation is generated, and a set of significant questions, among many possible ones, that users may ask about the domain data.

We described the steps that enable us to apply the approach that we proposed for *semantic-based annotation of engineering models* (*cf.* Chapter 4) to seismic interpretation models. We showed how to represent (i) seismic data as instances of the Engineering Metamodel; (ii) interpreted objects as instances of the Local Ontologies, and (iii) experts' interpretations themselves as instances of the Annotation Metamodel. As a result, all the seismic data and the associated interpretations were stored in one same repository.

As a consequence of this implementation, we showed how to perform the exploitation of the use case. We have demonstrated that the user is able to retrieve non-ambiguous information about the interpreted objects by means of the Annotation element. One part of the questions asked by the users can be answered by querying the annotation. This is notably the case when the user needs to differentiate one same object that was interpreted in different ways by several experts, or to retrieve all objects issued from one same interpretation cycle, or simply to be re-directed to the data files that are associated to the interpreted object.

We also described how the *A posteriori* approach of ontology integration (*cf.* Chapter 5) was applied to the use case. The local ontologies related to the seismic interpretation (Seismic and Well ontologies) were mapped to the Basic Geology ontology by means of *a posteriori case-of* relationship. This mapping was defined by the domain experts so that it has been possible establishing a subsumption relation (such as a subclass hierarchy) between concepts from pre-existing ontologies that were not related to one same domain of expertise. The *a posteriori case-of* relations allow to integrate the various geosciences domains using the Basic Geology as a common thread.

We then demonstrated the applicability of the subsumption relations established between local and global ontologies. The user formulates queries using the terms described in the global ontology (Geology) which are shared through the whole earth modeling activity. The *a posteriori case-of* relations define a navigable hierarchy between ontology concepts that were separately created. Therefore, the extended *SELECT* operator can navigate through all the hierarchies of subsumed classes (subclasses, *a priori case-of* classes and *a posteriori case-of* classes). Thanks to the subsumption links, the query automatically returns instances from the global and local ontologies. It then becomes possible to address multiple domains in one same query.

At the present stage, the main goal has been to check the usability of the methodology by domain users and the quality and relevance of the output. Consequently, we did not intend to optimize our approach in terms of speed of processing.

Discussion of the obtained results with domain experts showed that some work remains to be done for fully demonstrating the usability of our approach i.e. to make it easily operative by users in view of their particular goals.

In order to formulate the semantic-based queries presented in this chapter, the user is supposed to have a complete mastery of the query language (in this case OntoQL, but it could be the same for any query language, such as SPARQL). The PLIB ontology editor presented in Section 2.2 can be of help when formulating queries that demands ontology concepts to be joined. But the query designer still lacks a more ergonomic interface, and also a graphical tool for using *is-case-of* quantifiers, for example.

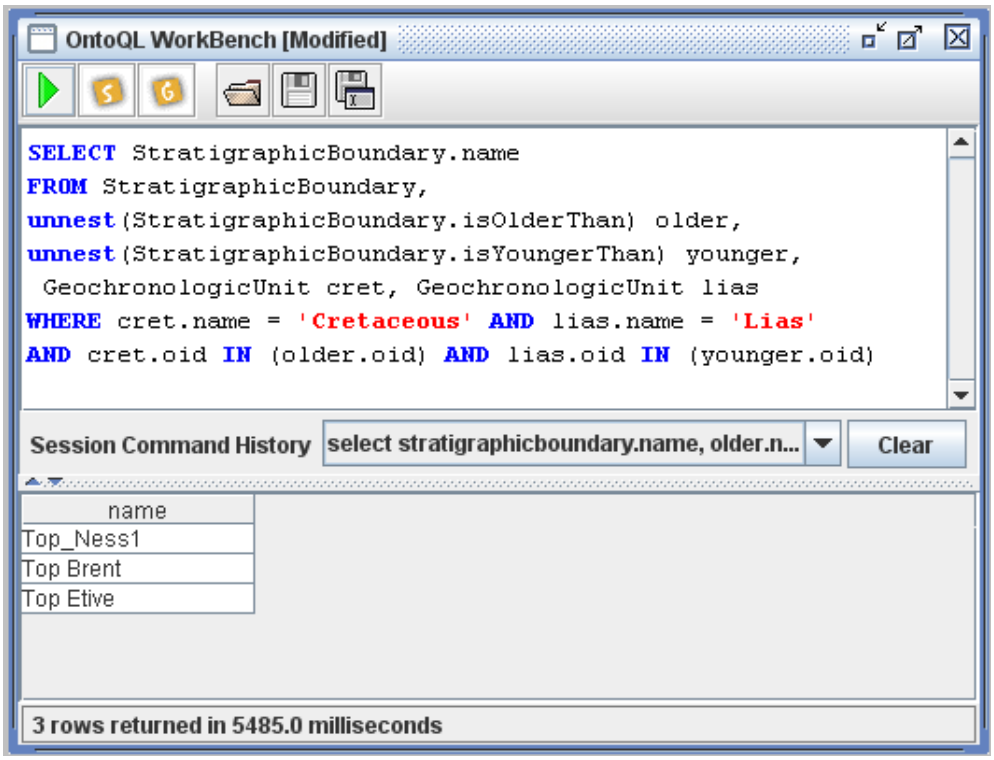
The results retrieved by the semantic-based queries are the answers that were expected by the experts who formulated the queries. This demonstrates that our software is able to automatically provide fully relevant answers to various types of questions. Before the implementation of the semantic-based framework, no information was available that could allow to answer these questions without asking them directly to the operators of the interpretation themselves. Thanks to the complementary approaches above implemented, we are able at present to formulate semantic queries that (i) integrate information from the various engineering models and (ii) retrieve information that were made explicit through annotations. The results obtained in this work strengthen our feeling that the considered methodology and that the implemented framework have an interest for engineering domains.

We also created SPARQL versions of the user questions, in order to perform a comparison to our approach. The conclusion is that, if we implement all ontologies, data and annotations as OWL objects in a OWL document, it is, indeed, possible to retrieve the same results for one part of the questions (those that query directly the concepts of the local ontologies). However, the queries that use the global vocabulary in order to retrieve local concepts from various ontologies are not possible to be defined in SPARQL, due to the lack of an *a posteriori case-of* operator. Another disadvantage is that in a RDF-based approach, we lack the metamodel level, which makes the methodology generic and thus applicable to other domains.

In relation to the results obtained from the OntoDB-OntoQL approach, the users would appreciate a means of creating correspondences between *instances*, that is, to be able to express that some instance of a *local object* is equivalent to some instance of a *global object*. The typical example is to create a correspondence between an instance of a geological horizon and an instance of a seismic horizon. This will be object of further work on the mapping relations, and will be described in the Future Works section.

Table 8.1: Q_1: Which horizons where identified that are younger than Lias, and older than Cretaceous ?

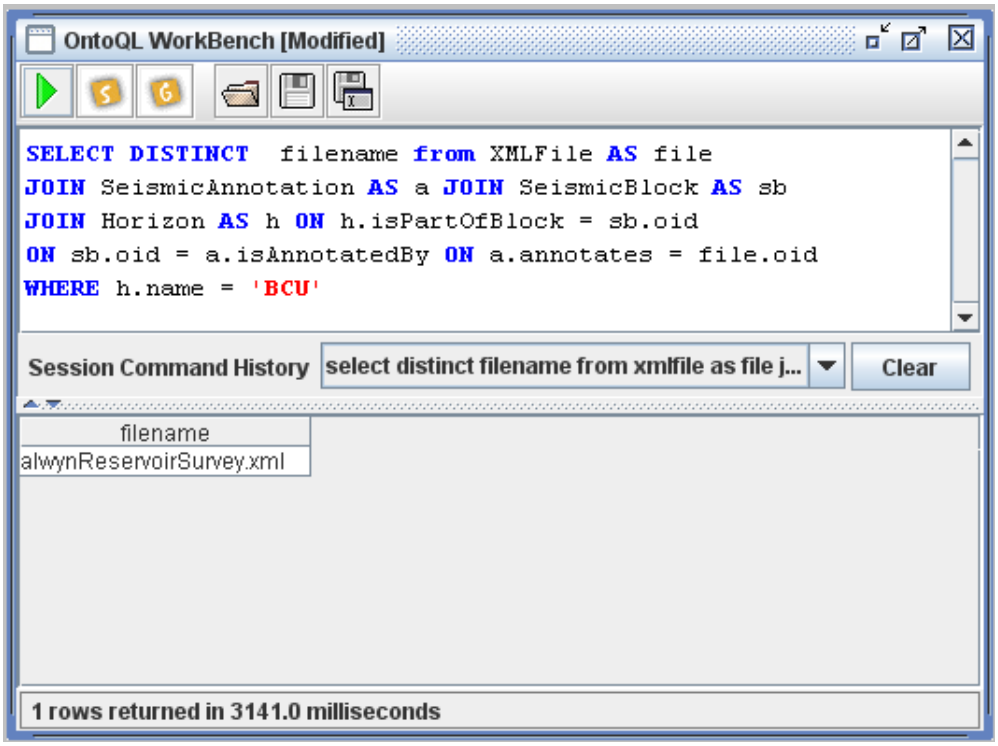
```
SELECT StratigraphicBoundary.name
FROM StratigraphicBoundary, unnest(StratigraphicBoundary.isOlderThan) older,
  unnest(StratigraphicBoundary.isYoungerThan) younger,
  GeochronologicUnit cret, GeochronologicUnit lias
WHERE cret.name = 'Cretaceous' AND lias.name = 'Lias'
AND cret.oid IN (older.oid) AND lias.oid IN (younger.oid)
```



Explanation. For the moment, the user is able to retrieve the geological horizons (*StratigraphicBoundary*) that have been *explicitly* dated as being younger than Lias and older than Cretaceous, that is, the horizons Top Brent, Top Etive and Top Ness1 resulted from the query.
The Geological Dating ontology and the inference rules described in Section 2.1.2.4 of Chapter 7 have been developed to allow the users to add temporal information about geological objects and also to infer derived temporal relationships from those that were explicited. However, the ontologies implemented in this work are not able to make use of inference rules. This is due to the fact that the OntoDB database do not dispose at present of a repository for storing inference rules and neither of an inference engine. The inclusion of inference rules in the OntoDB database is an issue that will be discussed in the Future Works section.

Table 8.2: Q_2: From which seismic image comes the horizon BCU ?

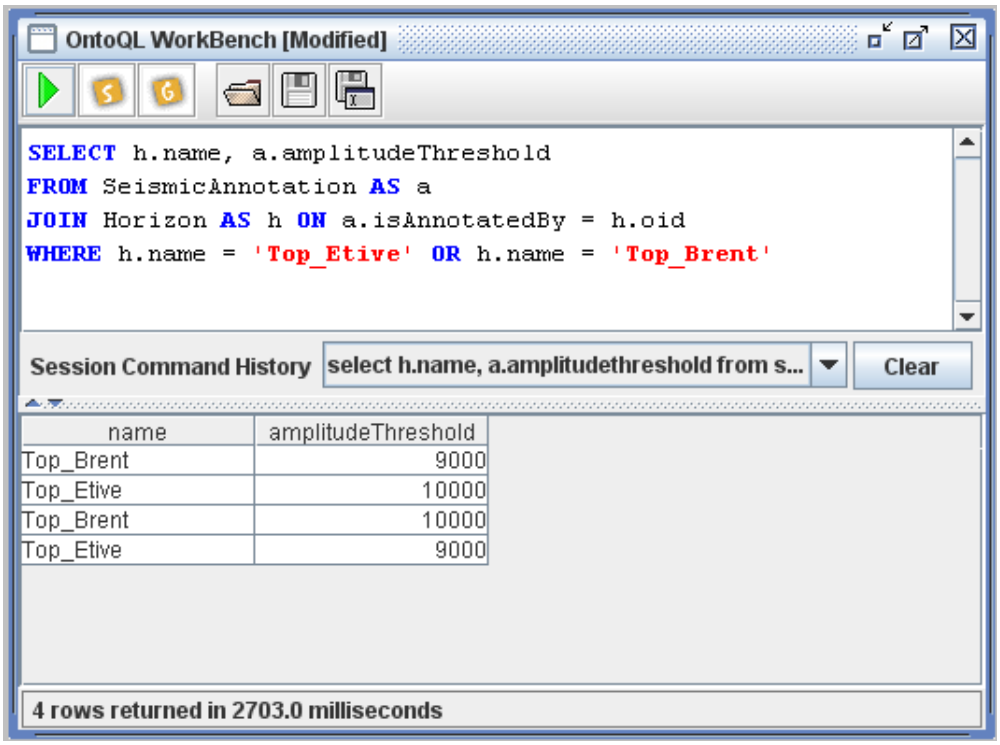
```
SELECT DISTINCT filename FROM DataFile AS file
JOIN SeismicAnnotation AS a JOIN SeismicBlock AS sb
JOIN Horizon AS h ON h.isPartOfBlock = sb.oid
ON sb.oid = a.isAnnotatedBy ON a.annotates = file.oid
WHERE h.name = 'BCU'
```



Explanation. This query selects the *filename* from an instance of some *DataFile*, which is *annotated* by some *SeismicBlock* containing the given *Horizon* 'BCU'. The result is the filename 'alwynReservoirSurvey.xml'

Table 8.3: Q_3: With which amplitude threshold the horizons Top Etive and Top Brent were detected ?

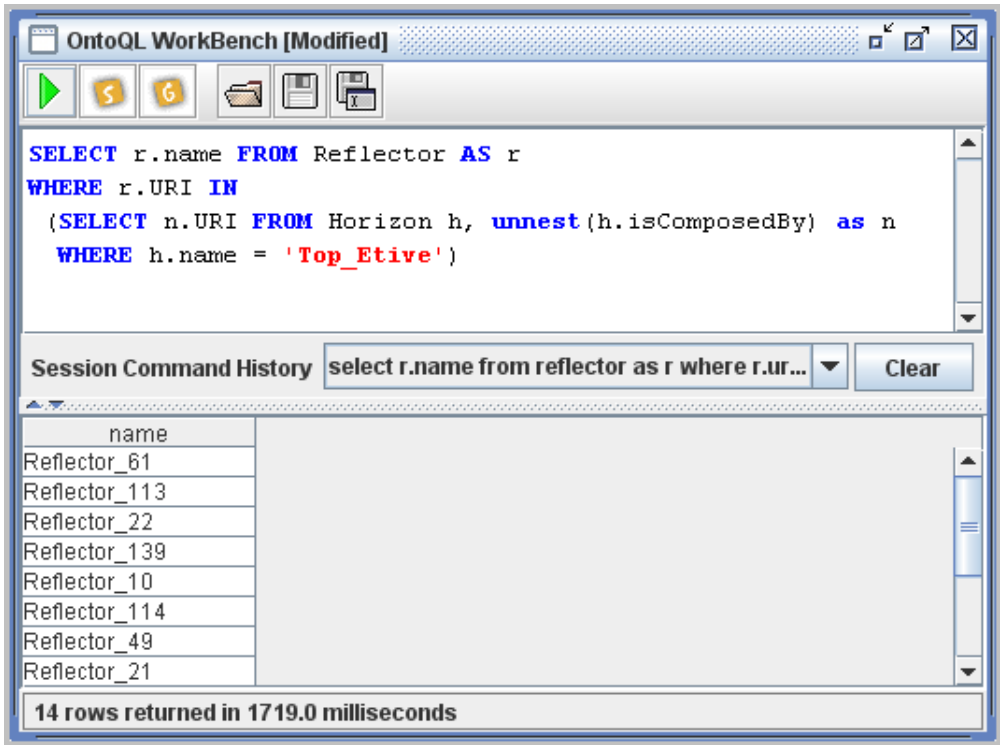
```
SELECT h.name, a.amplitudeThreshold FROM SeismicAnnotation AS a
JOIN Horizon AS h ON a.isAnnotatedBy = h.oid
WHERE h.name = 'Top_Etive' OR h.name = 'Top_Brent'
```



Explanation. This query selects the *amplitudeThreshold* property of the *SeismicAnnotation* instances that are related to the given *Horizon* instances 'Top_Etive' and 'Top_Brent'. The result is all the amplitude thresholds in which these horizons were detected, i.e. 9000 and 10000. It means that *two* process of detection were executed, one with each of the thresholds, which were both capable of detecting the given *Horizon* instances.

Table 8.4: Q_4: Which reflectors are associated with the Top Etive horizon ?

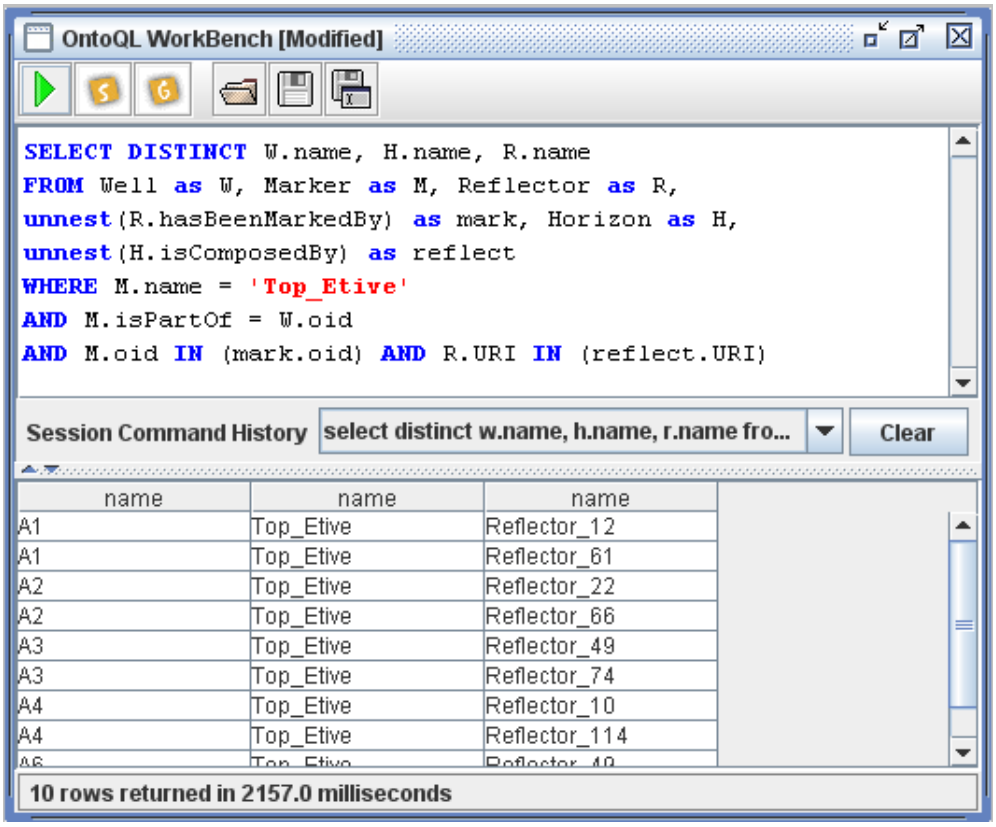
```
SELECT r.name FROM Reflector AS r WHERE r.URI IN
(SELECT n.URI FROM Horizon h, unnest(h.isComposedBy) as n}
WHERE h.name = 'Top_Etive')
```



Explanation. This query selects the *name* of the *Reflector* instances that compose the given *Horizon* instance 'Top_Etive'.

Table 8.5: Q_5: Which wells made possible the association of horizons extracted from the seismic image to the marker Top Etive ?

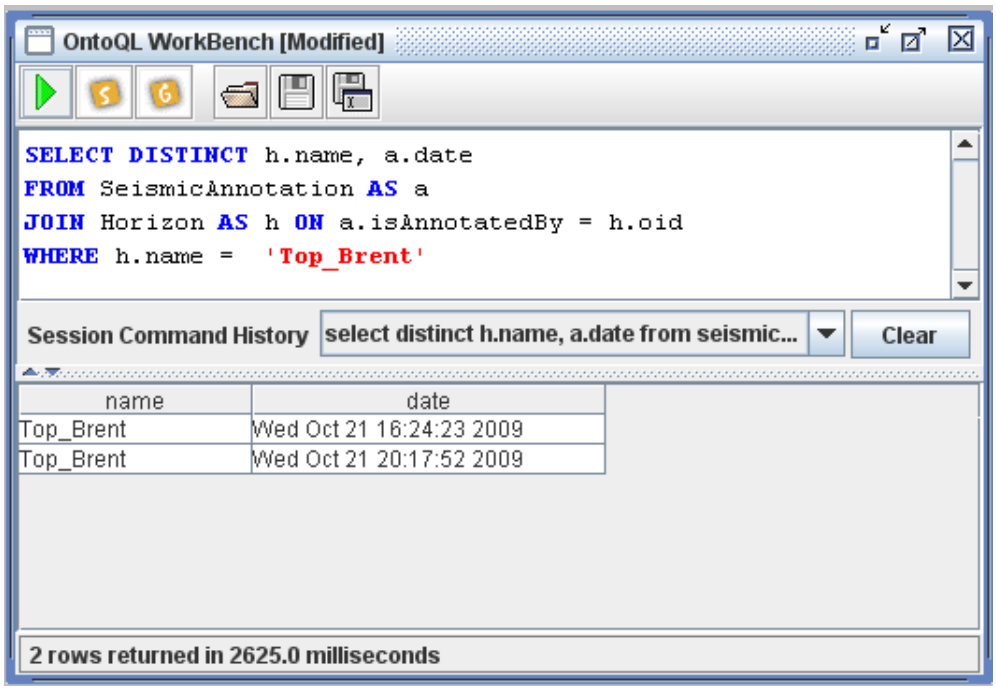
```
SELECT DISTINCT W.name, H.name, R.name
FROM Well as W, Marker as M, Reflector as R,
  unnest(R.hasBeenMarkedBy) as mark, Horizon as H,
  unnest(H.isComposedBy) as reflect
WHERE M.name = 'Top_Etive'
AND M.isPartOf = W.oid
AND M.oid IN (mark.oid) AND R.URI IN (reflect.URI)
```



Explanation. This query selects the *name* of the *Well* instances that are associated to the *Marker* instance 'Top_Etive', which have been used to mark *Reflector* instances that compose *Horizon* instances of the seismic image.

Table 8.6: Q_6: When was made the interpretation which allowed the identification of the horizon Top Brent ?

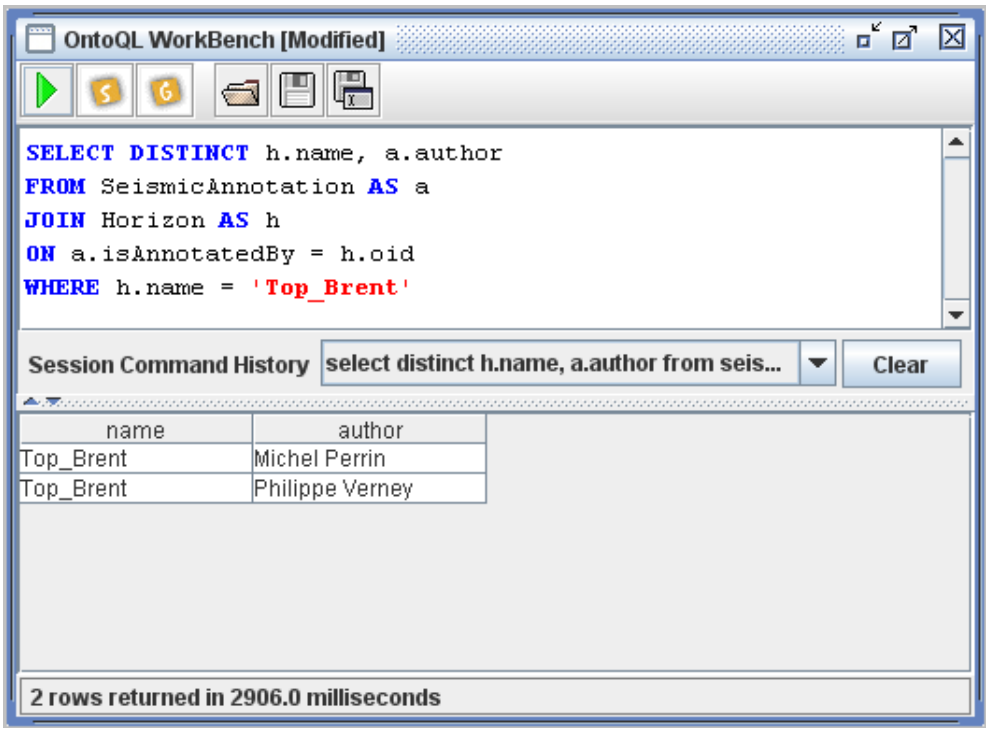
```
SELECT a.date FROM SeismicAnnotation AS a
JOIN Horizon AS h ON a.isAnnotatedBy = h.oid
WHERE h.name = 'Top_Brent'
```



Explanation. In this query, Top Brent is both a *seismic horizon* and a *geological horizon*. The annotation has been made during the seismic interpretation task, therefore, the user is searching here for the instance of the seismic concept.
This query selects the *date* property of the *SeismicAnnotation* objects that are related to the given *Horizon* instance 'Top_Brent'. The dates of the two different interpretations that have identified the given horizon are resulted.

Table 8.7: Q_7: Who carried out this interpretation ?

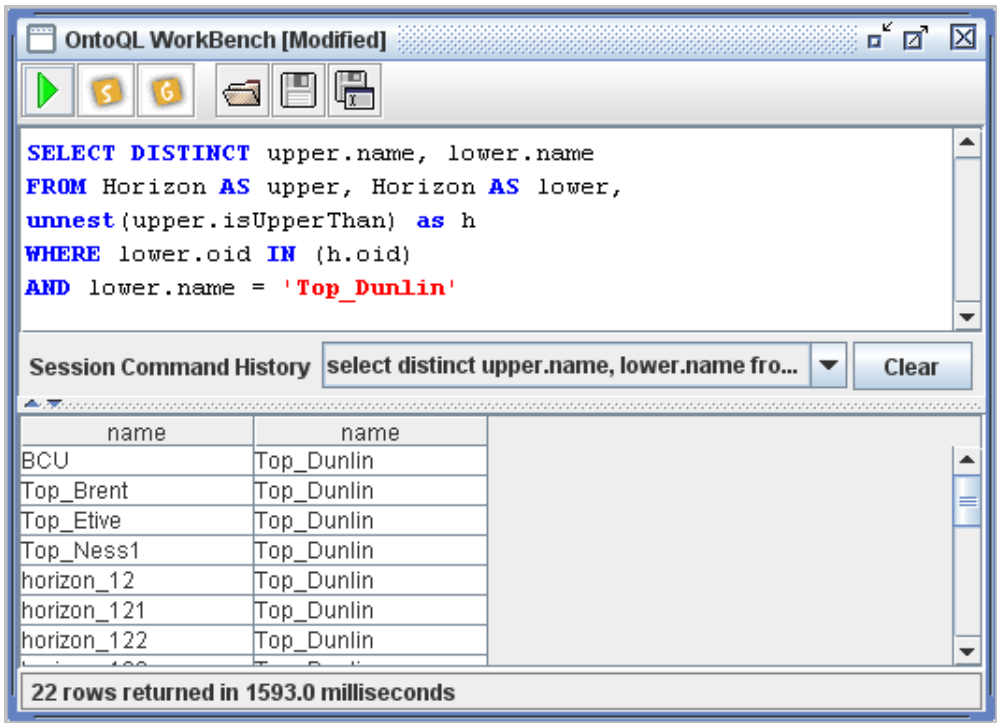
```
SELECT a.author FROM SeismicAnnotation AS a
JOIN Horizon AS h ON a.isAnnotatedBy = h.oid
WHERE h.name = 'Top_Brent'
```



Explanation. This query selects the *author* property of the *SeismicAnnotation* objects that are related to the given *Horizon* instance 'Top_Brent'. The names of the authors of the two different interpretations that have identified the given horizon are resulted.

Table 8.8: Q_8: Among all horizons identified during seismic interpretation, specify those which are younger (or older) than the Top Dunlin horizon.

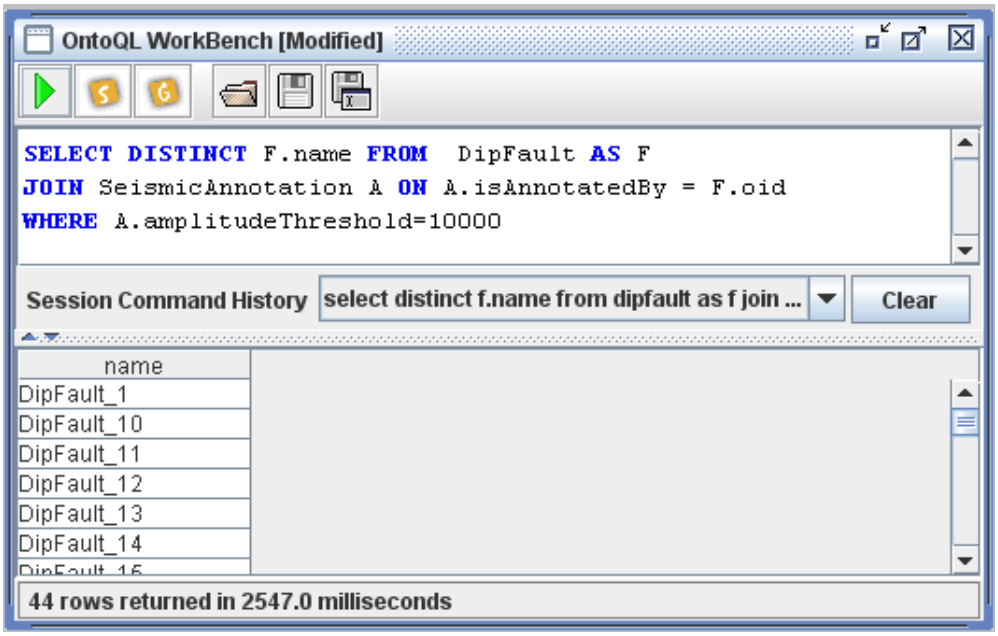
```
SELECT DISTINCT upper.name, lower.name
FROM Horizon AS upper, Horizon AS lower, unnest(upper.isUpperThan) as h
WHERE lower.oid IN (h.oid)
AND lower.name = 'Top_Dunlin'
```



Explanation. This query selects the *name* of all *Horizon* instances that are younger than the given *Horizon* instance 'Top_Dunlin'. It should be noticed that Top_Dunlin is both an instance of a *seismic horizon* and of an *geological horizon*. However, we emphasize here that the user is posing queries about the seismic interpretation objects, whose mutual relations are of type *isLowerThan*/*isUpperThan*, instead of *isOlderThan*/*isYoungerThan*. Since the user must employ the "vocabulary" defined by the seismic ontology relations, he/she must know that, in some cirscumstances, the seismic property that gives the idea of being *younger* than is the property *isUpperThan*, and inversely, the seismic property that gives the idea of being *older* than is the property *isLowerThan*. [55] In Section 2.2.3.1 we show how to execute this query using the exact vocabulary desired by the user, that is, using the relation *isYoungerThan* to retrieve the results.

Table 8.9: Q_9: Which faults were identified with an amplitude threshold 10000 ?

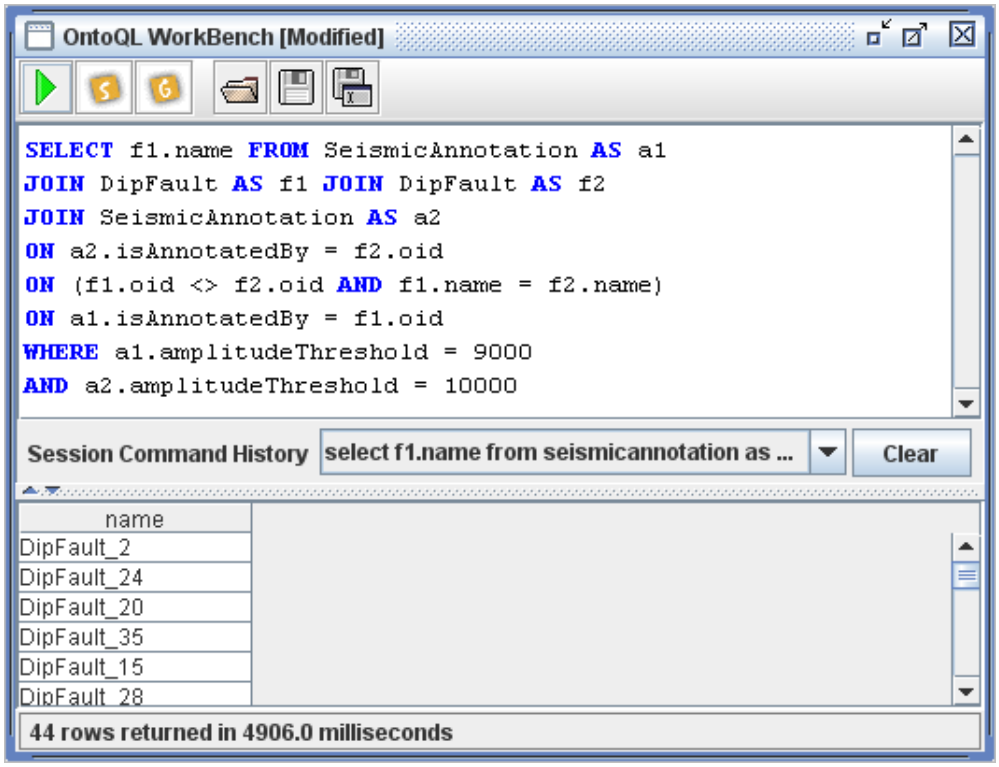
```
SELECT f.name FROM DipFault AS f
JOIN SeismicAnnotation AS a ON a.isAnnotatedBy = f.oid
WHERE a.amplitudeThreshold = 10000
```



Explanation. It should be noticed that the faults which is about the queries described here and in Table 8.10 are also seismic objects, and not geological objects. This query selects the *name* of all *DipFault* instances that are associated to some *SeismicAnnotation* whose *amplitudeThreshold* is equals to 10000.

Table 8.10: Q_{10}: Which faults were identified when using at the same time amplitude thresholds of 10000 and 9000 ?

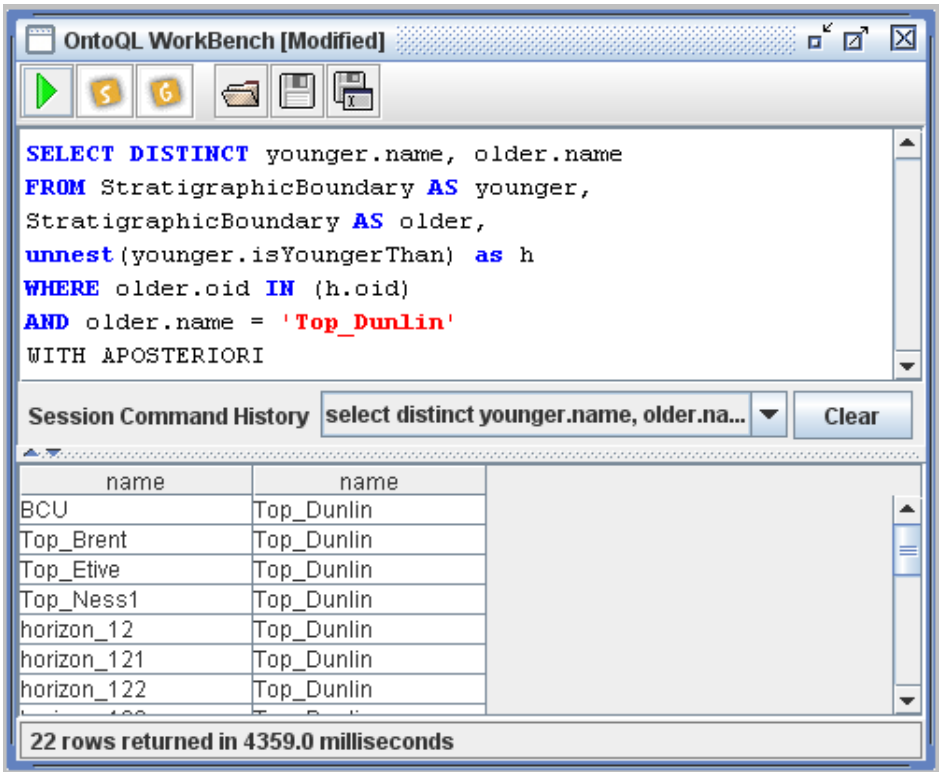
```
SELECT f1.name SeismicAnnotation AS a1
JOIN DipFault AS f1 JOIN DipFault AS f2
JOIN SeismicAnnotation AS a2 ON a2.isAnnotatedBy = f2.oid
ON (f1.oid <> f2.oid AND f1.name = f2.name)
ON a1.isAnnotatedBy = f1.oid
WHERE a1.amplitudeThreshold = 9000
AND a2.amplitudeThreshold = 10000
```



Explanation. This query selects the *name* of all *DipFault* instances that are at the same time associated to some *SeismicAnnotation* whose *amplitudeThreshold* is equals to 9000 and to some other *SeismicAnnotation* whose *amplitudeThreshold* is equals to 10000. It results *DipFault* instances that have been detected by two different identification processes: one using amplitude threshold of 9000 and the other using amplitude threshold of 10000.

Table 8.11: Q_8: Among all horizons identified during seismic interpretation, specify those which are younger (or older) than the Top Dunlin horizon.

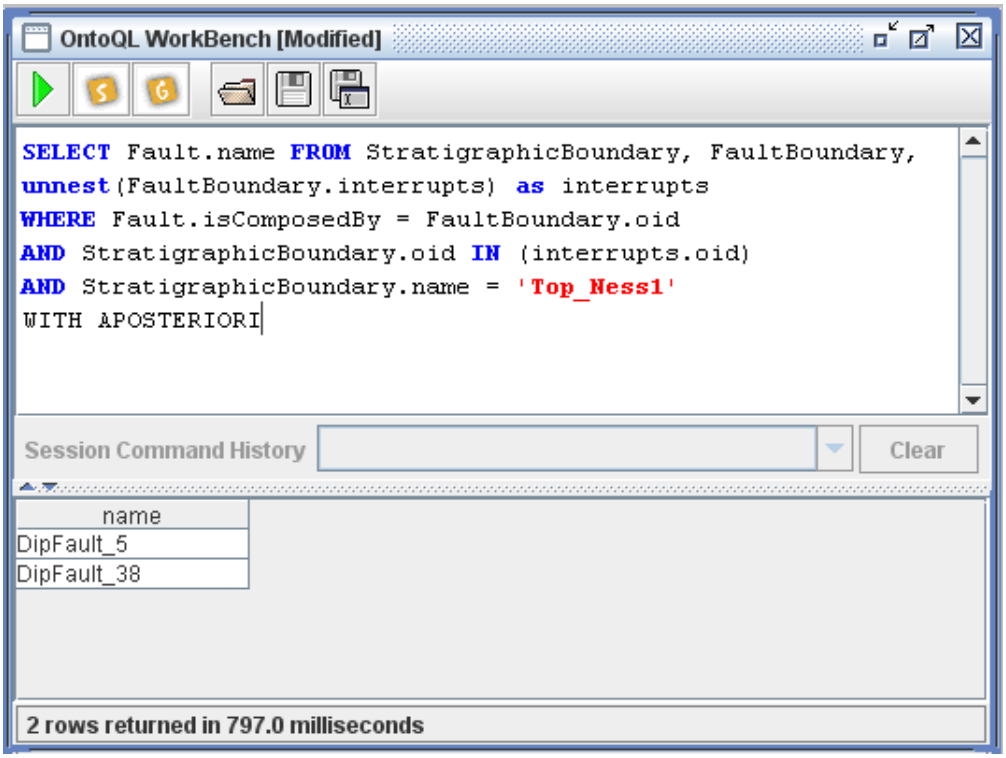
```
SELECT DISTINCT younger.name, older.name
FROM StratigraphicBoundary AS younger,
     StratigraphicBoundary AS older,
     unnest(younger.isYoungerThan) as h
WHERE older.oid IN (h.oid) AND older.name = 'Top_Dunlin'
WITH APOSTERIORI
```



Explanation. This user question presented apparently corresponds to the one that was already object of the query described in Table 8.8. However, while that query had to be reformulated by the user in terms of *isUpperThan*/*isLowerThan* relationships, it can now be directly formulated by the user in the language of geology in termes of *isYoungerThan*/*isOlderThan* relationships. In this case, the query gives as result all the instances of *geo:StratigraphicBoundary* plus the instances of the *a posteriori case-of* concepts of *geo:StratigraphicBoundary*, which are younger than the Top Dunlin horizon .
The answer is of course the same as that of query of Table 8.7 although it was obtained by a totally different procedure. The execution time required in the case of this query (4359 milliseconds) is significantly longer that that required for answering query of Table 8.7 (1593 milliseconds). However, this difference in the times of execution is the price to pay for allowing the formulation of queries using the user's own language.

Table 8.12: Q_{11}: Which faults interrupt horizon Top Ness 1?

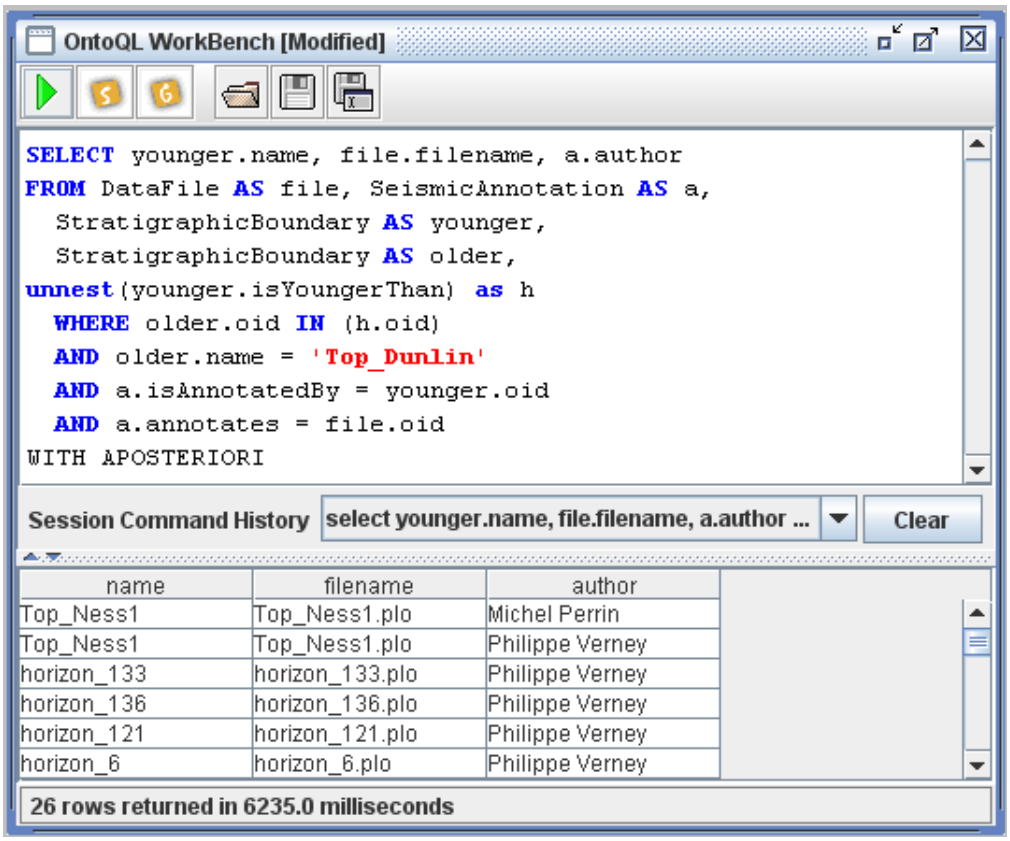
```
SELECT Fault.name FROM Fault, StratigraphicBoundary, FaultBoundary,
 unnest(FaultBoundary.interrupts) as interrupts,
 unnest(Fault.isComposedBy) as composed
WHERE FaultBoundary.oid IN (composed.oid)
AND StratigraphicBoundary.oid IN (interrupts.oid)
AND StratigraphicBoundary.name = 'Top_Ness1'
WITH APOSTERIORI
```



Explanation. This query retrieves all the instances of *geo:Fault* plus the instances of the *a posteriori case-of* concepts of *geo:Fault* which interrupt some instance (including the *a posteriori case-of* instances) of *geo:StratigraphicBoundary* whose name is Top Ness 1.

Table 8.13: Q_{12}: Retrieve the data files that represent the horizons that have been interpreted to be younger than the Top Dunlin horizon, and the author of this interpretation.

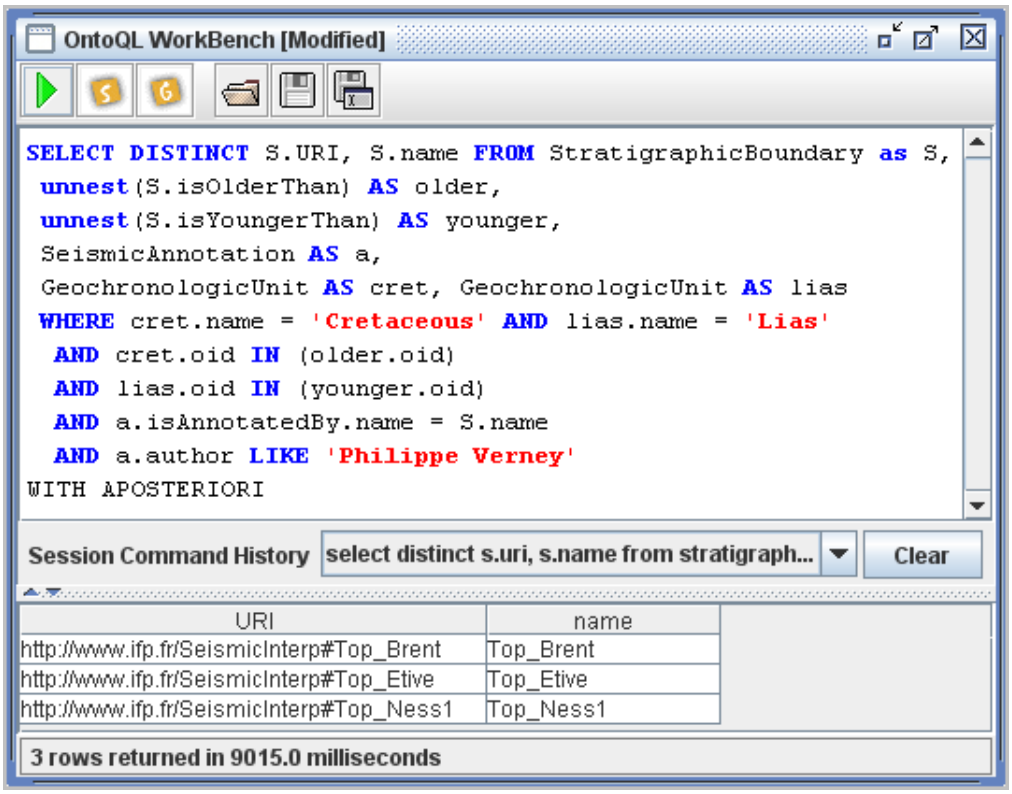
```
SELECT younger.name, file.filename, a.author
FROM DataFile AS file, SeismicAnnotation AS a,
  StratigraphicBoundary AS younger,
  StratigraphicBoundary AS older, unnest(younger.isYoungerThan) as h
  WHERE older.oid IN (h.oid) AND older.name = 'Top_Dunlin'
  AND a.isAnnotatedBy = younger.oid AND a.annotates = file.oid
WITH APOSTERIORI
```



Explanation. This user question presented corresponds to the query described in Table 8.11, but in this case we are able to find the original files that have been interpreted. The query above first searches the instances of geological horizons (using the *a posteriori case-of* quantifier) that are younger than the Top Dunlin horizon. After, it finds the *SeismicAnnotation* instances that are associated to the retrieved horizons and, from the annotation, retrieves the names of the data files associated to these horizons and the author of the interpretation.

Table 8.14: Q_{13}: Which reflectors were interpreted by Philippe Verney as having an age younger than Lias and older than Cretaceous ?

```
SELECT S.URI, S.name FROM StratigraphicBoundary as S,
 unnest(S.isOlderThan) AS older, unnest(S.isYoungerThan) AS younger,
 SeismicAnnotation AS a, GeochronologicUnit AS cret, GeochronologicUnit AS lias
WHERE cret.name = 'Cretaceous' AND lias.name = 'Lias'
 AND cret.oid IN (older.oid) AND lias.oid IN (younger.oid)
 AND a.isAnnotatedBy.name = S.name AND a.author LIKE 'Philippe Verney'
WITH APOSTERIORI
```



Explanation. A very significant question that is likely to be asked by geoscientists is related to finding which were the original objects (those issued from seismics, for example) that constitute the horizons that have a given geological age.
The above OntoQL expression selects all geological horizons that are younger than Lias and older than Cretaceous. After, it projects only the geological horizons whose name is the same than the name of local objects that were interpreted by Philippe Verney (i.e. that are annotated by *SeismicAnnotation* instances whose author is Philippe Verney). The answer can be both seismic objects or well objects, or objects from other local ontologies which have been mapped to the geology ontology concepts. The user is not obliged to choose, since the *a posteriori case-of* quantifier is employed.

M2 - Metamodel entities	M1 - Model classes	M0 - Instances
Class	151	4334
Annotation	1	3724
DataElement	5	2344
Total	157	10402

Table 8.15: Summary of number of elements created for the use case implemented in the OBDB

Conclusion and Future Work

Conclusion

The approaches proposed in this work are a contribution towards a complete semantic integration of engineering models. Among the several issues related to engineering model management, we particularly addressed here those related to:

- **Semantic annotation of engineering models**. The proposed annotation model enables to make explicit interpretations made about the *identification* of domain objects within engineering models.
- **Ontology integration**. Depending on their level of expertise, experts may identify some object as being more *general* or more *specific*, depending whether it is an instance of a more general ontology concept or of a more specific one. This required a means of aligning semantic unrelated ontologies.
- **Representation and persistence.** Thanks to metamodeling techniques it was possible to produce a uniform representation of ontologies, data and annotations. All these representations were persisted in an ontology-based database, which ensures the scalability of the proposal.
- **Querying.** When engineering models are annotated by experts using the domain ontologies, these models can be queried by users using significant vocabulary.
- **Application to an engineering domain.** The approaches proposed in this work were applied to a real use case in the activity of earth models building performed by petroleum companies.

Annotation of Engineering Models

The first contribution of this work is the proposal of an approach of *semantic-annotation of engineering models*, since none of the annotation tools proposed so far consider the semantic annotation issue from the perspective of *computer-based models*. A significant number of tools and frameworks are indeed available for providing ontology-based annotation, but, to the best of our knowledge, there presently exists no technique allowing to complete those models by formal comments or explanations, or to attach more semantics to the technical data produced by modeling tools. Although corporate knowledge can

be found for a large part in text repositories, such as project documentation and reports, some of this knowledge is also the result of the expertise that was introduced into engineering models by their builders. For instance, in the case of petroleum exploration, which is the object of the case study presented in the second part of this work, it appears that non-explicit interpretation carried out during earth modeling is the most important knowledge introduced into the models. Such strategic knowledge cannot be lost. Consequently, in view of the lack that exists in current annotation models, we proposed an annotation model fit for being applied to computer-based models in general.

In order to provide a means for processing these types of annotation, we proposed a model annotation approach which is composed of two metamodels:

- an *Engineering Metamodel*, which can be used as a common metamodel for the representation of any data model used by engineering models and also for transformations between different file formats;
- an *Annotation Metamodel*, which defines the *annotation* as a separate, explicit entity which is placed in the same abstraction level than the constructs for ontologies and data elements.

An annotation entity allows to create a link between an element of the engineering metamodel and the domain ontologies. The consequence of annotating engineering models is the creation of a set of ontological representations of the raw data that are used by these models.

Ontologies

One of the main challenges was to make an abstraction of the different types of information, languages, file formats, and to raise engineering models at the *knowledge level*. In view of this goal, we studied and applied methodologies presently available for making explicit the knowledge related to some domain and notably those related to the *ontology* research field. Ontology is one of the today hot keywords regarding the issue of describing data semantics. Just like computer programming languages, each ontology language proposes specific constructors that are more or less adapted to certain categories of issues. The main goal of some ontologies is *inference* while that of others is *structured characterization* of knowledge. Ontology engineers should choose the ontology model that is more adapted to the problems that they intend to solve. The choice of RDF triple as a representation instead of conventional databases was influenced by the desire to create modular and open databases that could be linked only through the associated URIs. Another benefit presented by RDF representation is portability. Any kind of data and metadata represented as triples is portable from one triple store to the other. On the contrary, in conventional databases data and metadata description depends on the implementation and on the vendor.

Ontology-based databases

The broad expressiveness of Semantic Web languages and their ability to provide various ways for expressing entities are also their weak point when considering the implementation of an actual database for managing large amounts of data. RDF-based databases are a young technology and may not support features that more mature relational database implementations have optimized long time ago, such as

performance, redundancy and transaction control.

For these reasons, in this work, we considered the implementation of *Semantic Web* ontologies in a more structured context, making use of an ontology-based database (OBDB) system implemented over a relational model. In view of this issue, the OntoDB OBDB was designed in order to provide a database architecture for storing ontologies and their instances. It stores ontology instances in an horizontal representation that ensures better performance when numerous properties are used in queries. It provides a *meta-schema part* that can be altered and increased, allowing the user to implement new constructs.

OntoQL is an ontology query language that provides definition, manipulation and query languages for both data and ontology. OntoQL is based on a core model algebra containing primitives supporting the exploration of different ontology models.

One of the most attractive characteristics of OntoDB/OntoQL is their metamodeling capability. This was capital for the implementation of our approach, since our goal was defining a general framework that would be independent of any particular application field and consequently potentially applicable to many engineering domains. Another advantage of OntoDB/OntoQL was that it enables the user to perform all operations about data and ontologies in one same tool. The user is able to query data using OntoQL, but also to modify, to insert and to delete data from the database, by simply using the Data Definition Language and the Data Modification Language of OntoQL. This is in contrast with RDF/OWL-based approaches, in which the user modifies the ontology using an ontology editor, which must be connected to the ontology repository in order to be queried. The approach used in this work allows to centralize all ontology management operations in the OntoDB repository.

Ontology integration

This work addressed the issue of how to integrate and exploit heterogeneous engineering models so as to offer a coherent view of different domains and allow the emergence of new knowledge relevant for the engineers. In theory, describing data resources by means of ontologies guarantees that these data resources are able to be adequately integrated. However, ontologies can be themselves sources of heterogeneity. In the case of information systems related to multidisciplinary domains, users must eventually deal with a bunch of different ontologies, each of which describes information related to one specific domain, with the result that ontologies themselves may contribute to increase heterogeneity. Consequently, in order to provide engineering models integration, one goal of the present work concerned the issue of handling ontology heterogeneity.

The first step towards a semantic integration process consists in identifying the nature of the heterogeneity. The problem that we handled in this work concerns the integration of sources coming from *different domains of interest* (such as Biology and Chemistry fields when considering for instance the Biochemistry domain, which integrates these two fields). In this case, integration is realized for allowing the emergence of new knowledge from disparate domains. This can be seen as a case of *perspectival representation*, in which sources from different fields represent models having different purposes with respect to the domain of interest.

The integration of *semantic unrelated domains* is not trivial, and, to our knowledge, it is not frequently

described in the literature concerning semantic integration systems. When dealing with concepts whose meanings are eventually orthogonal, automatic matching techniques are unhelpful. For this reason, in this work, the understanding of the semantic structure that integrate the ontologies is left to the domain experts, who are responsible for defining mappings between the different ontologies. Moreover, we wanted to prevent a *multi-ontology* integration structure, in which the correspondence between two ontologies would be directly established from one ontology to the other, so that the number of mappings would drastically increase for each new ontology to be integrated. Conversely, the integration architecture that we proposed is composed of several *Local ontologies* (LO), which formalize the semantic concepts of specific expertise fields involved in engineering modeling, and of a *Global ontology* (GO), which represents the common concepts shared by all local fields. The concepts of the LOs are then manually mapped to the concepts of the GO applying engineering domain rules.

The relationship that we proposed to be used for ontology mapping is an enrichment of the subsumption relation, the *is-case-of* relation, that is not typically implemented in ontology models. The reason for this is that we wanted to avoid defining *strong* subsumption relations, such as those of a subclass hierarchy. In this work, ontologies are kept completely independent, while being correlated by means of a *light* subsumption relation, which defines a concept as being a *case* of another. One advantage of this relation is that it can be created in a *a posteriori* fashion. This means that one can create a subsumption relation between concepts that have been already defined, without creating an existential dependence between them. The *is-case-of* relation creates partial inheritance hierarchies between concepts. The consequence is that one is not obliged to import/map all the properties of some subsuming (more general) concept, just those that are adequate for the subsumed (more specific) concept. This relation allows to define navigable hierarchies between concepts of different ontologies. The user is able to query a subsuming concept and receives as answer the instances of all the related subsumed concepts.

Application to Petroleum Engineering

We described a use case for the application of knowledge-based techniques. It concerns the *earth modeling workflows* performed by petroleum companies for building hydrocarbon reservoir models and more precisely *seismic interpretation*, which is the first in a chain of several interconnected and knowledge-intensive activities operated during these workflows.

We presented all the data sets related to this activity and the way in which interpretations are generated. We defined domains ontologies for describing the fields involved in the earth modeling workflow: the Basic Geology ontology, the Geological Time and Dating ontologies, the Seismic interpretation and the Well identification ontologies. These ontologies were first represented in the OWL language, and afterwards persisted in an ontology-based database. The Basic Geology ontology was chosen as the upper ontology (GO) to which specific ontologies (Seismic and Well local ontologies - LO) are mapped by means of *is-case-of* relations. A typical example of subsumption in these domains is is the one which results from defining the local concepts *seismic horizon* and *well marker* as a case of a *geological horizon*. The resulting subsumption relation allows to retrieve all the local objects by formulating a query using the global vocabulary. The advantage is that the user does not need to know which are the local ontologies mapped to the global ontology. So, in the case when the user is not an expert in the local domains, he/she can formulate a query by addressing the global concepts.

The *is-case-of* relation defines mapping in the concept level. When discussing the results of the use case concerning seismic interpretation, which we dealt with and, more particularly the implementation of the *a posteriori case-of* approach to the developed Geo-ontologies, the users insisted that they would like to be given a means of creating a subsumption link directly between *instances*. They need to be able to express that some instance of a *local object* (for instance a seismic horizon) is a case of some instance of a *global object*, (for instance a geological horizon). In the use case that we operated, this link was artificially created by adding search criteria when performing a query over the instances. For example, we retrieved some geological horizons that corresponded to given seismic horizons by specifying that their respective names were the same.

This issue clearly illustrates the kind of difficulties that we went trough when having to define the knowledge-model of some domain. Even when the domain specialists wish to share their expertise so that it can be acquired and formalized, they tend to present their knowledge using automatic shortcuts that cannot be perceived by knowledge engineers. These body of unarticulated knowledge that someone applies in daily tasks but is not able to describe in words have been called tacit knowledge by Nonaka et al. (1995). In this work, we succeeded to acquire and formalize the main objects that form the basic conceptualization of the domains involved in earth modeling, that is, the *ontologies*. Along with the ontologies, which represent the explicit part of knowledge, it is necessary to identify the tacit knowledge applied by experts, that is, the *way how* they perform the identification of objects. This would require work turned to the representation of reasoning and inferential knowledge, which was not the first goal of this work.

In collaboration with experts of the domain, we also defined among many possible ones a set of significant questions about the domain data, which were likely to be queried by users. We then demonstrated the applicability of the approaches proposed in our work for solving the knowledge management issues related to the use case. The typical user questions that we selected could not be handled without using a knowledge-management approach, because the answers to the related queries require crossing information issued from objects that were created when performing different activities such as seismic or well interpretation. For answering this type of questions by means of a knowledge-management approach, we had to establish relationships between concepts belonging to different ontologies. We also defined *annotation entities*, which can be queried by the users and enable them to retrieve the contexts in which interpretations were made. We finally demonstrated the operability of the knowledge oriented methodology that we propose by providing in an automatic way and in acceptable operating times, answers to the selected queries that were those expected by the users.

Future work

The present work must be considered an initial framework for allowing knowledge driven management of engineering domains, which guarantees a “first level” of knowledge management. This work has opened various perspectives concerning what remains to be done in order to develop a complete framework for engineering models integration.

Development of the three annotation processes

We implemented a use case of engineering model annotation in the case when the user performs a *white-box annotation* process. It still lacks the implementation of the two other identified annotation processes: the *black-box annotation* and the *intrusive annotation* processes.

Enhancing the annotation model

Another future research concerning annotation is the possibility of annotating *parts* of data artifacts. One way of doing that is using XPointers on XML files, in order to point the beginning and the end of the section to be annotated. In other file formats, the physical addresses inside the data file can be used. Other ways to improve the annotation model is to add access right to annotations and correctness values. We also leave for future work the exploration of mechanisms for specifying which authors have access to annotations and how this might affect the workflow.

Handling the *is-case-of* operation

The OntoQL language has been extended in order to support the creation of *is-case-of* relations between ontology concepts. However, it still lacks the possibilities of modifying and deleting some *is-case-of* relation. We must emphasize that those operations are only suitable over *a posteriori case-of* relations. The reason is that concepts that have been defined as *a posteriori case-of* other concepts, had not had their *structure* changed, since the existing properties were only *mapped* to each others. Modifying/deleting the *a posteriori case-of* relation will only have influence on the *results* of queries over the subsuming concept: instances of the previously subsumed concept might not be able to be retrieved anymore. On the contrary, when creating an *a priori case-of* relation, the structure of the subsumed concept is strongly affected by the subsumption, since it *imports* properties from the subsuming concept. The *a priori case-of* relation cannot be, thus, modified or deleted.

An important perspective for the *is-case-of* relation is the possibility of being used to create *mapping expressions*. Instead of using just the subsumption relation to create a correspondence between different concepts, it should be possible to define logical and mathematical expressions between the concepts. For example, one should state that the subsumed properties corresponds to twice the subsuming property ($propA = 2 * propB$). This possibility should be implemented as an extension of the *is-case-of* operator in the OntoQL language and of the *is-case-of* primitives in OntoDB.

Ontology integration

The strategy of ontology mapping applied in this work is that of creating correspondences in the conceptual level. A mapping relation (the *is-case-of* relation) is established between ontology concepts, and not between their instances. However, when faced with the results obtained from the OntoDB-OntoQL approach, the users expressed the wish of also creating correspondences between *instances*. Notably, they want to be able to express that some instance of a *local object* is equivalent to some instance of a *global*

object. In order to establish this relation, one needs to define a relation similar to the built-in OWL property *owl:sameAs*, which links an individual to an individual. Such an *owl:sameAs* statement indicates that two URI references actually refer to the same thing: the individuals have the same "identity".

However, this instance mapping is seen as an *interpretation made by the expert*. The consequence is that the mapping link will need to be *annotated*, in the same way as the *engineering models annotation*. Therefore, unlike the cited OWL property, we need to define instance-to-instance links that can have their own attributes, such as author, date and used tool. In the end, this will be represented simply as an extension in the annotation model in order to allow to create an annotation entity between instances of different ontologies. The biggest issue, however, is to identify together with the experts the way how these instances will be created and linked. Considering the use case that has been dealt with in Chapter 8, the way in which the expert produces an interpretation is the result of a complicated procedure[56]. This is the reason why we preferred to leave this work to be done as a future complement to the thesis.

Reasoning

During the work of ontology building, inference rules were defined that allow to discover new possible temporal relations between time scale elements from the relations established by the expert. However, the OntoDB database does not dispose at present of a repository for storing inference rules and neither of an inference engine. The next implementation of the OntoDB architecture, called *OntoDB2* (Fankam, 2008) will enable the user to define derivation functions, which are similar to domain-specific inference rules.

An alternative implementation that can replace the inference mechanism is to implement inference rules as *triggers*[57] in the database. Triggers can be encoded to be fired each time a new temporal relation is entered in the database. The trigger code should ensure the creation of every possible derived temporal relation from the furnished one. However, the trigger code must be defined so as to avoid too complex treatments (for example, recursivity).

Applicability to other domains

We can imagine that the approaches proposed in this work are potentially applicable to other engineering domains. From the experience brought by this work we state two criteria that should be present in such domains:

- Domains whose activities rely in computer-based models;
- Models whose objects can be described by concepts and properties in some ontology.

[56] In the procedure described by Verney (2009), an algorithm detects when a seismic horizon is *close* to some well marker according to a given threshold. If it is the case, the seismic horizon is given the name of the geological horizon corresponding to this neighbour well marker. The seismic horizon interpretation thus depends on a particular procedure which takes as input among others interpretations concerning well markers.

[57] A trigger is a procedural code that is automatically executed in response to certain events on a particular table or view in a database.

An example of engineering domain to which this approach could be applied is to *airplane tooling design*, due to the complexity of the activity and of the models involved.

Tooling: Geological Knowledge Editor

Moreover, possible future implementations are expected from the propositions made in this work.

At first, the major item that remains to be completed is the software tool designed by the IFP/ENSMP/ENSMA partners by the name *Geological Knowledge Editor* (GKE). The GKE was originally developed as a graphical interface that allowed expert users to create geological objects, such as horizons and faults, and to manually define the order in which they were arranged in a stratigraphical sequence. The output of the first version of GKE was a Geological Evolution Schema (GES), i.e. a graph-representation that formalizes the order of occurrence among various geological events (Perrin, 1998). The new version of the GKE that is presently being prepared at ENSMA will be a prototype, which will implement the main ideas proposed in this thesis.[58] We will describe them in the next sections.

GKE and Metamodels

The development of the new version of the GKE is based on the Eclipse Modeling Framework (EMF). EMF is a code generation facility for building Java applications based on model definitions. From a UML model specification described, EMF provides tools and runtime support for producing a set of Java classes, in order to provide viewing and a basic editor. The EMF framework will enable the GKE to implement the metamodels proposed in Part II of this thesis. The GKE will then be able to use the constructs of the Engineering Metamodel for representing the input and output data files of some interpretation task. Moreover, it will represent the interpretation provided by the user using the Annotation Metamodel. The flexibility and modularity acquired with this implementation will hopefully demonstrate the practical advantage of having proposed models in a meta-level.

GKE and Ontologies

The most time consuming process in this work was with no doubt defining the domain ontologies. Nevertheless, we are aware that this work is not finished. Each time a new use case is identified, we imagine new manners in which the concepts should have been organized, new possible specializations for concepts, or, most commonly, new attributes and properties to be added. Ontology building is a continuous process and we do not believe that there is a way to ensure that a final and complete ontology is achieved. Therefore, the tools that rely on domain ontologies need to be prepared for ontology evolution. This possibility is already being studied and put in practice with the development of the GKE. Notably, the EMF framework allows to use ontologies as the base model for the visual editors. This means that each time an ontology changes, the input forms automatically change accordingly. However, there not yet exists an appropriate solution for making evolving instances of a modified ontology. Evolution and versioning

[58] The new version of the GKE is being implemented by a group of students of the LISI laboratory, in the ENSMA, Poitiers, under supervision of Jean-François Rainaud and under my own consulting.

is a difficult issue, which has become an important topic of ontology research. Moreover, as already explained in the Conclusions of Chapter 7, the GKE will implement the geological time and dating ontologies as well as the geological codification, which will allow the user to find easy answers to all questions related to identification and correlation of geological time units and boundaries.

GKE and ontology-based queries

One tool that is notably lacking in the proposed work is a system that would enable the user to pick concepts in the hierarchy of the ontology in order to build queries about the various domains. We are aware that formulating queries can be a challenging experience for users that are not acquainted with computer-based languages. Moreover, the users need to have a vision of the domain ontology structure in order to choose the right concepts for their queries. The visualization of the ontology in a hierarchical structure was already part of the original version of GKE and will be kept. A new module will be added in order to create OntoQL queries for selecting instances from the chosen concepts, and also for adding search criteria to the selection: attribute filtering and choice of the concepts hierarchy in which to search (*is-a* , *a priori case-of* or *a posteriori case-of* hierarchy).

Appendix A

UML Overview

1 Introduction

The UML, or Unified Modeling Language(OMG, 2008), is a textual and graphical notation used to formalise our understanding of systems.[59] There is a growing interest in the use of UML class diagrams as a modeling language to represent domain ontologies (Guizzardi et al., 2004). This appendix presents an overview of the features of UML that are used for representing the ontologies developed in this work.

UML defines thirteen types of diagrams, divided into three categories: **Structure Diagrams**, **Behavior Diagrams** and **Interaction Diagrams**. In order to represent ontologies, we are interested in diagrams that represent static application structure (Structure Diagrams), more specifically, the **Class Diagram** and the **Object Diagram**.

- The **Class diagram** describes the structure of a system, by showing its classes, their attributes and operations, and the associations between classes.
- The **Object diagram** presents a set of objects and attributes, and the links between the objects.

In UML, an object represents a particular instance of a class. For this reason, in this work, *objects* are represented together to their *classes*, fusing the two diagrams in a Class and Object diagram.

2 UML constructs for classes and objects

In terms of ontology representation, we focus here on the most basic representation constructs of the UML profile.

In a class diagram, classes are depicted as a rectangle with three horizontal sections (see Figure A.1(1)):

- the class name (e.g. *Person*),

[59] see http://www.uml.org/

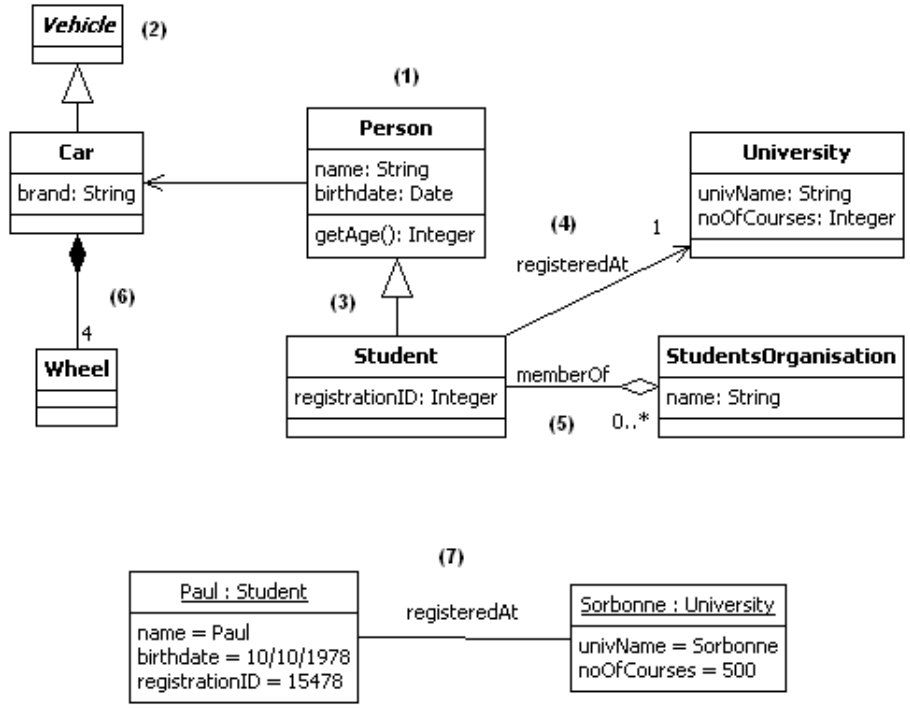


Figure A.1: Example of UML diagram: classes and objects

- the class attributes specified by their name (e.g. *name* and *birthdate*), type (e.g. *String* and *Date*) and visibility (public, by default),
- the class operations specified by name (e.g. *getAge*), argument list (which is empty), return type (e.g. *Integer*) and visibility (public, by default).

For the purposes of representing ontologies, we consider that: (i) all attributes can be considered to have public visibility (since ontologies are built to be shared) and (ii) ontology classes do not present operations. An special type of classes are *abstract* classes, which do not have instances. An abstract class name is depicted in italics (e.g. the class *Vehicle* in Figure A.1(2)).

There are three types of relationships that can be created between classes:

- *Generalisation*, which corresponds to the subsumption relation known as *is-a* or *subclass-of* in ontologies. It is represented by lines with a large arrowhead (a completed triangle) at the top pointing to the super class. It comprises inheritance of attributes and operations from the most general class (e.g. class *Person* is a generalisation of class *Student* in Figure A.1(3)).
- *Association*, represented by solid lines between two classes with an open arrowhead if the association is known by only one of the classes (e.g. class *Student* has an association named *registeredAt* with class *University* in Figure A.1(4)). Semantic relations in ontologies can be represented with the *association* construct.
- *Aggregation*, which is the typical *whole/part* relationship. It is depicted by a diamond at the

aggregate end of the link (e.g. class *Student* has an aggregation relationship named *memberOf* with class *StudentsOrganisation* in Figure A.1(5)). UML includes a stronger type of aggregation (composite aggregation), notated by a solid black diamond, which implies that the 'part' does not exist without the 'whole' (e.g. class *Wheel* has a composite aggregation relationship with class *Car* in Figure A.1(6)). However, we do not make a distinction between the two types of aggregation in ontologies.

The ends of association and aggregation relationships may be annotated with multiplicity indicators which denote how many instances of the class are expected within this association. Considering the multiplicity labels on the associations *registeredAt* and *memberOf*, an *Student* can be registered at only one *University*, and can be member of any number of *StudentsOrganisation* (including none). A *Car* can have exactly 4 wheels.

An object is depicted as a rectangle, with two horizontal sections (see Figure A.1(7)): in the top it shows the name of the instantiated object separated from the class name by a ':' and underlined, to show an instantiation (e.g. *Paul:Student*). In the botton part, the values of object's attributes are assigned using the notation *attributename = value* (e.g. *univName = Sorbonne*). A link between two objects is represented as a solid line, with no arrowheads or multiplicity labels (e.g. the link *registeredAt* between *Paul* and *Sorbonne*).

Appendix B

Geological Time Scale

1 International Stratigraphic Chart

The 2009 version of the International Stratigraphic Chart is given here for visualization (Figure B.1). The original version can be downloaded from the web site of the International Commission on Stratigraphy.[60]

[60] http://www.stratigraphy.org/

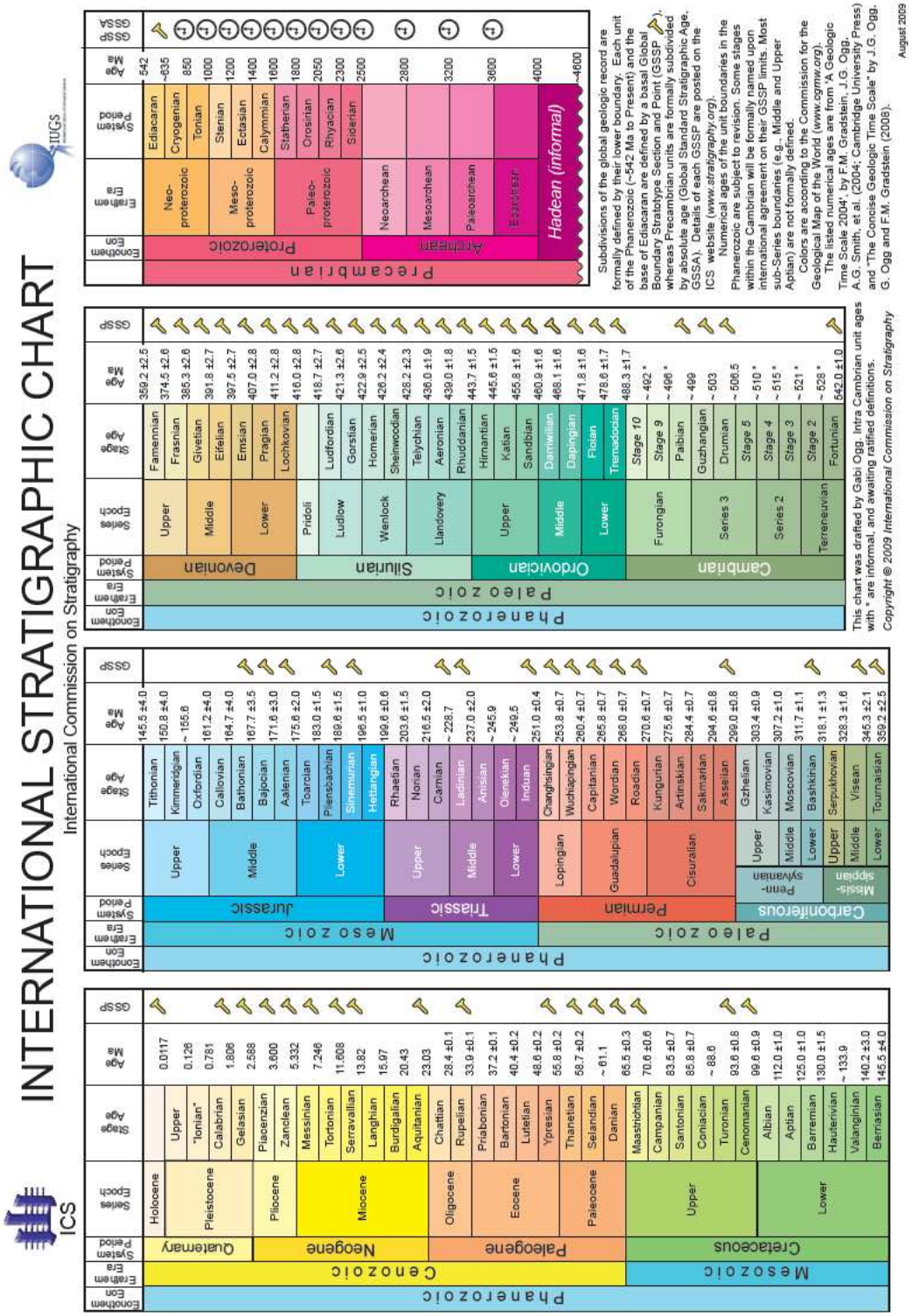


Figure B.1: International Stratigraphic Chart

Appendix C

SPARQL queries for the Seismic Interpretation Use Case

We present here the SPARQL version of the OntoQL queries presented in Section 2.2.2 of Chapter 8 in order to show that if a *Semantic Web* approach was chosen in detriment of the data-centric approach, it would be possible to create equivalent SPARQL queries. SPARQL was chosen because it is a W3C Candidate Recommendation towards a standard query language for the Semantic Web. We take back the list of user questions presented in Section 1.4 of Chapter 8, as follows.

Q_1 - Which horizons where identified that are younger than Lias, and older than Cretaceous ?

Q_2 - From which seismic image comes the horizon BCU ?

Q_3 - With which amplitude threshold the horizons Top Etive and Top Brent were detected ?

Q_4 - Which reflectors are associated with the Top Etive horizon?

Q_5 - Which wells made possible the association of horizons extracted from the seismic image to the marker Top Etive?

Q_6 - When was made the interpretation which allowed the identification of the horizon Top Brent?

Q_7 - Who carried out this interpretation?

Q_8 - Among all horizons identified during seismic interpretation, specify those which are younger (or older) than the Top Dunlin horizon.

Q_9 - Which faults were identified with an amplitude threshold 10000 ?

Q_{10} - Which faults were identified when using at the same time amplitude thresholds of 10000 and 9000 ?

Q_{11} - Which faults interrupt the horizon Top Ness 1?

Q_{12} - Retrieve the data files that represent the horizons that have been interpreted to be younger than the Top Dunlin horizon, and the author of this interpretation.

Q_{13} - Which reflectors were interpreted by Philippe Verney as having an age younger than Lias and older than Cretaceous ?

The Table 1 presents the SPARQL queries and their results for the questions Q_1 to Q_{13} in the list of user questions . We consider the given SPARQL prefixes `PREFIX s: <http://www.ifp.fr/SeismicInterp#>` and `PREFIX g: <http://www.ifp.fr/BasicGeology#>`.

Q$_n$	**SPARQL query**
Q_1	SELECT DISTINCT ?hname WHERE {?h rdf:type g:StratigraphicBoundary. ?h g:name ?hname. ?lias g:name "Lias". ?cret g:name "Cretaceous". ?h dating:isYoungerThan ?lias. ?h dating:isOlderThan ?cret}
Q_2	SELECT DISTINCT ?filename WHERE { ?block rdf:type s:SeismicBlock. ?h s:isPartOfBlock ?block. ?h s:name "BCU". ?annot s:annotates ?datafile. ?annot s:isAnnotatedBy ?block. ?datafile s:filename ?filename }
Q_3	SELECT DISTINCT ?hname ?thold WHERE { ?h rdf:type s:Horizon. {?h s:name "Top_Etive"} UNION {?h s:name "Top_Brent"}. ?annot s:isAnnotatedBy ?h. ?h s:name ?hname. ?annot s:amplitudeThreshold ?thold }
Q_4	SELECT DISTINCT ?hname ?rname WHERE { ?h s:name "Top_Etive". ?h rdf:type s:Horizon. ?h s:name ?hname. ?h s:isComposedBy ?r. ?r s:name ?rname. }
Q_5	SELECT DISTINCT ?mname ?wname WHERE { ?h rdf:type s:Horizon. ?h s:name "Top_Etive". ?h s:isComposedBy ?r. ?r s:hasBeenMarkedBy?m. ?m s:isPartOf ?w. ?m s:name ?mname. ?w s:name ?wname. }
Q_6	SELECT DISTINCT ?hname ?date WHERE { ?h rdf:type s:Horizon. ?h s:name "Top_Brent". ?h s:name ?hname. ?annot s:isAnnotatedBy ?h. ?annot s:date ?date. }
Q_7	SELECT DISTINCT ?hname ?author WHERE { ?h rdf:type s:Horizon. ?h s:name "Top_Brent". ?h s:name ?hname. ?annot s:isAnnotatedBy ?h. ?annot s:author ?author.}
Q_8	SELECT DISTINCT ?hname2 WHERE { ?h1 s:isUpperThan ?h2. ?h1 s:name "Top_Dunlin". ?h1 s:name ?hname1. ?h2 s:name ?hname2. ?h1 rdf:type s:Horizon. ?h2 rdf:type s:Horizon. }

Q_9	SELECT DISTINCT ?fname ?thold WHERE { ?f rdf:type s:DipFault. ?f s:name ?fname. ?annot s:isAnnotatedBy ?f. ?annot s:amplitudeThreshold ?thold. FILTER (?thold = 10000.0). }
Q_{10}	SELECT DISTINCT ?fname1 WHERE { ?f1 rdf:type s:DipFault. ?f2 rdf:type s:DipFault. ?f1 s:name ?fname1. ?f2 s:name ?fname2. ?annot1 s:isAnnotatedBy ?f1. ?annot2 s:isAnnotatedBy ?f2. ?annot1 s:amplitudeThreshold ?thold1. ?annot2 s:amplitudeThreshold ?thold2. FILTER ((?thold1 = 10000 && ?thold2 = 9000) && (?fname1 = ?fname2) && (?f1 != ?f2)).}
Q_{11}	SELECT DISTINCT ?fname ?thold WHERE { ?f rdf:type s:DipFault. ?f s:name ?fname. ?f s:isComposedBy ?hg. ?hg s:disconnects ?h. ?h s:name "Top_Ness1"}
Q_{12}	SELECT DISTINCT ?filename WHERE { ?h1 s:isUpperThan ?h2. ?h1 s:name "Top_Dunlin". ?h1 s:name ?hname1. ?h2 s:name ?hname2. ?h1 rdf:type s:Horizon. ?h2 rdf:type s:Horizon. ?annot s:isAnnotatedBy ?h2. ?annot s:annotates ?datafile. ?datafile s:filename ?filename.}
Q_{13}	SELECT DISTINCT ?rname WHERE {?r rdf:type s:Reflector. ?r s:name ?rname. ?lias s:name "Lias". ?cret s:name "Cretaceous". ?h rdf:type g:StratigraphicBoundary. ?h g:name ?hname. ?h s:isYoungerThan ?lias. ?h s:isOlderThan ?cret. ?annot s:isAnnotatedBy ?r. ?annot s:author "Philippe Verney". FILTER (?hname = ?rname)}

Table 1: User questions in SPARQL

The first choice to execute the SPARQL queries was to use CORESE, the Conceptual Resource Search Engine developed by the Edelweiss team of INRIA.[61]. However, CORESE typically processes only a subset of OWL Lite ontologies. The OWL documents output by the Seismic Interpretation module are incompatible with OWL Lite, since they make use of restrictions and class union.[62] It would be then nec-

[61] http://www-sop.inria.fr/edelweiss/software/corese/

[62] We could load the OWL document within CORESE, but some queries produced errors related to the union classes.

essary to create a simplified version of the OWL document. Another point is that when using CORESE as query interface, the ontologies are stored as an external file (OWL file) directly into the disk, and must be loaded in the engine's main memory. The load operation costs a significative time (the loading time is not told by CORESE interface). The queries are then performed over the loaded ontologies, but once the user finishes the application, the main memory is cleared. This approach becomes inefficient for very large ontologies, such as those of domains that produce huge quantities of data like engineering and scientific domains.

We decided to execute the queries in the SPARQL query panel in Protégé 3.3.1.[63] This panel allows the Protégé user to execute SPARQL queries over the loaded ontology only. The OWL document used was the OWL version of the seismic interpretation use case. The Seismic Interpretation engine outputs, for each interpretation cycle, one OWL document with the instances of seismic objects resulted from the interpretation. We fusioned two files that were resulted from two different interpretations. The instances in the fusioned OWL document correspond to the instances that were stored in the OntoDB ontology.

Concerning the formulation of the queries, the main difference between OntoQL and SPARQL queries is that objects in SPARQL queries do not need to be *typed*. An RDF-based repository is a gathering of triples that have the ontology as container. In OntoDB, on the contrary, each instance has a specific container: the table that represents the instance's superclass. In a RDF-based repository, any triple that satisfies the condition given by the clause *WHERE*, are included in the results, while in OntoDB, the user needs to explicitly indicate the name of the concept from which he/she wants to retrieve instances.

SPARQL may present an advantage because of its flexibility in relation to object typing, however, the language still lacks of some important operators proposed by SQL, such as *COUNT* and *GROUPBY*, which help the visualization of the results.[64] These operators are present in OntoQL, since it is an extension of SQL.

Using SPARQL queries the user is able to answer one part of the questions asked by the user, notably, those that query directly the concepts of the target ontology. For example, query Q_1 uses only concepts from the Basic Geology, and queries Q_2 to Q_{10} query directly the concepts of the local ontologies. However, the queries that use the *global vocabulary* in order to retrieve local concepts from various ontologies are not possible to be defined in SPARQL, due to the lack of *a posteriori case-of* operator. Queries Q_{11} to Q_{13} can be handled by addressing directly the concepts from the local ontologies, instead of those of the global ontology (Q_{13} in order to be answered demands the OWL document with the instances of the Basic Geology to be fusioned to the OWL document with the instances of local ontologies). But this looses the advantage of the *a posteriori case-of* strategy.

Another disadvantage is that in a RDF-based approach, we lack the metamodel level, which makes the methodology generic and thus applicable to other domains. As we explained in Section 4.3.1 of Chapter 2 RDF-based ontologies cannot have their metaclasses modified, unless they become OWL Full ontologies. But then they loose some guarantees (such as the decidability of basic inference) which OWL DL and OWL Lite provide for reasoning systems and which constitute an advantage when developing OWL ontologies.

[63] http://protege.stanford.edu/doc/sparql/

[64] The *COUNT* operator allows to retrieve the number of rows resulted by some query. The *GROUPBY* operator groups similar results by one or more columns.

Finally, other RDF-based repositories, such as SESAME (Broekstra et al., 2002) or the recently launched OWLIM (Kiryakov et al., 2005) could have been used in order to implement the database. But the objective of this work was not to make a comparison of the efficiency of different semantic repositories nor prove that the repository employed in this thesis is the best in efficiency. The approach OntoDB-OntoQL was used mainly because it provides metamodeling capabilities, also because it enables the storage of the data and knowledge in a fixed repository, that does not need to be loaded in main memory, and finally because it manages huge quantity of data. We believe that these points are important for the implementation of an engineering use case.

List of Figures

List of Tables

Glossary

Absolute dating : Geological age measured obtained by radiometric measurements, expressed in million years (My)
AI : Artificial Intelligence
AKSIO : Active Knowledge Systems for Integrated Operations

BSU : Basic Semantic Unit

CO2 storage site : Underground site fit for CO2 storage
Cross-section : A sketch showing the arrangement of geological terrains along a vertical section plane
CSV : Comma Separated Values: type of file in which data is separated by commas

DAML : DARPA Agent Markup Language: one of the first languages for representing ontologies
DCMI : Dublin Core Metadata Initiative - http://dublincore.org/
DDL : Data Definition Language
Deposit : The result of geological sedimentation
Dip : The local 3D orientation of a geological surface
DML : Data Manipulation Language
DQL : Data Query Language
Drilling : A means of exploring underground along a linear trajectory (vertical or inclined)

e-Wok Hub : Environmental Web Ontology Knowledge Hub
Earth model : A 3D (or 4D) model showing underground geological arrangement
Energistics : Energy Standards Resource Centre, the new name of POSC consortium
EpiSEM : Epicentre Shared Earth Model
Erosion : Removal of geological matter due to a surface mechanical effect
EXPRESS : Information modeling language defined in ISO 10303-11:1994

Fault : A planar or suplanar discontinuity affecting geological terrains
Fluid migration : Hydrocarbon migration from source rocks to a reservoir
FOAF : Friend of A Friend

GAV : Global-As-View

Geological map : A map (produced for instance by a Geological survey) showing the geology of a given region
Geological structure : Description of the geometrical features of a geological object and of its eventual deformations
Geological survey : A governmental institution responsible caring about geological heritage
Geological time : Time at the geological scale (millions and billions of years)
GEON : Geosciences Network
Geophysics : A geoscience for studying underground geology by means of physical methods
GeoSciML : Geoscience Markup Language
GIF : Graphics Interchange Format: a common bitmap image format
GML : Geography Markup Language
GPS : Global Positioning System
GTS : Geological Time Scale

Horizon : A sedimentary geological boundary
Horizon gap : The split of some geological horizon due to its interruption by a fault
HTML : Hyper Text Markup Language: the predominant markup language for web pages
Hydrogeology : A geoscience dedicated to the study of underground water

ICS : International Commission of Stratigraphy
IIP : Integrated Information Platform
ISO : International Organization for Standardization
ISS : International Stratigraphic Scale
IUGS : International Union of Geological Sciences

JPEG : Joint Photographic Experts Group: a file format commonly used for image compression

KM : Knowledge Management

LAV : Local-As-View
Lithology : A geoscience dedicated to the study of rocks as such (synonym: petrology)

Marine fossil : Fossil associated with a marine environment
MDA : Model Driven Architecture
MDE : Model Driven Engineering
Metamorphism : Mineralogical transformation of rocks submitted to temperature/pressure gradients underground
MOF : Meta-Object Facility
MPEG : Moving Pictures Expert Group: international standard for audio/video compression

NADM : North American Geologic Map Data Model

O3R : Oilfield Ontology Repository
O4OIL : Open Oilfield Ontology Organization
OBDB : Ontology-Based Database
ODM : Ontology Definition Metamodel

OMG : Object Management Group
On lap surface : A stratigraphical surface interrupting younger horizons
OWL : Web Ontology Language: a knowledge representation language, based on an RDF syntax, that provides a very complete set of elements for the description of ontologies

Palaeogeography : Regional description of past environments
Petrology : A geoscience dedicated to the study of rocks as such (synonym: lithology)
Petrophysics : Study of rock physical properties
PLIB : Parts LIBrary - Norme ISO 13584
POSC : Petrotechnical Open Standards Consortium

QBE : Query-By-Example
QVT : Query/View/Transformation

RDF : Resource Description Framework: the W3C specification for modeling information that is implemented in web resources
RDFS : Resource Description Framework Schema: a knowledge representation language, based on an RDF syntax, that provides the basic elements for the description of ontologies
Reflector : Part of a geological horizon on which seismic waves are reflected
Regional geology : Geological description of a given geological area
Relative dating : Chronological ordering of two geological items by a olderThan/younger relationship
RESCUE : REServoir Characterization Using Epicentre
Reservoir : An underground volume where fluids (oil, gas, water) accumulated
Reservoir model : A 3D earth model for reservoir description

Sedimentary basin : A marine area where sediments were deposited
Sedimentation : Mechanical accumulation of particles deposited by gravity
Sedimentology : A geoscience dedicated to the study of sedimentary processes
Seismics : A geophysical method based on the study of underground propagation of acoustic waves
SEM : Shared Earth Modeling
SPARQL : SPARQL Protocol And RDF Query Language
SQL : Structured Query Language
STEP : STandard for the Exchange of Product data - Norme ISO 10303
Stratigraphical age : Geological age determined in reference with a Geological time Scale (opposite Absolute age)
Stratigraphical model : An earth model describing the arrangement and the lithological content of geological units in a given underground volume
Stratigraphy : Study of stratigraphic successions
Stratotype : A reference set of sedimentary units used for defining a geological age
Structural Geology : Study of spatial arrangements of geological units and of rock and geological units deformations
Structural interpretation : Specifying the spatial and chronological relationships between geological objects
Structural model : A 3D earth model showing the spatial arrangement of geological surfaces in a given underground volume

Surface picking : Hand operated or automated sampling of a seismic horizon
SWEET : Semantic Web for Earth and Environmental Terminology

Tectonic boundary : A geological surface corresponding to some discontinuity (example: a fault)
Tectonics : A geoscience dedicated to the study of rock deformation processes

UML : Unified Modeling Language
Unconformity : A geological structure corresponding to a younger horizon interrupting older ones
URI : Uniform Resource Identifiers: a string of characters used to identify or name a resource on the Internet
URL : Uniform Resource Locator: the addressing system used in the WWW, which contains the method, the server and the path of the file to be accessed

W3C : World Wide Web Consortium: the main international standards organization for the World Wide Web
Well log : Registration of a given physical parameter along a well trajectory
Well marker : A point along a well trajectory interpreted as corresponding to the intersection between the well trajectory and some geological horizon
Well/Well bore : The linear possibly kilometer long cavity resulting from underground drilling
WITSML : Wellsite Information Transfer Standard Markup Language: a standard for transmitting technical data about drilling between organizations in the petroleum industry
WSDL : Web Services Description Language - http://www.w3.org/TR/wsdl20/
WWW : World-Wide Web: the hypertext-based Internet service used for browsing Internet resources

XML : eXtensible Markup Language
XPointer : XML Pointer Language

Bibliography

Relevant Publications

Y. Ait-Ameur, N. Belaid, M. Bennis, O. Corby, R. Dieng, J. Doucy, P. Durville, C. Fankam, F. Gandon, A. Giboin, P. Giroux, S. Grataloup, B. Grilheres, F. Hussson, S. Jean, J. Langlois, P. H. Luong, L. Mastella, O. Morel, M. Perrin, G. Pierra, J.-F. Rainaud, I. A. Sadoune, E. Sardet, F. Tertre, and J. F. Valiati. Semantic hubs for geological projects. In *ESWC Workshop on Semantic Metadata Management and Applications (SeMMA)*, Teneriffe, Spain, 2008.

L. Mastella, M. Perrin, M. Abel, J.-F. Rainaud, and W. Touari. Knowledge management for shared earth modelling. In *69th EAGE Conference & Exhibition incorporating SPE EUROPEC 2007*, London, 2007a. EarthDoc. URL: `http://www.earthdoc.org/detail.php?paperid=D021&edition=27`.

L. Mastella, Y. Ait-Ameur, M. Perrin, and J.-F. Rainaud. Ontology-based model annotation of heterogeneous geological representations. In J. C. Hammoudi, J. Filipe, and Slimane, editors, *4th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 290–293, Funchal, Madeira, Portugal, 2008-08-18 2008a. INSTICC Press. ISBN 978-989-8111-27-2.

L. Mastella, Y. Ait-Ameur, M. Perrin, and J.-F. Rainaud. Annotation à base ontologique de modèles: application aux modèles en géologie pour le stockage de co2. In *26ème Congrès INFORSID - Atelier Systèmes d'Information et de Décision pour l'Environnement*, pages 1–10, Fontainebleau, 2008b.

L. Mastella, M. Perrin, Y. Aït Ameur, M. Abel, and J. F. Rainaud. Formalising geological knowledge through ontologies and semantic annotation. In *70th EAGE Conference & Exhibition incorporating SPE EUROPEC 2008*, Rome, 9-12 June 2008c. EarthDoc. URL: `http://www.earthdoc.org/detail.php?paperid=I024&edition=40`.

L. Mastella, Y. Ait-Ameur, S. Jean, M. Perrin, and J. Rainaud. Semantic exploitation of engineering models: An application to oilfield models. In Springer, editor, *Proceedings of the 26th British National Conference on Databases: Dataspace: The Final Frontier*, volume 5588 of *Lecture Notes in Computer Science*, pages 203–207, Birmingham, UK, 2009a. Springer.

L. Mastella, Y. Aït Ameur, S. Jean, M. Perrin, and J. F. Rainaud. Semantic exploitation of persistent metadata in engineering models: application to geological models. In IEEE, editor, *Proceedings of 3rd International Conference on Research Challenges in Information Science (RCIS)*, pages 129–138, Fez, Morocco, 2009b. IEEE.

L. S. Mastella, M. Abel, L. F. D. Ros, M. Perrin, and J.-F. Rainaud. Event ordering reasoning ontology applied to petrology and geological modelling. In *IFSA 2007 - Soft Computing in Petroleum Applications Special Session*, volume 42, pages 465–475, 2007b.

M. Perrin, J.-F. Rainaud, L. S. Mastella, and M. Abel. Knowledge related challenges for efficient data fusion. In *Proceedings SEG/AAPG/SPE Joint Workshop in Data Fusion: Combining Geological, Geophysical and Engineering Data*, Vancouver, BC, 2007.

M. Perrin, P. Durville, S. Grataloup, L. Mastella, S. Lions, O. Morel, and J.-F. Rainaud. Knowledge issues for automatic identification of co2 storage sites by means of semantic web technology. In *EAGE CO2 Geological Storage Workshop Workshop on CO2 Sequestration*, Budapest Hungary, 28-29 September 2008. EarthDoc. URL: `http://www.earthdoc.org/detail.php?pubid=15267`.

J.-F. Rainaud, L. Mastella, P. Durville, Y. Ait-Ameur, M. Perrin, S. Grataloup, and O. Morel. Two use cases involving semantic web earth science ontologies for reservoir modeling and characterization. In *W3C Workshop on Semantic Web in Oil & Gas Industry*, Houston, US, 9-10 December 2008. URL: `http://www.w3.org/2008/11/ogws-agenda.html`.

References

S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *2nd International Workshop on the Semantic Web*, 2001.

J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

H. A. Babaie, J. S. Oldow, A. Babaei, H. G. A. Lallemant, and A. J. Watkinson. Designing a modular architecture for the structural geology ontology. *SPECIAL PAPERS-GEOLOGICAL SOCIETY OF AMERICA*, 397:269, 2006.

D. Beckett and T. Berners-Lee. Turtle - terse rdf triple language - w3c team submission 14 january 2008, 14 January 2008 2008. URL: `http://www.w3.org/TeamSubmission/turtle/`.

L. Bellatreche, G. Pierra, D. N. Xuan, D. Hondjack, and Y. A. Ameur. An a priori approach for automatic integration of heterogeneous and autonomous databases. In *Lecture Notes in Computer Science - Database and Expert Systems Applications*, volume 3180/2004, pages 475–485. Springer, Verlag, 2004.

L. Bellatreche, N. Dung, G. Pierra, and D. Hondjack. Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. *Computers in Industry*, 57(8-9):711–724, 2006.

M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual reasoning distilled. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(3):279–305, 2000.

S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *ACM Sigmod Record*, 28(1):54–59, 1999.

T. Berners-Lee, Mit/Lcs, R. Fielding, U. C. Irvine, L. Masinter, and X. Corporation. Uniform resource identifiers (uri): Generic syntax. *The Internet Society*, RFC 2396, 1998.

T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific american*, 284(5):28–37, 2001. URL: http://www.w3.org/2001/sw/.

D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An annotation management system for relational databases. *The VLDB Journal The International Journal on Very Large Data Bases*, 14(4):373–396, 2005.

L. Bi, Z. Jiao, and S. Fan. Ontology-based information integration framework for mechatronics system multi-disciplinary design. In *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008*, pages 831–836, 2008.

S. Bloehdorn, K. Petridis, C. Saathoff, N. Simou, V. Tzouvaras, Y. Avrithis, S. Handschuh, I. Kompatsiaris, S. Staab, and M. Strintzis. Semantic annotation of images and videos for multimedia analysis. In *The Semantic Web: Research and Applications*, volume 3532/2005 of *Lecture Notes in Computer Science*, pages 592–607. Springer, 2005.

R. Blumberg and S. Atre. The problem with unstructured data. *DM Review*, 13:42–49, 2003. URL: http://www.dmreview.com/issues/20030201/6287-1.html.

L. H. Boff and M. Abel. Autodesenvolvimento e competências: O caso do trabalhador de conhecimento como especialista. In R. Ruas, C. S. Antonello, and L. H. Boff, editors, *Aprendizagem Organizacional e Competências*, pages 70–86. Bookman, Porto Alegre, 2005.

P. Bouquet, F. Giunchiglia, F. Harmelen, L. Serafini, and H. Stuckenschmidt. C-owl: Contextualizing ontologies. In *The SemanticWeb - ISWC 2003*, Lecture Notes in Computer Science, pages 164–179. Springer Verlag, 2003/// 2003. URL: http://www.springerlink.com/content/2bkbjnxx5r0a897w.

B. Braunschweig and J.-F. Rainaud. Semantic web applications for the petroleum industry. In *2nd European Semantic Web Conference - ESWC 2005*, Heraklion, Greece, 2005.

D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema - w3c recommendation 10 february 2004, 10 February 2004 2004. URL: http://www.w3.org/TR/rdf-schema/.

D. Brickley and L. Miller. The friend of a friend (foaf) project, 2000. URL: http://www.foaf-project.org/.

S. Brockmans, R. Volz, A. Eberhart, and P. Loffler. Visual modeling of owl dl ontologies using uml. In *ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2004.

J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Lecture Notes in Computer Science - In The Semantic Web - ISWC 2002*, volume 2342, pages 54–68. Springer, 2002.

A. Brush, D. Bargeron, A. Gupta, and J. Cadiz. Robust annotation positioning in digital documents. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 285–292. ACM New York, NY, USA, 2001.

P. Buneman, R. Bose, and D. Ecklund. Annotation in scientific data: A scoping report. Technical report, Technical Report. University of Edinburgh, 2005, Edinburgh, May 1, 2005 2005.

J. Bézivin. On the unification power of models. *SoSyM - Software and Systems Modeling*, 4(2):171–188, 2005.

Y. Callec, D. Janjou, T. Baudin, C. Luquet, J. M. Pellé, and P. Laville. Echelle des temps geologiques. BRGM, 2006.

D. Calvanese, G. Giacomo, and M. Lenzerini. Ontology of integration and integration of ontologies. In *Proceedings of the 2001 Description Logic Workshop (DL 2001)*, pages 10–19, 2001.

M. Carrara and N. Guarino. Formal ontology and conceptual analysis: a structured bibliography. Technical report, Dept. of Philosophy, University of Padova and LADSEB-CNR, Padova, March 1999 1999. URL: `http://old.ulstu.ru/people/SOSNIN/umk/Basis_of_Artificial_Intelligence/mirrors/www.ladseb.pd.cnr.it/infor/ontology/Papers/Ontobiblio/Ontobiblio.doc`.

J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM New York, NY, USA, 2005.

S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

F. Chum. Use case: Ontology-driven information integration and delivery: A survey of semantic web technology in the oil and gas industry. W3C Semantic Web Use Cases and Case Studies, 2007/06/04 2007. URL: `http://www.w3.org/2001/sw/sweo/public/UseCases/Chevron/`.

D. Core. Dublin core metadata initiative, 2000. URL: `http://dublincore.org/`.

L. Cosentino. *Integrated Reservoir Studies*. Institut Français du Pétrole Publications. Gulf Publishing Company, Paris, editions technip edition, 2001.

S. Cox and S. Richard. A formal model for the geologic time scale and global stratotype section and point, compatible with geospatial information transfer standards. *Geosphere*, 1(3):119–137, 2005. URL: `http://geosphere.geoscienceworld.org/cgi/content/abstract/1/3/119`.

M. Cutkosky, R. Engelmore, R. Fikes, M. Genesereth, T. Gruber, W. Mark, J. Tenenbaum, and J. Weber. Pact: An experiment in integrating concurrent engineering systems. *IEEE Computer*, 26(1):28–37, 1993.

K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, 2003.

I. Davies, P. Green, S. Milton, and M. Rosemann. Using meta models for the comparison of ontologies. In *EMMSAD 2003*, 2003.

J. Davis and D. Huttenlocher. Shared annotation for cooperative learning. In *The first international conference on Computer support for collaborative learning table of contents*, pages 84–88. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1995.

L. F. De Ros, M. Abel, L. Mastella, K. Goldberg, F. Victoreti, and E. Castro. Advanced acquisition and management of petrographic information from reservoir rocks using petroledge system. In *AAPG Annual Convention and Exhibition*, Long Beach, CA, 2007.

H. Dehainsala. *Explicitation de la sémantique dans les base de données : Base de données à base ontologique et le modèle OntoDB*. PhD thesis, ENSMA et Université de Poitiers, Poitiers, May 2009 2007.

H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA' 07)*. Springer, 2007.

C. Diamantini and N. Boudjlida. About semantic enrichment of strategic data models as part of enterprise models. In *Business Process Management Workshops*, volume 4103/2006 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.

R. Dieng-Kuntz and N. Matta. *Knowledge Management and Organizational Memories*. Kluwer Academic Pub, 2002.

A. Doan and A. Halevy. Semantic-integration research in the database community. *AI magazine*, 26(1): 83–94, 2005.

Energistics. Witsml - the wellsite information transfer standard markup language, 2007. URL: `http://www.witsml.org`.

J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007.

J. R. Fanchi. *Shared Earth Modeling: Methodologies for Integrated Reservoir Simulations*. Gulf Professional Publishing, 2002.

C. Fankam. Ontodb2: support of multiple ontology models within ontology based database. In *Proceedings of the 2008 EDBT Ph.D. workshop*, volume 326, pages 21–27, Nantes, France, 2008. ACM.

C. Fankam, S. Jean, and G. Pierra. Numeric reasoning in the semantic web. In *ESWC First International Workshop on Semantic Metadata Management and Applications (SeMMA)*, volume 346, pages 84–103. CEUR Workshop Proceedings, 2008. URL: `http://ceur-ws.org/Vol-346/7.pdf`.

J. Farrell and H. Lausen. Semantic annotations for wsdl - w3c recommendation 28 august 2007, 2007. URL: `http://www.w3.org/TR/sawsdl/`.

R. Fikes and T. Kehler. The role of frame-based representation in reasoning. *ACM*, 1985.

R. Fikes, P. Hayes, and I. Horrocks. Owl-ql: a language for deductive query answering on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):19–29, 2004.

R. Fjellheim and D. Norheim. Aksio - an application of semantic web technology for knowledge management in the petroleum industry. In *4th International Semantic Web Conference - ISWC 2005*, Galway, Ireland, 2005.

A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with dolce. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, volume 2473 of *Lecture notes in computer science*, pages 223–233. Springer, 2002.

D. Gasevic, D. Djuric, and V. Devedzic. Bridging mda and owl ontologies. *Journal of Web Engineering*, 4(2):119–134, 2005.

F. Geerts, A. Kementsietsidis, and D. Milano. imondrian: A visual tool to annotate and query scientific databases. In *Advances in Database Technology - EDBT 2006*, volume 3896 of *Lecture Notes in Computer Science*, pages 1168–1171. Springer, 2006.

F. Giunchiglia and P. Shvaiko. Semantic matching. *Knowledge Engineering Review*, 18(3):265–280, 2003.

F. Gradstein, J. Ogg, and A. G. Smith, editors. *A Geologic Time Scale 2004*. Cambridge University Press, 2004.

P. Grenon, B. Smith, and L. Goldberg. Biodynamic ontology: applying bfo in the biomedical domain. *Studies in health technology and informatics*, pages 20–38, 2004.

P. Grosso, E. Maler, J. Marsh, and N. Walsh. Xpointer framework - w3c recommendation 25 march 2003, 2003. URL: `http://www.w3.org/TR/xptr-framework/`.

T. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2): 199–219, 1993.

T. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human and Computer Studies*, 43(5/6):907–928, 1995. URL: `http://citeseer.ist.psu.edu/gruber93toward.html`.

T. Gruber. Ontology (to appear in the encyclopedia of database systems). In L. L. Özsu and M. Tamer, editors, *Encyclopedia of Database Systems*, volume 23. Springer-Verlag, 2008.

T. Gruber, J. Tenenbaum, and J. Weber. Toward a knowledge medium for collaborative product development. *Artificial Intelligence in Design*, 1992.

N. Guarino. *Formal ontology in information systems*. IOS Press, 1998.

G. Guizzardi, G. Wagner, and H. Herre. On the foundations of uml as an ontology representation language. In *Engineering Knowledge in the Age of the SemanticWeb*, volume 3257/2004 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2004.

A. Gómez-Pérez, A. Moreno, J. Pazos, and A. Sierra-Alonso. Knowledge maps: An essential technique for conceptualization. *Data & Knowledge Engineering*, 33(2):169–190, 2000.

S. Handschuh, S. Staab, and A. Maedche. Cream: creating relational metadata with a component-based, ontology-driven annotation framework. In *Proceedings of the 1st international conference on Knowledge capture*, pages 76–83. ACM New York, NY, USA, 2001.

D. L. M. Harmelen and F. van. Owl web ontology language overview. Technical report, W3C Recommendation, 10 February 2004, 10 February 2004 2004. URL: `http://www.w3.org/TR/owl-features/`.

S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pages 1–15, 2003.

S. Harris and N. Shadbolt. Sparql query processing with conventional relational database systems. In *Web Information Systems Engineering - WISE 2005 Workshops*, volume 3807 of *Lecture notes in computer science*, pages 235–244. Springer, 2005.

J. Heflin and J. Hendler. Searching the web with shoe. In *AAAI Workshop on Artificial Intelligence for Web Search (WS-00-01)*, pages 35–40. AAAI Press, Menlo Park, CA, 2000. URL: `http://www.cs.umd.edu/projects/plus/SHOE/pubs/aiweb2000.ps`.

I. Herman and S. Stephens. Semantic web education and outreach interest group: Case studies and use cases, 2007. URL: `http://www.w3.org/2001/sw/sweo/public/UseCases/`.

V. Hubka. *Theory of Technical Systems*. Berlin/Heidelberg: Springer-Verlag, 1984.

S. Jean. *OntoQL, un langage d'exploitation des bases de données à base ontologique.* PhD thesis, ENSMA, Poitiers, Octobre 2007 2007.

S. Jean, G. Pierra, and Y. Ait-Ameur. Ontoql: an exploitation language for obdbs. In *VLDB PhD Workshop*, volume 29, 2005.

S. Jean, Y. Ait-Ameur, and G. Pierra. Querying ontology based databases. the ontoql proposal. In *18th International Conference on Software Egineering and Knowledge Engineering (SEKE-2006)*, pages 166–171, 2006a.

S. Jean, A.-A. Yamine, and G. Pierra. Querying ontology based database using ontoql (an ontology query language). In R. Meersman, Z. Tari, and et al., editors, *Proceedings of On the Move to Meaningful Internet Systems (ODBASE)*, volume 4275 of *Lecture Notes in Computer Science*, pages 704–721. Springer, 2006b.

S. Jean, Y. Ait-Ameur, and G. Pierra. An object-oriented based algebra for ontologies and their instances. In S. B. Heidelberg, editor, *Advances in Databases and Information Systems*, volume 4690/2007 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2007.

J. Kahan, M. R. Koivunen, E. Prud'Hommeaux, and R. R. Swick. Annotea: an open rdf infrastructure for shared web annotations. *Computer Networks*, 39(5):589–608, 2002.

Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowledge Engineer Review*, 18(1):1–31, january 2003 2003. URL: http://eprints.ecs.soton.ac.uk/10519/01/ker02-ontomap.pdf&e=10342.

A. Kalyanpur, J. Hendler, B. Parsia, and J. Golbeck. Smore-semantic markup, ontology, and rdf editor. In *Proceedings of 3rd International Semantic Web Conference (ISWC-2004), Japan (Poster)*, 2004.

A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, 2004.

A. Kiryakov, D. Ognyanov, and D. Manov. Owlim - a pragmatic semantic repository for owl. *Lecture Notes in Computer Science*, 3807:182, 2005.

A. Kleppe, J. Warmer, and W. Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley, 2003.

G. Klyne and J. J. Carroll. Resource description framework (rdf). Technical report, W3C Recommendation, 10 February 2004, 2004. URL: http://www.w3.org/TR/rdf-concepts/.

C. Kobryn. Uml 2001: a standardization odyssey. *Communications of the ACM*, 42(10), 1999.

K. Lin and B. Ludaescher. Geon: Ontology-enabled map integration. In *2004 ESRI International User Conference - ESRI Professional Papers*, Redlands, California, 2004. URL: http://gis.esri.com/library/userconf/proc04/abstracts/a1796.html.

Y. Lin, D. Strasunskas, S. Hakkarainen, J. Krogstie, and A. Solvberg. Semantic annotation framework to manage semantic heterogeneity of process models. In S. B. Heidelberg, editor, *Proceedings of the 18th Conference on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 433–446. Springer, 2006.

B. Ludäscher, K. Lin, S. Bowers, E. Jaeger-Frank, B. Brodaric, and C. Baru. Managing scientific data: From data integration to scientific workflows. *GeoInformatics, Data to Knowledge*, pages 109–129, 2006.

A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, volume 2473 of *Lecture Notes in Computer Science*, pages 69–75. Springer, 2002.

Z. Malik, A. Rezgui, and A. K. Sinha. Ontologic integration of geoscience data on the semantic web. In *Geoinformatics 2007 Conference*, San Diego, California, 2007.

C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An introduction to the syntax and content of cyc. In *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pages 44–49, 2006.

T. Mens and P. Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

B. Motik. On the properties of metamodeling in owl. *Journal of Logic and Computation*, 17(4):617, 2007.

B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between owl and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):74–89, 2009.

R. D. Murphy, L. Mark, S. Rugaber, V. Balaji, J. Chastang, L. Cinquini, C. DeLuca, D. Middleton, and Sylvia. Earth system curator: metadata infrastructure for climate modeling. *Earth Sci Inform*, 1(3-4): 131–149, 2008.

N. A. G. M. D. M. S. C. Nadm. Nadm conceptual model 1.0 - a conceptual model for geologic map information (us geological survey open-file report). Technical Report U.S. Geological Survey Open-File Report 2004-1334, North American Geologic Map Data Model, 2004. URL: `http://pubs.usgs.gov/of/2004/1334`.

W. Nejdl, M. Wolpers, and C. Capelle. The rdf schema specification revisited. In *Workshop Modellierung 2000*, 2000.

I. Nonaka, H. Takeuchi, and H. Takeuchi. *The knowledge-creating company: how Japanese companies create the dynamics of innovation*. Oxford University Press, New York, 1995. ISBN 0195092694.

D. Norheim and R. Fjellheim. Aksio - active knowledge management in the petroleum industry. In *ESWC2006 - 3rd European Semantic Web Conference*, Budova, Montenegro, 2006.

N. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University - Knowledge Systems Laboratory, Stanford, CA, 2001.

N. F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record, Special Issue on Semantic Integration*, 33(4), December 2004 2004. URL: `http://citeseer.ist.psu.edu/wache01ontologybased.html`.

G. Olsen, M. Cutkosky, J. Tenenbaum, and T. Gruber. Collaborative engineering based on knowledge sharing agreements. *Concurrent Engineering*, 3(2):145–159, 1995.

S. Omdal. The integrated information platform (iip) for resevoir and subsea production systems. Technical report, Norwegian Oil Industry Association (OLF) 2005, POSC IntOPS SIG Regional Meeting, May 2006 2006.

OMG. Meta object facility (mof) core, January 2006 2006. URL: `http://www.omg.org/spec/MOF/index.htm`.

OMG. Unified modeling language (uml), February 2009 2008. URL: `http://www.omg.org/spec/UML/index.htm`.

OMG. Xml metadata interchange (xmi), December 2007 2009. URL: `http://www.omg.org/spec/XMI/index.htm`.

OpenSpirit. Openspirit corporation, 2000. URL: `http://www.openspirit.com/`.

J. Pan and I. Horrocks. Metamodeling architecture of web ontology languages. In I. C. e. al., editor, *The emerging semantic web: selected papers from the first Semantic Web Working Symposium*, pages 21–45. IOS Press, 2002.

J. Pan and I. Horrocks. Rdfs(fa) and rdf mt: Two semantics for rdfs. In *Proc. ISWC 2003*, Lecture notes in computer science, pages 30–46. Springer, 2003.

M. Park, J. Lee, C. Lee, J. Lin, O. Serres, and C. Chung. An efficient and scalable management of ontology. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07) - LNCS*, volume 4443, pages 975–980. Springer, 2007.

A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web*, pages 553–562. ACM New York, NY, USA, 2004.

M. Perrin. Geological consistency: an opportunity for safe surface assembling and quick model exploration. In *GOCAD ENSG Conference - 3D Modeling of Natural Objects : A Challenge for the 2000's*, volume 3, pages 4–5, Nancy, France, june 1998 1998.

M. Perrin, B. Zhu, S. Schneider, and J.-F. Rainaud. Ontology-driven applications for geological modeling. In B. Braunschweig, M. Alvarado, R. Banares-Alcantara, and L. Sheremetov, editors, *IJCAI Workshop on Intelligent Computing in Petroleum Industry*, pages 76–81, Acapulco, Mexico, 2003. Mexican Petroleum Institute.

M. Perrin, B. Zhu, J.-F. Rainaud, and S. Schneider. Knowledge-driven applications for geological modeling. *Journal of Petroleum Science and Engineering*, 47(1-2):89–104, 2005/5/15 2005. URL: `http://www.sciencedirect.com/science/article/B6VDW-4FSCVBF-1/2/044aa925f698fa145cb1d44e1accd254`.

G. Pierra. Context-explication in conceptual ontologies: The plib approach. In *Concurrent Engineering (CE 2003)*, pages 243–254, Madeira, Portugal, 2003/// 2003.

G. Pierra. The plib ontology-based approach to data integration. In *Proceedings of the 18th IFIP World Computer Congress (WCC'2004)*, Toulouse, France, August 2004 2004. Springer.

Posc. Epicentre specification v. 3.0. Petrotechnical Open Software Corporation, 2001. URL: `http://www.posc.org/Specifications/Epicentre_V30/index.html`.

V. Quint and I. Vatton. An introduction to amaya. *World Wide Web Journal*, 2(2):39–46, 1997.

E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

J. F. Rainaud. A short history of the last 15 year's quest for it interoperability in the petroleum e&p industry. *Oil & Gas Science and Technology - Special issue - Software Interoperability for Petroleum Applications*, 60(4):597–605, 2005. URL: `http://www.ifp.fr/information-publications/publications/a-short-history-of-the-last-15-year-s-quest-for-it-interoperability-in`.

J.-F. Rainaud, M. Perrin, and Y. Bertrand. Innovative knowledge-driven approach for shared earth model buidling. In *67th EAGE Conference & Exhibition incorporating SPE EUROPEC*, Madrid, Spain, June 13, 2005 2005.

R. G. Raskin and M. J. Pan. Knowledge representation in the semantic web for earth and environmental terminology(sweet). *Computers & geosciences*, 31(9):1119–1125, 2005.

S. M. Richard. Geoscience concept models. In A. K. Sinha, editor, *Geoinformatics: Data to Knowledge*, pages 81–107. Geological Society of America Special Paper 397, 2006.

J. Rothenberg. The nature of modeling. In L. E. Widman, K. A. Loparo, and N. R. Nielsen, editors, *Artificial Intelligence, Simulation, and Modeling*. New York, John Wiley and Sons, Inc, 1989.

N. Sandsmark and S. Mehta. Integrated information platform for reservoir and subsea production systems. *Proceedings of the 13 thProduct Data Technology Europe Symposium (PDT 2004), October, Stockholm*, 2004.

D. A. Schenck and P. R. Wilson. *Information modeling: the EXPRESS way*. Oxford University Press, USA, 1994.

D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

R. Schroeter, J. Hunter, and D. Kosovic. Vannotea - a collaborative video indexing, annotation and discussion system for broadband networks. In *Knowledge Markup and Semantic Annotation Workshop, K-CAP 2003*, Sanibel, Florida, 2003. URL: `http://ezproxy.library.uq.edu.au/login?url=http://www.itee.uq.edu.au/~eresearch/papers/2003/schroeter-kcap03.pdf`.

A. Seaborne and E. Prud'hommeaux. Sparql query language for rdf. Technical report, W3C Recommendation, 15 January 2008, 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

C. Sperberg-McQueen. Xml and semi-structured data. *ACM Queue*, 3(8):34–41, December 8, 2005 2005.

P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus ontology engineering. *ACM SIGMOD Record*, 31(4):12–17, 2002.

M. Stefik. The next knowledge medium. *AI magazine*, 7(1):34–46, 1986.

D. Stenmark, V. Technol, and S. Gothenburg. Information vs. knowledge: The role of intranets in knowledge management. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences - HICSS*, pages 928–937, 2002.

G. Toye, M. Cutkosky, L. Leifer, J. Tenenbaum, and J. Glicksman. Share: a methodology and environment for collaborative productiondevelopment. In *Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 33–47, Morgantown, WV, USA, 1993.

A. Tripathi and H. Babaie. Developing a modular hydrogeology ontology by extending the sweet upper-level ontologies. *Computers and Geosciences*, 34(9):1022–1033, 2008.

T. Tudorache. *Ontologies in Engineering: Modeling, Consistency and Use Cases*. VDM Verlag, 2008.

V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):14–28, 2006.

M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996. URL: `http://citeseer.ist.psu.edu/uschold96ontologie.html`.

M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Record*, 33(4):58–64, 2004. URL: `http://www.sigmod.org/sigmod/record/issues/0412/12.uschold-9.pdf`.

M. Uschold, M. King, A. I. A. Institute, and U. o. Edinburgh. *Towards a Methodology for Building Ontologies*. Artificial Intelligence Applications Institute, University of Edinburgh, 1995.

P. Verney. *Modélisation géologique 3D : formulation de la géologie du modèle à partir de données de sismique 3D par une méthode semi-automatique supervisée*. PhD thesis, Ecole des Mines de Paris, 17 September 2009 2009.

D. Vrandecic, V. J., P. Haase, D. T. Tran, and P. Cimiano. A metamodel for annotations of ontology elements in owl dl. In *2nd Workshop on Ontologies and Meta-Modeling*, 2006.

H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information: A survey of existing approaches. In H. Stuckenschmidt, editor, *Proceedings of the IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108 –117, Seattle, WA, 2001.

H. Wang, N. Noy, A. Rector, M. Musen, T. Redmond, D. Rubin, S. Tu, T. Tudorache, N. Drummond, and M. Horridge. Frames and owl side by side. In *9th International Protégé Conference*, page 4, Stanford, CA, 2006.

C. Welty and D. Ferrucci. What's in an instance? Technical Report Technical Report 94-18, RPI Computer Science Dept, 1994.

K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient rdf storage and retrieval in jena2. In *Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB'03)*, pages 131–150, 2003.

M. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part-whole relations. *Cognitive Science: A Multidisciplinary Journal*, 11(4):417–444, 1987.

D. Xuan, L. Bellatreche, and G. Pierra. Ontology evolution and source autonomy in ontology-based data warehouses. *Entrepôts de Données et l'Analyse en ligne (EDA-06)*, pages 55–76, 2006.

S. Zdonik and G. Mitchell. Encore: An object-oriented approach to database modelling and querying. *Data Engineering*, 14(2):53–57, 1991.

A. Zhdanova and P. Shvaiko. Community-driven ontology matching. In *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2006.

Q. Zheng, K. Booth, and J. McGrenere. Co-authoring with structured annotations. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 131–140. ACM New York, NY, USA, 2006.

Z. Zuo and M. Zhou. A distributed description logic approach to semantic annotation. In *Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Application and Technologies, PDCAT*, pages 219–221, 2003.

Exploitation sémantique des modèles d'ingénierie : application à la modélisation de réservoirs de pétrole.

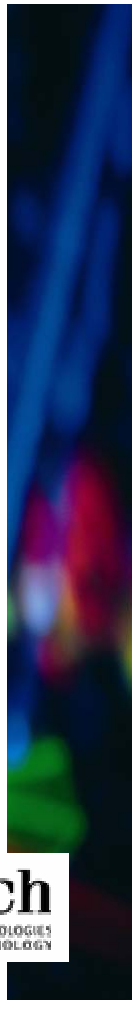
RESUME: Ce travail propose des solutions innovantes en vue de *l'exploitation des modèles d'ingénierie hétérogènes*. Il prend pour exemple le domaine de la *prospection pétrolière*. Les stratégies de prospection sont élaborées à partir de représentations tridimensionnelles du sous-sol appelées *modèles géologiques*. Ceux-ci reposent sur un grand nombre de données hétérogènes générées au fur et à mesure de la conduite de l'exploration par des activités telles que la prospection séismique, les forages, l'interprétation des logs de puits. A fin d'optimisation, les utilisateurs finaux souhaitent, pouvoir retrouver et réutiliser à tout moment les données et les interprétations attachés aux différents modèles successivement générés. Les approches d' *intégration des connaissances* susceptibles d'être mises en œuvre pour résoudre ce défi, doivent être dissociées aussi bien des sources et des formats de données que des outils logiciels en constante évolution. Pour cela, nous proposons d'utiliser *l'annotation sémantique*, technique courante du Web sémantique permettant d'associer la connaissance à des ressources au moyen d' « étiquettes sémantiques ». La sémantique ainsi explicitée est définie par un certain nombre d' *ontologies de domaine*, qui, selon la définition classique, correspondent à autant « de spécifications formelles de la conceptualisation » des domaines considérés. En vue d'intégrer les modèles d'ingénierie considérés, nous proposons une architecture, qui permet de relier des concepts appartenant respectivement à des ontologies locales et à une *ontologie globale*. Les utilisateurs peuvent ainsi avoir une vision globale, intégrée et partagée de chacun des domaines impliqués dans chaîne de modélisation géologique. Un prototype a été développé qui concerne la première étape de la chaîne de modélisation (*interprétation séismique*). Les expérimentations réalisées prouvent que, grâce à l'approche proposée, les experts peuvent, en utilisant le vocabulaire de leur domaine d'expertise, formuler des questions et obtenir des réponses appropriées.

MOTS CLES : Intégration et interopérabilité de modèles, Ontologies, Base de Données à Base Ontologique, Méta-modélisation, Annotation sémantique, Modélisation de réservoir pétrolier

Semantic exploitation of engineering models: application to petroleum reservoir models.

ABSTRACT: This work intends to propose innovative solutions for the *exploitation of heterogeneous models in engineering domains*. It pays a special attention to a case study related to one specific engineering domain: petroleum exploration . Experts deal with many petroleum exploration issues by building and exploiting three-dimensional representations of underground (called *earth models*). These models rest on a large amount of heterogeneous data generated every day by several different exploration activities such as seismic surveys, well drilling, well log interpretation and many others. Considering this, end-users wish to be able to retrieve and re-use at any moment information related to data and interpretations in the various fields of expertise considered along the earth modeling chain. Integration approaches for engineering domains needs to be dissociated from data sources, formats and software tools that are constantly evolving. Our solution is based on *semantic annotation*, a current Web Semantic technique for adding knowledge to resources by means of semantic tags. The "semantics" attached by means of some annotation is defined by ontologies, corresponding to "formal specifications of some domain conceptualization". In order to complete engineering model exploitation, it is necessary to provide *model integration*. Correspondence between models in the *ontology level* is made possible thanks to semantic annotation. An architecture, which maps concepts from *local ontologies* to some *global ontology,* then ensures that users can have an integrated and shared global view of each specific domain involved in the engineering process. A prototype was implemented considering the *seismic interpretation activity*, which corresponds to the first step of the earth modeling workflow. The performed experiments show that, thanks to our solution, experts can formulate queries and retrieve relevant answers using their knowledge-level vocabulary.

KEYWORDS : Model integration and interoperability, Ontologies, Ontology-based databases, Meta-modeling, Semantic annotation, Petroleum reservoir modeling.



MINES ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY