Dynamic situation monitoring and Context-Aware BI recommendations

Raphaël Thollot

▶ **To cite this version:**

Raphaël Thollot. Dynamic situation monitoring and Context-Aware BI recommendations. Other. Ecole Centrale Paris, 2012. English. NNT : 2012ECAP0020 . tel-00718917

HAL Id: tel-00718917

https://theses.hal.science/tel-00718917

Submitted on 18 Jul 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ecole Centrale Paris

DOCTORAL SCHOOL
SCIENCES FOR THE ENGINEER

PhD Thesis

to obtain the title of

Doctor of Ecole Centrale Paris

Defended by
Raphaël Thollot

Dynamic Situation Monitoring and Context-Aware BI Recommendations

prepared at Ecole Centrale Paris, MAS Laboratory, and SAP Research, Business Intelligence Practice

defended on 2012, April 3rd

Jury:

Advisor:	Marie-Aude Aufaure	-	Professor at Ecole Centrale Paris
President:	Jacky Akoka	-	Professor at CNAM
Reviewers:	Franck Ravat	-	Professor at Université Toulouse 1 Capitole
	Stefano Rizzi	-	Professor at University of Bologna
Examinators:	Yannick Cras	-	PhD, chief architect at SAP
	Patrick Marcel	-	Associate professor at Université François Rabelais Tours

Abstract

The amount of information generated and maintained by information systems and their users leads to the increasingly important concern of information overload. Personalized systems have thus emerged to help provide more relevant information and services to the user. In particular, recommender systems appeared in the mid 1990's and have since then generated a growing interest in both industry and academia. Besides, context-aware systems have been developed to model, capture and interpret information about the user's situation, generally in dynamic and heterogeneous environments.

Decision support systems like Business Intelligence (BI) platforms also face usability challenges as the amount of information available to knowledge workers grows. Remarkably, we observe that only a small part of personalization and recommendation techniques have been used in the context of data warehouses and analysis tools. Therefore, our work aims at exploring synergies of recommender systems and context-aware systems to develop personalization and recommendation scenarios suited in a BI environment.

In response to this, we develop in our work an open and modular situation management platform using a graph-based situation model. Besides, dynamic aspects are crucial to deal with context data which is inherently time-dependent. We thus define two types of active components to enable dynamic maintenance of situation graphs, activation rules and operators. In response to events which can describe users' interactions, activation rules defined using the event-condition-action framework are evaluated thanks to queries on underlying graphs, to eventually trigger appropriate operators.

These platform and framework allow us to develop and support various recommendation and personalization scenarios. Importantly, we design a re-usable personalized query expansion component, using semantics of multi-dimensional models and usage statistics from repositories of BI documents like reports or dashboards. This component is an important part of another experimentation we realized, Text-To-Query. This system dynamically generates multi-dimensional queries to illustrate a text and support the knowledge worker in the analysis or enrichment of documents she is manipulating. Besides, we also illustrate the integration and usage of our graph repository and situation management frameworks in an open and extensible federated search project, to provide background knowledge management and personalization.

Résumé

Le volume des données créées et gérées par les systèmes d'information et leurs utilisateurs augmente régulièrement, conduisant à la problématique croissante de la surinformation. Pour répondre aux défis posés par l'accès à

l'information dans de grands volumes de données, les systèmes personnalisés visent à proposer des données et des services plus adaptés à l'utilisateur. Les systèmes de recommandation (SR), apparus au milieu des années 1990, sont un cas particulier de ces systèmes personnalisés. Depuis, les SR ont suscité un intérêt croissant tant dans la communauté académique que du cté des industriels. Par ailleurs, des systèmes contextuels ont été développés dans le but de modéliser, capturer et interpréter l'information relative à l'environnement de l'utilisateur. Systèmes contextuels et SR partagent donc un même but, celui de fournir les données et les services les plus adaptés à la situation de l'utilisateur, généralement dans un environnement dynamique et hétérogène.

Les systèmes d'aide à la décision tels que les outils de Business Intelligence (BI) présentent eux aussi des difficultés relatives à leur utilisation, en particulier du fait de la quantité et de la complexité des données accessibles aux utilisateurs. Il est cependant notable que seules quelques rares techniques héritées de systèmes de recommandation ont à ce jour été appliquées dans le domaine des entrepts de données et des outils d'analyse. Notre travail consiste donc à explorer des synergies pouvant résulter de la combinaison de SR et de systèmes contextuels, à des fins de personnalisation dynamique dans les outils de BI.

En réponse à ces challenges, nous développons dans notre travail une plateforme ouverte et modulaire permettant la gestion des situations ou contextes utilisateurs. Cette plateforme repose principalement sur un modèle de situation à base de graphes. Par ailleurs, la dynamique des interactions implique une dépendance inhérente au temps des informations contextuelles. Nous définissons donc deux types de composants actifs, règles d'activation et opérateurs, responsables de la gestion de l'évolution des graphes de connaissances. Les règles sont construites selon le modèle évènement-condition-action (ECA) et sont évaluées en réponse aux divers évènements reus par la plateforme. L'évaluation d'une règle consiste à valider ses conditions grâce à l'exécution d'un certain nombre de requêtes sur les graphes de données, afin de déclencher l'exécution d'opérateurs appropriés.

La plateforme modulaire proposée avec un framework de développement nous a permis de démontrer divers scénarios de personnalisation et de recommandations. Nous présentons en particulier un composant personnalisé d'expansion de requêtes multidimensionnelles. Ce composant exploite d'une part la sémantique des modèles multidimensionnels et d'autre part des statistiques d'usage dérivées de collections de rapports et tableaux de bords existants. Le composant d'expansion de requêtes est utilisé par exemple dans Text-To-Query (T2Q), un SR suggérant des requêtes et visualisations adaptées, générées dynamiquement afin d'illustrer un document texte (non structuré). T2Q a pour objectif d'aider l'utilisateur à analyser et enrichir les documents sur lesquels il travaille. Enfin, nous décrivons l'intégration de notre plateforme dans un projet de recherche fédérée d'information. La

plateforme est en particulier utilisée comme support pour la gestion de la connaissance relative aux utilisateurs. Celle-ci nous permet d'élaborer une stratégie de personnalisation de la recherche via la définition de préférences appliquées aux sources d'information.

Acknowledgements

I first thank SAP Research for supporting this work and providing me with a rich industrial environment to experiment. More personally, I thank the many people who contributed to make this PhD a great experience for me.

I wish to express my deepest gratitude to my advisors at Ecole Centrale Paris and SAP.

I would really like to thank Marie-Aude Aufaure for her gentle and comprehensive guidance throughout these years. Her continuous support and academic advices on research directions were precious to go forward with my work.

I am also grateful to Yannick Cras for his kind management and the many inspiring discussions we had and, I hope, we will keep having. It was a great opportunity for me to benefit from the computer science expertise he delivers with passion.

In the SAP-sphere, I had the chance to work with various people that all contributed to make work days instructive and enjoyable. I would like to thank Chahab Nastar for taking me on board in the BI practice, for the open discussions we had and, in particular, for shedding some light on the *paradox of choice*. Thank you to the cross-prototyping and BI practice teams, to the whole search task force of the Business Web project and to coffee-break-colleagues.

I am also thankful to the whole BI team of the MAS laboratory, in Ecole Centrale Paris, for its good atmoshpere and those numerous lunches at the irreplaceable Chinese restaurant.

I am particularly thankful to Franck Ravat and Stefano Rizzi for accepting to review my PhD thesis. I would also like to thank Yannick Cras and Patrick Marcel who both accepted to be jury examinators. And thank you to Jacky Akoka for the honor he made me by accepting to be president of the jury.

I am lucky enough to have met amazing friends and they have always been a great help. It proved true once again and it is fortunate for my mental health that they were here.

The permanent support of my family was extremely precious to me. Thank you to my parents and sisters to whom I owe so much. I am excited to welcome three recent or soon-to-be nephews.

Last but not least, Anaïs already has more than my gratitude for the many little things, and the big ones, that she brings in our lives. Thank you for being there, yourself, genuinely adorable.

Contents

Chapter I

Introduction

Contents

1 General introduction

The amount of information generated and maintained by information systems and their users leads to the increasingly important concern of information overload. Personalized systems have thus emerged to help provide more relevant information and services to the user. In particular, recommender systems appeared in the mid 1990's and have since then generated a growing interest in both industry and academia. Besides, context-aware systems have been developed to model, capture and interpret information about the user's situation, generally in dynamic and heterogeneous environments.

Decision support systems like Business Intelligence (BI) platforms also face usability challenges as the amount of information available to knowledge workers grows. Remarkably, we observe that only a small part of personalization and recommendation techniques have been used in the context of data warehouses and analysis tools.

Therefore, our work aims at exploring synergies of recommender systems and context-aware systems to develop personalization and recommendation scenarios suited in a BI environment.

2 Context and proposed approach

Our work has been significantly influenced by the industrial environment that supported and fostered it, that is the BI practice of SAP Research.

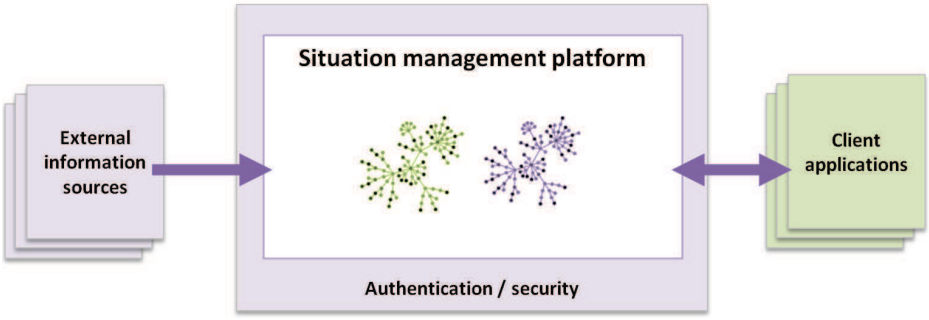

Figure I.1: Overview of the proposed situation management platform which aggregates information from different sources and exposes consolidated situation models.

SAP is a software editor providing important enterprise tools including, for instance, enterprise resource planning (ERP), customer relationship management (CRM) and BI systems.

In this context, we explore capabilities offered by recommender systems and context-aware systems to assist knowledge workers as they interact in varied ways with several communicating applications. Our aim is to propose a shared personalization platform enabling dynamic adaptation across multiple clients or business applications, based on various information systems (or sources).

Dynamic adaptation bases on a context model to capture and interpret users' interactions with their environments. However, context being a broad notion (as will be discussed in Chapter II, Section 3), we consider the user's situation as a filtered and thus more easily interpretable view of such interactions. One of our goals is to ensure that user modeling efforts can be shared between different applications. We thus propose a graph-based situation model, leveraging semantic technologies to ease inter-operability and handle the diversity of available resources. Besides, we define active components for our situation management platform, to deal with dynamic aspects of situation monitoring and more generally graphs maintenance.

If personalization can be observed from a very generic angle, we focus more specifically on BI platforms dedicated to multi-dimensional data analysis. We explore, for instance, the application of our platform to provide personalized assistance in *ad-hoc* query and reporting workflows. To conclude, Figure I.1 illustrates the high level overview of the proposed situation management platform which aggregates various sources and exposes situation models to client applications.

3 Thesis organization

Let us briefly present the organization of this manuscript.

We first review related work in Chapter II. The review is structured so as to reflect the three main fields of research related to our work. First, we introduce recommender systems and the most common techniques developed in this area. Besides, we present more recent work that attempted to introduce contextual parameters into the recommendation process. Then, we describe contributions related to more generic context-aware systems. These systems base on important efforts to generalize the notions of context and situation and aim at being shareable and reusable across different applications. The last area that we discuss concerns data warehouses and multi-dimensional analysis. We present in particular some approaches that were adopted to provide a certain degree of personalization and recommendations in the context of data warehouses.

Next, we introduce in Chapter III the requirements imposed on the situation management platform that we envision. These requirements lead to the definition of our graph-based situation model, meant to enable a high level of expressiveness and deal with the heterogeneity of possible information sources. To ensure data privacy and users' data isolation, our work relies on a graph repository framework that we describe in details. Furthermore, our situation model exposes dimensions that help categorize situation-related information and expose more structured information to consuming applications.

Chapter IV deals with situation dynamics or more precisely mechanics used to maintain graphs in response to events, using operators and activation rules described in the event-condition-action framework. This chapter also describes the architecture of the proposed framework, from the underlying graph repository to dynamic components.

With a focus on data warehouses and multi-dimensional analysis, Chapter V introduces multi-dimensional models and their semantics. On top of these, we exploit usage statistics that can be derived from repositories of existing BI documents. This includes the definition of users' preferences and similarity as well as a measure of co-occurrences between BI entities (measures and dimensions). Eventually, semantics and statistics are combined to propose a personalized component for multi-dimensional query expansion.

To illustrate our generic framework and BI-specific notions previously discussed, Chapter VI describes a certain number of experimentations that were realized. We first introduce Text-To-Query, a system dynamically generating structured analytics related to a text document the user is reading. Besides, we further discuss the use of our framework to enable personalization at different levels, in a federated search project.

To conclude, we briefly summarize our work and contributions in Chapter VII. The discussion includes hints and suggestions for future work related

to our situation management platform and its applications.

Chapter II

Recommender systems, context-awareness and BI recommendations

Contents

1 Introduction

As information systems contain and generate ever increasing amounts of data, users face the tedious task of choosing among many possibly huge sources of information those likely to satisfy their needs. This problem, called *information overload*, led to the active development of personalized

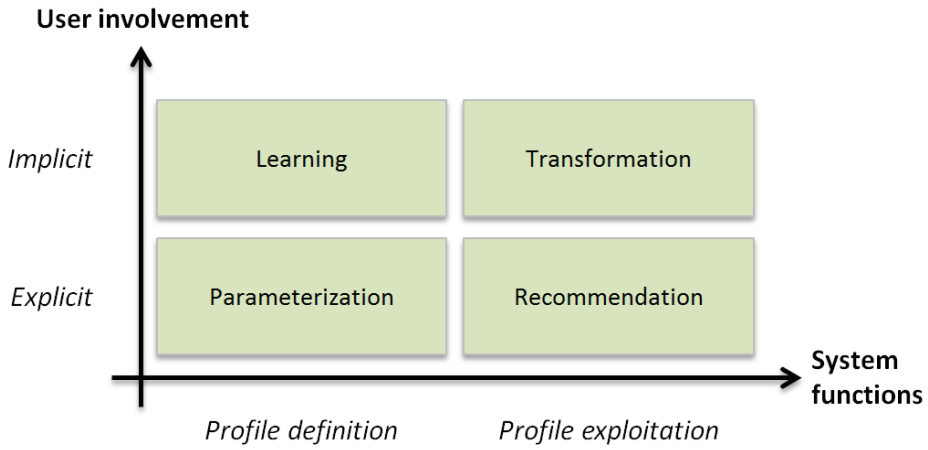

Figure II.1: Classification of personalization techniques proposed by Bentayeb et al. [BBF+09], according to user involvement and system functions.

systems which aim at adapting information and services to the user, at content or design levels. Content-tailoring may be considered to filter or adapt content to present more relevant and adapted information to the user. Besides, different users may have distinct design preferences, e.g., in terms of layout and colors, or more generally theming options.

Bentayeb et al. classify *personalization* or personalized systems along two axis, user involvment and system functions [BBF+09], as depicted in Figure II.1. Personalization fundamentally relies on user models which represent key characteristics of the user like her hobbies, interests, goals, etc. User models are meant to identify the active user and collect various pieces of information about her, ideally so they can be understood and exploited by both humans and machines. Previous work on user modeling led to the development of Generic User Modeling Systems (GUMS), first meant to isolate user modeling efforts from application-specific logic. These systems have been thoroughly reviewed by Kobsa in [Kob07].

A particular approach of personalization has been adopted with *recommender systems* (RS). These systems aim at providing suggestions of different content, somehow related to the user's current interests. RS present a remarkable personalization technique as they are designed to introduce a sense of novelty and serendipity so as to encourage exploration and discovery [HKTR04]. RS can be found in many systems nowadays but their popularity emerged in particular from collaborative or social Web sites.

However, most RS have relied on heavy pre-computations which reduced their ability to dynamically adapt to changing user needs. On the other hand, *context-aware systems* have mainly focused on context modeling and acquisition, that is to leverage knowledge about the user's environment and

dynamic behaviours to provide timely information and services. Research in this area has mainly been driven by mobile or ubiquitous systems, for instance with intelligent environments like a meeting room [CFJ03].

It can be observed that the two research areas of RS and context-aware systems fundamentally share the same goal of providing personalized information and services to end-users. As a result, RS and context-aware systems can be mutually beneficial, leading to present improved and more contextual suggestions [CGF07]. Our work thus aims at exploring the synergies of these different personalization techniques, in particular in the context of large data warehouses and multi-dimensional data analysis.

The rest fo this chapter is structured as follows. Section 2 presents the recommendation problem and the main techniques that were developed to implement RS. Then, Section 3 defines the notions of context and situation to introduce context-aware systems, their architecture and dynamic capabilities. Section 4 introduces multi-dimensional data analysis with data warehouses and different forms of personalization and recommendations. Eventually, we summarize this discussion with a particular focus on connections between these three research areas.

2 Recommender systems

Information systems and their users generate ever increasing amounts of data which leads to the infamous *information overload* problem. RS were born in this context to help users find and discover interesting items in vasts spaces of available resources [MMN02].

2.1 The recommendation problem

RS have grown increasingly popular in both industry and academia to become an important research area. These systems aim at helping the exploration of vast collections of resources by users. In an e-commerce context, suggestions can be used to encourage additional sales, e.g., in the form of sale bundles or related items. The major focus in this field has been the use of ratings assigned by users to items so as to suggest other items of interest and thus encourage exploration.

Given a set of users U and a set of items I, the recommendation problem is commonly formulated as the maximization of a utility function $f : U \times I \mapsto R$, where R is a totally ordered set of recommendations [AT05]. The core problem addressed by a RS is the coverage of the utility function f. Indeed, a user only rates some of the items. Estimating missing ratings then allows to determine and recommend those with the highest scores.

An item $i \in I$ may be represented with a certain number of features or characteristics. Characteristics of a song may be the artist, the year, the album, the genre, etc. Other types of items may obviously be described

using different features, for instance a selection of important keywords in a Web page. Similarly, a user $u \in U$ is commonly described by the means of a profile to enable further personalization.

In the rest of this section, we present the main techniques that were developed to enable personalized recommendations of varied items.

2.2 Recommendation methods

The research area of RS has been very active and many techniques were developed to provide users with recommendations. Major techniques have been thoroughly reviewed by Adomavicius et al. [AT05]. These techniques are commonly classified into *content-based*, *collaborative filtering* (CF) and *hybrid* approaches. Let us first introduce CB methods which originate from Information Retrieval (IR) and Information Filtering (IR) areas.

2.2.1 Content-based filtering

Originating from IR and IF fields, content-based (CB) filtering methods base on the representation of items features and user preferences in a vector space model [MRS08]. Initially considered for search in text documents, these techniques can be used for a broad range of applications.

The vector space model. The adopted representation of documents or items is a vector of weighted keywords. This representation is common in the IR field, and an item (or document) can be noted as an array of weighted keywords: $d = \{w_{k_1,1}, \ldots, w_{k_n,n}\}$. Weights denote the importance of a given keyword in a document. A commonly used weighting scheme is that of *term frequency* (TF) - *invert document frequency* (IDF) [Sal88]:

$$\begin{aligned} w_{k,d} &= TF(k,d) \times IDF(k) \\ &= \frac{f_{k,d}}{\max_{k'} f_{k',d}} \times \log\left(\frac{N}{n_k}\right) \end{aligned} \tag{II.1}$$

Where N is the total number of docs and n_k is the number of documents where the keyword k appears. The IDF contribution aims at reducing the importance of keywords that appear in many documents. Such keywords do not help much in the filtering process since they are less discriminatory. It may also be noted that a normalized frequency is used for keywords in the scope of one document. This is meant to give equal chances to short and long documents.

Given two vectors $\overrightarrow{w_u}$ and $\overrightarrow{w_i}$ representing respectively the user preferences and the item. The utility function f can then be defined using a distance metric between these two vectors. For instance, the *cosine similarity measure* (or variants) is commonly used:

$$f(u,i) = \cos(\overrightarrow{w_u}, \overrightarrow{w_i}) = \frac{\overrightarrow{w_u} \cdot \overrightarrow{w_i}}{\|\overrightarrow{w_u}\| \times \|\overrightarrow{w_i}\|} \quad \text{(II.2)}$$
$$= \frac{\sum_k w_{k,u} w_{k,i}}{\sqrt{\sum_k w_{k,u}^2} \sqrt{\sum_k w_{k,i}^2}}$$

Other techniques. Previously described techniques and heuristics are directly derived from IR, but other methods have also been considered. For instance, statistical and machine learning methods can be used to learn models and predict items' utility based on them.

In particular, the naïve Bayes classifier is a common probabilistic approach which can be used to associate a page to a certain category. A simple and binary example of such categories is to classify pages as relevant or irrelevant for the user [PBMW97]. Let C denote a category, and $k_1, \ldots, k_n$ the keywords of the page to classify. The problem is to evaluate the probability $P(C \mid k_1, \ldots, k_n)$. Under the keywords independance assumption, this probability is proportional to:

$$P(C \mid k_1, \ldots, k_n) \propto P(C) \prod_{i=1}^{n} P(k_i \mid C) \quad \text{(II.3)}$$

Remarkably, all individual probabilities $P(k_i \mid C)$ in the above formula can be evaluated with learning on a training set. Eventually, the method consists in computing the probability for each category C and assigning the page (or item) to the one with the highest value. It has been shown that even though the keyword independence assumption cannot be used in many scenarios, the naïve Bayes classifier gives good accuracy results [PBMW97].

The naïve Bayes classifier is a simple example but other techniques have been developed in the active field of IR and may also be considerd for use in CB recommender systems.

Feature extraction and user preferences. CB methods adopt the same keyword-based representation for both users' interests and items' content. The feature extraction phase consists in analyzing an item's content when it is added to the system. A text document can be parsed to determine its keywords and associated weights. However, a broad range of other items may be considered by using various feature identifiers as keywords.

On the other hand, a simple model of user preferences can be materialized with weighted keywords. These keywords and their associated weights can be elicited using a variety of methods which correspond to explicit and implicit data collection. Implicit methods include web usage mining, for

instance using click-through data or the user's browsing history. Explicit preferences can also be employed to help determine more accuretely keywords weights, e.g., through forms asking users to rank keywords. Ranking keywords is not possible in practice though as it introduces an important work load for the user. More commonly, users can be asked to order their favorite categories, should items be categorized. This approach is for instance adopted in the personalized news service proposed by Google[1].

Benefits and limitations. Analyzing items' content presents the great interest of making it possible to recommend items as soon as they are added to the RS. Contrary to collaborative methods (discussed in the next section) CB methods do not suffer from the new item problem. Besides, CB suggestions are valuable in situations where users' requests need to be served, e.g., in the form of keyword-based queries.

By nature, CB methods rely on a representation of features or characteristics describing an items' content. The feature extraction phase is key but depends highly on the type of resource being processed. IR techniques are well adapted to process text documents. However, automatic features extraction may be much more difficult with multimedia content like images or videos.

Another issue presented by CB recommender systems is the lack of novelty and surprise introduced in recommendations. Indeed, such systems recommend items similar to those the user has liked in the past.

Finally, if CB methods do not suffer from the *new item* problem, they face issues with *cold-start* users. These users are those new to the system and they have no associated profile or preferences. Therefore, the system cannot offer personalized recommendations to a new user, as long as collected data are not sufficient. Recommendations may be bootstrapped with less personalized results though, e.g., simply with the most popular items (see for instance Youtube[2]).

2.2.2 Collaborative filtering

Collaborative recommender systems rely on the assumption that a user is bound to be interested by what other users with similar tastes or interests liked. CF methods have been developed to ease information extraction using the *collective intelligence* [Ala08]. This is an important aspect of CF techniques since they do not rely on deep content analysis and feature extraction, but primarily on ratings which remain superficial and are contributed by users themselves.

[1] http://news.google.com
[2] http://www.youtube.com

User-based filtering. So called memory-based methods use the whole set of ratings given by users to predict missing ratings. The base structure for these methods is the user-item ratings matrix $R = U \times I$. The cell $r_{u,i}$ of this matrix is the rating assigned by user u to the item i. Then, various techniques are employed to extrapolate missing ratings and recommend items with the highest ratings. A rather general example of aggregation function which uses ratings given by other users can be, with K a normalization constant:

$$r_{u,i} = \overline{r_u} + K \sum_{u' \in U} sim(u, u') \times (r_{u',i} - \overline{r_{u'}}) \tag{II.4}$$

In this aggregated rating, each contibution is weighted using a similarity metric between users. In a first approximation, the $sim(u, u')$ factor can be omitted but this leads to take into account users that may have very little in common with the active user u, and may thus produce less relevant recommendations. To compute similarity between users, most popular approaches use the set of items co-rated by two users u and u', and noted $I_{u,u'}$. The *cosine similarity* introduced in equation II.2 can also be used in this context: $sim(u, u') = \cos(\overrightarrow{r_u}, \overrightarrow{r'_u})$. Contrary to CB methods which compute the cosine of the angle formed by two vectors of weighted keywords, the approach of CF methods is to use vectors of ratings, i.e., the vector space is not the same. Another important metric is the *Pearson correlation*, most common to determine users' similarity [SK09]:

$$sim(u, u') = \frac{\sum_{i \in I_{u,u'}} (r_{u,i} - \overline{r_u})(r_{u',i} - \overline{r'_u})}{\sqrt{\sum_{i \in I_{u,u'}} (r_{u,i} - \overline{r_u})^2 \sum_{i \in I_{u,u'}} (r_{u',i} - \overline{r'_u})^2}} \tag{II.5}$$

In particular, this correlation coefficient does not only integrate absolute ratings but takes into account deviation from average user ratings. This addresses the fact that different users may use the ratings scale differently.

Item-based filtering. The user-item approach to CF was to consider similarity between users. However, it has been shown that focusing on correlations between items was also important [SKKR01]. The rationale behind this is for instance that the consultation of an item can trigger further exploration by the user. Amazon[3] is a popular example of the item-based approach, and focuses on the fact that an item can be co-purchased by two users (see Figure II.2).

A simple approach is to build a matrix M where cell $m_{i,j}$ contains the number of times items i and j have both been bought by a same user. Then,

[3] http://www.amazon.com

Figure II.2: When a user adds an item in her shopping cart, Amazon.com proposes recommendations of the form "users who bought this also bought...".

from this matrix, it is possible to filter a given row M_i (or a column) to determine the top-N items that are most often bought with item i. However, this first simple method presents two main issues. First, recommendations made using this process correspond to most popular items, not necessarily the most relevant ones. Moreover, this method does not define a continuous numerical prediction to rank recommendations between them.

Consequently, Sarwar et al. proposed to build the $item \times item$ matrix using explicit user-item ratings instead of co-consulted items. The problem then becomes the transformation of the $U \times I$ ratings to derive items similarity. As opposed to user-item methods which base on users' similarity, missing ratings are this time estimated using ratings attributed to similar items:

$$r_{u,i} = \overline{r_i} + K \sum_{i' \in I} sim(i, i') \times (r_{u,i'} - \overline{r_{i'}}) \tag{II.6}$$

Eventually, the Pearson correlation or cosine similarity may be used to compute items' similarity. However, in the case of item-based CF, the difference of ratings scales between users is not taken into account by the simple cosine similarity. The *adjusted cosine similarity* is therefore preferred and derived from the Pearson correlation to integrate ratings deviation [SKKR01].

User-item and item-item approaches are very similar, but one or the other can be better, depending on the application context [CB07]. The item-item method was presented by Sarwar et al. as a solution to cope with

the fact that the set of users may be less stable (more rapidly changing) than the catalogue of items. Besides, it is particularly adapted for cases where the number of users is much greater than the number of items: $|U| \gg |I|$ [SKKR01].

Other model-based approaches. Previously presented methods are called *memory-based* methods, in that they rely on the whole set of ratings to make their predictions. Another direction in research work has been to build and learn models from collections of ratings to predict missing ones.

First, various probabilistic models have been proposed. Here again, the popular Bayesian model can be used to evaluate conditional probabilities, e.g., if the rating prediction bases on the *expected value*:

$$r_{u,i} = E(r_{u,i}) = \sum_{r=0}^{n} P(r_{u,i} = r \mid r_{u,i'}, i' \in I) \tag{II.7}$$

Clustering techniques can also be used to improve CF algorithms. These methods may be exploited in varied ways but they are most of the time considered as a preliminary step to apply, e.g., memory-based techniques on smaller sets of similar users or items. This can help address computational complexity and thus scalability issues. For instance, the RecTree system proposed by Chee et al. uses *k-means* clustering (with $k = 2$) to recursively building a tree structure, and recommendations are made within the leaf where the active user belongs [CHW01].

Model-based techniques are regarded as having a better combinatory complexity [PHG00]. On the other hand, memory-based approaches present good accuracy results and are reactive to changes in the data. Therefore, these two approaches have been combined in hybrid methods [AT05].

Benefits and limitations. CF techniques present a certain number of important advantages. First, they provide a sense of novelty and diversity in their recommendations, in particular with the user-item approach which focuses on users' similarity, not items'. Besides, they do not require in-depth analysis of items' content and can therefore be employed to recommend any kind of item.

However, it is pointed out that CF techniques present various important challenges [SK09]. The first one and most important being the extreme sparsity of the $user \times item$ matrix. Indeed, in many commercial applications, the RS has to work with very large datasets, and users only rate a small number of items. In particular, this sparsity leads to significant *cold-start* and challenges, be it for new items or new users. Indeed, new items that have not been rated by enough users cannot be recommended. Moreover, recommendations presented to new users are not likely to be relevant, as the browsing and rating history of these users is initially empty. The sparsity problem

has been addressed using a variety of methods, including dimensionality reduction techniques (e.g., Singular Value Decomposition, SVD) [SKKR00] to obtain an approximate ratings matrix of lower rank.

2.2.3 Hybrid methods

Numerous optimizations or variants have been proposed for CB and CF methods independently. Beyond isolated improvements, CB and CF methods are often combined to overcome some of the shortcomings of both types of systems, resulting in hybrid methods. In this section, we briefly introduce these methods but we do not give much emphasize as they are in essence combinations of previously described techniques. Adomavicius et al. proposed the following classification of hybrid methods based on possible combinations [AT05]:

- Methods that combine results obtained from separate CB and CF systems.
- Methods that incorporate CB techniques in a CF method.
- Methods that incorporate CF techniques in a CB method.
- Methods that develop a unified model integrating some characteristics of both approaches.

As an example of the third approach above, let us consider a CB system representing items (e.g., news articles) and users' profiles using vectors of weighted keywords. The system could also exploit ratings given by users to articles and form a hybrid recommendation strategy. For instance, items' similarity may be measured using the cosine similarity on keyword vectors, hence eliminating the new item problem thanks to preliminary content analysis. On the other hand, users' similarity may base on the Pearson correlation of rating vectors, the cosine similarity of keyword-based profiles or eventually a (linear) combination of both.

2.3 Extensions and perspectives

In this section, we present a certain number of approaches that have been considered to improve existing RS. First, an important trend is to leverage trust networks to improve the selection of similar or close users. Then, semantic-based systems help broaden the flexibility and expressiveness of recommendation queries. Finally, modeling and exploiting the user's context has been an important research direction to provide further personalized recommendations.

2.3.1 Trust networks

As we presented the main CB and CF recommendation approaches, the importance of selecting users similar to the current one was highlighted. Remarkably, recent work have focused on the use of *trust networks* to refine this selection and narrow down the collaborative contribution to users more trustworthy for the current one. In particular, it can be argued that trust networks help deal with critical sparsity and cold-start issues. Indeed, trust data can be exploited to augment the overlap between information on users' in the system, which in turn helps increase its coverage. Jamali et al. indicate that trust-based approaches use ratings provided by direct or indirect neighbors and may lead to lower precision, as the distance between users increases [JE09]. For this purpose, the authors propose a random-walk model which allows them to propose a confidence indicator for their recommendations.

Massa et al. proposed the use of a trust network using data from the *epinions.com* service [MB04, MA04]. With this service, users are invited to review items but also to rate reviewers and indicate how helpful they judged their reviews. Using this already established trust network, Massa et al. compare two users using their distance, that is the number of edges separating them in the graph. In this approach, users explicitly build their "webs of trust", which can in the end be aggregated to build a global trust network. Massa et al. use this trust network, in conjunction with traditional CF techniques to provide a final trust-based CF ranking.

Beyond this, O'Donovan et al. focused on the automatic inferrence of trust data from ratings [OS05]. The authors argue that the collaborative contribution in items' scoring should also depend on the item itself. The reason for this is that a given user may be more or less relevant to rate different items. The computational model they propose for trust scores distinguishes the *consumer* user, *producer* users and *items*, which leads to a 3-dimensional scoring. The consumer is the active user, the one recommendations are destined to. On the other hand, producers are users selected to contribute to items ratings. Finally, the last dimension of their model stands for the items to recommend.

2.3.2 Semantic-based recommendations

Another important area of research for RS is that of knowledge-based methods, which pursue the use of reasoning to maximize the match between an item's features and the user's interests.

Initially, logic- an case-based reasoning were considered [Bur00] but knowledge-based RS have mainly been enabled by the emergence of Semantic Web technologies. These technologies aim at easing the representation and exchange of knowledge in distributed and heterogeneous envi-

ronments [EGCB09]. The common ground of these semantic technologies is a rich typed graph model (RDF). On top of this, schemas can defined using RDF-S and OWL frameworks and complex queries can be expressed using the SPARQL query language [DKDA07]. Most commonly, knowledge bases are defined as ontologies within these frameworks but other methods may be used [Gha02]. Ontologies in RS are particularly well suited to help inter-operability in heterogeneous environments, that is with distributed resources of varied types [PdCDL08]. Ziegler et al. presented a RS for such distributed environments. In their approach, agents interact and share partial trust and rating functions over a set of common products [Zie05]. In particular, this system considers all user and rating data distributed but eventually, recommendations are computed locally.

Aside from distributed environments which represent a particular configuration, ontologies may be used in CB or hybrid approaches to improve the representation and understanding of users' interests and items' content [CFV07, ZStL04]. Besides, logic rules can be defined (in the schema itself) to use inferencing capabilities and derive additional (and potentially missing) information [BFPAGS+08]. As an example, this may help deal with partial user preferences and reduce the cold-start issue. In order to address the issue of changing users' interests, Loizou et al. present a system that dynamically chooses input and output spaces for recommendation based on the query, recommendation and rating history [LD06]. The work presented in this paper focuses on the use of an ontology to incorporate contextual elements about items' content and the recommendation process. Integrating contextual aspects in RS has also been an active research area, which we now discuss separately.

2.3.3 Introducing context in recommender systems

RS still need to base on a deeper understanding of the user and his context to offer varied and adapted recommendations [AT05]. So called context-aware RS aim at improving the information filtering process by taking into account some elements about the user's context, like her location, agenda, social environment and preferences. However, actually used context attributes are largely dependent on the application [WS07]. It is worth noting that RS and context-aware systems fundamentally share the same goal, that is to offer relevant personalized information and services [CGF07]. It is common to distinguish between information about the user profile and the context. While profiles describe users themselves, context knowledge describes the environment they interact with and may bear more dynamic changes.

An example of context-aware RS is the CARS system proposed by Abbar et al., defining a *personalized access model* to information, with various personalization concepts and services [ABL09]. In particular, the following context attributes are considered: date, time (of the day), device, IP address

(for geo-location), browser and operating system. Values of these parameters are organized into hierarchies, so authors then use *agglomerative hierarchical clustering* and *k-means* to cluster log records into separate contexts.

Beyond custom systems and models, the remaining of this section presents two important approaches to introduce some degree of context-awareness, respectively in CF and CB systems.

Multi-criteria ratings and multi-dimensional methods. In CF systems, simple ratings do not carry rich semantics which limits their exploitation for fine-tuned recommendations. One approach in this regard is called *multi-criteria ratings* and enables the association of multiple ratings to a given item in different categories. A restaurant could be rated according to the service, the atmosphere, the food, etc. Similarly, books, hotels and many other items can be rated with regards to various aspects. Recommendations with multi-criteria ratings may simply be broken down and use (linear) combinations of the techniques previously described, for single-criterion ratings.

More generally, Adomavicius et al. proposed to extend the usual $U \times I$ recommendation space with a *multi-dimensional* model [AT01]. The idea is to associate a snapshot of the user's context at the time the rating is made. In particular, the authors argue that recommendations should base on additional contextual information to better adapt, e.g., to different user tasks. Let $D_1, \ldots, D_n$ denote a set of dimensions, the utility function is then re-defined as: $f : D_1, \ldots, D_n \mapsto R$, with R the ordered set of recommendations. This model bases on the OLAP paradigm (see Section 4), so "recommendation cubes" are built and the authors defined a Recommendation Query Language (RQL) to query them [ATZ05], as a subset of the OLAP algebra. An example of RQL query can be:

```
RECOMMEND    Movie TO User
FROM         MovieRecommender
BASED ON     PerosnalRating
```

Multi-criteria ratings or multi-dimensional models can help with a finer prediction of missing ratings based on various aspects (or dimensions). However, it can be argued that most RS still lack the ability to adapt dynamically and respond to users' interests shifting over time [LD06]. Ontology-based approaches intend to leverage the expressiveness of Semantic Web technologies to incorporate relevant contextual elements in the recommendation process.

Ontology-based approaches. Closer to CB approaches, ontologies are considered as a way to reconciliate heterogeneous background information and reason over similarity between entities [LD06]. This suits particularly context information which is broad and lies in varied sources [Dey01] (see Section 3).

Kim et al. propose a layered ontology which describes different aspects: (a) products, (b) records or shopping history, (c) the location and (d) the consumer [KK07]. In particular, the authors define preference scores for elements of the ontology – at instance and class levels – to select products of interest. Instances are simply scored using shopping records and an aggregated score is computed for classes, taking into account the concepts hierarchy.

With their COReS system, Costa et al. argue for the inclusion of domain-specific preferences [CGF07]. Traditional CF techniques rely on the whole set of ratings to determine users' similarity and thus recommend items. This has significant implications and reduces the ability of the system to react to different tastes in different domains. For instance, if two users liked the same movies, they would likely get recommendations for the same restaurants, even though they may have different culinary preferences. In response to this, COReS uses domain ontologies to augment context information and perform suitable reasoning.

The *News@hand* system presented by Cantador et al. bases on a controlled and structured vocabulary – defined in an ontology – to represent both items' features and users interests [CBC08]. The hybrid recommendation process integrates these representations, along with semantic relations described in the ontology. Moreover, *News@hand* proposes recommendations of news items using two different models: one for long term user's interests and one for short term adaptation to the user's context. News articles are automatically retrieved and annotated with weighted concepts, using TF-IDF-like weights. Then, inside a session, the user's reading activity is analyzed to determine what the authors call the *semantic runtime context*. Similarly, this structure is represented as a set of concepts, with weights decreasing as time passes.

To conclude, we reckon RS could greatly benefit from an increased context-awareness, for instance to enable further personalization and dynamic adaptation. However, context modeling and management is a complex domain in itself. In the next section, we thus review related work in the area of context-aware systems.

3 Context-awareness

In the previous section we presented work related to RS and in particular a trend which pushes for the integration of some context sensitivity. Context is a very general term that led to the active development of a corresponding research area. In this section, we define context and context-aware computing and we review related work in this area.

3.1 Definition, characteristics and usage of context

The notion of context appeared in different disciplines like artificial intelligence, databases, natural language processing [AS97], etc. However, this research area has been mainly driven by work on context-aware computing and underlying modeling issues.

3.1.1 Preliminary definitions

Let us first introduce definitions of important terms which stand for rather abstract concepts and have thus led to varying definitions.

Context. Dey and Abowd present the *context* as a key element to enrich and increase human interactions using an implicit common understanding, e.g., of everyday situations [DA99]. One can easily imagine that the notion of context is thus extremely broad and complex, in particular when applied to human-computer interaction. As a result, authors have provided varying definitions in previous work, trying to balance between the generality of the notion and concrete operational needs [ZL07].

Definitions by examples are the most simple and operational since they just enumerate attributes that can be used and considered as context. Among possible attributes, location has been the most exploited in previous context-aware systems [KPVOGM05]. For instance, Schilit et al. presented in 1994 the ParcTab system using as context information the user's current location (or room) and the device [SAW94]. Therefore, the type of device and its underlying capabilities are also commonly used attributes. Mobile computing is indeed an important use case of context-aware systems [CK00].

However, context cannot be limited to device and location [SBG99] and it is important for application designers to understand what context is and how it can be used. Dey gave the following definition of context which is now commonly accepted:

> "*Context* is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." *(Anind K. Dey)* [Dey01]

Context-aware computing. Schilit and Theimer first coined the phrase *context-aware* to denote applications that react to the user's location, objects nearby and changes to these properties over time [SAW94]. In their vision, an application is said context-aware if it uses or reacts to knowledge about who you are, where you are or what/who is around you. The definition by Dey appears to be more general and thus more operational in varied application scenarios:

> "A system is *context-aware* if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." *(Anind K. Dey)* [Dey01]

Situation. The complexity and broadness of context led to the introduction of a higher-level abstraction, called *situation* and sketched by Dey as a description of the states of relevant entities [DA99]. A simple but remarkable definition of the situation was later formulated by Yau et al. as follows:

> "A *situation* is a set of context (information) in the application over a period of time that affects future system behavior." *(Yau et al.)* [YL06]

In particular, this definition indicates that designers of context-aware systems should identify and focus on subsets of context information that influence a given behavior of their system. Besides and most importantly, the time aspect is crucial and the definition highlights the fact that context knowledge is inherently time-dependent [Sch06a], among other characteristics that we now present.

3.1.2 Characteristics of context information

Context information or knowledge presents characteristics which need to be carefuly analyzed as they push requirements on representation formalisms and context models [HIR02, BBH+10].

Sources and heterogeneity. Previous definitions of both context and situation reveal the important variety of resources and facts which may contribute to the description of a user's environment. This is in particular true since context information often originates from heterogeneous sources in distributed environments. Among many possible examples, we can mention private corporate information systems (CRM, ERP, BI platform, etc.), web services, publicly accessible ontologies, etc.

Henricksen et al. recognize three main sources of context information based on how and where it is produced, be it from sensors, human users or derived through further computation [HI04]. User-provided information may be rather static or dynamic. Static facts are rarely changing and explicitly provided by the user or obtained from (controlled) background knowledge. Therefore, they are regarded as highly accurate which differs from dynamic information, more often out-of-date or incomplete. Besides, derived information can also introduce errors or imprecisions. Sensors of context information will be further discussed along with context data acquisition (see Section 3.1.3).

Incompleteness and uncertainty. Context data has often been presented as imperfect by nature, be it unknown, ambiguous, imprecise or errorneous [HI04]. In their extensive survey on context modelling and reasoning techniques, Bettini et al. state that "one of the key requirements of context-awareness is capturing and making sense of imprecise, and sometimes conflicting data, about the physical world" [BBH+10].

We argue that the caution paid to uncertainty has mainly been motivated by the mere origin of most context-aware systems. Indeed, systems that aim at modeling context to provide adapted information and services mostly emerged from ubiquitous computing and so-called intelligent environments. Systems in this area primarily made use of networks of sensors to obtain information about the physical world, which is particularly prone to incompleteness or inaccuracy. Although uncertainty is an interesting topic, we do not consider this is so important when dealing with dedicated software sensors.

Time variability. The above mention of static and dynamic facts is a more general and common issue in context modeling. We previously mentioned that context knowledge is inherently time-dependent [Sch06a] and the aim of context-aware applications is to timely react and adapt to possible environment changes. Bettini et al. therefore insisted on the important *timeliness* of context information and argued that it should natively be captured by context models [BBH+10].

The problem of context information validity evolving over time is referred to as the *aging*. It can be noted that the aging is not uniform accross different aspects of the context [Sch06a]. That is, the validity of different facts evolves at different paces over time, from static to rapidly changing. For instance, the user's hobbies and her employer are rather static facts whereas her location or current task are highly volatile.

Resulting from the evolution of information validity over time, context history management is also another valuable feature to consider. In particular, storage or persistence of the context composes a history, a rich resource which can be used to enable reasoning over time [SLP04]. For instance, the user's interaction history may be mined to detect common patterns and provide predictive capabilities [Sch06a]. If history sounds like simple persistence, it brings considerable value to systems willing to adapt and opens an important door to apply reasoning and machine learning techniques.

Eventually, Schmidt mentioned that imperfection and dynamics are two aspects of context information that have been poorly explored by research so far [Sch06a].

Security and privacy. Context information is by nature often personal and thus particularly sensitive. Consequently, models, architectures and

frameworks need to integrate security and privacy as a core requirement of their design [CK00].

Tatli suggested three main requirements and proposed a context data model to handle privacy-related issues [Tat06]. An interesting particularity of this model is the use of blurring techniques, e.g., to return range values rather than exact ones.

Even if "technology in itself is rarely inherently bad (...) it can be used for good or bad purposes" [WFG92]. Therefore, varying regulations in different countries control storing and usage of users' personal data. These legal aspects often impose important limitations and should also be considered in the design of context-aware systems.

3.1.3 Acquisition of context information

As mentioned previously, context information originates from different sources and can be acquired in a variety of ways at different levels. This results in distinct acquisition approaches described below [BDR07]:

Direct sensor access. In this approach, the application has direct access to a sensor (in the hardware sense of the term). Examples of these sensors include but are not limited to a GPS, a motion detector, a luminosity sensor, etc. For instance, most modern smartphones integrate a GPS which proves extremely useful for many location-based applications.

Middleware infrastructure. Robust software design and development methods push for encapsulation and componentization, which leads to layered approaches for context-aware applications. This is in particular interesting to build components responsible for hiding the unnecessary complexity of context details, often very granular (e.g., GPS coordinates). The layered architecture of context-aware middlewares and applications will be further discussed in Section 3.3.1.

Context server. Pushing the approaches above further, context data management is significantly improved using a client-server architecture. The context server consolidates partial context information and enables sharing with multiple clients or devices. Moreover, the server can relieve clients from resource-intensive processing and perform consolidated consistency checking.

Context information is either explicit – that is provided by the user – or implicit [SBG99]. Information is said implicit when it is derived or collected from a range of sources called sensors [HIR02]. Indulska et al. proposed to classify sensors as *physical*, *virtual* and *logical* [IS03]. The authors initially considered location-related sensors but this classification holds more generally and can be described as follows [BDR07]:

Physical sensors. Many physical sensors are rather commonly available nowadays. This includes hardware sensors like cameras, microphones, GPS, touch-sensitive screens, etc. These sensors prove useful in scenarios which require to capture information about the user's physical environment.

Virtual sensors. A sensor is said virtual if it is an application or software source of context information. For instance, observation plugins are used in [Sch06b] to obtain information on users' actions and manipulated content in an unobtrusive manner.

Logical sensors. Logical sensors provide an additional layer of abstraction and may combine information from other sensors. Logical sensors can provide information consolidated at a higher-level, or inferred using reasoning techniques.

3.2 Context and situation modeling

In the previous section, we introduced the notions of context and situation and presented, in particular, characteristics of context information which push requirements on models. For these notions, this section discusses models which are required from machine-representation and computational perspectives. It should be noted that an interesting survey and analysis framework for context models was published by Strang et al. [SLP04], later extended and structured by Bolchini et al. [BCQ+07].

3.2.1 Different formalisms

Research on context modeling produced different types of models. Some used simple key-value pairs which lack semantics, others relied on object models which may be hard to share between client applications. Finally, more recent works have proposed ontology-based models that enable a uniform representation (e.g., using RDF) and improve reasoning[LCCH10]. This section describes the main types of formalisms that have been considered to model context.

Key-value models. The most simple context models are based on *key-value* pairs to represent named attributes [SLP04]. These attributes capture various characteristics of interest, for instance the most common one, the user's location.

Key-value pairs are often used as they are particularly easy to manage. However, they often lack semantics and are most of the time used to perform some exact matching.

Markup scheme models. Markup scheme models rely on a markup-text representation of context data, for instance as XML. The aim of this approach was to encourage a richer representation of facts and enable more sophisticated querying and reasoning.

For this purpose, most markup-based approaches rely on a hierarchical data structure. Even though tree structures convey more semantics than simple key-value attributes, we reckon that choosing between markup-based and other models is more of a serialization problem and may/should be decorrelated from the core expressiveness of the model. For instance, RDF graphs that describe ontology-based models (described later) may also be serialized as XML. To conclude, markup approaches often lead to custom or proprietary definitions and models which discourage sharing and re-usability.

Object models. Key-value pairs and markup text are rather common approaches, easily shared between platforms, operating systems, etc. Object models present the advantatage of being pragmatic and commonly understood by application developers. However, this formalism for models lacks a common representation suitable for sharing and re-use in distributed environments using, e.g., different programming languages [CFJ03].

The particularity of object-oriented models is to easily enable encapsulation. This allows to hide some of the complexity of low-level context information, for instance by providing convenience methods to access filtered data. More generally, object-oriented formalisms only enable the exploitation of context knowledge through pre-defined interfaces.

Semantic models. Context aims at representing users, their environments and interactions between all concerned entities. More generally, Kofod-Petersen et al. argue that context is knowledge, not a special type of information [KPC06]. Therefore, context modeling should greatly benefit from research in the area of knowledge representation. For instance, description logics enable symbolic knowledge representation and reasoning. In particular, the OWL-DL subset of the OWL language – used to define ontologies – is a description logic [BBH+10].

The first approaches to model context using ontologies introduced this as a necessity to better normalize and formalize context information [OA97]. More precisely, Strang et al. indicate that ontologies provide a uniform representation formalism for the core concepts of the model, plus any kind of subconcepts or facts [SLP04].

There are varied examples of context model using ontologies. Again, context is a broad notion and authors do not seem to converge towards a commonly accepted context ontology. As an example of ontology-based model, the CoBrA-ONT ontology presented by Chen et al. in [CFJ03] describes entities and relations of interest in the domain of an intelligent meet-

ing room. Gu et al. also presented an ontology-based model for intelligent environments, focusing on the following classes of entities: person, location, activity and computational entity [GWPZ04]. Sieg et al. use background knowledge ontologies and model the context with scores assigned to concepts of these ontologies [SMB07]. These scores are dynamically adjusted using spreading activation, in response to the user's dynamic behaviour.

Heckmann et al. reconciliated user, context and resources models into the notions of situation [Hec05b] and situated interaction [Hec05a]. Situation statements were introduced by Heckmann et al. as a homogeneous data structure to represent the unit of information in situation modeling. These statements extend simple *subject-predicate-object* triples with additional metadata and reference entities described in ontologies like the general user modeling ontology (GUMO) [Hec05b].

Strang et al. summarized their analysis of different model formalisms, placing more hope in ontology-based approaches [SLP04]. The authors argue that ontologies are best suited to handle characteristics of context information previously presented.

3.2.2 Categories or dimensions of context information

Since context information is broad, structuring it is key to enable an easier interpretation – in particular by client applications [DA99] – with, for instance, general pre-defined categories. Various model dimensions have been proposed in the literature. The most simple categorizations distinguished between personal and environmental context knowledge [GS01]. Schmidt proposed a context model structured according to three categories: self, activity and environment [Sch02].

With a more operational focus, Zimmermann et al. propose the following five fundamental categories of information to extend the context definition [ZL07]:

1. *Individuality*: this category contains knowledge about the entity of interest, be it a real user, an applicative agent, etc.

2. *Time*: time-variability of context information is a key aspect (see Section 3.1.2) and this category represents time-related information, for instance the user's time zone or the current time would stand in this category. Besides, the time dimension may be further structured, e.g., exposing work hours, holidays, etc.

3. *Location*: location has often been regarded as a key parameter of context. Location can be physical (e.g., GPS coordinates) or virtual (e.g., the IP address in a network), absolute or relative.

4. *Activity*: the user's needs are significantly determined by her current activity, which may be represented with goals, tasks and actions. Ac-

tivity theory has also been used to model activities in [KPC06], but little has been done to indicate how this information can be captured.

5. *Relations*: the relations category provides structure to entities by representing their interactions or dependencies. Zimmermann distinguishes social, functional and compositional relations [ZL07].

In their data-oriented survey of context models [BCQ+07], Bolchini et al. propose to categorize aspects captured by existing models around *space*, *time* and related *history*, *subject* of the model and the *user profile*.

To conclude, it is commonly acknowledged that structuring context information using dimensions make the model more operational. If such general dimensions have a real interest, we reckon that pragmatic context-adaptation will most of the time involve application-specific categorization in more sophisticated models.

3.3 Context management

The previous sections presented characteristics of context information and models that are designed to represent, interpret and reason on this knowledge. This section presents context-aware systems and discusses their architecture, common context management tasks performed and eventually dynamic aspects.

3.3.1 Architecture

Architecture applies at different levels for context-aware systems. First, the internal perspective focuses on the core architecture of systems managing users' context data, often described in a layered manner. Then, frameworks and services can be proposed to help consume context information and build context-aware clients in heterogeneous environments.

Proposed context-aware systems may be either centralized or distributed [HIMB05]. However, centralized architectures have the great advantage of facilitating information completeness and consistency management. For instance, the Context Broker Architecture *CoBrA* bases on a central broker which collects and aggregates data from heterogeneous sources [CFJ03]. Reconciliation can be helped by using ontologies and reasoning, as in [LCCH10].

Context management layers. More generally, architectures developed for context-aware systems usually base on a layered approach, even though naming of layers and functional ranges may vary [BDR07]. In particular, decoupling context acquisition from its usage is key to handle data diversity and encourage reusability accross context-aware applications.

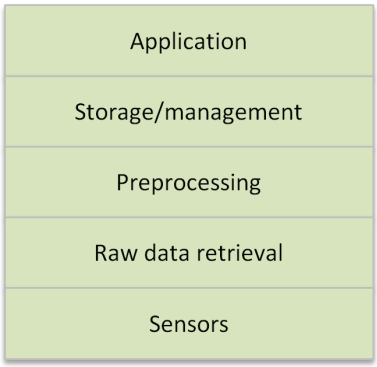

Figure II.3: The five layers of context-aware systems high-level architecture.

Ailisto et al. presented the following five layers in [AAH+02]: physical, data, semantic, inference, application. More recently, Baldauf et al. proposed in [BDR07] the five layers depicted in Figure II.3:

Sensors. This first layer represents sensors, physical, virtual or logical (see Section 3.1.3).

Raw data retrieval. This layer stands for components taking care of access to raw data from sensors. Most of the time, drivers will be necessary when dealing with physical sensors.

Preprocessing. Preprocessing components are responsible for the data transformation that may, in some cases, be necessary to obtain a uniform representation (or formalism).

Storage/management. In particular, the homogeneous representation helps with storage of context knowledge and may enable reasoning. This layer involves various management tasks that will be further described in the next section.

Application. The context model and some system services can be consumed by client applications using controlled interfaces, be they programmatic or Web services.

Middleware, frameworks and services. At the application level in the layers previously described, clients may consume context-related information thanks to middlewares/frameworks or services. This is interesting for instance to hide details about raw context information like GPS coordinates and ease context consumption and adaptation.

The *Context Toolkit* proposed by Dey is an example of such framework [DAS01]. This toolkit exposes three main types of components to build context-aware systems: widgets, interpreters, and aggregators.

CASS is an example of server-based context management framework that was presented by Fahy et al. [FC04]. The authors argue that serparating context-based inferences and behaviours from application code paves the way to context-aware services configurable by users. About CASS, Bolchini mentions that it "manages both time and space, taking into account the context history, and provides context reasoning; it does not contain user profiling capabilities." [BCQ+07].

It is undoubtedly true that frameworks facilitate the development of context-aware applications, e.g., by providing re-usable and extensible code. However, this approach often lacks the flexibility (client-wise) of Web services. For instance, two different devices in heterogeneous environments may not allow the same programming language but they would likely both be able to consume Web services via HTTP requests. For instance, SOCAM – for Service-Oriented Context-Aware Middleware – aims at enabling the rapid prototyping and development of context-aware mobile services [GPZ05]. SOCAM defines context providers and interpreters to handle data acquisition and processing.

These examples of context-aware systems and middlewares are far from an exhaustive list, but more complete surveys have already been presented, for instance in [BCQ+07, BDR07].

3.3.2 Dynamic management

Time-variability is a key characteristic of context information (see Section 3.1.2). A complete context-aware system approach thus needs to handle context construction but also dynamic maintenance aspects. In this section, we review work related to context usage at runtime and dynamic management aspects in particular.

Context usage and management. Context information acquisition has been thoroughly investigated by previous research (see Section 3.1.3). The goal of data acquisition is first to build and initialize the context model so the system can benefit from its dimensions. However, Zimmermann et al. mention that "something is context because of the way it is used in interpretation, not due to its inherent properties" [ZL07]. Therefore, context usage probably matters more than its acquisition.

Application developped using context-aware systems – in particular via defined frameworks – generally consume information in either *synchronous* or *asynchronous* manner [BDR07]. Synchronous context consumption involves frequent polling which may be particularly resource intensive and leaves to the client the task of determining differences. On the other hand, the asynchronous consumption mode bases on the *publish-subscribe* pattern for event management. Subscribers or clients register so they can be notified by the system when a specific event occurs, as in [DSP09].

Dynamic evolution. Context-aware systems target dynamic adaptation of information and services. Such systems therefore need a strategy to handle active components and knowledge evolution. However, dynamic context management has rather poorly been addressed by previous research [Sch06a].

Euzenat et al. approach the problem of dynamics with a focus on active components. They argue in [EPR08] that context-aware systems in distributed environments require the ability to dynamically integrate new and sometimes unexpected entities, be they real users and objects or computational entities. In particular, the discussion adopts a data representation point of view. It highlights the fact that ontologies provide a homogeneous knowledge representation and ease the dynamic definition and integration of new types of data. On the other hand, the definition of dynamic computational entities responsible for actual data management and manipulation is almost ignored.

As discussed in Section 3.1.2, the validity of context information evoles over time. History management is thus an important component of context management systems. In particular, the history is composed of persisted context knowledge, which thus needs to integrate time-related metadata, e.g., a timestamp and a lifetime. If history sounds like simple persistence, it brings considerable value to systems willing to adapt and opens an important door to perform reasoning over time or apply machine learning techniques [SLP04].

In event-driven systems and for instance active databases, the ECA framework has been used to define active rules with clear declarative semantics: `ON` *event* `IF` *condition* `DO` *action*. In case a certain event occurs and a specified condition is satisfied, the desired action is eventually performed [TSM01]. This framework seems particularly well adapted to approach the issue of context knowledge maintenance. In the ECA framework, Laitaki presents the use of SPARQL-based rules to perform reasoning on context data represented in RDF graphs [Lai07]. Interestingly, the ECA framework has also been considered to introduce dynamics in data warehouses and multi-dimensional analysis [TSM01]. In a more recent work, Ravat et al. make use of the ECA framework to perform dynamic query personalization, in response to OLAP operations (events) performed by the user to move from one analysis context to another. Query personalization and recommendations will be further discussed in the next section.

To conclude, beyond pragmatic concerns related to actual context management, Zimmermann introduces two interesting notions about context dynamics: transition and sharing [ZL07]. First, capturing and modeling context transitions would be extremely valuable, for instance to determine a change of focus from the user. Second, sharing is defined as what happens when two contexts, e.g., from different entities, base on the same information or knowledge. In their discussion, Zimmermann et al. focus on the value brought by additional knowledge or inferred relations but they do not

address privacy and security issues. Even though this discussion on context transition and sharing is interesting, concepts introduced still lack a proper formalization and tangible implementations.

4 Data warehouses and recommendations

In this section we present previous work related to multi-dimensional data warehouses (DW), for personalization and recommendations in particular. First, Section 4.1 introduces DWs and On-Line Analytical Processing (or OLAP) approaches. Then, we review in Section 4.2 methods that have been considered to integrate personalization in warehouses. Last, Section 4.3 discusses different kinds of recommendations proposed for DW-related content.

4.1 Data warehouses and OLAP

In this section, we introduce the basics of the multi-dimensional model supported by DWs. Due to its important interest in both academia and industry, research on data warehousing produced many in-depth studies, so our aim here is not to be exhaustive but rather to introduce the key concepts used in the area.

4.1.1 Data warehouses and the multi-dimensional model

Let us first present DWs and the switch from relational to multi-dimensional storage and modeling of data.

From relational to multi-dimensional databases. Relational database management systems (RDBMS) are extensively used in production systems. The design of relational schemas is often optimized for On-Line Transaction Processing (OLTP) thanks, e.g., to a high degree of normalization. The requirements of OLTP applications (consistency, recoverability and concurrency) are quite different from those of OLAP applications [CD97]. In particular, OLAP systems aim at supporting the decision making process of knowledge workers thanks to sophisticated analysis that aggregate large amounts of data [CT98]. Chaudhuri mentions that for this purpose, "historical, summarized and consolidated data is more important than detailed, individual records" [CD97]. OLAP analysis and operations will be further described in the next section.

At the heart of the OLAP approach is the DW, a large repository containing historical data from several sources of an organization, for analysis purposes [PJ01]. Data are usually extracted from various production systems to be transformed and eventually loaded in the DW; this *extract-transform-load* (ETL) process and related technologies have been reviewed in [Vas09]. To conclude on the materialization of DWs, Rizzi argues that if centralized

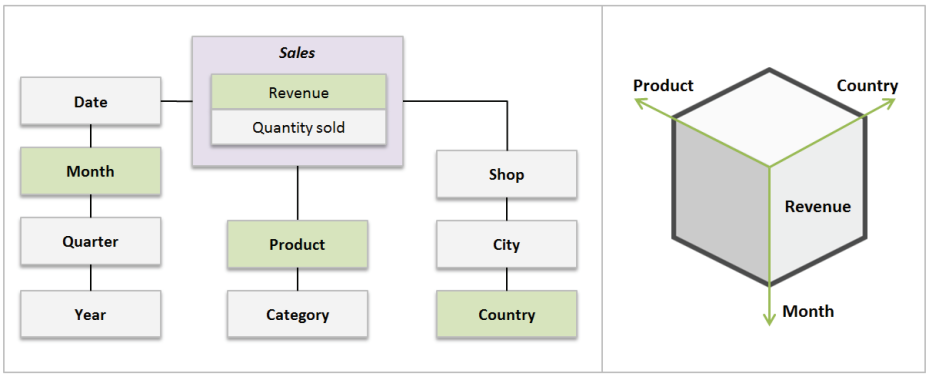

Figure II.4: Simple example of a *Sales* fact table defining two measures *Revenue* and *Quantity sold*. These measures can be analyzed against 3 dimensions (at the finest level): *Product*, *Date* and *Shop*.

architectures are well adapted for warehouses of isolated organizations, they no longer fit the needs of collaborative BI networks [Riz11].

The multi-dimensional model. DWs and OLAP approaches are supported by the *multi-dimensional model* which was introduced to provide improved analytical capabilities and to enable the interactive analysis of large amounts of data. Multi-dimensional modeling methodologies have been thoroughly reviewed by Romero et al. in [RA09]. A certain number of formal definitions and models were proposed for the multi-dimensional space and the OLAP algebra [GRB11, PMT08, NNT01, Vas98, CT98], which operations will be introduced in the next section. In short, the multi-dimensional data model organizes data in cubes and a DW can be seen as a collection of related cubes [PJ01]. Cells of these cubes contain numerical facts called *measures*. These measures are defined in *fact tables* and can be analyzed against any number of *dimensions*. Let us briefly describe these key elements:

Dimensions are core concepts of multi-dimensional models and aim at providing different observation angles around facts of interest. Dimensions are used to select and aggregate data at the desired level. Dimensions are indeed organized into hierarchies and each instance (or value) of a dimension belongs to a particular level [PJ01].

Facts or fact tables contain tuples which may for instance describe unit sales transactions. Each tuple is defined by a certain number of pointers to dimensions and numerical indicators [CDG01], and is thus represented as a point in the multi-dimensional space.

Measures are numerical values represented in fact tables and are commonly associated with an aggregation function [PJ01], like *sum*, *average*, etc. To conclude, Figure II.4 illustrates a fact table, `Sales`, defining two measures (`Revenue` and `Quantity sold`) attached to various dimensions like `Country`, `Date`, etc. The right hand side of the picture illustrates a 3-dimensional cube aggregating the `Revenue` measure.

To conclude about the multi-dimensional model, it should be noted that issues may arise regarding the aggregation of measures on different dimensions. These are called additivity or more generally summarizability issues and may result in erroneous aggregations, for which a taxonomy is proposed by Horner et al. [HS05]. Our aim is not to be exhaustive regarding summarizability issues here but rather to introduce and mention them, as they have been thoroughly reviewed by Mazón et al. in [MLT09]. Briefly though, Pedersen et al. present three classes of measures based on how they aggregate, *additive*, *semi-additive* and *non-additive* measures [PJ01]. Additive measures can be aggregated along any dimension whereas semi-additive measures can only be aggregated against certain dimensions. Eventually, there are also non-additive measures which cannot be aggregated at higher level. In particular, Mazón et al. discuss the fact that summarizability issues have to be taken into account at modeling time and compatibility of measures and dimensions may be addressed using functional dependencies [MLT09, RA09].

4.1.2 OLAP - On-Line Analytical Processing

OLAP tools and applications aim at enabling interactive yet powerful data analysis. Analaysis can be performed thanks to a set of core operations that are exposed by OLAP servers, resulting in actual queries over the materialized DW.

OLAP servers and querying. Chaudhuri presents in detail the two main server approaches that have been adopted to provide BI clients with analytical capabilities [CDG01]. First, ROLAP – for relational OLAP – servers exploit relational databases as storage backends. This support is important since relational databases are common in the industry and already handle large amounts of data. If relational databases may be used for storage purposes in some DW implementations, Pedersen et al. argue that "the only robust solution is to use database technology that offers inherent support for the full range of multidimensional data modeling" [PJ01]. MOLAP – for multi-dimensional OLAP – have thus been proposed and offer native optimized multi-dimensional storage. Eventually, these two types are often combined into hybrid OLAP (HOLAP) servers. Indeed, ROLAP servers often

perform better when data is relatively sparse, whereas MOLAP servers are more efficient with dense data [CDG01].

BI applications need to query the warehouse for actual data, and queries on the DW have to be expressed and evaluated in accordance to the underlying storage engine. Should a relational backend be used, queries may be expressed in SQL [Gor03]. On the other hand, OLAP cubes and native multi-dimensional storage may be queried using the Multi-Dimensional eXpressions (MDX) language.

OLAP operations. An important aspect of OLAP is to allow the interactive exploration of vast amounts of aggregated data, thanks to the following core operations [PJ01]:

Slice and dice consists in the selection and aggregation of information to summarize numerical measures along the different levels of dimensions hierarchies [Vas98]. Selection involves in particular the application of filters on both measures and dimensions values.

Drill down and roll up denote a way to navigate in and explore data cubes. In particular, hierarchies of dimensions can be navigated up and down to obtain respectively higher- or lower-level aggregates. This navigation is enabled by hierarchies which represent partial orders between dimensions [CT98].

Drill across is the operation combining two cubes that share a common dimension. This operation is implemented by a join in the relational agebra.

Ranking in queries allows to retrieve for instance the top/bottom n cells. Numerous queries may involve this operation, for instance to retrieve the 5 shops that best sell a given product.

Rotate. The rotation of a cube results in the observation of given facts under different dimensions.

4.2 Personalization of data warehouses

In this section we review methods used to personalize data warehouses at different levels. In particular, we focus on those based on user profiles and the definition of user preferences.

4.2.1 Personalization approaches

Bentayeb et al. present personalization in rather general terms and classify some OLAP-related techniques according to the matrix illustrated in Figure II.1 [BBF+09]. The authors define *configuration* and *adaptation* as

the main ways to address personalization. Configuration allows the user to explicitly modify her profile whereas adaptation consists in implicit learning of the user profile by the system. Both configuration and adaptation help perform some transformation of content or layout and provide personalized recommendations.

More specifically, the different approaches considered to provide a certain degree of personalization in DWs have been reviewed by Kozmina et al. [KN10]. In particular, the authors propose a classification of OLAP personalization techniques into five categories. First, methods using *preferences* allow query personalization at the schema level. Most importantly, users can express their preferences on measures, dimensions and hierarchies. These preferences may be expressed either quantitatively or qualitatively, thus conveying more or less semantics (see Section 4.2.2). Then, *static and dynamic* personalization are distinguished, function of the moment personalization is actually achieved, at design or at run-time. Personalization at design time is usually less reactive to dynamic changes. Another personalization aspect lies in visualizations proposed to the user. Kozmani places in this category visual query editors which relieve users from complex SQL/MDX generation using, e.g., simple drag-and-drop of measures and dimensions. Bellatreche also proposed to integrate visualization constraints into their user profile [BGM+05]. Eventually, Kozmina highlights two techniques to provide *recommendations* of multi-dimensional queries, based on users' sessions or preferences. Query recommendation will be more thoroughly discussed in Section 4.3.

4.2.2 User profiles and preferences

We introduced the main types of personalization approaches considered to enable user-adaptive OLAP exploration of a DW. Let us now focus on user profiles that are commonly used to achieve this, and in particular preferences which may be defined with quantitative or qualitative approaches.

User profile. User profiling or modeling for personalization has been extensively studied, in particular in the fields of information filtering and retrieval [RT09]. Methods aim at maximizing the matching between returned data and some characteristics of the user model.

Kozmina et al. present in [KN10] a composite user profile, aggregating various types of profile about spatial and temporal aspects, the user, her interactions, preferences and recommendations. Each profile type contains a certain number of attributes collected from different information sources. Examples of considered attributes include the user's roles (from system and organizational perspectives), her location, the device being used, etc. User profile data is obtained either explicitly from the user, or implicitly by learning from various sources (context, static, activity and analysis data).

The user profile proposed by Bellatreche et al. in [BGM+05] is defined as the combination of preferences and visualization constraints. Preferences will be described later in this section. From the visualization perspective, the device being used is part of what the authors call context data. This parameter is important, e.g., to determine the screen size and thus the amount of information which can be displayed.

OLAP preferences. Golfarelli and other authors argue that if a lot of work has been done on database preferences, preferences for data warehouses and OLAP analysis have not attracted so much attention [GR09, KN10]. However, there are some exisinting approaches, often classified in *quantitive* and *qualitative* preferences [BBF+09].

Ravat et al. propose a quantitative model of preferences to ease the user's navigation in multi-dimensional data [RT09]. The user can express her preferences by associating weights to objects of the multi-dimensional model. The authors then propose a rule-based system – using the *event-condition-action* (ECA) framework – on top of these numerical preferences to assign priority weights to schema elements, based on the current OLAP operation being performed. As a result, personalized visualizations and OLAP manipulations can be proposed to the user.

Kozmina et al. also presented a quantitative model for OLAP preferences in [KN10]. Besides, it may be noted that the proposed model distinguishes between schema-specific and report-specific preferences. Indeed, a user may have a certain preference for a schema element in a given report and another more general for other documents and tools. Preferences on schema elements are qualified using two attributes, the degree of interest and a weight. The degree of interest is a categorical attribute and ranges from very low to very high while the weight is a real in $[0; 1]$.

On top of the visualization constraint previously mentioned, the user profile proposed by Bellatreche et al. in [BGM+05] includes the definition of preferences. Preferences are simply defined as a total pre-ordering of dimensions' values. This approach can still be regarded as quantitative but adopts a relative perspective using ordering rather than absolute weights. The authors then derive a total order between sub-cubes of the warehouse. Eventually, their *PersoVisu* algorithm provides personalized visualizations that (a) maximize the user's preferences w.r.t. the represented cube and (b) satisfy the visualization constaint.

MyOLAP is a significative example of qualitative approach, introduced by Golfarelli et al. in [GRB11]. In particular, the authors present the following specificities of OLAP preferences:

- they can be expressed on numerical domains as well as categories,
- they can address the desired level of aggregation in hierarchies and

- they can refer to atomic as well as aggregated data.

Taking these into account, Golfarelli et al. introduce MyOLAP as an algebra to express preferences as soft constraints. Preference-specific operators and keywords are used to extend MDX queries, for instance to mention the user prefers data such that the aggregated value of a given measure falls in a certain window.

Jerbi et al. also introduced a qualitative model for preferences in multidimensional databases [JRTZ08, JRTZ09a]. These preferences only address dimensions but base on strict partial orders (s.p.o.) at two levels. The first s.p.o. is used to compare two dimensions whereas the second one compares, inside a given dimension, two attributes or values. Remarkably, the authors also model the *analysis context* in order to define context-aware preferences. This specific notion of context will be further discussed in the next section.

4.3 Query recommendations

As a particular use case of personalization, we review in this section techniques for query recommendation. Such techniques can be used to ease user-adaptive exploration of a DW and reduce the time taken by analysts to get to data of interest.

4.3.1 Recommendation types

Various types of OLAP recommendations can be considered and have been classified by Jerbi et al. as follows [JRTZ09b]:

Anticipatory recommendations. These recommendations aim at determining, given the current state of an OLAP analyis, what is most likely the next step or operation to perform.

Alternative results. Alternative results can be provided to the user to suggest other related data. These results may or may not be related to the last operations performed by the user. Concretely, these results help explore areas of cubes that had not necessarily been visited before.

Interactive assistance for query design. Last, interactive assistance aims at helping the user in the query design process with suggestions, for instance to build queries and ad-hoc reports. Jerbi et al. mention that this recommendation scenario has attracted little attention from the research community, compared to the first two [JRTZ09b].

4.3.2 Query recommendation

Stefanidis et al. proposed a taxonomy to categorize database recommendation techniques in (a) *current-state*, (b) *history-based* and (c) *external*

sources approaches [SDP09]. Methods (a) exploit data as well as metadata (schema), be it from the current query being performed or the database itself. Methods (b) leverage the history of users' queries contained in logs, which may be structured in analysis sessions. Eventually, methods (c) may exploit data from sources other than the database to perform further personalization, e.g., by building more complete profiles.

More precisely focused on data warehouses, techniques employed for query recommendations have been recently reviewed by Marcel et al. [MN11]. In particular, the authors provide a formal framework to represent query recommendation techniques as a function of various parameters, $Recommend(L, cs, I, P, f)$. L is a log or set of sessions, cs is the current state of the current session, I the data warehouse instance, P the user profile and f an expectation function. The expectation is a function associating a real number to (or scoring) data of the warehouse. Marcel et al. divided methods for query recommendations in those (a) based on user profiles, (b) using query logs, (c) based on expectations and (d) hybrid ones.

Methods based on user profiles. Marcel et al. define this category for methods that take the user profile (e.g., preferences) into account, on top of the common current analysis session. Such techniques can be formally represented as $Recommend(\emptyset, cs, I, P, \emptyset)$. They include the user profile (e.g., preferences) in the recommendation process to maximize the user's interest for suggested queries [GRB11, KN10]. User profiles and preferences have been presented in more details in Section 4.2.2. It may be noted though that preferences or the profile may define visualization-related constraints, which prove valuable since queried data eventually aims at being displayed to the user [BGM+05].

Methods based on query logs. Methods in this category are represented as $Recommend(L, cs, I, \emptyset, \emptyset)$. Techniques involved build on logs of queries in various analysis sessions and mainly address predictive recommendations of forthcoming queries. Logs are composed of queries posed by users of the system (through different clients) and are stored by the server.

Analysis sessions are positioned as first-class citizens of these recommendation approaches. Jerbi et al. used the definition of OLAP analysis as a sequence of contexts which represent successive states or queries [JRTZ09b, JRTZ09a]. More precisely, the analysis is modeled as a graph where nodes denote contexts and edges represent OLAP operations performed to go from one to the other. A context stands for a query and is represented using a tree structure. Root nodes are measures and dimensions used in the analysis, while leaves of the tree represent their values, be they numerical or categorical. The authors apply these notions in particular to provide query recommendations leveraging the user's preferences, as

previously discussed.

Sapia et al. proposed to model query logs using a probabilistic Markov model [Sap99]. The authors then exploit this to determine probabilities of forthcoming queries. Clearly, query recommendations consist in selecting those that maximize the probability in the chain model.

Another approach is to use a distance metric between sessions to extract recommended queries from past sessions similar to the current one. Giacometti et al. mention that the log can be extremely large and "sparse", in the sense that many queries may not be posed several times by users [GMN08]. Therefore, the authors propose to first group queries into clusters called partitions of the log. Each query is associated to a given class or cluster, and *generalized sessions* are determined by replacing queries in sessions with their attached classes. The current and former sessions are matched using an adaptation of approximate string matching principles. Candidate query recommendations are selected and ranked among queries of past sessions.

Methods based on expectations. Methods for query recommendation that fall in this category are represented as $Recommend(\emptyset, cs, I, \emptyset, f)$. They base on *discovery-driven* analysis and rely on two models, one for data the user has already visited in the past and another one for unseen data [MN11]. The major assumption of these techniques is that the user will be interested by parts of data where the two models most significantly differ. The last parameter f of the *Recommend* function, is the expectation function and may represent this deviation.

Sarawagi et al. introduced the discovery-driven analysis of multi-dimensional data [SAM98] and proposed two different types of operators to guide users toward zones of a cube that present unexpected data. The first type tries to find either higher-level data that confirms an observed trend or lower-level data to explain differences. Sathe et al. further discuss these aspects in [SS01]. Second, techniques based on an adaptation of the maximum entropy principle present the user with most unexpected data compared to what she has already seen [Sar00].

More recently and to renew with the discovery-driven approach of OLAP analysis, Giacometti et al. propose in [GMNS09] to model current and past analysis sessions using pairs of cells (i.e. at the data level) containing significant and unexpected differences. If the current session leads the user to observe the same difference as a former session, the latter is used to recomend other significant and related differences.

Hybrid methods. To conclude on query recommendation techniques, previously described approaches may be combined in various ways with hybrid methods [GMNS09]. Methods in this category are represented as

the complete recommendation function $Recommend(L, cs, I, P, f)$. To the best of our knowledge and according to Marcel et al., the only really hybrid method described in previous work was proposed by Giacometti et al. in [GMNS09]. Indeed, the presented technique processes query logs to determine analysis sessions and associate a goal to them. Eventually, recommended queries are those that pertain to past sessions sharing the same goal with the current one.

5 Summary and discussion

In this chapter we presented our review of previous work in three different but somehow connected areas: recommender systems (RS), context-aware systems and multi-dimensional data analysis. We want to highlight the fact that our aim was not to be exhaustive in any of these directions, but rather to identify zones for possible synergies.

RS are designed to provide users with suggestions of items bound to be interesting to them. The two major recommendation techniques are content-based (CB) and collaborative filtering (CF). CB methods rely on the vector space model and represent both items and users with vectors of weighted keywords. They thus present the great advantage of being easily configurable with queries. However, a CB approach requires content analysis on items which implies to develop potentially sophisticated feature extraction techniques. On the other hand, collaborative techniques have mainly based on superficial information (like ratings) gathered from a large number of users, thus reducing the need for feature extraction. CF methods usually build on ratings matrices to determine various distance or similarity metrics. For instance, the user-based approach first determines users similarity based on users' ratings pattern. Then, a rating is computed for items the active user has not yet seen, aggregated from ratings given by similar users. CF methods are extremely interesting since, at minima, they require no knowledge of items content. This makes them easily re-usable with varied types of resources. However, collaborative techniques suffer from so called cold-start issues, be it *new item* or *new user* problems. On the other hand, CB methods avoid the new item case using content extraction. Eventually, recent work have addressed the need to integrate more context sensitivity in recommendation algorithms. In particular, semantic-based approaches are promising in this regard but do not seem to have reached the maturity, scalability and stability levels of CB or CF techniques.

A lot of work has also been done on context-aware systems which are designed to adapt to various characteristics of the user and her environment, physical as well as virtual. Context-aware systems have often focused on more dynamic aspects of personalization and aim at taking more than the user herself into accout. Researchers in this field commonly agree on

a definition of the context provided by Dey et al., which considers it as "any information the can characterize the situation of an entity" [DA99]. This definition – which is arguably said operational – reveals how broad the notion is. Therefore, it is no surprise that researchers did not seem to converge towards a commonly accepted model of context. Many different models have thus emerged, using formalisms with more or less flexibility and expressivity, mainly key-value pairs, object models and ontologies. In particular, it has been argued that semantic technologies are the most appropriate to meet heterogeneity and reasoning requirements imposed by context-aware systems [SLP04]. Remarkably, this is also a promising direction for future recommender systems integrating more and more context knowledge [CGF07, CBC08].

The important influence of mobile and more generally ubiquitous computing in research on context-aware systems brought interesting challenges. In particular, ubiquitous systems often involve hardware elements like devices or physical sensors, e.g., in so called intelligent environments [CFJ03]. Unfortunately, this proves rarely useful when designing adaptive applications in a pure software environment for knowledge workers or business users. Besides, several proposed context models intend to go extremely far in personalization, e.g., by representing the user's psychological state. Again, this kind of settings raises serious doubts as to their applicability in business environments since information (a) needs to be captured and (b) needs to be interpreted in a useful and meaningful way, not to mention obvious security and privacy issues.

In Section 4.1 we presented the multi-dimensional model used in data warehouses (DW) and OLAP analysis tools. In particular, we presented the core concepts of dimensions, facts and measures which compose the common star schema of the warehouse. Measures are numerical indicators of a fact table and they can be analyzed against a certain number of dimensions. Queries can be built to query for actual of the DW and usually depend on the underlying storage engine, be it relational or purely multi-dimensional. Modern DW solutions propose query design tools which enable the formulation of queries by non-expert users and hide the complexity of SQL/MDX generation [PMT08]. A way to ease the query design process is to simply let the user select measures and dimensions from a list to compose his queries, and eventually add some sort/filtering criteria. However, many real-life production systems define complex multi-dimensional models which may involve up to thousands of obejcts, measures and dimensions. Consequently, it is thus difficult for users to navigate this data and find the appropriate information. Therefore, query designers could be significantly improved by using personalization and recommendation techniques. This most likely generalizes to other analysis tools.

Different disciplines and areas produced varying definitions of context. multi-dimensional data analysis in BI is no exception. The notion of *analysis*

context is considered in the field of multi-dimensional modeling and OLAP analysis. An analysis session is defined as a sequence of contexts, where OLAP operations (drill-down, etc.) are used to go from one to the other. This leads to a graph representation of sessions. Nodes of this graph are contexts or queries, and edges denote OLAP operations. This notion of context is interesting and allows for instance to recommend the most likely forthcoming query given the current session. However, this definition is obviously much more limited than generic context models presented in Section 3.

Eventually, it is worth noting that recommendation techniques for multi-dimensional queries presented in Section 4.3 only poorly leverage generic methods developed in the field of RS. To the best of our knowledge, there has been no previous work using content-based or collaborative filtering. We thus consider that query recommendation and more generally access to BI tools and data could significantly benefit from RS techniques. Besides, most work have proposed language-specific approaches, e.g., using MDX operators to express preferences in queries [GRB11]. On the other hand and from a commercial standpoint, BusinessObjects (now SAP) has from its origin adopted a source-agnostic approach, based on so called *universes* or semantic layer. This layer stands for the multi-dimensional model and isolates clients from underlying storage and connection issues. We thus consider it is important to provide techniques for personalization and recommendation that are platform-independant.

6 Conclusion

In this chapter we reviewed work related to three research areas: (a) RS, (b) context-aware systems and (c) data warehouses and OLAP analysis. We identified interesting connections between RS and context-aware systems, in particular at the modeling point of view since both types of system rely on user modeling. Context-aware systems are more generic from the knowledge representation perspective and encompass broader sets of information, about the user and her environment. Besides, context-aware systems adopt a more dynamic position which makes them interesting in scenarios involving interactive personalization. However, traditional RS techniques have proved highly scalable and can provide valuable results, in particular in situations of exploration or discovery. Last but not least, we presented data warehouses and the multi-dimensional model, which organizes data in cubes. Facts are represented as points in the multi-dimensional space and refer to numerical measures and dimensions. Dimensions can be organized into hierarchies to define navigation paths and are used, e.g., in roll-up/drill-down OLAP operations. We observed that only recently techniques for query recommendations emerged and that most approaches are platform- or language-specific. To conclude and to the best of our knowledge, techniques considered so far for

query recommendations have not explored the use of traditional RS methods, namely content-based and collaborative filtering.

Chapter III

Graph-based Situation Modeling

Contents

1 Introduction and requirements

We underlined in the review of related work (see Chapter II) that personalization commonly bases on some – more or less dynamic – user modeling. Modeling the user and her context or situation is key in many applications to serve timely and personalized recommendations of data or services.

This section presents our graph-based situation model leveraging a secure and modular *Graph Repository* (GR) of heterogeneous resources. Before going in any further detail, we first describe requirements imposed on such system, in the context of corporate deployments regarded as our main application scenario.

Applications inter-operability. As opposed to the experience of a user limited to Web sites inside a browser, a business user evolving inside a company's network may interact in varied ways with several communicating applications. Ideally, these applications would share the same knowledge and representation of the user's interests and intentions to better assist her and ease inter-operability. Such user-centric applications are called *situational applications* and they emphasize the need to inter-operate on the basis of a shared situation model.

Information heterogeneity. Several modeling approaches have been considered to address the context representation issue and were discussed in more details in Chapter II, Section 3. Definitions of both context and situation reveal the important variety of resources and facts which may contribute to the description of a user's environment. The need to handle heterogeneous resources and relations between them leads to graphs as a natural representation.

Security and privacy. Situation modeling and monitoring is about capturing and interpreting users' personal characteristics and interactions, which means the underlying data is particularly sensitive. Users' privacy thus needs to be respected and enforced by the system. If user data has to be protected, security also has to be observed from a different angle. Indeed, a given user may not be allowed to see all resources available in her corporate network. This is the case, for instance, with critical HR or financial information.

Dynamic adaptation. Eventually, our aim with situation modeling is dynamic adaptation to serve timely and personalized recommendations of resources to end-users. Therefore, we need to take into consideration the fact that users' interests or goals may evolve rapidly as they interact with various applications. Time-dependency thus has to be integrated at the heart of the representation of situation-related knolwedge.

To conclude, various requirements push for an expressive model for situation knowledge, allowing a homogeneous representation of heterogeneous resources. Therefore, a graph-based model seems particularly adapted, in particular to represent and operate on semantics of such information. Different formalisms – from simple key-value pairs to ontology-based models – have been introduced for context modeling and were discussed in Section 3.2 of Chapter II. Strang et al. summarized their analysis of these formalisms and placed more hope in ontology-based approaches [SLP04], arguing that they are best suited to handle characteristics of context information.

The rest of this chapter is organized as follows. Section 2 presents the graph-based situation model and the underlying graph repository framework

that we propose. Further characteristics of our model are then introduced in Section 3, in particular situation dimensions. Last, we discuss in Section 4 the proposed model according to the analysis framework presented by Bolchini et al. [BCQ+07].

2 A graph-based situation model

The requirements previously introduced impose constraints on our system, which lead to graphs as a natural and homogeneous representation. Graphs are indeed very well suited to handle information diversity and represent a variety of resources and relations between them. For instance, this was a major argument in favor of the rich typed graph model RDF, to handle distributed sources of heterogeneous resources. In this section, we first introduce *situation statements* which are a major component of our graph model.

2.1 Situation statements

Sitting on top of raw context data, situation models aim at characterizing the user and her environment, or interactions between her and the "rest of the world". For this purpose, Heckmann introduced *situation statements* as a homogeneous data structure used to represent the unit of information in situation modeling [Hec05b], based on Semantic Web technologies. Statements offer an appropriate answer to handle the heterogeneity of resources and interactions as well as time-dependency and privacy aspects.

2.1.1 Definition and notations

Our work bases on the statement structure proposed by Heckmann to elaborate on an aggregated and dynamically maintained view of users' situations. Situation statements are extended triples representing assertions formed of a subject S, a predicate P and an object O, like `Marge` *read* `document1`. Table III.1 presents other examples of such statements.

Basic triples are augmented with additional metadata M to take into account temporal constraints, privacy settings, origin (or source) and confidence of the information. A statement can then be noted as a tuple (S, P, O, M), with $M = (origin, t, l, s_{owner}, s, c)$. We describe below these metadata:

Timestamp t and lifetime l. The timestamp t and the expected lifetime l are used to handle time-related validity constraints. In a first approximation, the timestamp can be seen as the date and time the statement was created. The lifetime indicates during how much time the statement holds valid. Statements may be flagged with an infinite lifetime to indicate they are always true, regardless of time constraints.

♯	Subject	Predicate	Object	Origin
S_1	`Marge`	*read*	`document1`	`EmailClient`
S_2	`document1`	*mention*	`Country`	`EntityExtractor`
S_3	`document1`	*mention*	`Q3`	`EntityExtractor`
S_4	`Q3`	*instanceOf*	`Quarter`	`EntityExtractor`
...				
S_k	`Marge`	*hasLocation*	`Paris`	`MobileApp`
S_{k+1}	`Marge`	*reportsTo*	`Bruno`	`SocialNetwork`
...				

Table III.1: Examples of (partial) statements, some of which constitute Marge's situation as represented by the graph in Figure III.1.

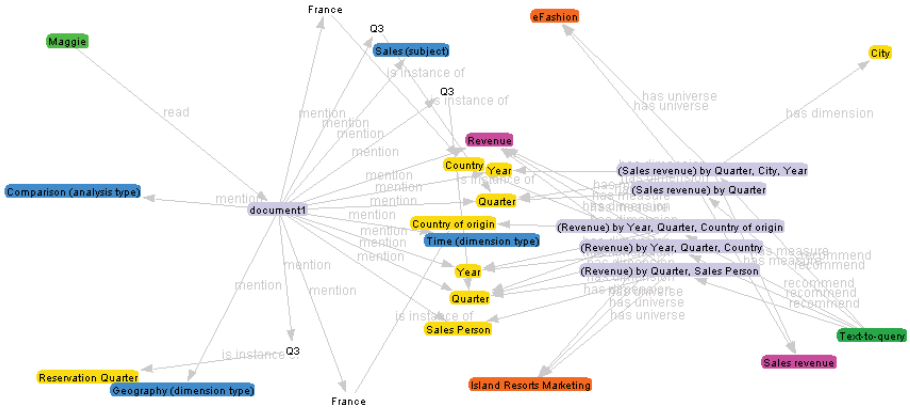

Figure III.1: Example of situation graph for the user `Marge`.

Privacy and ownership. The privacy setting s in combination with the ownership s_{owner} indicates whether this statement is public, private or has custom access authorizations. By default, all situation statements are owned by the concerned user/agent and are strictly private. Sharing some information with other users has to be, here again, the result of the user opting in explicitly.

Origin and confidence. We define *agents* as resources of the system which can create statements (providers, users, operators, client applications, etc.). The origin refers to the *agent* that created the statement. Finally, the confidence attribute c allows agents to qualify the reliability level of the statement, which is key in distributed environments involving situational applications inside a corporate network.

2.1.2 Situation graphs

At a certain point in time, the situation model can be seen as a collection of valid facts or statements which takes the form of a graph. Such a graph is illustrated in Figure III.1, it can be centered on the concerned user or agent to reflect its particular importance. In this figure, the agent `Marge` is represented by the green node in the top left corner. This graph is formed thanks to a certain number of statements, some of which are enumerated as examples in Table III.1. These statements reveal for instance that `Marge` is reading `document1`, an unstructured document mentionning a certain number of entities. This example is going to be discussed in depth in Chapter VI, Section 2 with the description of the Text-To-Query system. Text-To-Query analyzes text documents the user is to suggest related queries on multi-dimensional data warehouses.

In the definition proposed by Heckmannn, statements reference resources generally available and described in shared ontologies. Even if statements introduce ownership and visibility attributes, it is unclear how security is going to be ensured by the system. Indeed, a situation management platform may require the definition and development of various components. Heckmann et al. do not propose a mechanism to let users control what data can be consumed by these components. However, ontology-based models and semantic technologies are well suited to offer a shared representation and understanding of heterogeneous resources. We thus present our approach to base statements – and more generally graph data – on a secure and modular *graph repository* (GR). The GR is a user-centric framework for personalized graph data management, enabling in particular the isolation of users' data for security and privacy purposes.

2.2 Graph Repository

We introduced situation statements that we use as a data structure to represent the unit of information in situation modeling. As said before, statements can be seen as qualified relations (or edges) in graphs where nodes describes various available resources. These resources may come from a great variety of source systems, which leads us to define a modular and extensible *Graph Repository*.

2.2.1 Source systems and personalized providers

Various information systems available inside a corporate network may contribute to general knowledge about available resources, and users in particular. For instance, an LDAP directory can describe employees and relations between them. Although other popular online social networks may also be used, this wasn't the focus of our work as they seem less appropriate when dealing with private resources of an organization and users that are

employees of the same organization. Other classic examples of enterprise applications include ERP, CRM and BI platforms. A CRM system could be used to extract relations between account managers and customers. Such information may be useful in various scenarios, for instance with an automatic message redirection system in a support center. Moreover, BI platforms are an important kind of system that we leverage in several personalization and recommendation scenarios, as described in Chapters V and VI.

As many applications and systems may contribute to background knowledge about users and available resources, our aim is to define an modular *graph repository* (GR), enabling progressive using modules called *providers*, e.g., by connecting to source systems. The modularity of providers – and more generally the GR – will be further discussed from an architecture standpoint, later in Chapter IV, Section 3.1.

Most source systems impose their own security constraints like access-control rules. Duplicating their data and maintaining security policies over duplicated data would be particularly complex and costly. This leads us to opt for a delegated security enforcement policy. Users of our platform can thus choose (or refuse) to give their credentials to leverage connections to existing systems through dedicated providers. This *opt-in* approach is key to minimize privacy issues and ensure users control what data is collected and manipulated in their situation model. By construction, the GR is an inherently personalized structure and may contain private data. Security is therefore a key aspect and the access to the GR requires agents to be authenticated.

2.2.2 Definitions

Being a personalized structure, the GR is dedicated to one user u and should be noted GR_u. For the sake of clarity, the user indication is omitted in the following definition. The GR can be defined as an aggregation of several graphs $G_i = (N_i, S_i, P_i)$, defining sets of nodes N_i and statements (or edges) S_i identified by URIs. Each graph G_i is populated and maintained by a provider P_i, and a provider may manage any number of graphs.

Available types of nodes and statements are defined in the GR schema. The GR initially provides a core schema T_{base}, defining basic types of nodes and statements. A provider P_i has the ability to declare a custom schema T_i when registering to the repository. The provider-specific schema T_i has to extend T_{base} to define new types of nodes and statements. To enable schema-level reasoning, the complete type schema T is obtained by merging core and provider-specific schemas: $T = T_{base} \cup (\bigcup_{i=1}^{m} T_i)$. Eventually, with $G = \bigcup_{i=1}^{m} G_i$, the graph repository can be noted $GR = (G, T)$.

The base type schema T_{base} initially describes core types of nodes and relations (or edges) but may be enriched by providers. The three main types defined in T_{base} are `Node`, `Statement` and `Agent`. The `Agent` type is used

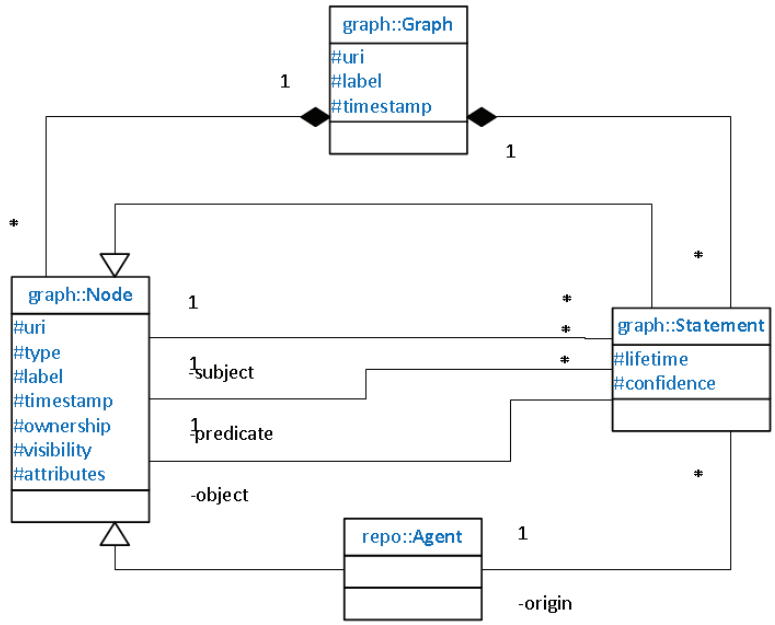

Figure III.2: Class diagram - Overview of the graph model used and exposed by the Graph Repository.

to denote any entity interacting with our system, be it a `UserAgent` or an `ApplicationAgent`. Beyond providers and client applications, we will later introduce another kind of `ApplicationAgent`, *operators* (see Chapter IV, Section 2.3). Briefly, operators are used in our dynamic management platform to maintain situations' current views or graphs.

2.2.3 Graphs schema and the GR ontology

We introduced the extensible GR to expose homogeneous graphs of resources and statements for our situation model. We now describe more precisely the actual graph model.

From an internal object model perspective, graphs are exposed as depicted in the class diagram of Figure III.2. This diagram is only showing main attributes and does not reflect all implementation-related methods such as getters, setters, etc. Most importantly, every node is identified by a *URI*, a *type* and *ownership*/*visibility* attributes. Besides, other attributes can be considered, in the form of typed name-value pairs. Statements extend nodes so they inherit the same attributes. We presented in Section 2.1 statements as semantically qualified relations between a *subject* and an *object*. In the internal object model of Figure III.2, statements are represented as a n-ary association between four nodes: a *subject*, a *predicate*, an *object* and an *origin*.

The graph model previously introduced is meant to be source agnostic. Therefore, it does not require a specific storage facility, be it a relational database or a triple store. However, for serialization and exchange purposes, an RDF representation proves useful and well adapted. We thus defined a simple ontology for the GR schema. An RDF-S ontology enables schema merging and simple hierarchical reasoning on types, for instance to infer sub-classes and sub-properties chains. It is also possible to consider OWL ontologies to define schema with, for instance, transitive properties or relations. An example of transitive relations will be discussed with functional dependencies in multi-dimensional modeling, in Chapter V, Section 2.2. We present below an example of an individual `document1` in the `Email` node class:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:grepo="http://www.sap.com/situation/grepo/0.97#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:about="http://.../document1">
    <rdf:type rdf:resource="http://...#Email"/>
    <rdfs:label>Meeting this afternoon</rdfs:label>
    <grepo:timestamp>12345</grepo:timestamp>
  </rdf:Description>
</rdf:RDF>
```

In RDF, statements are simple triples (subject-predicate-object) but they can be further qualified using *reified statements*. A reified statements is itself a resource (thus identified by a URI) of type `rdf:Statement`. It has three main properties, a subject, a predicate and an object. The statement metadata can be represented using additional properties. We illustrate below a statement of the assertion "`Marge` *read* `document1`", e.g., originating from a plugin in an `EmailClient`:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:grepo="http://www.sap.com/situation/grepo/0.97#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:about="http://.../statement123">
    <rdf:type rdf:resource="http://...#Statement"/>
    <rdf:subject rdf:resource="http://.../users/marge"/>
    <rdf:predicate rdf:resource="http://.../read"/>
    <rdf:object rdf:resource="http://.../document1"/>
    <grepo:origin rdf:resource="http://.../EmailClient"/>
    <grepo:ownership rdf:resource="http://.../users/marge"/>
    <grepo:visibility>private</grepo:visibility>
```

```
    <grepo:timestamp>12345</grepo:timestamp>
    <grepo:lifetime>5</grepo:lifetime>
  </rdf:Description>
</rdf:RDF>
```

The use of reified statements is not mandatory since another custom type of node could have been defined to denote statements. However, it happens to be particularly useful to deal with implementation- and performance-related concerns. For instance, the Jena API for Semantic Web technologies makes use of reified statements to limit the important overhead introduced by additional statements properties.

For extensibility purposes, the schema ontology can be enriched by registered providers. For instance, providers can define additional types of nodes and statements. These types may integrate specific attributes. In the example above, an `Email` type could be defined as a sub-class of `Document`, defining in particular "from", "to" and "body" attributes (or properties in the RDF terminology). Below is the previous description of `document1` with these additional attributes:

```
<rdf:RDF ...
    xmlns:gr-mail="http://www.sap.com/situation/grepo/mail#">
  <rdf:Description rdf:about="http://.../document1">
    ...
    <gr-mail:from>nick@email.com</gr-mail:from>
    <gr-mail:to>marge@email.com</gr-mail:to>
    <gr-mail:body>Hi, I'm not going to be available this
    afternoon, can we re-schedule later?</gr-mail:body>
  </rdf:Description>
</rdf:RDF>
```

Already existing ontologies may be used under the condition of a small data transformation, in particular for statements which will have to be extended with appropriate metadata. Besides, individuals defined in these ontologies will also have to correspond to types defined in the aggregated schema.

2.2.4 Graphs factories

The actual architecture and implementation of the graph repository will be more precisely discussed in Chapter IV, Section 3.1. Briefly though, graphs are populated thanks to specific modules which can be used to connect to various source systems. These modules, called providers, may re-use or define new *graph factories* for actual graph creation and more generally graph data management. Our aim is to isolate graph capabilities from actual

data storage. Examples of possible implementations of graph factories are briefly presented below:

RDF database. The graph is originally represented as RDF triples and these triples are stored either in a dedicated triple store or in a relational databse.

RDF in-memory. The graph is originally represented as RDF triples and *a priori* only held in-memory.

XML/RDF file. The graph is originally represented as RDF triples and triples are stored in a file.

Custom. Graph data are stored in a custom format (for instance in a relational database) and transposed in the GR model by the provider.

Clearly, it is possible to form hybrid strategies to connect and adapt to various remote systems. As an example, a provider may use Web services to query a remote system, convert data to the GR model on the fly and populate RDF graphs held in-memory, e.g., for caching purposes.

3 Situation model characteristics

We introduced the graph foundations of our situation model with the personal and extensible GR. In this section, we present our approach to structure this information and present (a) comprehensible dimensions to applications consuming the exposed model and (b) utility methods for components consuming the internal model.

3.1 Situation model overview

At a given point in time, the user's situation is represented by a collection of statements. These statements represent raw facts or assertions, possibly at different granularity levels. Therefore, it is important to also enable exploitation of the situation model at a higher level of abstraction and expose, e.g., information structured in categories of specific interest.

A *situation* is a graph centered on the concerned *agent*, augmented with a simple *profile* and a certain number of *dimensions*. Figure III.3 illustrates these main components of the internal object model, hiding lower level graphs, nodes and statements. We present below a short description of these different elements:

Agent. A situation concerns a specific user or agent and is centered on this particular node. Most times, the agent is going to be a user. However, it may be interesting in some scenarios to monitor and automate tasks for applicative agents.

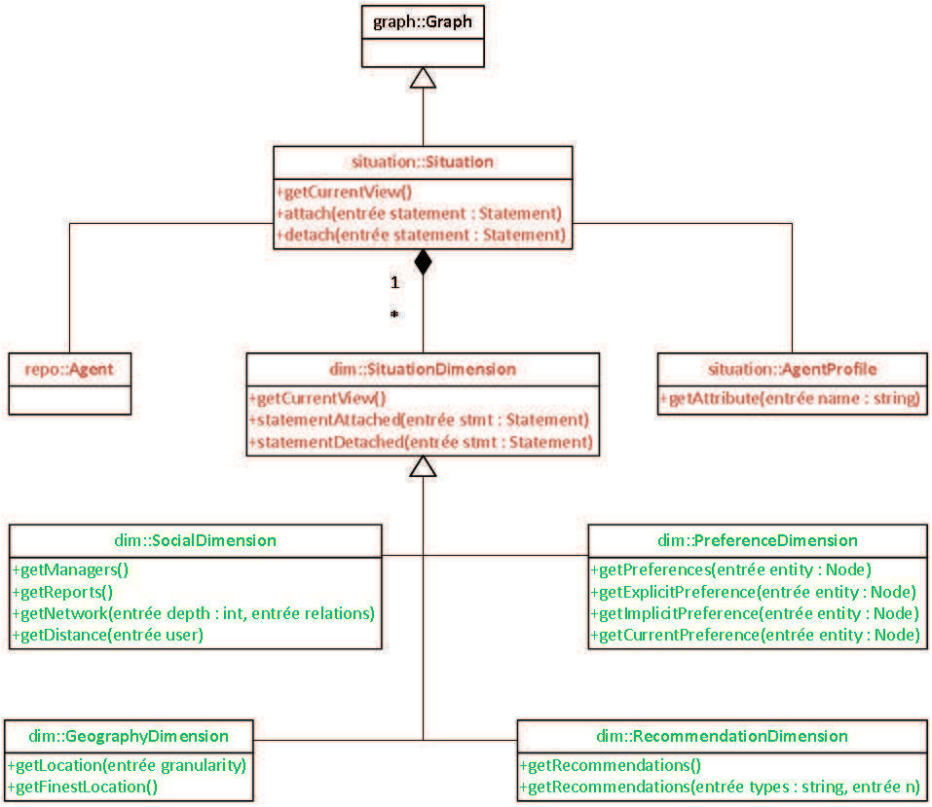

Figure III.3: Class diagram - Overview of the internal object-based situation model, including core dimensions and personalization-related dimensions.

Profile. The `AgentProfile` class associated to a situation is a simple container for typed key-value attributes. It may be used to store and retrieve simple literal values like the user's organization name, her job title, phone number, email addresses, etc.

Dimensions. A `SituationDimension` defines a group of selected situation statements, and aims at representing coherent sets or categories. The following categories will be defined in more details in Section 3.2: core dimensions of our model (*social* and *geographical* aspects) and personalization-related dimensions (*preferences* and *recommendations*).

3.1.1 Current view

The time aspect is an inherent part of our model thanks to statements metadata which express time-dependant validity (timestamp and lifetime). From a dynamic point of view, situation graphs evolve to reflect users' interactions and changes in their environment.

At a given time t_{now}, the situation graph exposes statements that are currently valid and part of the user's situation. However, deprecated statements may still reside in the underlying data as they can be kept for history purposes. To cope with the separation between currently valid and outdated statments and ease data querying, `Situation` objects expose the `getCurrentView()` method on top of usual graph operations. This method filters out deprecated statements.

Dynamic aspects of our system are further discussed in Chapter IV. In particular, so called *operators* are components responsible for ongoing situation maintenance. We now describe how our homogeneous model can be used to reconciliate current and history data.

3.1.2 History management

Statements generated by the system are managed and stored using GR factories previously described and can be regarded as "knowledge of the past". Metadata related to time allow to query for statements that were valid at a certain time in the past $t_{past} \leq t_{now}$. On top of this core functionality to query raw data, our situation structure may also expose specific history elements which may be helpful to various applications. Clearly, such elements are largely dependant on the use case being considered. An example of application will be presented (along with other experimentations) in Chapter VI, Section 3. Among possible history elements, the presented federated search system could use previous queries or questions posed by the user with learning approaches.

To sum up, history of past context information can be fully exploited with queries on the GR. On the other hand, from a more dynamic perspective, the platform continuously maintains a view of currently relevant context information.

3.2 Agent profile and core dimensions

Predicates play a central role in determining the semantics of characteristics and interactions reflected by statements. Based on this observation, a simple categorization of statements enables us to expose consolidated and structured dimensions in the situation model. This structure may be useful at two levels. First, internal modules of our dynamic situation monitoring platform can leverage utility methods exposed by dimension objects of our model. Second, from an external client perspective, this allows for situation data to be queried in a more fine-grained manner. For instance, client applications may query for situation data present in one given dimension.

3.2.1 User profile

Although it is not precisely a dimension in the sense of the internal object model previously presented (i.e. a collection of statements), the user profile is an important component of an agent's situation.

An `AgentProfile` instance is associated to each `Situation` object created and monitored by the system. This profile can be used to store and retrieve typed key-value attributes meant to describe various characteristics of the agent with literal values. Examples of such attributes can be anything like the user's birth date, her first and last names, her job title and company name, etc. Besides, additonal contact information like the user's email addresses or phone numbers may be of interest.

In the RDF terminology, attributes values are denoted as *literals* and can be of different types (int, date, etc.). Each resource can have any number of attributes, represented by *data properties*, as opposed to *object properties* which stand for relations between two resources.

3.2.2 Social dimension

Knowledge about the user's social environment is key to further personalize information and services. Therefore, the `SocialDimension` is among major dimensions of our model. It has been briefly presented in previous sections that relations making up the user's social network may be obtained from various sources. First, popular online social networks may be used. In our case, focused on corporate deployment scenarios, users are employees of a same organization and they may share private resources of this organization. Therefore, a minimal view of the user's social network is derived from an LDAP directory or, more generally, any enterprise directory. For instance, such directories may contain information about hierarchical relations between employees. The construction and maintenance of the user's social network is not really the focus of our work and will not be further discussed. We leverage an existing prototype called *Social Network Analyzer*[1]. This prototype enables the exploration of multiple relations between people which can be represented using various predicates like *reportsTo*, *hasBusinessContact*, *worksWith*, etc. To sum up, any kind of statement between two `UserAgent` nodes is a potential candidate for the social dimension, *a priori* independant of the precise predicate. As an example, Figure III.4 illustrates `Marge`'s social network, including different relations with users at a maximum distance of 2.

In this particular dimension, the following commodity functions can be defined:

- `getNetwork(depth, relations)` is the most generic method. It gets

[1] http://sna-demo.ondemand.com/

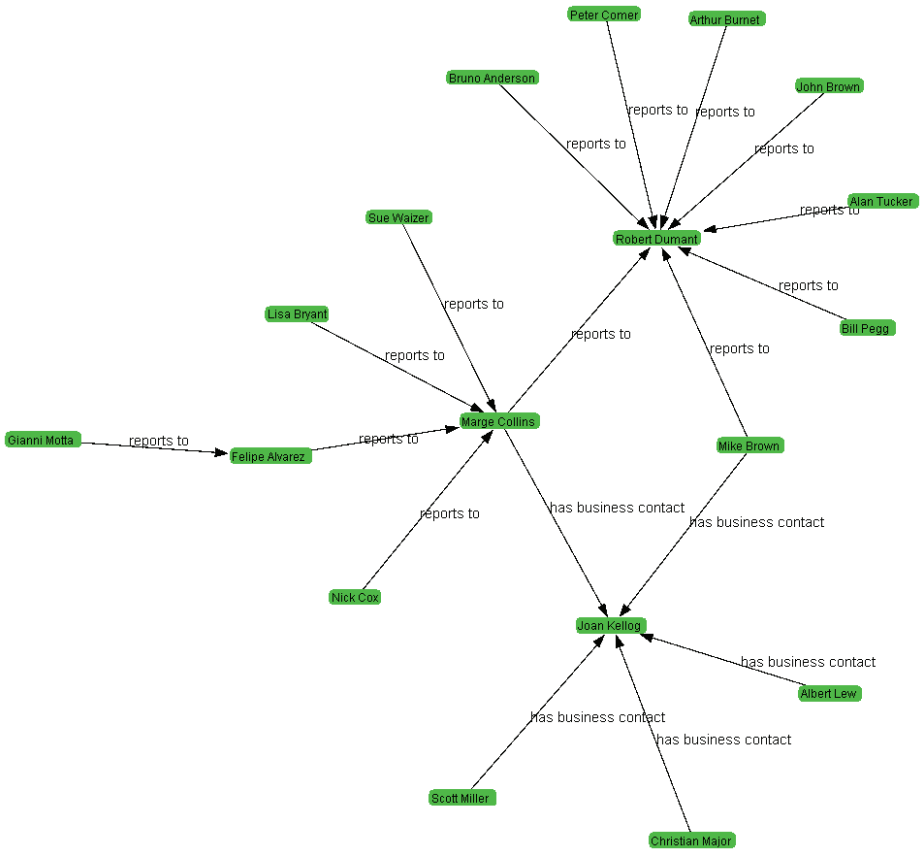

Figure III.4: Relations constituting the social network for the user `Marge`. Only users at a maximum depth of 2 are displayed.

relations to entities that are not further than `depth` from the current user. Only allowed types of `relations` are returned.

- `getDistance(userB)` returns the distance between the current user and `userB`, i.e., the number of edges between the two.
- `getManagers()` returns the user's managers, i.e., people she reports to. This method is a simple shortuct to extract relations starting from the current user, of type *reportsTo*.
- `getReports()` returns people that report to the current user. Again, this is a simple commodity method to return relations of type *reportsTo* and ending on the current user.

These last two methods are simple examples but other types may clearly be considered, e.g., to exploit *worksWith* and *hasBusinessContact* predicates.

3.2.3 Geography dimension

The `GeographyDimension` is another key dimension exposed in our model. We define the *hasLocation* predicate to handle geographical context information. It can be used to describe the user's location in statements like `Marge` *hasLocation* `(45.194, 5.733)` or `Marge` *hasLocation* `India`. The object (or target) of such statements are nodes of the `Geography` type which can be subclassed to define different granularity levels (`Country`, `Region`, `City`, etc.). In the `GeographyDimension`, two simple commodity methods may be used to retrieve – among all possible geography statements – those that describe the user's location:

- `getLocation(granularity)` can be used to get the agent's current location at the desired level of granularity, should it be available.
- `getFinestLocation()` is a shortcut to directly retrieve information about the user's location, at the finest granularity level available.

This dimension ensures consistency at different levels. Consider for instance a new statement that informs the system about the current user's city. If previous knowledge is conflicting with the new information, then other more fine-grained statements will be re-evaluated. Besides, in the next chapter, we will introduce operators responsible for dynamic situation management. In particular, a `ReverseGeocoding` operator may be used to determine the country, state, region, city and address from GPS coordinates. Such precise position is now very commonly available on modern mobile devices.

3.3 Personalization-related dimensions

Our aim with this model is to enable personalization in various *situational applications* and eventually provide recommendations of potentially heterogneous resources.

3.3.1 Preference dimension

First, different applications may be willing to exploit personal user preferences in order to adapt their content and/or design. It is worth noting, here again, that so called preferences are often dependant on the application scenario considered. For instance, a news reader aggregating RSS feeds may store preference scores associated to various sources of information. A music delivery platform may be interested by the user's favorite artists and albums, etc.

Concretely, users' preferences may be described using specific *preference statements*, that is a relation between the current user and a specific

resource, based on predicates like *prefers*, *rates*, *likes*, *dislikes*, etc. For instance, the user's favorite language for proposed content could be defined using a statement like `Marge` *prefers* `Locale-EN`, indicating that `Marge` prefers to read English RSS feeds. The origin of the statement can be used to distinguish preferences expressed by and for different applications.

The `PreferenceDimension` presents the main method `getPreferences(node)`, which returns all statements indicating (positive or negative) preferences regarding a given `node`. An additional *weight* attribute can be used with these statements to weigh or simply order preferences. Like all context information, preference statements can be either explicitly given by the user or implicit, e.g., derived from the analysis of the user's history.

Explicit preferences. Forms may be used, e.g., to let the user express her preference for different data sources. Users' feedback can be captured with statements like `userA` *rates* `item123` (weight = 0.8) or `Marge` *likes* `item123`.

Implicit preferences. Preferences of the user regarding certain resources may also be obtained implicitly, for instance derived from the analysis of the user's interactions (browsing history, click-through data, etc.). Derived preference statements usually have a lower confidence attribute than explicit ones. In Chapter V, Section 3, we define implicit preferences for OLAP entities like *measures* and *dimensions*. In this scenario, implicit preferences are determined using occurrences of these entities in charts or tables the user commonly uses.

3.3.2 Recommendation dimension

In the situation management framework which we will present in the next chapter, *operators* are components responsible for dynamic maintenance of situation graphs (see Chapter IV, Section 2.3). In particular, operators benefit from an access to the user's situation model to provide personalized and dynamically adapted recommendations, in response to events. The situation model we propose thus integrates recommendations as a core dimension.

Recommendation statements may be used by dynamic components like operators to present varied suggestions to the user. Recommendations can, for instance, be described thanks to *recommends* or *hasRecomendation* predicates, in statements like `operator1` *recommends* `item123` (origin = `operator1`) or `userA` *hasRecommendation* `GoToSleep` (origin = `operatorHealthCare`). Operators may use (or not) the confidence attribute to assign a score to the recommendation statement. Other simple descriptive statements may be added by operators to provide explanations or further information on the item being recommended, be it a document, an action, etc. For instance, in the previous example, `operator1` could have mentioned

the fact that `item123` *hasAuthor* `userB`, and `userB` happens to be a frequent collaborator.

The `RecommendationDimension` mainly exposes the method `getRecommendations(types, n)`. This method returns the `n` best ranked recommendation statements, suggesting resources of desired `types`.

4 Summary and discussion

We have seen how the aggregated situation model can be built out of atomic statements thanks to generic statement and graph structures. Statements may be grouped for convenience purposes in different dimensions. In this section, we discuss our graph-based situation model against the analysis framework proposed by Bolchini et al. [BCQ+07].

4.1 Modeled aspects

Key aspects of our model have been described in Section 3.2. In this section, we discuss these with the categorization proposed in the analysis framework previously mentionned.

Geography and time. These two aspects are often considered of particular importance. The *hasLocation* predicate can be used with various `Geography` nodes to describe the user's location at different granularity levels. The time aspect in an inherent part of our model through statements metadata which express time-dependent validity (timestamp and lifetime).

Subject and user profile. The subject of the model is the point of view adopted to represent information which, in our case, is user-centric. The user's profile is built from statements which describe her characteristics (personal features and preferences) and interactions. Bolchini et al. also mention the possibility for the model to expose user classes. As of today, we do not expose such classes in the proposed model.

Context history. Generated statements are managed by the Graph Repository, which acts as a memory and vehicle for context knowledge. Statements have time-related attributes and enable the reconstruction of a user's situation graph at any given point in time. Therefore, the current user's situation may depend on past context information, based on the application needs.

4.2 Representation features

Representation features are general characteristics of the model itself.

Type for formalism. The proposed model relies on a graph-based formalism. The choice of representation formalism is driven by its appropriateness to serve the application needs. Our main application is *data tailoring* with recommendations. Recommendations are meant to help information exploration and we consider graphs are not only applicable but also very well suited to back this scenario with semantics and high level of expressiveness.

Flexibility and formality. Our model flexibility lies in the possibility to define custom types of nodes and statements. Besides, predicates preside statements semantics and new ones can be created, e.g., to describe application-specific knowledge. However, non-formally defined predicates will be difficult to interpret and discourage information sharing.

Context granularity and constraints. Our model allows the representation of context information at different levels of granularity. This was in particular illustrated on the example of the *hasLocation* predicate used with different geographical nodes like `Country`, `City`, etc. Finally, context constraints can be expressed and continuously evaluated at runtime. For this purpose, activation rules will be presented with dynamic aspects in Chapter IV, Section 2.

4.3 Context management and usage

This section discusses the way the model is built, managed and exploited. However, these aspects are mostly addressed in our work thanks to components responsible for the dynamic maintenance of the platform. These aspects will be covered in more details in chapter IV.

Construction. The situation model is centrally maintained in the situation platform, by aggregating relevant knowledge from potentially distributed source systems (through providers).

Reasoning. Reasoning is considered by Bolchini et al. as the ability to "infer properties or more abstract context information" [BCQ+07]. In our framework, custom operators can be developed to interpret the semantics of specific entities and statements. These operators may produce additional knowledge by creating new statements. In that sense, the proposed framework allows reasoning on context data. As an example, an operator could determine the city from GPS coordinates with reverse geo-coding techniques or services.

Information quality, ambiguity and incompleteness. Quality monitoring is particularly critical when context information is acquired from physical sensors. However, it is not as crucial in our application

scenario (recommendations for business users in situational applications). This is why our framework delegates various tasks of quality monitoring to operators. For instance, an operator could be activated in case two ambiguous statements are detected, e.g., `Marge` *hasLocation* `position1` and `Marge` *hasLocation* `position2`, such that `position1` and `position2` are distant.

Automatic learning features. Machine learning techniques could be employed on history data, e.g., to detect interaction patterns and help perform an automatic categorization of situations. However, our work does not fully addres this possibility and could be considered in future research.

Multi-context modeling. Bolchini et al. define it as the representation"in a single instance of the model all the possible contexts of the target application". Our situation model exposes a single homogeneous graph of currently valid statements, categorized in major dimensions. Applications consuming this model (e.g., operators or platform client applications) are free to query and filter data to keep only facts of interest and adapt accordingly.

5 Conclusion

In this chapter, we introduced the requirements imposed on context or situation modeling systems, in particular in our use case dedicated to corporate deployment scenarios. These major requirements are meant to handle (a) applications inter-operability, (b) resources heterogeneity, (c) security and privacy and (d) dynamic adaptation. They impose conditions on the kind of system we are considering and led us to propose a graph-based situation model.

The unit of information in this model is represented by a homogeneous data structure called situation statements and first introduced by Heckmann et al. [Hec05b]. Situation statements extend simple subject-predicate-object triples – common with Semantic Web technologies like RDF – with additional metadata to capture time dependency, security and privacy, etc. We presented the graph repository (GR) framework to handle and manage graph data for different users, taking into account security aspects with actual user data isolation. The GR is populated thanks to an extensible set of providers which can connect and adapt to various remote systems.

Besides, we presented a higher level and more structured situation model. On top of the GR structure which represents a personalized knowledge container, an agent situation is composed of a graph of interactions, a profile and a certain number of dimensions. Various dimensions are exposed by our model and present categories of situation knowledge. Core dimensions

describe social and geographical aspects, whereas personalization-related dimensions capture the user's preferences and recommendations provided to the user.

In the next chapter, we will focus in particular on dynamic aspects of situation modeling and management. In particular, we will introduce extensions to the GR framework to define active components like activation rules and operators. Activation rules can be used to trigger appropriate operations in response to certain types of business events, under varying conditions.

Chapter IV

Dynamic Situation Management Framework

Contents

1 Introduction

In the previous chapter we introduced our graph-based situation model, relying on the homogeneous statement structure of the *Graph Repository* (GR). Besides, we defined situation dimensions to expose aggregated and structured sets of statements. We presented in particular social, geography, preferences and recommendations dimensions but others may be considered for specific use cases.

In this chapter, we focus on dynamic situation management, that is how to continuously monitor agents' interactions and help maintain situation

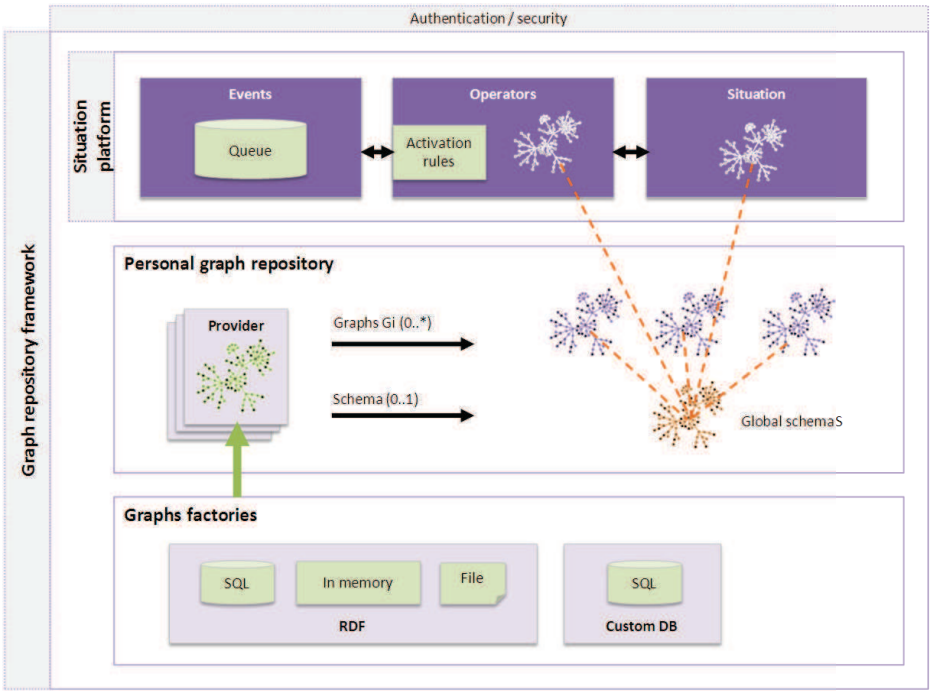

Figure IV.1: Architecture overview of the situation management platform and its major components: events, activation rules and operators.

graphs as events occur in their environments. For this purpose, we propose a modular and extensible framework based on the following key notions: *business events*, *activation rules* and *operators*. These components are represented in the general architecture diagram of Figure IV.1.

This section presents business events and the *event-condition-action* (ECA) model, used to define active rules in event-driven architectures. Section 2 discusses our implementation of this model, in particular the expression and evaluation of activation rules to trigger operators. Then, we review in Section 3 the modular architecture of our framework, from the underlying GR to more dynamic components.

1.1 Buisness events and situation dynamics

Business events are at the core of dynamic situation management. Events may describe characteristics of the user's environment – for instance the user's location in a statement like `Marge` *hasLocation* `India` – or interactions with existing resources – like `Marge` *read* `document1`. Events can be raised by internal components of the platform or sent by client applications to contribute to the aggregated view of a user's situation. Figure IV.7 shows examples of client applications. As an example, an add-in for a commonly used email client could be designed to submit events similar to `Marge` *read*

document1. In the end, the situation management component needs to react to these events to take appropriate actions in impacted situation graphs.

The examples above seem to indicate that an event is fairly similar to a statement, that is a qualified interaction between resources. In some cases though, an event may need to convey more semantics to be meaningful, e.g., in the form of background knowledge. For instance, the statement Marge *read* document1 could be completed with additional information about the author of the document, as with the statement document1 *hasAuthor* Nick.

Therefore, we can distinguish between *atomic* (or simple) events and *complex* events. An atomic business event is composed of a single statement. On the other hand, we define a complex business event as the graph formed by a collection of statements which represent an interaction plus additional information. The RDF listing below presents a sample business event which actually contains two statements, Marge *read* document1 and document1 *hasAuthor* Nick:

```
<rdf:RDF xmlns...>
  <rdf:Description rdf:about="http://.../statement123">
    <rdf:type rdf:resource="http://...#Statement"/>
    <rdf:subject rdf:resource="http://.../marge"/>
    <rdf:predicate rdf:resource="http://.../read"/>
    <rdf:object rdf:resource="http://.../document1"/>
    <grepo:origin rdf:resource="http://.../EmailClient"/>
    <grepo:timestamp>12345</grepo:timestamp>
    <grepo:lifetime>5</grepo:lifetime>
  </rdf:Description>
  ...
  <rdf:Description rdf:about="http://.../statement127">
    <rdf:type rdf:resource="http://...#Statement"/>
    <rdf:subject rdf:resource="http://.../document1"/>
    <rdf:predicate rdf:resource="http://.../hasAuthor"/>
    <rdf:object rdf:resource="http://.../Nick"/>
    <grepo:origin rdf:resource="http://.../EmailClient"/>
    <grepo:timestamp>12346</grepo:timestamp>
    <grepo:lifetime>500</grepo:lifetime>
  </rdf:Description>
</rdf:RDF>
```

1.2 The ECA framework

In event-driven architectures, the *event-condition-action* (ECA) structure can be used to define active rules, with clear declarative semantics [BFMS06]:

ON *event* IF *condition* DO *action*.

Rules are used to react to specific *events* and trigger additional processing (*action*), assuming some other *conditions* are met. We use this ECA framework to define active rules in our system. Let us briefly introduce the three components of such rules:

Event. The *event* part is used to filter events the activation rule will react to. For instance, a given rule could react to events which match the interaction pattern "a user is reading an unstructured document".

Condition. The *condition* part allows to express additional conditions which can be evaluated thanks to queries on the GR. The GR thus serves as a vehicle for contextual information. Various conditions may be expressed, for instance dependencies between agents as in "`agentB` applies if `agentA` *hasProcessed* `document1`".

Action. Eventually, the *action* part is used to determine which operations or modifications should be applied to situations (and more generally graphs), in response to the event being interpreted.

We define in our framework two important components to react to business events, *activation rules* and *operators*. The actual expression of rules and their evaluation is further discussed in Section 2. A rule positively evaluated triggers the execution of an operator (see Section 2.3).

2 Activation rules and operators

In this section, we present *activation rules* which are the application of the ECA framework previously presented to define rules in our dynamic situation situation management framework.

2.1 Rules expression

We introduce the semantics and expressiveness of activation rules based on the ECA framework, meant to help define active rules in event-driven systems.

2.1.1 E - Filtering events

In an ECA rule, the *event* part enables a first level of filtering on events the rule may react to. Events may represent very different types of interactions plus potential background knowledge, and it is likely that only few rules are concerned by a given business event. Event filtering is thus an important aspect during the rule evaluation process as it helps dismiss some rules in an efficient way, by reducing actual evaluations to the minimum necessary. Rules evaluation may indeed become costly when the number of

rules increases or complex querying is involved. The rule evaluation process is further described in Section 2.2.

Therefore, the first part of an activation rule is an event filtering pattern. An event filtering pattern is defined as a combination of four conditions imposed on the subject, the predicate, the object and the origin of statements, noted $C_{subject}C_{predicate}C_{object}, C_{origin}$. Let us consider a first event filtering pattern, which keeps only those that introduce the geographic location of a entity, for instance in a statement like `Marge` *hasLocation* `India`:

`ANY` *hasLocation #GeographyNode*, `ANY`

In this example, the reserved keyword `ANY` is used to denote an absence of condition imposed on the subject and the origin. The only accepted predicate will be *hasLocation*. It may be noted that we leave out full URIs in examples for the sake of clarity. The object condition filters out nodes which type is not `GeographyNode`. This type is one of those accessible through the global *Graph Repository* schema, introduced in Chapter III, Section 2.2.3. In particular, this schema may be augmented by providers of the GR and enables the definition of hierarchies of types. `Country` and `City` are two examples of types defined as children of the `GeographyNode` type.

Let us now consider a more precise event filtering pattern to only react to events that describe the location of the current user. For this purpose, we introduce the `USER` keyword which refers to the current for whom a rule is being evaluated:

`USER` *hasLocation #GeographyNode*

In this example and because there is no specific constraint on the origin, the corresponding condition can be omitted. When the condition on the origin is not omitted, it is possible to restrict, e.g., to information emerging from mobile application agents:

`USER` *hasLocation #GeographyNode*, *#MobileApp*

2.1.2 C - Condition expression

On top of conditions imposed on the event itself, an activation rule may define additional constraints for the execution of a given operator. Let us present briefly examples of conditions that may be used to define such constraints in our framework.

An *execution guard* is a simple condition preventing the execution of the operator. Execution guards can be manifold, from simple authorization to more complex dynamic operationalconstraint. Below is a first condition example that enables the execution of an operator if and only if the current user explicitly authorized it:

`USER` *authorizes* `operatorA`, `USER`

This example is particular since rules do not have to make it explicit. The authorization control is indeed taken care of by the dynamic management framework before the operator is executed. Another example of condition may be used to avoid an operator being executed twice in a situation, in response to a single event.

`NOT(operatorA` *hasProcessed* `EVENT.object`, `operatorA)`

The `EVENT` keyword is used to denote the event being evaluated or interpreted. The four parts of the corresponding statement are obtained through the following accessors: `EVENT.subject`, `.predicate`, `.object` and `.origin`. It is also worth noting that using this kind of condition to express a dependency is inherently dependant on the assumption that `operatorA` will create such a statement when it has finished its processing.

Similarly, activation rules may also be used to define constraints in the sense of *operators dependencies*. For instance, if an `operatorB` requires the execution of a previous `operatorA` to first perform some operations on the event target (or object):

`operatorA` *hasProcessed* `EVENT.object`

2.1.3 A - Taking action

In the ECA framework, the *action* part of a rule is meant to determine actions to be taken in response to the initial triggering event. These actions can be manifold but often, in the fields of active databases or Semantic Web technologies, the action consists in data insertion, modification or deletion. In our framework, activation rules are meant to trigger the execution of operators which will be presented in further details in Section 2.3

As we previously presented, activation rules are continuously evaluated in response to events thanks to queries executed on the GR. An activation rule triggers the execution of a given operator with additional parameters.

To define the action part of our rules, we introduce two reserved keywords, `CALL` and `WITH`, used with the following syntax:

`CALL` *operator* `WITH` *parameters...*

Should a rule be positively evaluated, the specified *operator* is executed with the following *parameters*. Parameters of this list (separated by commas) result from the event filtering and condition evaluation process, using bound variables. Rules evaluation is discussed in the next section.

Considering the previous ongoing rule example, we now illustrate the call to a *stemming* operator, taking as single parameter the unstructured document mentioned by the event (in the corresponding statement object):

CALL *stemming* WITH EVENT.object

Stemming is a text analysis operation which consists in reducing words to their roots. Other operators may be developed to perform other text analysis operations like named entity recognition (NER). To conclude, below is a complete example of rule used to activate a stemming operator when the current user is reading an unstructured document (e.g., an email).

```
<rule>
  <event>
    USER "http://.../read" "http://.../#UnstructuredDocument", ANY
  </event>
  <condition>
    NOT("http://.../stemming" "http://.../hasProcessed" EVENT.object)
  </condition>
  <action>
    CALL "http://.../stemming" WITH EVENT.object
  </action>
  <description>
    Stems an 'unstructured document' the user is reading
  </description>
</rule>
```

It is worth noting that new rules can be added like modules or plugins, most commonly by developers providing new components to the situation platform. Rules are created to define a sequence of operations to perform during the interpretation of a given type of event. Thanks to their loosely coupled integration with operators, rules may also be defined later on, e.g., as new use cases emerge for a certain operator.

2.2 Rules evaluation

In the previous section, we presented the simple expression language we use to define activation rules and trigger appropriate operators in response to various events. We now describe how rules are actually being evaluated in our dynamic system.

Events occur and are submitted in the scope of the user's GR. Rules are thus evaluated for one given user. The evaluation process is divided in two phases for performance reasons, the event filtering and the evaluation of additional conditions:

Event filtering. Events can be filtered without the cost of a query on the GR, by simply validating conditions on the event object being processed (or interpreted).

Condition evaluation. On the other hand, the evaluation and validation of additional conditions requires to query the GR. As previously discussed, the evaluation cost increases with the number of rules and the complexity of resulting queries.

During the rule evaluation process, `USER` and `EVENT` markers of the language are dynamically replaced by the system with URIs of the current user and the event being processed. Similarly, markers of event components (`EVENT.subject`, `EVENT.predicate`, etc.) are determined by the system before evaluation.

Additional conditions of a rule are evaluated thanks to queries on the GR, which serves as a vehicle for context information or background knowledge. Concretely, the type of query and the way it is being executed depends on the underlying graphs factories (see Chapter III, Section 2.2.4). Indeed, a graph backed by a relational database will eventually be queried using the SQL language. On the other hand, an RDF triple store can be queried using the SPARQL query language.

Therefore, in our framework, a special component is responsible for the transformation of an activation rule, from the custom format (described in this section) to a SPARQL or SQL query. This component is the `ActivationRuleEvaluator` and its use in the event interpretation workflow will be further discussed along with other components in Section 3.2.3.

Let us consider a rule triggering the named entity extraction (`NER`) operator in reaction to events of the type "the user is reading a document authored by a person she manages, and the `stemming` operator has processed this document first":

```
<rule>
  <event>
    USER "http://.../read" "http://.../#UnstructuredDocument"
  </event>
  <condition>
    EVENT.object               "http://.../hasAuthor"    ?author
    AND ?author                "http://.../reportsTo"    USER
    AND "http://.../stemming"  "http://.../hasProcessed" EVENT.object
  </condition>
  <action>
    CALL "http://.../ner" WITH EVENT.object, ?author
  </action>
  <description>
    Applies NER operator if the user is reading a document
    authored by a person she manages, and the document
    has been stemmed first.
  </description>
```

```
</rule>
```

In the example above, the event filter assumes that the event itself does not contain information regarding the author of the document. The event filter can thus simply be validated without the cost of a query to the GR. However, the condition validation imposes to query the GR and the example illustrates the use of a bound variable `?author` to form a complex pattern, in an approach similar to SPARQL queries. We illustrate below the SPARQL query which can be obtained by transforming the condition of the previous activation rule:

```
PREFIX rdf :   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX grepo : <http://www.sap.com/graphrepo#>

SELECT ?author
WHERE {
  EVENT.object    grepo:hasAuthor     ?author
  ?author         grepo:reportsTo     USER
  grepo:operatorA grepo:hasProcessed  EVENT.object
}
```

The result of the evaluation for this query is a value (or set of values) for the `?author` variable. The evaluation fails if there is the result set is empty. Otherwise, the NER operator is invoked with parameters resulting from both event filtering and condition validation phases.

2.3 Operators

We presented in details the language used to express rules and how they are evaluated on the GR in reaction to events, with actual queries. In this section, we define and discuss operators, which execution is triggered by activation rules with parameters resulting the rule evaluation process.

2.3.1 Definition

Operators are key components which maintain situations graphs (and more generally graphs of the GR) by adding, updating or deleting statements. These operations represent either explicitly declared facts (e.g., `Maggie` *reading* `document1`) or knowledge resulting of additional processing (for instance a NER result, like `document1` *mentions* `entity1`).

An operator *op* takes as input a situation graph G_{sit}, an event e and parameters *params* to return the updated situation G'_{sit}:

$$op : (G_{sit}, e, params) \rightarrow G'_{sit}$$

Table IV.1: Examples of operators. This include system operators and specific ones related to some of our experimentations (see Chapter VI).

Operator	Description
Expiry checker (system)	Checks statements of a given situation to determine and remove those that are outdated. Outdated statements are simply removed from the current view but persisted to constitute historical data.
Geolocalize	Determines the approximate GPS coordinates (latitude and longitude) of a given geographic entity, which can be a precise address, a city, a country, etc.
GeoIP	Determines the city from an IP address. Various external web services can easily be used for this purpose.
ReverseGeocoder	Reverses GPS coordinates to determine the closest address or more simply the city, department, state, etc.
Stemmer	Applies stemming to unstructured resources. Stemming consists in text normalization and reduces words to their roots. For instance, the sentence "it is raining cats and dogs" would be reduced to "it be rain cat and dog".
NER	Performs Named Entity Recognition on text resources, extracting entities with various dictionaries. Extracted entities are mapped to objects of the data warehouse the user can see.
Query recommendation	Combines results of entity extraction with the semantics of a business domain to suggest meaningful queries. These last 3 operators will be further discussed with the Text-To-Query system, in Chapter VI, Section 2.

From the framework standpoint, similar to providers for the GR, operators are defined in a modular way. The situation platform manages a pool of registered operators which can be developed to interpret the specific semantics of various providers and/or events. Implementation wise, new operators can easily be developed and registered as plugins, according to the architecture presented in Section 3.2.2.

2.3.2 Examples of operators

Many operators can be considered to perform a broad range of tasks. Table IV.1 lists examples of operators implemented throughout our various experimentations, further described in Chapter VI. Operators can be combined and reused so that more complex operations can be divided and im-

plemented in a modular way. Operators can be developped and added to the platform like other types of modules or plugins, as will be discussed with the architecture of the small GR in Section 3.1.

In this table, operator 1 (`ExpiryChecker`) is a system operator and is triggered by internal events to perform graph cleaning, removing outdated statements from situation graphs. Operators 2-4 are operators meant to handle various types of geographic information processing. In particular, they are used to gain more information about the location of particular entities. Operators 5-7 are related to text processing and the query recommendation engine *Text-To-Query* presented in Chapter VI, Section 2.

3 Situation framework and services

In the previous chapter we introduced our graph-based situation model, the requirements imposed on our system and the foundations of our framework, with the GR. In this chapter we developed dynamic aspects of the system, reacting to business events thanks to dynamic components, namely activation rules and operators.

The overall architecture of the complete system was illustrated in Figure IV.1. Beyond this overview, this section describes in further details the actual architecture of our framework. This proves particularly important since the framework we are providing aims at being consumed and enriched by further developments.

3.1 Graph repository framework

The *graph repository* is a major component upon which the situation management platform is built, as represented in Figure IV.1. From an internal framework perspective, the architecture of the GR and its most important components are depicted in the class diagram of Figure IV.2.

3.1.1 Platform and plugins modularity

At the highest level of our architecture is the `Platform`. The most important role of the platform is to enable the desired level of modularity, in particular for the following components that have been introduced in our system: providers, graphs factories and later with dynamic aspects, operators and activation rules.

Implementation wise, our platform is embodied as a set of bundles (or plugins) running in an OSGi environment. OSGi – or Open Services Gateway initiative framework – is a powerful dynamic component management system. Applicative components are described in the form of bundles and can be remotely installed, started, stopped, updated and uninstalled dy-

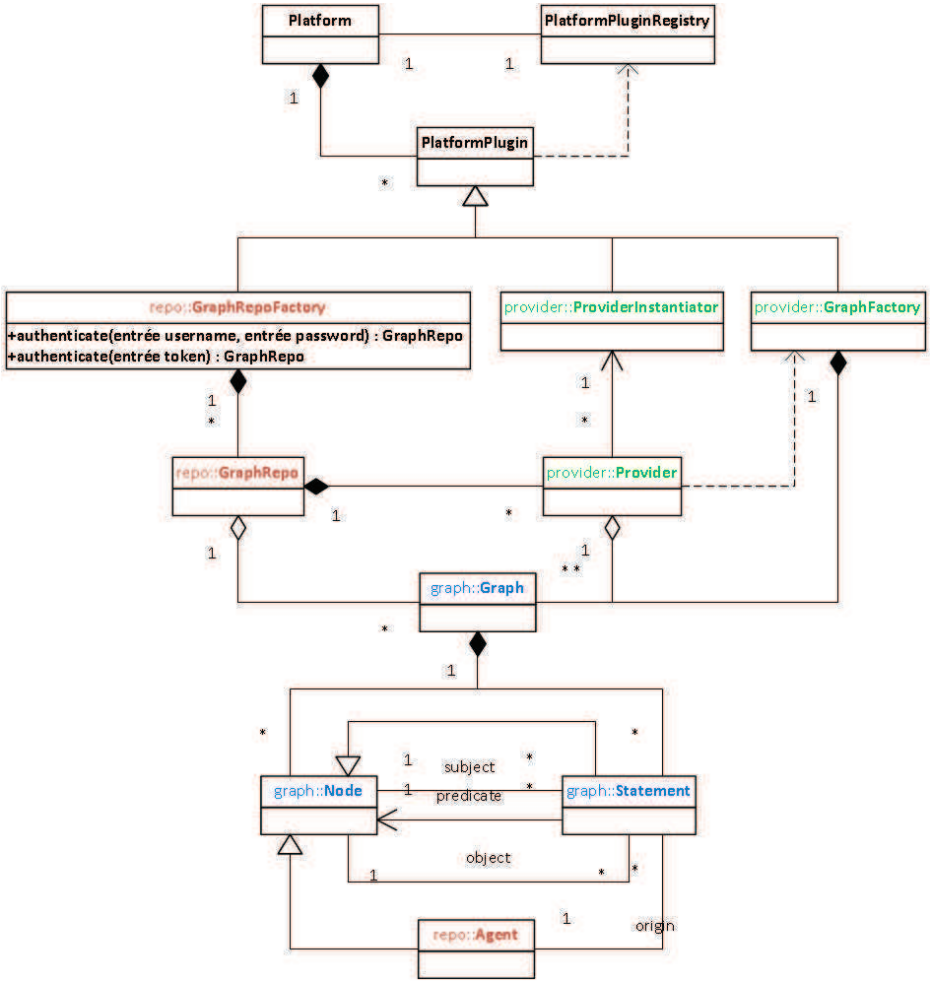

Figure IV.2: Class diagram - Core components of the modular *Graph Repository* architecture. Providers and graphs factories are modules which can easily be extended.

namically without interruptions[1]. Besides, the OSGi specifications define a shared service registry which allows bundles to publish and access services. Our framework bases on the Eclipse Equinox[2] reference implementation of OSGi R4 specifications.

Concretely, in line with the class diagram of Figure IV.2, our `Platform` is the OSGi runtime. In such a runtime, the `PlatformPluginRegistry` may easily be implemented using the OSGi bundle context which gives access to the services registry. Besides, it may be noted that all elements pushed in our system architecture are actual instances of `PlatformPlugin`. We now introduce the higher level implementation of the GR based on capabilities of the `Platform`.

3.1.2 Personalized and secure graph repositories

First and foremost, the `GraphRepoFactory` is the entry point to the GR framework deployed in the platform previously described. All plugins installed and registered in the platform may access the GR factory service to retrieve personal graph repositories, through an authentication process.

Authentication. The `GraphRepoFactory` component handles agents' authentication. However, this responsibility may be delegated to another dedicated authentication component, e.g., higher at the `Platform` level. The result of a favorable authentication is a token used to create and manage personal `GraphRepo` instances for different users – or more generally agents – logged in the system.

For each authenticated user, the `GraphRepoFactory` creates a dedicated `GraphRepo`, initialized with registered providers and graphs factories. These providers are meant to populate graphs of specific resources accessible to the user and may connect to remote source systems (see Chapter III, Section 2.2.1). It is important to note that a `GraphRepo` is initialized with providers, instantiated by the factory according to a security model that we now present.

Authorizations and credentials. Providers are meant to feed graphs of data which may be of interest to personalize services and information for the current user. For the sake of security and privacy, users need to be able to allow or reject the use of some providers, according to an explicit *opt-in* approach. As an example, a user may accept a provider connecting to her account in a BI platform but may on the other hand reject the CRM provider. These authorizations are represented by the relation `User` *authorizes* `ApplicationAgent` in Figure IV.3.

[1] http://en.wikipedia.org/wiki/OSGi
[2] http://www.eclipse.org/equinox/

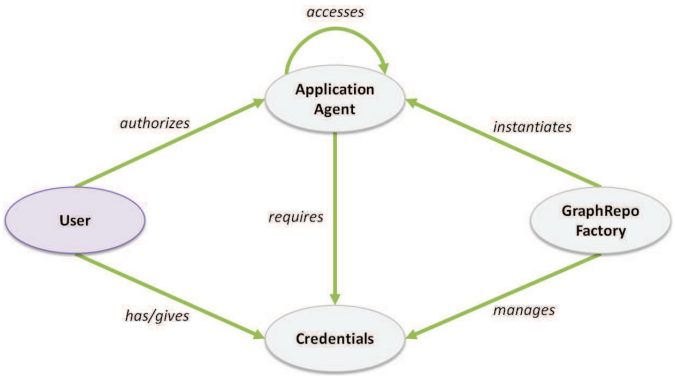

Figure IV.3: Security model of the `GraphRepoFactory`, managing credentials for different applicative agents authorized by users.

More generally, this figure illustrates the security model adopted by the platform to enable or not the instantiation of various application agents (plugins). Including dynamic components described in this chapter, we have introduced the following types of plugins: providers, graphs factories, operators and activation rules. Thanks to authorizations, users can control the composition of their own personalized `GraphRepo`. In case an application require the user's credentials to connect to a remote system, Figure IV.3 indicates that credentials are given by users and managed by the `GraphRepoFactory`. Another important aspect described in the model is that an application agent may leverage data provided by other agents. In such case, the provider must declare its dependencies, represented by the relation *accesses* in the model. The GraphRepoFactory determines the applicability based on the user's authorizations and available credentials. Finally, it may be noted that some providers could expose generally accessible (or public) data and would therefore not require credentials.

3.1.3 Providers and graphs factories

The GR architecture presented in Figure IV.2 introduces `Provider` and `GraphFactory` components. As previously discussed, providers are instantiated by the graph repository factory according to users' authorizations. A provider is instantiated in the scope of a personalized `GraphRepo` and is meant to represent personal user data or any other background knowledge.

The `GraphFactory` is an important component which handles the actual creation and management of graphs. This interface and our containing framework are meant to expose source-agnostic graphs. This provides an abstraction above the actual data storage, be it a relational database, a dedicated RDF triple store, etc. As a storage facility, the `GraphRepoFactory` is responsible for usual *create*, *read*, *update* and *delete* (CRUD) operations on

graphs.

In particular, we provide an RDF-based implementation which enables the creation of graphs managed in-memory, in files or by a relational database. Besides, data can be queried using the SPARQL language but we also expose a certain number of convenience methods to iterate over filtered nodes or statements. This implementation is enabled by the popular library for the Semantic Web, Jena[3].

3.2 Situation management framework

In Section 2 of Chapter III, we defined the user's situation as a collection of currently valid statements. The time aspect is an important aspect and therefore, situation graphs are meant to evolve at a possibly rapid pace, in response to events.

We previously discussed the foundations of the GR framework. Let us now elaborate more on dynamic components of our situation management framework. Figure IV.1 illustrates the three building blocks of the architecture of our situation platform: situations, operators with activation rules and events. In this section, we describe these aspects, with respect to the class diagram of the framework depicted in Figure IV.4.

Let us briefly remind the roles of components responsible for dynamic maintenance of the situation model described in Chapter III:

Events. An `Event` is a statement (or collection of statements) raised in a given situation, representing interactions plus additional knowledge. Events are queued to be processed asynchronously by an `EventProcessor`.

Activation rules. An activation rule triggers an operator in response to an event, assuming some other conditions are met. Rules can be defined in any plugin and are mainly evaluated thanks to queries on the GR.

Operators. As a result of rules evaluation, an operator may be executed with certain parameters. The operator is then able to update the situation graph in response to the triggering event.

3.2.1 Situation provider

We defined situations as specific graphs centered on the concerned agent. As all graphs in the GR framework, a `Situation` is therefore created and managed by a particular provider, the `SituationProvider`. This provider creates situation graphs using the RDF-based factory, backed by a relational database for data storage or simply by a pure in-memory model. These

[3] http://jena.sourceforge.net/

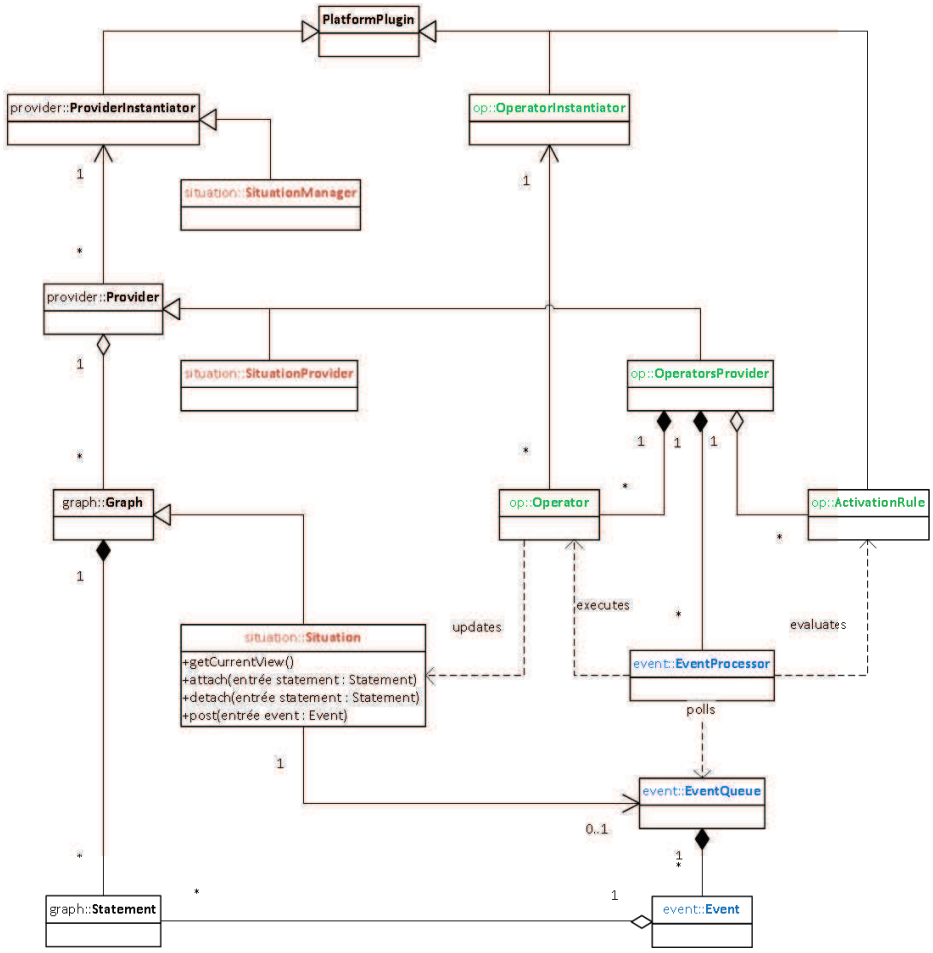

Figure IV.4: Class diagram - Overview of the situation framework. The main dynamic components are events, activation rules and operators.

graphs contain all nodes and statements that contribute to define the dynamic view of situations, obtained through the `getCurrentView()` method exposed in our object model (see Chapter III, Section 3).

The current view is dynamically mainted by operators which add, udpate or remove statements from situations. The term *remove* matters here as statements are not actually deleted. Instead, the situation provider manages statements lifetime accordingly and old statements are stored for history purposes. In our homogeneous model, it is possible to query for both currently valid and historical data. Concretely, it is possible to evaluate a certain query at any time (now or in the past) using dedicated *timestamp* and *lifetime* attributes.

Among operators that dynamically maintain situation graphs, the `ExpiryChecker` introduced in Table IV.1 scans statements of the current view to determine and remove outdated ones. The `SituationProvider` is responsible to trigger this operator by raising internal events at a determined frequency, e.g., every minute. An example of such event could be expressed as follows:

`SituationProvider` *requires* `ExpiryChecker`

For the sake of completeness, we illustrate below the complete rule that triggers the `ExpiryChecker` operator in response to this event. The operator does not require any parameter so `WITH` *parameters* is omitted in the action part of the rule:

```
<rule>
  <event>
    "http://.../situation-provider" "http://.../requires"
    "http://.../expiry-checker", "http://.../situation-provider"
  </event>
  <condition/>
  <action>
    CALL "http://.../expiry-checker"
  </action>
  <description>
    Calls expiry checker upon request by situation provider
  </description>
</rule>
```

3.2.2 Operators and activation rules

Operators are very similar to providers in that their instantiation is handled by the `GraphRepoFactory` according to the previously discussed security model, illustrated in Figure IV.3. Operators are implemented as modules or plugins and they are instantiated in the scope of a given `GraphRepo`, so

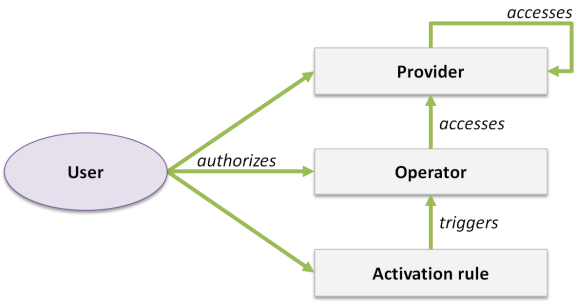

Figure IV.5: Three levels of authorizations and personalization for users. Providers, operators and activation rules may be enabled (or disabled) to compose personalized and dynamic GR.

they apply to the concerned user's situation. Users can authorize or reject an operator and give credentials, should the operator require some. Indeed, just like providers, operators may also connect to remote systems on behalf of the user, for instance to use external services.

Operators instantiated by the `GraphRepoFactory` are managed by the `OperatorsProvider` along with other `ActivationRule` plugins. Plugins defining activation rules are more simple since they are instantiated only once, not for each user or agent. The rule evaluation process in response to business events is further discussed in the next section. A rule is supposed to trigger an operator and thus requires that the user has authorized this operator. Besides, we add another level of security and privacy by enabling the same authorization pattern at the rule level. To sum up, the three levels of authorizations – providers, operators and activation rules – are summarized in Figure IV.5 and enable a full customization of the user's GR.

3.2.3 Events management and rules evaluation

Finally, we present components of Figure IV.4 responsible for events management and the actual rules evaluation process. Figure IV.6 illustrates in a sequence diagram how major components of our system interact to proceed to asynchronous events interpretation.

An `Event` can be posted by any agent in a given situation. This event is then added to the `EventQueue` – attached to the situation – so it can be processed later on. The queue is regularly polled and events are processed by a variable number of `EventProcessor` in an asynchronous manner. Processors access the pool of `ActivationRules` available through the `OperatorsProvider`.

The actual rule evaluation process is handled by the `ActivationRuleEvaluator` which uses transformations to obtain, e.g., SPARQL queries. Rules evaluation and their translation into SPARQL (or

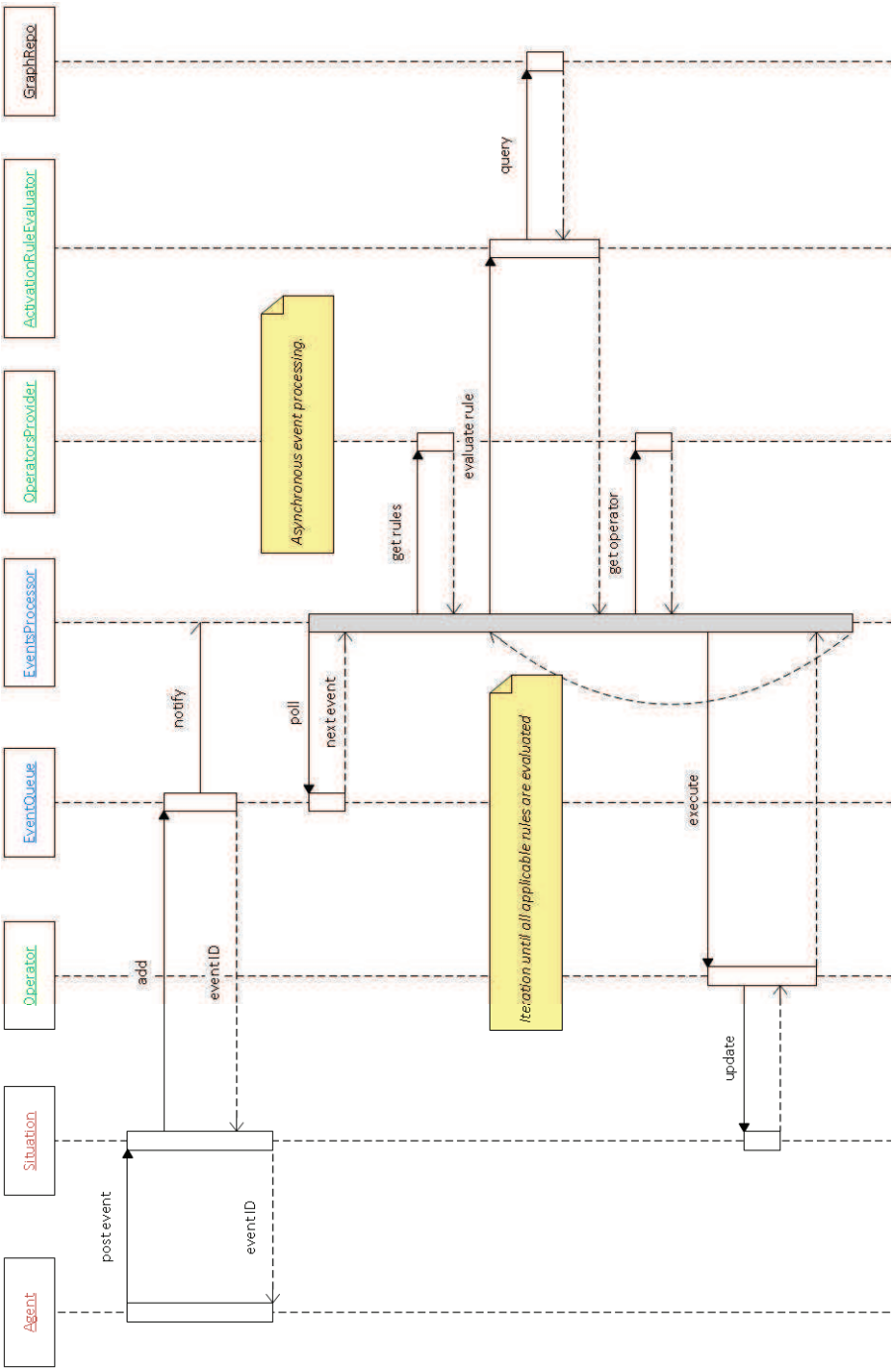

Figure IV.6: Sequence diagram for rules evaluation in reaction to an event posted in a given situation. The `EventsProcessor` polls the queue until it is empty to evaluate rules thanks to queries on the GR.

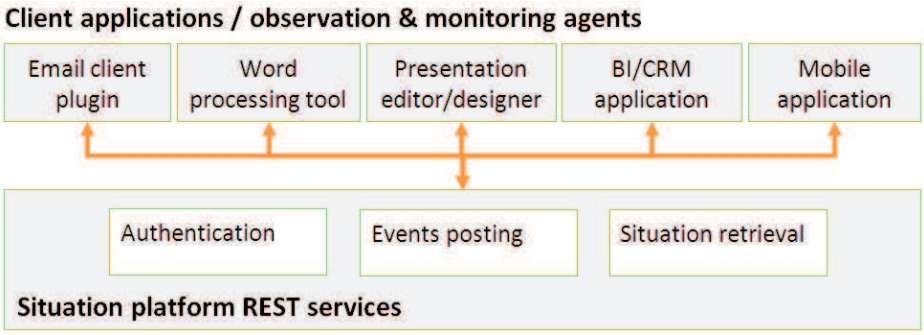

Figure IV.7: Examples of client applications interacting with the situation management platform to provide personalized and dynamic adapatation.

SQL) queries have been discussed in more details in Section 2.2. Assuming a given rule is positively evaluated, the targeted `Operator` is executed to update the `Situation` in response to the initial triggering event.

3.3 Client situation services

Previous sections described our GR and situation management frameworks from an internal perspective. In this section, we briefly present services exposed to client applications. Figure IV.7 illustrates examples of client applications which interact with the three main services or methods exposed: *authentication*, *events posting* and *situation retrieval*.

Implementation wise, the situation management bundle embeds a simple web application. This application exposes REST services which can be consumed as JSON or XML and bridges with selected capabilities of the underlying framework and situation model. Functionalities exposed by the client API are clearly limited compared to the full situation model which can be retrieved *in-proc* within the OSGi runtime.

Authentication. The `/auth` method (GET) is used to let users authenticate. It takes as parameters the user's credentials and it returns a token, required by other methods to access and query the personalized GR.

Situation retrieval. The `/situation` method (GET) returns the list of statements which constitute the user's current situation graph. Additional parameters may be used to filter returned statements and retrieve only facts of interest (e.g., in a certain dimension).

Events posting. The `/event` method (POST) lets clients contribute to the description of users' interactions by sending events. The body of the request describes the event itself, in the RDF format for improved inter-operability.

4 Summary and discussion

In the previous chapter we introduced a graph-based situation model to enable a homogeneous representation and exploitation of varied resources and interactions. We presented the foundations of the GR framework to meet our requirements, in particular in terms of security and modularity. In this chapter, we first introduced dynamic components that compose our situation management framework, implemented as extensions of the GR framework. Most importantly, activation rules and operators are active components used to react to business events, be they submitted by client applications (e.g., using REST services) or raised internally. Moreover, we discussed the expression and evaluation of activation rules in Section 2. These rules are described in a specific format and are used to trigger various operators. Operators can be developed to perform a broad range of operations in order to maintain situations (or more generally graphs). Then, we described in more details the actual architecture of our GR and situation management frameworks. In particular, we implemented the GR foundations in an OSGi runtime to enable a high level of modularity, with plugins that can be dynamically installed, started and stopped.

In Section 2.1, we described the expression of activation rules, using a specific format then translated into queries for the GR. This custom format structures rules using the ECA approach. In particular and beyond the initial *event* filtering, the *condition* part of a rule defines additional constraints that need to be verified to perform the *action*, implemented as an operator. Our aim was to maintain the storage independance and define an intermediary format for rules, in particular the *condition* part. However, it rapidly appeared that rules would greatly benefit from an expressive graph query language, for instance using bound variables, very briefly introduced in the last example of Section 2.2. Should the graph storage be implemented as a triple store, the SPARQL query language is particularly well adapted. Our initial attempt to define an intermediary language appeared to be an impediment for rapid and iterative improvements, as our reference implementation of the GR was RDF-based. It would have been more productive and less redundant to leverage to SPARQL queries and use the full expressivity of this language from the very beginning.

The continuous rules evaluation process described in Section 3.2.3 enables both linear (or deterministic) and non-linear evolution scenarios. Linear evolution scenarios are those which produce a determined sequence of operations in response to a given event. As a simple example, a linear evaluation scenario will be illustrated by the combination of several operators in Chapter VI, Section 2. In this example, three operators are activated in a pre-determined order in response to events of the type `Marge` *read* `document1`. The mentioned document is first stemmed before entity extraction techniques can be applied and these results are eventually com-

bined to recommend queries related to `document1`. In non-linear evolution scenarios, operations performed may not be determined in advance as continuous rules evluation may not give the same result twice, assuming some conditions changed during the evaluation. Although non-linear evolution scenarios would be extremely interesting to consider, it appeared difficult to showcase interesting real-life scenarios. Indeed, such a scenario would most likely involve an important degree of integration with a variety of source systems, and thus lead to important development and demonstration costs.

Even though it was not precisely discussed in this chapter, reasoning is an important feature enabled and exposed by the GR framework. Components developed in the context of this framework may use various capabilities like transitive closure or custom rule-based reasoning. Reasoning can be helpful in some case, e.g., to infer additional knowledge from existing statements which may be incomplete. However, we do not enable systematic reasoning on generated graphs since it can be extremely costly and represent an important burden for scalability. Therefore, the framework allows the construction of graphs extending existing ones in order to control thoroughly the reasoning process, for instance using volatile (purely in-memory) graphs. As an example, in Section 2.3 of Chapter V, we will present the use of minimal OWL and rule-based reasoning techniques in the context of multi-dimensional domain models (OLAP). Besides, we will discuss the reasoning performance with regards to two different implementations to insist on the importance of performance-related considerations.

5 Conclusion

In conclusion, we presented in this chapter extensions of the GR framework introduced in Chapter III. These extensions provide a set of extensible components to enrich the GR with dynamic behavior in an event-driven approach. Activation rules are used to react to varied business events and trigger appropriate operators. Business events are composed of one or more statements (in a graph) and may describe an interaction plus related knowledge. Activation rules are expressed according to the ECA framework and defined using a specific format. These rules are interpreted at runtime during the evaluation process and translated into queries on the GR. Should the GR reference implementation base on RDF graphs, queries may be expressed using the SPARQL graph query language. Positively evaluated rules trigger with resulting parameters the execution of appropriate operators. Operators are components responsible for the dynamic maintenance and adapation of situations (and more generally graphs). Like providers, the exploitation of activation rules and operators is controlled by the security model of the GR framework. Users' authorizations apply at these three different levels to eventually allow the definition of *personalized and active graph repositories*.

We reckon that the combination of the core GR framework with dynamic components brings a very interesting perspective, for personalization and dynamic adaptation in heterogeneous environments. As part of future work, we consider the following two directions are of particular interest and should be investigated. First, if linear evolution scenarios are already valuable, non-linear ones would more extensively demonstrate the range of capabilities offered by our dynamic situation management framework. We thus consider an important focus of future research should be on the definition and validation of such scenarios. Secondly, as a result of more complex scenarios being considered, the language used to express activation rules should be enriched and simplified, for instance by basing on regular SPARQL syntax. Besides, given the aggregagted nature of the GR, conditions of rules may have to be verified thanks to queries on multiple graphs. We did not investigate more closely on this issue as it is already a feature of SPARQL specifications[4] with named graphs (the FROM NAMED syntax).

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

Chapter V

Semantics and Usage Statistics for BI Recommendations

Contents

1 Introduction

Data warehouses are designed to integrate and prepare data from production systems - the Extract Transform and Load (ETL) process - to be analyzed with Business Intelligence (BI) tools. These tools now enable users to navigate through and analyze large amounts of data thanks to a significant effort from IT and domain experts to first model domains of interests. However, exploiting these multi-dimensional models may become challenging in important deployments of production systems. Indeed, domain models can grow extremely complex with thousands of BI entities, measures and dimensions used, e.g., to build OLAP cubes.

1.1 Situation modeling for BI personalization and recommendations

In Chapters III and IV we introduced our graph-based situation model and components for continuous monitoring and maintenance in response to business events, which may be used to describe users' interactions or more generally dynamic behaviors.

This chapter aims at presenting the application of our graph repository (GR) and situation management frameworks to BI concepts, to facilitate personalized data access. This personalization comes in various forms but we focus in particular on the assistance that can be given to users during the query design process, which is key to help them reach data they are looking for.

Personalization and recommendations involve a broad range of techniques, some of which were discussed in the review of related work, in particular in Chapter II, Section 2. In this chapter, we elaborate for instance on occurrences of BI entities and matrices of co-occurrences, users' preferences and users' similarity.

1.2 Application to the query design problem

In common reporting and analysis tools, users can design data queries using some kind of query panel. For instance, a user may drag and drop measures and dimensions she wants to use to create a new visualization or report, e.g., showing the `Sales revenue` aggregated by `City`. Given the number of available measures and dimensions, this selection can be tedious and helping the user to build her query rapidly becomes crucial. Therefore, our aim is to address the problem of query construction, through iterative suggestions and selection of measures and dimensions.

We define the query expansion problem as a function QE taking as input a user u, the current query q and additional parameters $params$. This function returns a collection of scored queries (q_i, s_i) such that, for i from 1

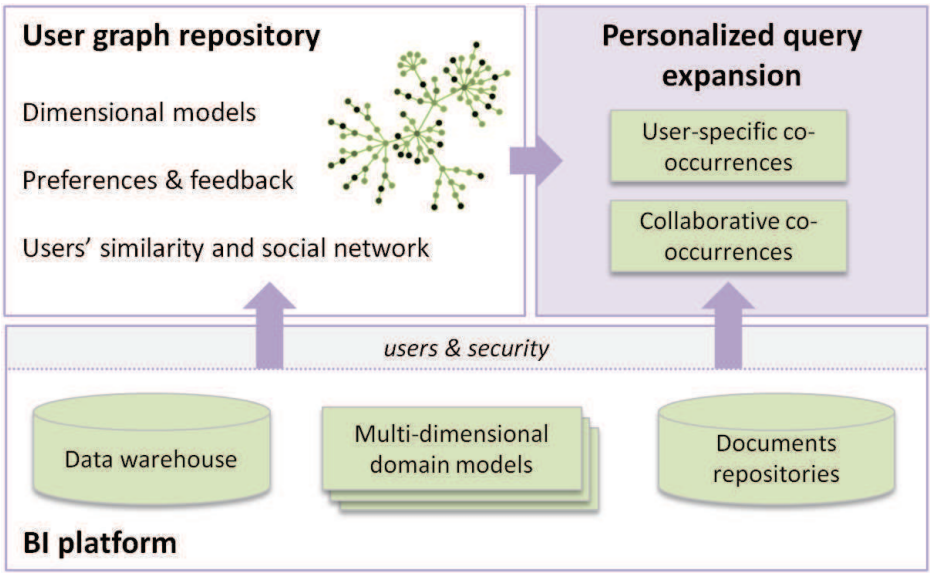

Figure V.1: Architecture overview of the proposed personalized query expansion system for multi-dimensional models.

to n, $|q_i| = |q| + 1$ and $q \subset q_i$:

$$QE\colon (u, q, params) \mapsto \{(q_1, s_1), \ldots, (q_n, s_n)\}$$

In response to this problem, this chapter presents various concepts involved in the design of an interactive and personalized query expansion system. Our method leverages semantics of multi-dimensional models, collaborative usage statistics derived from repositories of BI documents and user preferences to iteratively suggest relevant measures and dimensions.

Figure V.1 illustrates the main components involved in the architecture of our system, further discussed in this chapter. The remainder of the chapter is organized as follows: Section 2 introduces multi-dimensional domain models and their semantics. Section 3 introduces preferences and the definition of a similarity measure between users. Then, Section 4 presents a collaborative measure of co-occurrence between entities of these models, from a repository of BI documents. Eventually, Section 5 presents our personalized query expansion component and its architecture, according to an implementation based on the GR and situation managements frameworks.

2 Semantics of multi-dimensional domain models

Domain models can be designed on data warehouses, e.g., to enable an easier querying system for non-expert users. The modeling phase aims at defining

concepts of the business domain with key indicators (*measures*) and axis of analysis (*dimensions*) [JM06, chap. 1].

2.1 Measures and dimensions

Domain models are defined and used to query the warehouse for actual data and perform calculations. A warehouse may be materialized as a relational database, and queries thus have to be expressed accordingly, for instance as SQL. From a calculation point of view, it is also possible to build multi-dimensional OLAP cubes on top of a business domain model. Dimensions are represented as axis of a cube and measures are aggregated inside the different cells. It may be noted that the term "cube" refers to a convenient image but these structures are not limited to three dimensions. Queries can be expressed on these cubes, e.g., with Multi-Dimensional eXpressions (MDX). In modern warehouses, business domain models provide a predefined space of aggregates that reflects the organizational structure, key performance indicators, and other important information for a company's line of business. Furthermore, they provide SQL/MDX generation algorithms to enable non-expert users to formulate ad-hoc queries. End users can manipulate objects of these models (measures and dimensions) using common business terms (such as "customer" or "revenue"), rather than technical query languages.

Measures are numerical facts that can be aggregated against various dimensions ([BCG+08]). For instance, the measure `Sales revenue` could be aggregated (e.g., from unit sales) on the dimension `Country` to get the revenue in different countries. When analyzing data, e.g., for reporting, users can go deeper in a hierarchy of dimensions or aggregate at a higher level dimension to summarize data. These two operations are respectively called *drill down* and *roll up*. Data can also be filtered on some specific values of a dimension (*keep only* or selection) to restrict the scope of analysis.

To sum up, multi-dimensional domain models define measures and dimensions which are key business entities used to explore and query a data warehouse. Additionally, these domain models may also define hierarchies of dimensions and functional dependencies between measures and dimensions.

2.2 Functional dependencies and hierarchies

A functional dependency between two objects `A` and `B` (measures or dimensions) of a multi-dimensional model is a structural – and thus schema-related – relation, noted `A` *determines* `B`. As a simple example, a given instance of `City` determines a related instance at the higher `State` level. Another example that involves a measure and a dimension is to say that knowing a `Customer`, the `Sales revenue` he generates can be determined (e.g., aggregated from unit sales in a fact table). Functional dependencies are transitive: if `City` determines `State` which determines `Country`, then `City` determines

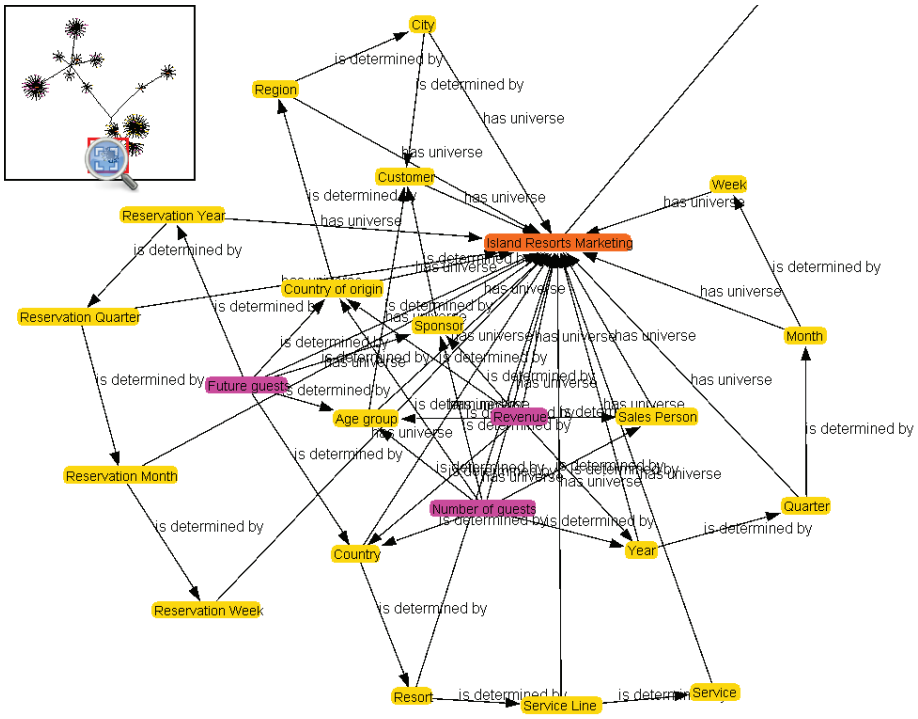

Figure V.2: Hierarchies and functional dependencies between some measures (purple nodes) and dimensions (yellow), described in an *Island Resorts Marketing* domain model (red) of a data warehouse.

`Country`.

Figure V.2 illustrates various hierarchies and dependency chains between objects of a multi-dimensional domain model. The most simple case of dependencies is when all measures are determined by all dimensions. This happens for instance when using a minimal dataset, e.g., reduced to one flat fact table like a spreadsheet. In such a case though, there is no dependency known before hand between dimensions, because of the lack of structuring schema.

The knowledge of functional dependencies is important when designing queries. For instance, they can be used to ensure queries do not contain incompatible objects, which would eventually prevent their execution. Let us consider an example of tourism-related domain model which defines a `Reservation year` dimension. This dimension may be used to query for the number of upcoming reservations in different places. On the other hand, it cannot determine the `Sales revenue` since this measure depends on billing rather than reservation dates. As an illustrating use case, the query expansion system presented later in this chapter (see Section 5) makes use of functional dependencies during the construction of its iterative suggestions.

In particular, objects incompatibility is used to filter the set of candidate entities before scoring.

Even though functional dependencies are important, multi-dimensional domain models do not necessarily capture and expose this metadata. Hierarchies of dimensions are more common though, usually exploited in reporting and analysis tools to enable *drill down* and *roll up* operations. For instance, if a `Year - Quarter` hierarchy is defined, the result of a user drilling down on `Year` 2010 is a similar query with the `Quarter` dimension, filtered on `Year` = 2010.

2.3 Reasoning about dependencies and hierarchies

If hierarchies of dimensions can be used to determine minimal dependency chains, techniques are required to help with automatic detection of functional dependencies. We now present our semi-automatic approach to ease a partly manual definition of functional dependencies, enriching existing domain models.

2.3.1 Transitivity and custom rules

The approach presented by [RCARM09] is to create domain-specific conceptual schemas in the form of ontologies and use *DL-Lite* inferencing capabilities. However, the definition of multi-dimensional business domain models is a costly process in which many BI users have already invested. Therefore, our method is different and meant to sit on top of already existing multi-dimensional models, in particular using defined measures and dimensions. We assume hierarchies are already represented in these models. Besides, our aim is to minimize as much as possible the manual definition of some functional dependencies, and infer additional relations using reasoning capabilities.

For this purpose, we mainly exploit the fact that functional dependencies are transitive plus two additional rules, noted R1 and R2. Figure V.3 illustrates derived knowledge resulting from the application of these two rules, described as follows:

R1. *Hierarchies imply functional dependencies.* As an example, knowing that `Week` *hasParent* `Quarter`, it can easily be deduced that `Week` *determines* `Quarter`.

R2. *If a dimension D_1 determines a measure M_1 and another dimension D_2, then D_2 determines M_1.* For instance, in Figure V.3, `Store` determines both the measure `Quantity Sold` and the dimension `City`. As a result, it can be inferred that `City` determines `Quantity Sold`.

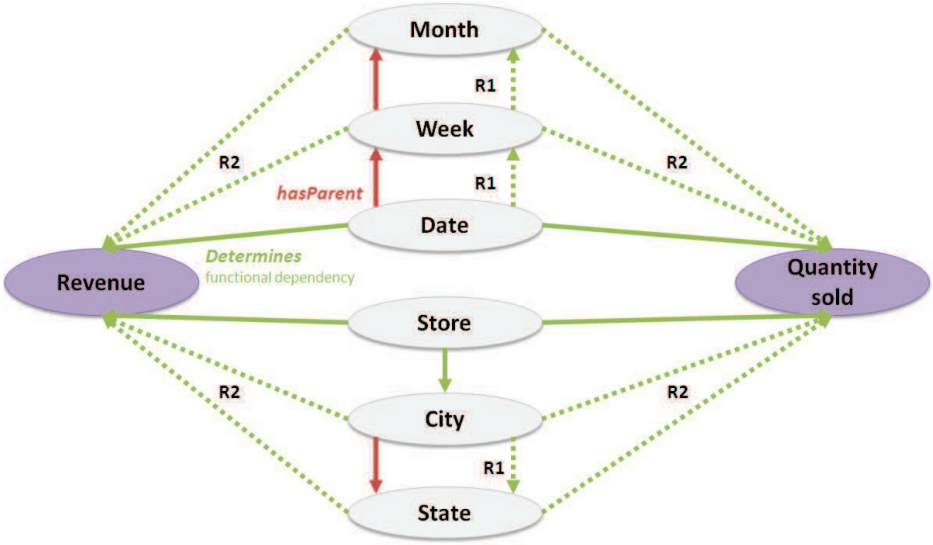

Figure V.3: Reasoning rules applied on an example with various measures (purple) and dimensions (gray). Dashed relations are statements resulting from rule-based reasoning (rules R1 and R2).

2.3.2 Reasoning in the Graph Repository framework

Let us come back to our graph repository (GR) framework which was introduced in Chapter III and further discussed in Chapter IV. Briefly, every user has its own GR composed of various graphs. These graphs are populated by *providers* and maintained by *operators*, two different kinds of plugins or modules. Graphs can be created and backed by different *graphs factories*.

In particular, our default implementation is an RDF-based graph factory. On top of this specific factory, reasoning is enabled in different ways thanks to a framework for Semantic Web technologies (the Jena API[1]). First, usual OWL reasoning may be applied. Taking the example of functional dependencies previously discussed, the transitivity of these relations is natively represented in an OWL ontology using the corresponding attribute on the object property (or predicate) `determines`. This information is represented and available in our framework through schemas of the GR (see Chapter III, Section 2.2.3).

Another interesting reasoning aspect comes with rules which can be defined using a SPARQL-like syntax. The simple rule R1 and the more complex one R2 presented above would be expressed as follows:

```
[R1:    (?a <urn:grepo/slayer#hasParent> ?b)
        -> (?a <urn:grepo/slayer#determines> ?b) ]
```

[1] http://incubator.apache.org/jena/

```
[R2:   (?a <urn:grepo/slayer#determines> ?m)
       (?m rdf:type <urn:grepo/slayer#Measure>)
       (?a <urn:grepo/slayer#determines> ?c)
       (?c rdf:type <urn:grepo/slayer#Dimension>)
       -> (?c <urn:grepo/slayer#determines> ?m) ]
```

To sum up, full-flavored OWL and rule-based reasoning capabilities are exposed through the GR framework, assuming the underlying RDF graph factory is used. It is important to highlight the fact that the complexity and cost of reasoning can in some cases make it inapplicable, even with rather small underlying datasets. However, well selected reasoning rules may be used with controlled datasets to obtain very valuable results or inferences in reasonable time. Section 5.2.1 further discusses these performance issues that need to be taken into consideration. In particular, the discussion bases on a comparison of two possible implementations for the set of reasoning techniques previously described.

We present in Section 5 our personalized query expansion component. For this purpose, we need to ensure suggested queries contain compatible objects so as to allow their execution. Functional dependencies allow us to determine compatible measures and dimensions. From the consumption point of view, hierarchies and dependencies that result from the reasoning process can be queried like any other graph of the GR.

3 Users' preferences and similarity

Preferences are more and more common to help personalize information and services for users. However, the problem of expressing preferences for multi-dimensional query entities has only recently been approached. In this section we discuss the possibility to express user preferences at the level of multi-dimensional models. This approach differs from the previous work by Golfarelli et al. [GRB11] since we do not go down to the underlying query language, be it SQL (on top of a relational database) or MDX (on top of OLAP cubes). Besides, our discussion focuses on simple numerical values which are applied to BI entities but could easily be considered with other scenarios and datasets. On top of these preferences, we use the Pearson correlation to define a similarity measure between users. This metric was presented with other usual techniques for recommender systems (RS), in Chapter II, Section 2.

3.1 User preferences and feedback

We call general preferences simple numeric values that indicate the user's preference for a given resource or entity. Even though this approach could

be extremely general, the definition of these indicators is illustrated here in a BI context.

We distinguish *explicit* and *implicit* preferences, respectively noted $pref_{u,expl}$ and $pref_{u,impl}$. For a given entity e, we define the user's preference function $pref_u$ as a linear combination of both preferences, where γ and δ are coefficients such that $\gamma + \delta = 1$:

$$pref_u(e) = \gamma \cdot pref_{u,impl}(e) + \delta \cdot pref_{u,expl}(e) \tag{V.1}$$

3.1.1 Explicit preferences

Explicit preferences are feedback received from the user, e.g., in the form of ratings (in $[0, 1]$) assigned to measures and dimensions in different models.

Entity level. Let us note $r_{u,e}$ the rating given by u to e (or `null` if e has not been rated yet) and $\overline{r_u}$ the average rating given by u to all entities. We simply define the user's explicit preference of an entity e as:

$$pref_{u,expl}(e) = \begin{cases} r_{u,e} & \text{if } r_{u,e} \neq \texttt{null} \\ \overline{r_u} & \text{otherwise} \end{cases}$$

Domain/model level. It is common in real life scenarios that users actually manipulate several multi-dimensional models. Two models may refer to the same underlying data source with a different view on it, or simply to two distinct sources. Given this fact, it is of importance to let users express their preference for a certain model M compared to others. We thus extend the previous definition with ratings given by a user to a multi-dimensional model M:

$$pref_{u,expl}(M) = \begin{cases} r_{u,M} & \text{if } r_{u,M} \neq \texttt{null} \\ \frac{1}{|M|} \sum_{e \in M} pref_{u,expl}(e) & \text{otherwise} \end{cases}$$

In this definition, we see that in case there is no explicit vote for the model M, the default value is derived from explicit votes at the lower entity level. Some could argue that this definition is no longer very explicit, since the rating is already aggregated from other explicit values.

3.1.2 Implicit preferences

Implicit preferences can be derived from a variety of sources, for instance by analyzing logs of queries executed in users' sessions [GMN08]. In this section we present another approach using occurrences of entities in a repository of BI documents.

If hierarchies and functional dependencies are structural or navigational relations, they are not specific to a given user and they do not convey any information regarding the way users actually explore available data. On the contrary, repositories of documents mainly bring usage statistics. Our aim is thus to leverage these statistics – combined with semantics of multi-dimensional models – to provide personalized assistance and recommendations. The structure of BI documents will be more precisely discussed in Section 4. Briefly though, reports and dashboards may contain chart or tables and BI entities may be referenced by these charts.

Entity level. From this perspecitve, we consider that occurrences of BI entities in documents manipulated by the user can be an interesting and simple indicator of her preferences. Let us note $occ_u(e)$ the set of charts and tables which reference an entity e, in u's personal collection of documents. We define implicit preferences of the user u for the entity e as the normalized frequency of e in documents:

$$pref_{u,impl}(e) = \frac{|occ_u(e)|}{\max_{e'} |occ_u(e')|}$$

The notions of entities occurrences and frequency in documents can clearly be linked to content-based recommender systems and at its root to information retrieval techniques. A more sophisticated formula may thus use TF-IDF-like weights for entities. In particular, as discussed in Section 2.2.1 of Chapter II, the *inverse document frequency* (IDF) part of a TF-IDF weight is meant to reduce the importance of entities that appear very often. Such entities are indeed less discriminatory in the filtering process. However, we do not adopt this approach since we prefer to keep simple normalized scores which can more easily be interpreted and linearly combined (aggregated). Besides, we do not consider occurrences (and co-cooccurrences) at the document level (e.g., dashboard) but rather at the chart/table level, which represents more fine-grained units of sense. This point will be further discussed in the next section.

Domain/model level. Similar to what we did with explicit preferences, we now define implicit preferences at the model level. First, it should be noted that the formula above is computed for each entity inside a given model, noted M. In particular, the term $\max_{e'} |occ_u(e')|$ refers to the maximum number of occurrences among all entities of M. In the following two equations, we extend the definitions of implicit and then general preferences at the model level:

$$
\begin{aligned}
pref_{u,impl}(M) &= \frac{1}{|M|}\sum_{e\in M} pref_{u,impl}(e) \\
pref_u(M) &= \gamma \cdot pref_{u,impl}(M) + \delta \cdot pref_{u,expl}(M)
\end{aligned}
$$

3.2 Users similarity

We previously defined numerical preferences which take into account explicit feedback from the user as well as implicit indicators derived from usage analysis. Based on this, we now introduce users' similarity in terms of BI data consumption.

In Chapter II, Section 2.2.2, we presented various definitions of similarity measures between users for recommender systems. In particular, the most commonly used with collaborative filtering systems is the *Pearson correlation*. Usually, this metric is used with a matrix of ratings $R = User \times Item$. To minimize cold-start issues and increase the coverage of our measure, we use the matrix of numeric preferences $(P_{i,j})$, such that $P_{i,j} = pref_{u_i}(e_j)$. Given the set of users $U = \{u_1, \ldots, u_m\}$ and the set of entities $E = \{e_1, \ldots, e_n\}$:

$$
(P_{i,j}) = \begin{pmatrix}
pref_{u_1}(e_1) & pref_{u_1}(e_2) & \cdots & pref_{u_1}(e_n) \\
pref_{u_2}(e_1) & pref_{u_2}(e_2) & \cdots & pref_{u_2}(e_n) \\
\vdots & \vdots & \ddots & \vdots \\
pref_{u_m}(e_1) & pref_{u_m}(e_2) & \cdots & pref_{u_m}(e_n)
\end{pmatrix}
$$

Let us consider two users u and u' and let $E_{u,u'}$ denote the set of entities accessible to both u and u'. We define the similarity measure between these two users, $sim(u, u')$, using the Pearson correlation as follows:

$$
sim(u,u') = \frac{\sum_{e\in E_{u,u'}} (pref_u(e) - \overline{pref_u})(pref_{u'}(e) - \overline{pref_{u'}})}{\sqrt{\sum_{e\in E_{u,u'}} (pref_u(e) - \overline{pref_u})^2 \sum_{e\in E_{u,u'}} (pref_{u'}(e) - \overline{pref_{u'}})^2}} \tag{V.2}
$$

Table V.1 illustrates a sample of 5 users with their (randomly generated) preferences regarding a partial set of the measures and dimensions. Based on these preferences, Figure V.4 represents the computation of users similarity matrices using the Pearson correlation and the cosine similarity.

The Pearson correlation takes values in $[-1; 1]$ and thus brings a sense of polarity. Positive values indicate users that tend to rate the same items in the same way, relative to their average rating. Conversely, negative values show users that often react in opposite directions. For instance, the

Preference	Category	City	Lines	Quantity sold	Quarter	Sales revenue	State	Store name	Year	**Avg**
User 1	0,29	0,2	0,24	0,87	0,84	0,74	0,5	0,48	0,92	0,56
User 2	0,74	0,12	0,86	0,53	0,01	0,76	0,03	0,51	0,83	0,49
User 3	0,35	0,98	0,92	0,47	0,54	0,62	0,15	0,9	0,73	0,63
User 4	0,35	0,81	0,53	0,29	0,44	0,27	0,76	0,12	0,12	0,41
User 5	0,91	0,88	0,3	0,13	0,46	0,01	0,34	0,1	0,76	0,43

Table V.1: Matrix of numeric users preferences regarding certain dimensions and measures. These values include explicit and implicit contributions to preferences.

cosine similarity indicates that users $U1$ and $U3$ are very similar whereas the Pearson correlation says the contrary. It can be observed from the data in Table V.1 that $U1$ and $U3$ tend to rate entities in opposite ways.

The Pearson correlation is invariant to linear combinations of the input vectors. Given two vectors u and v and any real numbers $\alpha, \beta, \gamma, \delta$, $pearson(u, v) = pearson(\alpha \cdot u + \beta, \gamma \cdot v + \delta)$. This proves important to determine vectors that vary in the same way rather than comparing vectors in absolute values. To conclude, it may be noted that the cosine similarity could be refined using the so called *adjusted cosine similarity*, which takes into account deviations from users' average preferences (or ratings).

4 Usage statistics in BI documents

In Section 2, we presented multi-dimensional domain models and their semantics. In particular, functional dependencies and hierarchies provide very structural knowledge regarding associations between BI entities. Beyond this, some BI platforms propose repositories of documents (such as reports and dashboards) which can be used to compute actual usage statistics for measures and dimensions. This kind of information is extremely valuable in our use case, since the query expansion problem (as formulated in Section 1) implies to find the best candidate to associate to a given set of measures and dimensions. Therefore, we describe in this section a measure of co-occurrence between BI entities.

4.1 Structure of BI documents and co-occurrence

We use the structure of BI documents to define co-occurrences between measures and dimensions. For instance, BI reports are roughly composed of sec-

PEARSON	U1	U2	U3	U4	U5
U1		0,08	-0,92	-1,68	-0,95
U2	0,08		0,70	-1,78	-0,33
U3	-0,92	0,70		-0,28	0,09
U4	-1,68	-1,78	-0,28		0,83
U5	-0,95	-0,33	0,09	0,83	

COSINE	U1	U2	U3	U4	U5
U1		0,76	1,61	0,94	1,03
U2	0,76		1,68	0,77	1,03
U3	1,61	1,68		1,10	1,21
U4	0,94	0,77	1,10		1,26
U5	1,03	1,03	1,21	1,26	

Figure V.4: Comparison of users similarity matrices obtained with Pearson correlation and cosine similarity. Computation is based on users and preferences described in Table V.1.

tions which may contain charts, tables, text areas for comments, etc. Charts and tables define important units of sense. Measures and dimensions associated in a same table/chart are likely to be strongly related and represent an analysis of specific interest to the user, which she may have created herself.

Similarly, dashboards can be composed of different pages or views. These views can also contain charts and tables. Figure V.5 illustrates an example of dashboard presenting three charts in a view:

1. Sales Revenue, Quantity Sold by Quarter,

2. Sales Revenue by Category and

3. Sales Revenue by City, Lines.

Reports and dashboards are examples but, more generally, any BI document with visualizations referencing measures and dimensions could be used to derive consolidated co-occurrences or usage statistics. The graph model introduced in Chapter III enables a homogeneous representation of these documents' structure. For instance, Figure V.6 illustrates the graph representation of the dashboard *USA Sales View* and its associated charts. A chart can be used in one or more dashboards and these relations are represented by the *hasDashboard* predicate. Besides, *hasDimension* and *hasMeasure* predicates are used to indicate BI entities a chart relates to.

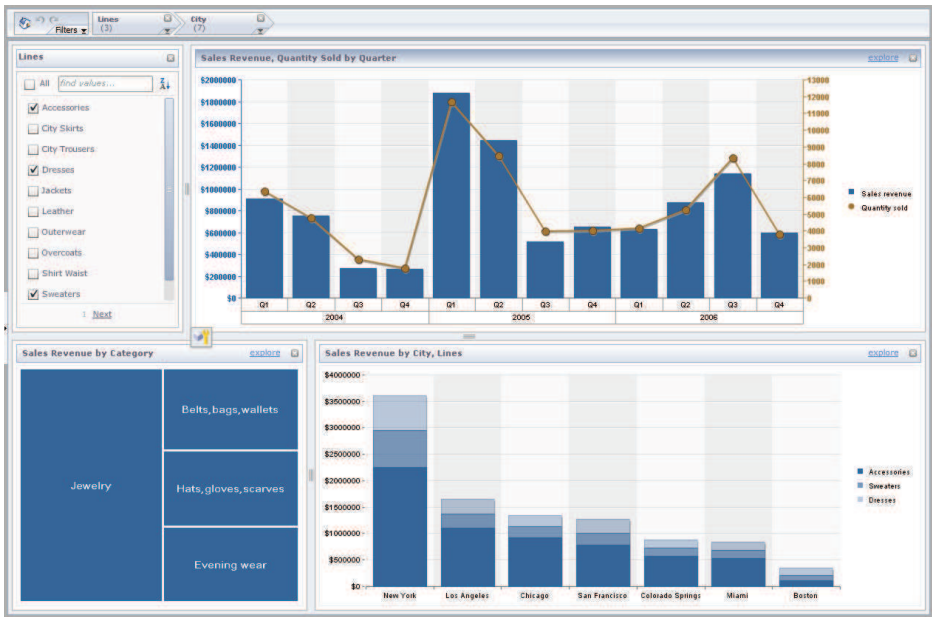

Figure V.5: A sample dashboard view, *USA Sales View*. This view defines 3 charts presenting analysis of sales revenue according to various dimensions.

4.2 Security and personal co-occurrence measure

Security is an important aspect to take into consideration since BI platforms provide access control rules to business domain models and documents built on top of them. Consequently, different users may not have access to the same models and at a more fine-grained level to the same measures and dimensions. Besides, repositories contain reports or dashboards generated by and shared (or not) between different users of the system. As a result, the measure of co-occurrence that we define in this section is inherently personalized.

Let us consider a user u and let $occ_u(e)$ denote the set of charts and tables – in documents visible to the user u – referencing a BI entity e (measure or dimension). We define the co-occurrence of two entities e_i and e_j as the Jaccard index of the sets $occ_u(e_i)$ and $occ_u(e_j)$. The Jaccard index is a simple but commonly used measure of the similarity between two sample sets:

$$cooc_u(e_i, e_j) = J(occ_u(e_i), occ_u(e_j)) = \frac{|occ_u(e_i) \cap occ_u(e_j)|}{|occ_u(e_i) \cup occ_u(e_j)|} \qquad \text{(V.3)}$$

Table V.2 presents a matrix of co-occurrences that can be computed with measures and dimensions appearing in charts, in a collection of dashboards. This matrix $(C_{i,j})(u)$ is symetric and defined as follows for all

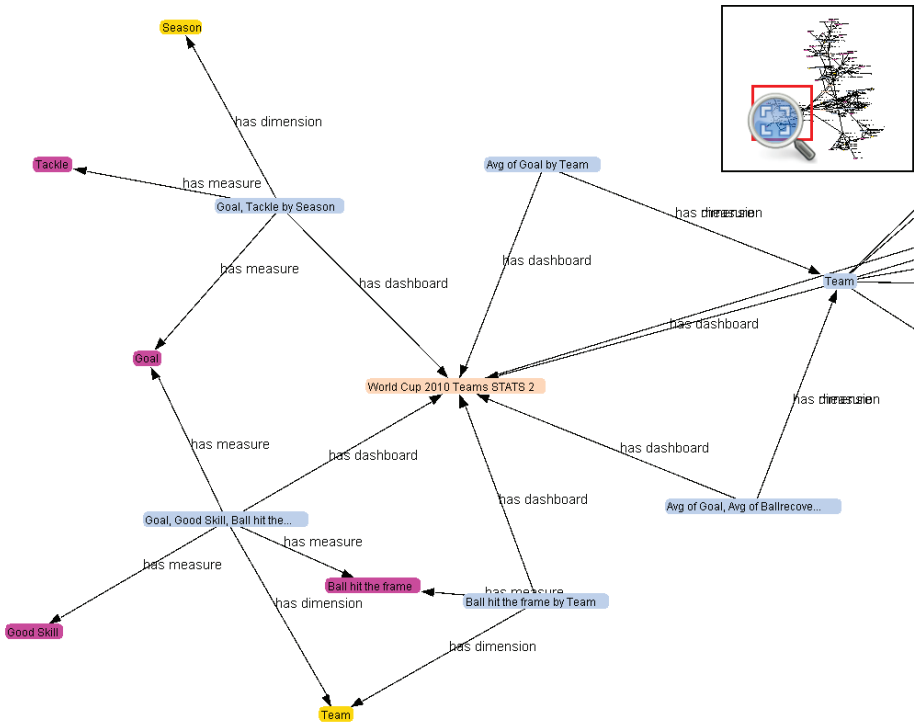

Figure V.6: Graph describing a dashboard (orange), *USA Sales View*, its associated charts (blue) and referenced measures (purple) and dimensions (yellow).

entities $\{e_1, \ldots, e_n\}$:

$$(C_{i,j})(u) = \begin{pmatrix} 1 & cooc_u(e_1, e_2) & \cdots & cooc_u(e_1, e_n) \\ cooc_u(e_2, e_1) & 1 & \cdots & cooc_u(e_2, e_n) \\ \vdots & \vdots & \ddots & \vdots \\ cooc_u(e_n, e_1) & cooc_u(e_n, e_2) & \cdots & 1 \end{pmatrix}$$

The k-th line (or column) of this matrix can be used to determine other entities (measures or dimensions) that most co-occur with e_k. For instance, according to Table V.2, the 4 dimensions most often associated to the measure `Sales Revenue` are `Quarter`, `Category`, `State` and `Year`.

Additionally, a weighted average of vectors formed by any two lines k and l can be considered to determine entities that most co-occur with entities e_k and e_l. However, when considering vectors sums, one needs to keep in mind entities compatibility. Indeed, the entity e_k may co-occur with two entities e_l and e_p, mutually incompatible.

Cooccurrence	Category	City	Lines	Quantity sold	Quarter	Sales revenue	State	Store name	Year
Category	1	0	0	0	0	0,25	0	0	0
City	0	1	0,33	0	0	0,13	0	0	0
Lines	0	0,33	1	0,2	0	0,22	0	0	0
Quantity sold	0	0	0,2	1	0,5	0,22	0	0	0,67
Quarter	0	0	0	0,5	1	0,38	0,25	0	0,67
Sales revenue	0,25	0,13	0,22	0,22	0,38	1	0,25	0	0,25
State	0	0	0	0	0,25	0,25	1	0	0
Store name	0	0	0	0	0	0	0	1	0
Year	0	0	0	0,67	0,67	0,25	0	0	1
Average	0,03	0,06	0,09	0,20	0,22	0,21	0,06	0	0,20

Table V.2: Symetric matrix of co-occurrences (in a personal collection of dashboards) between various dimensions and measures of a multi-dimensional domain model.

4.3 Collaborative co-occurrence measure

4.3.1 Cold-start users and coverage

In recommender systems, the *coverage* is similar to the recall in information retrieval and represents the share of items that can actually be compared (and thus recommended). This metric is thus commonly used to evaluate collaborative RS, introduced in Chapter II, Section 2.2.2. Formula V.3 presents a problem for *cold-start* users, i.e. those new to the system. Indeed, these users do not have stored documents from which co-occurrences can be computed. Recommender systems address this issue by introducing a collaborative contribution in items' ratings, e.g., in collaborative filtering methods.

A collaborative contribution in the final item ranking is key to improve the system's coverage and enable the exploration of resources previously unknowned (or unused) by the user. A simple approach could consist in using the average of the previous co-occurrence measure for the set of all $users$, noted $\overline{cooc}(e_i, e_j)$, as follows:

$$cooc_{simple}(u, e_i, e_j) = \alpha \cdot cooc_u(e_i, e_j) + \beta \cdot \overline{cooc}(e_i, e_j) \qquad \text{(V.4)}$$

This definition is a simple weighted average of the personal and the global co-occurrence scores, where α and β are coefficients to be adjusted such that $\alpha + \beta = 1$.

4.3.2 Using the social/trust network

Formula V.4 broadens the collaborative contribution to "the whole world". In this case, all users have an equal contribution, even though some may not relate at all to the current user. Jamali et al. suggest that this could be significantly improved by considering the user's social network and, e.g., favoring users close to her [JE09]. The benefits of narrowing the collaborative contribution down to close users reside at two levels: (a) results are more precisely personalized (b) potential pre-computation can be made more resource-efficient (memory, CPU, etc.).

Let us note $SN(u)$ the set of users in u's social network, which can be filtered, e.g., to keep only users up to a given maximum distance. We propose the following refined co-occurrence measure:

$$\begin{aligned} cooc(u, e_i, e_j) &= \alpha \cdot cooc_u(e_i, e_j) \\ &+ \frac{\beta}{|SN(u)|} \cdot \sum_{u' \in SN(u)} \frac{1}{d(u, u')} cooc_{u'}(e_i, e_j) \end{aligned} \quad \text{(V.5)}$$

This measure $cooc(u, e_i, e_j)$ is only defined for entities e_i and e_j exposed to the user u by access control rules. However, it may be extended for convenience to all entities, e.g., by considering a default value of 0. In this co-occurrence measure, the contribution of each user u' is weighted by the inverse of the distance $d(u, u')$.

The knowledge of relations between users can be obtained from a variety of sources, including popular social networks on the Web. However, this does not necessarily match corporate requirements since users of the system are actual employees of a same company. Therefore, we consider the use of enterprise directories (LDAP, Active Directory, etc.) to extract relations between users. For instance, such directories usually contain information regarding the company's hierarchical organization, that is relations of the type "A *reports to* B". If the social network and its actual construction are not the main focus of our work, it was discussed along with the social dimension of our situation model, in Chapter III, Section 3.

4.3.3 Similarity-based and hybrid approaches

We now investigate the usage of the users' similarity measure defined in the previous section to propose an alternative collaborative approach. In Formula V.5, the collaborative contribution is weighted function of the distance between users. However, it could be argued that close users (in terms of their distance in the social graph) are not necessarily those with the most common habits when it comes to data consumption. Two sales managers in different countries under different hierarchical chains may be distant of, say, 5 degrees. Even if their interests were similar, the inverse distance factor

would considerably reduce the importance of the second user in the final scoring function of the first.

In line with traditional techniques used in user-based collaborative RS (described in Chapter II, Section 2.2.2), we thus propose the following alternative co-occurrence measure:

$$\begin{aligned} cooc(u, e_i, e_j) &= \alpha \cdot cooc_u(e_i, e_j) \\ &+ \frac{\beta}{|U'|} \cdot \sum_{u' \in U'} sim(u, u') \cdot cooc_{u'}(e_i, e_j) \end{aligned} \quad \text{(V.6)}$$

In Formula V.6, U' denotes the set of users most similar to the current user u, determined using the similarity function as follows: $U' = \{u' \in U, sim(u, u') \geq K\}$, where K is a constant threshold. This step filtering the nearest users may be used to reduce the computation cost and improve its precision. However, in a first approximation it is possible to simply consider $U' = U$. To conclude, it would also be possible to consider hybrid approaches – using a similarity function combining users' distance and their preferences – but we did not investigate this aspect further due to lack of time.

5 Personalized query expansion

So far we presented in this chapter particular semantics of multi-dimensional models and examples of usage statistics that can be derived from existing repositories of BI documents.

In this section, we describe our approach to design a personalized query expansion component leveraging models semantics, co-occurrences and user preferences previously defined. Besides, we discuss the architecture of our system using the GR and situation management frameworks described in Chapters III and IV.

5.1 Query expansion

The aim of our system is to assist the user in the query design phase by offering suggestions of measures and dimensions she could use to explore data. When she selects a measure or a dimension, it is added to the query being designed and suggestions are refreshed to form new consistently augmented queries. For instance, if the user selects the measure `Sales revenue`, the system proposes to add dimensions to form queries like `Sales revenue` by `Year`, `Sales revenue` by `Category`, etc.

5.1.1 Ranking candidate entities

To complete a given a query $q = \{e_1, \ldots, e_n\}$ with an additional measure or dimension, we need to find candidate entities and rank them. In a first

Algorithm 1 Query expansion: expands an initial query q for a given user u with additional parameters *params*.

```
1: QueryExpansion(q, u, params)
2: candidates ← getCandidates(q, params)
3: rankedQueries ← initialize list
   {Rank each candidate entity}
4: for j = 1 to |candidates| do
5:    q' ← q ∪ {candidates[j]}
6:    rank ← rank_u(candidates[j], q)
      {Append generated query}
7:    rankedQueries.add(q')
8: end for
9: return sort(rankedQueries)
```

approximation, candidate entities, $c_j, j = 1..p$, are those defined in the same domain and compatible with every e_i, determined using functional dependencies (see Section 2.2).

We then use the following personalized function to rank each candidate c_j, using one of the co-occurrence measures defined in Section 4 and noted *cooc*:

$$rank_u(c_j, q) = \begin{cases} pref_u(c_j) & \text{if } q = \emptyset \\ pref_u(c_j) \cdot \frac{1}{n} \sum_{i=1}^{n} cooc(u, c_j, e_i) & \text{otherwise} \end{cases} \quad \text{(V.7)}$$

To conclude with the notation of the query expansion problem introduced in Section 1, we define our component QE as:

$$QE\colon (u, q, params) \mapsto \{(q_1, rank_u(c_1, q)), \ldots, (q_p, rank_u(c_p, q))\}$$

The complete query expansion process is summarized in Algorithm 1. This algorithm is pretty straightforward and mainly meant to present more formally the steps required towards final suggestions. In particular, the whole method begins line 2 with the selection of candidate entities. This selection depends on the inital query q to deal with compatibility issues and also additional parameters.

5.1.2 Candidates filtering and parameters

Beyond ranking, suggestions of the query expansion component can be fine-tuned using various parameters. In particular, the following parameters may be used to filter the set of entities which are candidates to expand the initial query:

- The maximum number of results can be configured to limit the number of variant queries that will be generated as recommendations or suggestions. Assuming this number is set to 3, the output of the expansion component will be the list of the 3 best-ranked queries.

- The type of suggested entities can be limited to measures and/or dimensions. This aspect is interesting to drastically filter the set of entities in some cases. For instance, it seems reasonable to assume that once the user has selected a measure, she is going to expect dimensions to be suggested rather than other measures.

- The domain can be restricted to a given list of dimensional models. The initial selection or configuration of this parameter should not be imposed on the user since it requires one more manual step before suggestions can take place. However, it is possible to determine it automatically when the initial query q is not empty, using the fact that queries can be executed in the scope of one given model.

- Suggested dimensions can be grouped by and limited to certain hierarchies. This may be used to reduce the number of suggestions and encourage the user explore varied axis of analysis.

5.2 Architecture overview

In Section 1, Figure V.1 gave an overview of the architecture of our query expansion component. In this section, we present this architecture more specifically with the integration of the GR and situation frameworks presented in Chapters III and IV.

Figure V.7 illustrates this framework-based implementation of the concepts described in this chapter. Providers are important components in this framework which allow the creation of graphs with user-related data. For instance, the *social provider* was presented with the definition of the social dimension in our situation model, in Chapter III, Section 3.3. Let us now describe other providers involved.

5.2.1 Multi-dimensional models provider

First and foremost, the *multi-dimensional models provider* is responsible for the creation of graphs which describe models available to the current user. Entities represented in these graphs are measures and dimensions, linked by relations (or statements) of two types: *hasParent* and *determines*.

Hierarchies and functional dependencies represent structural relations which semantics were introduced in Section 2. Among other points, we discussed the use of two reasoning techniques to augment existing models in a semi-automatic approach and help complete these graphs. At this point,

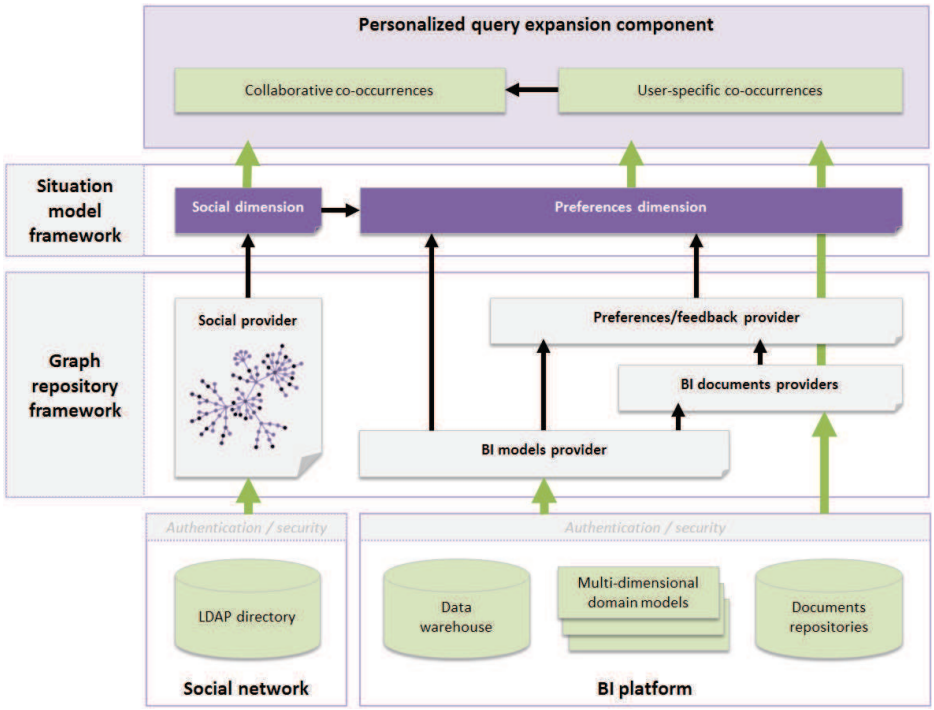

Figure V.7: Proposed integration of the personalized query expansion component with graph repository and situation management frameworks.

it could have been noted that the transitivity of functional dependencies – represented with the *determines* predicate – could also be integrated using a third custom rule R3:

```
[R3:    (?a <urn:grepo/slayer#determines> ?b)
        (?b <urn:grepo/slayer#determines> ?c)
        -> (?a <urn:grepo/slayer#determines> ?c) ]
```

The reason not to use this solution (with an additional rule) lies in practical implementation performance. We compared the time taken by the reasoning processs using two solutions: (1) rules R1 and R2 with OWL-MINI reasoning for transitivity and (2) rules R1, R2 and R3.

Figure V.8 illustrates the comparison of these two solutions, in terms of reasoning execution time function of the number of nodes. For this test, we randomly generated graphs with the given number of nodes and twice as many relations. The implementation of underlying RDF graphs, SPARQL querying and reasoning is enabled by the popular Jena framework for Semantic Web technologies[2].

[2]http://jena.sourceforge.net/

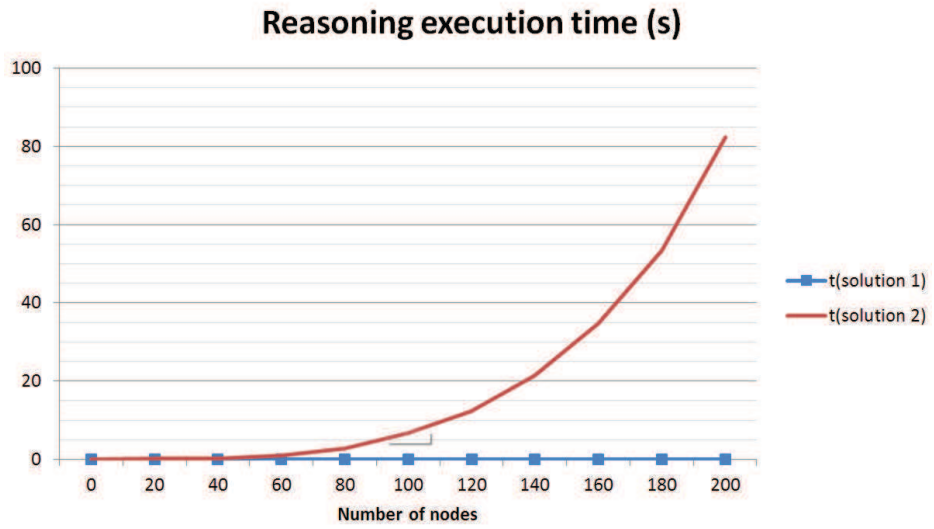

Figure V.8: Comparison of two proposed reasoning solutions to determine full dependencies in graphs populated from multi-dimensional models.

To conclude we evaluated the cost of reasoning using the first solution (which happens to be the most efficient) when the number of dimensions and measures grows. The result of this evaluation is presented in Figure V.9. In particular, we observe that the reasoning time augments almost linearly (below the second) for up to 13000 entities. This cost is easily acceptable and covers most real-life scenarios with large models.

5.2.2 BI documents provider

Next comes the *BI documents provider*. This component connects to existing repositories of the BI platform in order to describe dashboards, reports and their content in graphs. Documents may reference table or charts, which themselves reference measures and dimensions as discussed in Section 4.1. Figure V.6 illustrates the graph representation of a dashboard with referenced charts, measures and dimensions.

One thing we did not discuss previously with the provider for multi-dimensional models is the security aspect. Indeed, the BI platform requires user credentials to allow authentication. For this purpose, the graph repository framework allows users to enter required credentials, should they decide to enable the corresponding provider. In this context, the BI documents provider leverages this feature and exploits the user's credentials to log in a (remote) BI platform. Credentials – and more generally the security model applied to components of the GR – have been described more thoroughly in Chapter IV, Section 3.1.

To conclude, we leveraged in our experimentations the connection to

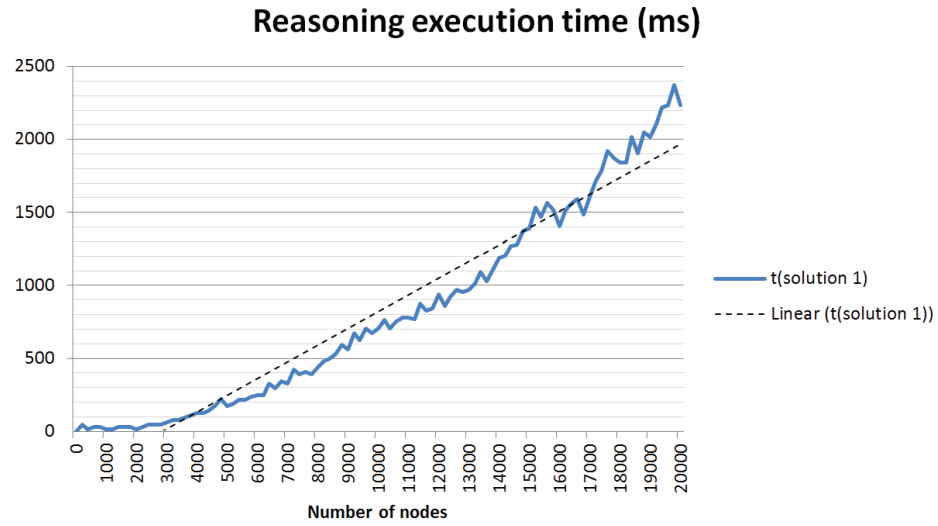

Figure V.9: Evaluation of the reasoning time function of the size of the multi-dimensional model (number of nodes, measures or dimensions).

Exploration Views, a dashboarding solution which used to be a prototype and is now included with SAP Business Objects Explorer. A demonstration version of this solution is available online[3].

5.2.3 Preferences provider

The last provider that needs to be presented in the architecture diagram of Figure V.1 is the *preferences provider*. This provider manages users' preferences which come with explicit and implicit contributions (see Section 3.1).

At least, this provider includes dependencies on the multi-dimensional models provider, to enumerate available measures and dimensions. On top of this, a dependency on the BI documents provider allows implicit preferences to be computed using occurrences of these measures and dimensions in documents.

Finally, the user's feedback (or explicit preferences) is maintained thanks to a graph, created and persisted using the appropriate graph factory (see Chapter III, Section 2.2.4).

6 Summary and discussion

In Section 2, we presented multi-dimensional domain models defined in data warehouses to enable the exploration of large amounts of data thanks to dedicated BI analysis tools. These models introduce in particular measures and

[3]http://exploration-views.ondemand.com

dimensions which inter-relations – hierarchies and functional dependencies – carry specific semantics. These dependencies are key in a query generation process and help for instance to determine objects compatibility. Besides, we described the use of BI documents repositories to derive valuable usage statistics for entities of these models. This allowed us to define several personalization-related indicators like users' preferences and users' similarity (in Section 3) and co-occurrences between entities (in Section 4). Several variants of the co-occurrence measure were proposed, from the most simple and personal to more elaborate ones, introducing for instance a collaborative contribution. Eventually, Section 5 discussed the utilization of these concepts, applied for the design of a personalized query expansion component. Such a component proves useful in various BI experimentation scenarios, as will be discussed in Chapter VI.

Repositories of BI documents are at the root of a certain number of indicators presented in this chapter. In our approach, we leverage the structure of these documents to define occurrences (and co-occurrences) in the scope of a table or a chart. We illustrated the exploitation of repositories of BI documents as one source of usage statistics. Besides, other sources of usage data could clearly be used, the most common example being logs of users' sessions and queries. More generally, personalization and recommendations for multi-dimensional queries have been discussed in the review of related work, in Chapter II, Section 4.

Golfarelli et al. described MyOLAP to express BI-specific preferences, in a qualitative approach. The authors define these preferences as soft constraints used to improve the scoring of most similar data. Interestingly, preferences can be expressed on both numerical and categorical domains, that is on measures and dimensions. For instance, it is possible with MyOLAP to define preferred aggregation levels (in a hierarchy of dimensions) and intervals (for values of a measure). These preferences are defined as MDX operators which enable the expression of soft constraints in potentially complex queries. The reverse side of the coin is the increased complexity of the query design process which assumes a certain know-how from end-users, in particular the ability to write MDX queries.

To sum up, we reckon that our quantitative approach presented here offers an interesting trade-off between the generality of the model and the expressivity of recommended queries. Simple numerical indicators allow us to easily integrate social contributions and rely on efficient and scalable methods, emerging from traditional recommendation techniques. On the other hand, our approach does not leverage the full expressivity of query languages, be it SQL or MDX. This introduces limitations to the complexity of queries that can be recommended by our system but presents the significant advantage of being source agnostic.

7 Conclusion

In conclusion, we presented in this chapter a personalized query expansion system that leverages (a) semantics of multi-dimensional domain models, (b) usage statistics derived from (co-)occurrences of *measures* and *dimensions* in repositories of BI documents and (c) users' preferences and similarity. This system and its experimentation with a prototype of interactive query designer will be presented in further details in Chapter VI, Section 3.

This chapter introduced a certain number of concepts for personalization and recommendations in a BI context. It would be interesting and particularly valuable to concentrate future work on the evaluation of the proposed query expanson system. In particular, an extensive study and comparison of the measures presented in Sections 3 and 4 would help determine and refine the most appropriate ones. To do so, the first step would consist in the acquisition (or generation) of an appropriate dataset. This dataset should at least include the definition of multiple users and a sufficient number of documents (reports or dashboards), to ensure that occurrences and co-occurrences defined are meaningful. Ideally, these users would also be described in a social graph so the collaborative contribution can be evaluated based on distances in the graph. This would for instance enable the comparison of the two co-occurrence measures propsoed in Formulas V.5 and V.6.

Importantly, our approach bases on statistical indicators derived from occurrences of entities in BI documents. On one hand, such indicators prove valuable and offer a rather generic model – in particular because they are source agnostic – but on the other hand, this limits the expressivity of queries that can actually be personalized or recommended. We illustrated the use of these concepts with a query expansion component. However, it is worth noting that users' preferences and their similarity (in terms of BI data consumption) could be employed in diversified scenarios. Therefore, as part of future work, we would like to investigate other promising applications like, for instance, a question answering system allowing the expression of business queries in natural language [KBA11]. This system could exploit users' preferences to help with the entity disambiguation process or query reformulation. Besides, the users' similarity measure would be valuable to introduce a social scoring function for answers brought by the system.

Finally, we reckon that recommendations in the context of data warehouses and BI platforms could benefit much further from techniques developed in the area of recommender systems. However, taking into account the specific semantics of multi-dimensional models is key to provide relevant structured analytics.

Chapter VI

Experimentations with BI Recommendations and Personalization

Contents

1 Introduction

The work that we presented in Chapters III to V aims at defining a framework – or set of re-usable and extensible components – to help build personalized and context-sensitive applications.

We introduced our graph-based *situation* model in Chapter III, in particular the underlying *graph repository* (GR). The GR serves as a container

for personal background knowledge and a vehicle for context-related information. Besides, *dimensions* are built on top of the situation, represented as a graph centered on the current user. Dimensions help expose categorized and aggregated information with additional convenience methods. For instance, *social* and *geography* dimensions can be used to retrieve respectively the user's social network and her current location.

The situation management framework presented in Chapter IV described active components for continuous monitoring and dynamic adaptation of situation graphs. In short, *business events* can be sent to the situation platform (or raised internally), which queues them to perform asynchronous processing. In response to these events, *activation rules* are used tro trigger specific *operators*, responsible for graphs adaptation.

Then, Chapter V presented semantics of multi-dimensional models commonly used in data warehouses and BI tools. Besides, we defined usage metrics or statistics that can be derived from repositories of documents like dashboards and reports. These indicators represent (a) occurrences and co-occurrences of measures and dimensions, (b) users' preferences according to these entities and (c) users' similarity.

To conclude, the aim of this chapter is to show case and evaluate our approach using different BI-related experimentation scenarios. These scenarios were implemented as proof-of-concepts to evaluate the applicability of our GR and situation management frameworks, and its effectiveness for personalization and recommendations. We distinguish two types of experimentations, *exploratory* and *search-related* scenarios.

1.1 Exploratory recommendation scenarios

The first kind of scenario we consider is said exploratory in that it is meant to suggest to the user resources which are related to what she is currently reading, writing, etc. The aim is to enable a certain form of exploration of resources available to the user and likely of interest.

In line with this approach, we will present *Text-To-Query* (T2Q), a system suggesting structured analytics that are dynamically generated to illustrate a textual content. For instance, the user might get charts and tables built from data in her corporate warehouse to illustrate an email she is reading or enrich a presentation she is working on.

In the T2Q example, the entry point for the user to get suggestions of analytics is a text (an unstructured document). On the other hand, assuming a user is currently interested in a given analysis, we will present a method leveraging our framework to suggest related visualizations among those already defined in a collection (or repository) of BI documents like dashboards and reports.

To conclude with exploratory recommendation scenarios, we will illustrate our approach with simple examples of recommendations exploiting the

user's location. This information is obtained through a mobile application.

1.2 Search-related recommendations and personalization

We described exploratory recommendation scenarios which contribute to broaden the user's vision on a given topic and help her discover resources. We now introduce *search-related* scenarios meant to assist the user to formulate her queries and to respond to them.

We will present a case study around an ongoing federated search project and describe the use of our GR and situation management frameworks to enable personalization and recommendations at different levels. In particular, techniques developed in Chapter V led to the definition of personalized query expansion component, which we use to enable auto-completion in an interactive query designer. Besides, federated search involves the aggregation of multiple sources of information which may introduce confusion for end-users. We present how our framework contributes to the personalization of search results thanks to simple user preferences.

The rest of this chapter is organized as follows: Section 2 presents the T2Q system and a set of clients in the form of add-ins for office applications. Finally, Section 3 presents personalization in the federated search project mentioned above.

2 Text-to-query

The description of the knowledge worker's environment by Schwarz highlights the fact that most of the information generated within companies is unstructured data [Sch05]. The amount of knowledge available in a company in unstructured form has been estimated to 80-85% by business analysts. Moreover, the Internet has become a fundamental source of information for business users, mostly unstructured as well.

An important share of unstructured data is not going to change inside companies and on the Web, because it is a natural communication means. Therefore, it can often be helpful to bring relevant structured data to a user, based on unstructured content she's manipulating, like emails, reports, presentations, web pages, etc. Next-generation Business Intelligence (BI) platforms should allow a smooth transition in both directions: unstructured - structured and structured - unstructured data. Given that most users are not database or BI experts, we focus on bringing structured data based on an unstructured users' input. This can serve to broaden the user's view on a given topic, or provide more insight on mentioned concepts.

The T2Q system we present in this section is a step to tackle this challenge in the context of large data warehouses. Our aim is to provide users with relevant aggregated content from a data warehouse based on unstructured textual content. In particular we investigate how to generate multi-

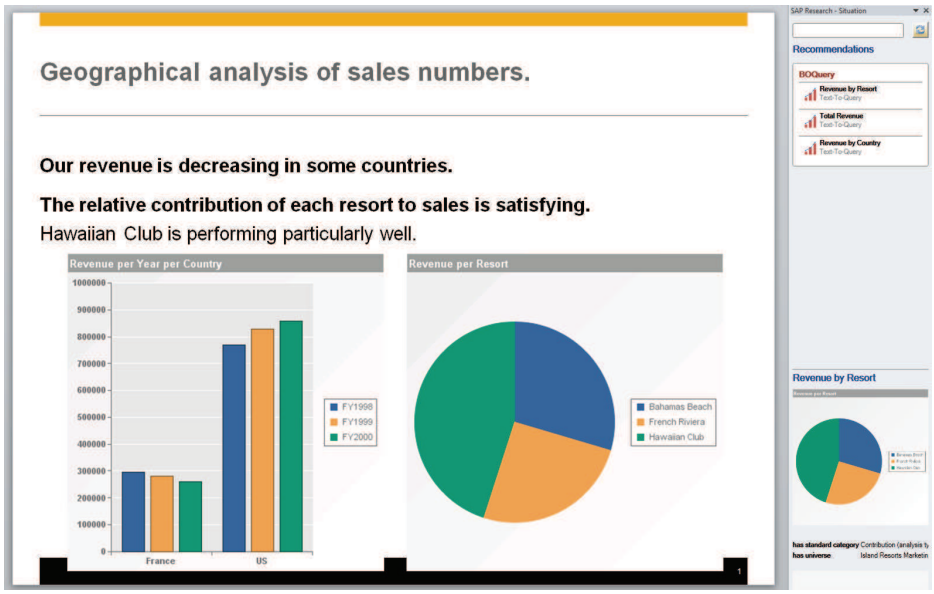

Figure VI.1: Text-To-Query exposed to the end-user as an add-in for PowerPoint. Charts suggestions appear in the right panel and can be dragged and dropped in the slide.

dimensional queries related to a text and how to automatically choose the best visualization for their results. Figure VI.1 illustrates a sample client surfacing T2Q recommendations as an add-in for an office application.

2.1 Dictionary generation for entity extraction

In Chapter V we introduced the semantics of multi-dimensional domain models defined in data warehouses. In order to map textual contents to these models and automatically generate structured queries, our systems exploits periodically re-generated dictionaries, which are applied to a text using the standard named entity recognition (NER) framework described by [Hul99] (SAP BusinessObjects Text Analysis). In this section, we briefly introduce how we generate, enrich and leverage these dictionaries.

2.1.1 Dictionaries for data warehouses metadata

In general our dictionaries represent the core concepts of a data warehouse and some auxiliary information:

Measures such as `Revenue` are usually the expected result of a query. Our system generates one dictionary for measures of each domain model of the data warehouse.

Dimensions such as `Country` are the core primitives that are used in technical queries. They specify the level of granularity that measures can be aggregated on (e.g., `Revenue` per `Country`). Again, we generate one dictionary with dimensions for each domain model.

Dimension values are used to filter the output of a query (e.g., `Revenue` in *Germany*). For each dimension, our system maintains a separate dictionary with its values (or instances). Therefore, when analyzing textual contents, we can also infer from the occurrence of a value the corresponding dimension.

Standard analysis categories define fixed terms that are used frequently in certain contexts and provide hints on the intention of the query. In particular, this dictionary contains weak time and geographic mentions such as "here", "there", "before", "year". Standard analysis categories and their usage are presented in detail in Section 2.2.

2.1.2 Relaxing dictionaries for entity extraction

Previous work has stressed the fact that entities in a text are very often referenced in a form that is not exactly the one present in the dictionary [ACK09]. This is a well known issue for entity extraction but also an important concern in the duplicate detection and record matching research areas, surveyed by [EIV07]. For this reason, [CGX09] presents a developed method to facilitate approximate entity matching by automatically generating variants for a reference entity from a collection of documents.

The issue of syntactical heterogeneity is an important challenge but not the main focus of our work. However, we facilitate two techniques for relaxing the dictionaries. First, our system leverages synonyms and orthographic variants, which are maintained in resources like WordNet and EuroWordNet. Therefore, we can identify dimension values such as *United States of America* (in a `Country` dimension) by terms like "US". Second, we apply stemming on the dictionary tokens, such that we can identify the dimension `Country` in a text like "the revenue in countries that...". Both, synonyms and stems are compiled into dictionaries.

We exploit metadata generated in the pre-processing phase – data warehouse dictionaries and trigger words associated to standard analysis categories – to analyze incoming texts, and return potential technical queries. Therefore, the text is passed to the NER framework which applies all necessary pre-processing such as stemming and dictionary matching. As result, we derive per sentence a set of identified measures, dimensions, dimension values and trigger words for standard analysis categories.

We introduced the use of NER techniques in conjunction with data and metadata from warehouses to recognize business entities mentioned in text

documents. In the next section, we describe our algorithm to dynamically build and recommend meaningful queries out of extracted entities.

2.2 Query generation from a text

In this section, we introduce the notions of *standard analysis categories* and *entity analysis context*. These are used in the dynamic query generation algorithm that we eventually present.

2.2.1 Standard analysis categories

Definition. On top of automatically generated categories (see Section 2.1), we define *standard analysis categories* (SAC) to extract analysis intentions from a text. These categories are those that are frequently used in querying, reporting and analysis. We associate to each category terms and expressions that relate to it. This kind of dictionary describing key aspects of analysis intentions is not part of common data warehouses and had to be defined manually. Among standard categories, we distinguish:

Dimension types. Analyzing data and reporting on it often involves classic dimension types, such as `Geography`, `Time` or `Organization`. It is for instance very common to query for the distribution of a given indicator per country. Terms associated to the `Geography` category can be very specific like "country" or "city", or more vague like "where", "here", etc.

Analysis types. Analysis type categories describe general analysis intentions such as `Trending`, `Comparison`, `Contribution` and `Ranking`. These analysis types can be used to help refine the suggested visualization by selecting the most adapted chart, as discussed later in Section 2.2.3.

Subject types. Subject types define broad common business topics, such as `Sales`, `Finance`, `Performance`, etc. They belong to standard categories as they are bound to be reusable in most businesses or industries. "Customer", "client", "sell", "revenue", "buy", "product" are some of the general terms related to the `Sales` subject type. Subject types are the most costly to define among standard categories and the most tightly bound to the business domain.

Mapping domain entities to SACs. Our aim with SACs is to enable rather abstract requests like "`Sales` analysis / `Geography` / `Time`", which could result in the actual query *Revenue per Country per Year*. Doing so requires a mapping between objects of the domain model and SACs. For instance, dimensions like `Country`, `City` are mapped to `Geography` and `Year`, `Quarter` and `Month` are mapped to the `Time`. The last operation performed after the dictionary creation process discussed previously is thus the categorization of measures and dimensions of the domain model in available SACs.

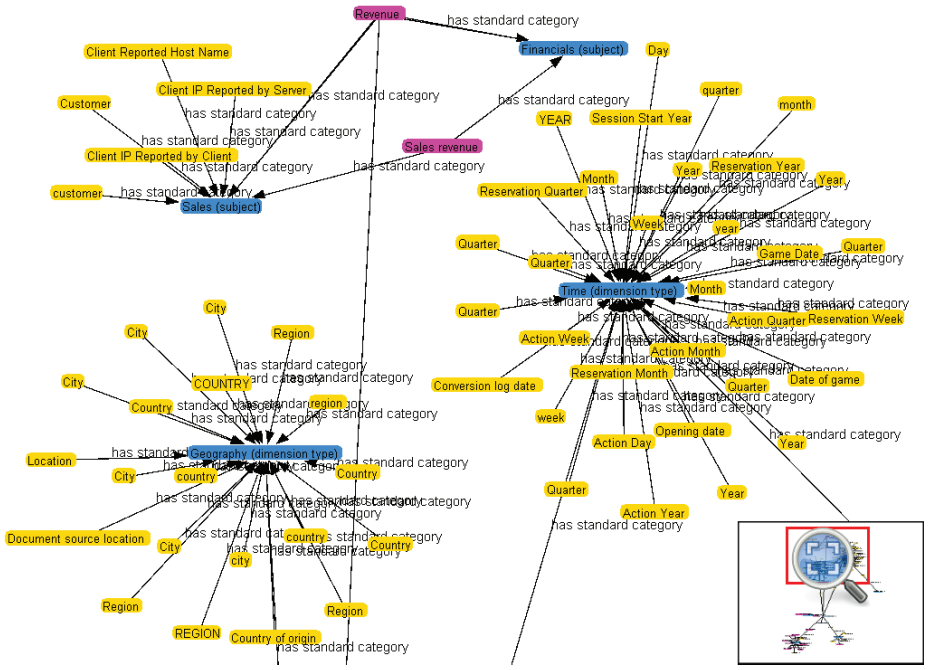

Figure VI.2: Mapping of some measures (purple) and dimensions (yellow) to Standard Analysis Categories (blue).

We use the dictionary of SACs to extract entities in the names of measures and dimensions (using previously discussed text normalization techniques). As an example, dimensions like `Reservation year`, `Quarter` or `Event month` are associated to the standard dimension type `Time`. Objects are associated to all SACs that could be extracted and, as a result, the association *domain object – SAC* is a one-to-many relation. Figure VI.2 illustrates the results of this mapping applied to some dimensions and measures of the warehouse we used for test and demonstration purposes.

The method presented here is extremely simple and poses two main issues. First, the dictionary of SACs is key but enriching it to cover new standard subjects and dimension types requires some domain knowledge. The second difficulty regarding this process is that it is error-prone. In particular, polysemous words may result in incorrect associations [BHH+10]. To deal with such erroneous associations, administrators have the ability to manually validate results using, e.g., Protégé[1] since graphs of associations are serialized as a simple RDF file. Even though this appears as an extremely simple method, we reckon this mapping does not necessarily induce an excessive knowledge-design phase since we operate in the context of the controlled (thus limited) vocabulary defined in models of a data warehouse.

[1] http://protege.stanford.edu/

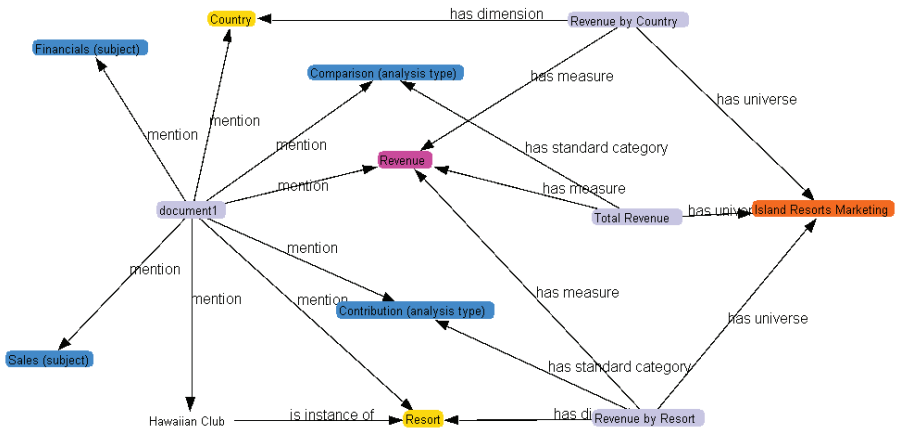

Figure VI.3: Graph view of entities extracted in *document1* (left part), which corresponds to the presentation slide illustrated in VI.1.

2.2.2 Entities analysis and query completion

In previous sections, we presented our approach to create dictionaries from multi-dimensional models. Then, we introduced SACs which are artifacts used to help infer analysis intentions thanks to a pre-defined dictionary. In this section, we describe the actual query generation process which starts by performing entity extraction on the input text.

The entity analysis context. In Section 2.1 we indicated that the output of the NER phase is a collection of entities which can be grouped by sentence (and by paragraph). Let $E = NER(text) = \{e_1, \ldots, e_p\}$ denote the set of entities – domain models' entities as well as SACs – extracted in the input *text*, containing n sentences. In the graph of Figure VI.3, extracted entities are represented by edges starting from the node `document1` (on the left hand side). This document stands for the presentation page depicted in Figure VI.1.

We define the *entity analysis context* (EAC) at document and sentence levels. At the document level, we simply have $EAC(text) = E$, that is the set of all entities. Then, at the sentence level, we consider the following subsets:

- entities explicitly mentioned in sentence j and
- entities mentioned in previous sentences, filtered and flagged as propagated by the *propagate* method which we discuss later in this section.

Eventually, with $E_j = \{e \in E, sentence(e) = j\} \subseteq E$, we define successive EACs as follows:

$$EAC_j(text) = \begin{cases} E_j \cup propagate(EAC_{j-1}(text)) & \text{if } j > 1 \\ E_j & \text{if } j = 1 \end{cases} \quad \text{(VI.1)}$$

Entity propagation. Let us first consider the sample sentence "our revenue is decreasing in some countries". The measure `Revenue` and the dimension `Country` are explicitly mentioned in this sentence. The simple query `Revenue per Country` can then easily be recommended. However, on top of these entities, the verb "decrease" belongs to the standard dimension type `Time`. We thus have $EAC_0 = \{\texttt{Revenue}, \texttt{Country}, \texttt{Time}\}$. We interpret the weak time mention as an indication that the query could be completed by using a dimension mapped to the `Time` SAC. We generalize this and consider that all extracted SACs should be represented by at least one object of the data warehouse. This allows for instance to serve abstract requests like "`Sales` analysis / `Geography` / `Time`".

Let us complete our previous example with a second sentence to illustrate entities propagation: "Our revenue is decreasing in some countries. It is increasing in France though.". Assume that, for the first sentence, the `Time` SAC has been represented by the dimension `Year` during SACs representation. The second sentence also mentions the `Time` SAC via the verb "increase". Therefore, we will favor the use of the same dimension to increase consistency across suggested queries. More formally, the *propagate* method of formula VI.1 adds to EAC_{j+1} entities from EAC_j that were used to represent a SAC. To distinguish between entities explicitly mentioned and those propagated, entities are flagged with the initial sentence and inclusion cause (explicit mention, SAC representation, query extension, etc.).

Query completion. During the iterative construction of EACs for different sentences, we need to be able to complete them with required measures and dimensions. In the previous example, we need to be able to complete EAC_0 by choosing the most appropriate dimension to represent the standard dimension type `Time`.

In Chapter V we introduced semantics of multi-dimensional models plus aditional usage statistics which can be derived from repositories of BI documents such as dashboards and reports. Using these concepts, Secion 5 of the latter chapter then presented a personalized query expansion component which we leverage here. This component QE takes as input the concerned user u, a query q to be completed and parameters $params$ whih can be used to filter candidate entities:

$$QE\colon (u, q, params) \mapsto \{(q_1, s_1), \ldots, (q_n, s_n)\}$$

Using these notations with our example, the initial query q is the set of measures and dimensions in EAC_0 and parameters simply indicate that we

require a dimension mapped to the `Time` SAC. In short, this adds up to find the dimension that:

- is mapped to the standard dimension type `Time`,
- is compatible with all business entities of EAC_0, as determined using functional dependencies (see Chapter V, Section 2.2),
- is preferred by the user (see preferences in Chapter V, Section 3),
- maximizes the co-occurrence with business entities of EAC_0 (see Chapter V, Section 4).

The query expansion component may also be used to return the top k candidates, which leads to offer suggestions of aternative queries. An immediate way to do so is to clone the current EAC for each of the k candidates. This technique may be paricularly useful in the case of limited entity extraction results. For instance, assume only the `Revenue` measure could be detected, leading to the simple query for the total aggregated value of this measure. It is interesting to consider the suggestion of related queries to help the user explore related analytics, like `Revenue per Country`, `Revenue per Year`, etc.

2.2.3 Text-To-Query algorithm

We introduced the notions of *standard analysis categories* and *entity analysis context*, which we use in conjunction with our query expansion component to compose queries related to a text document. In this section we present the final Text-To-Query (T2Q) algorithm.

In Algorithm 2, lines 1-12 correspond to query generation and completion techniques previously discussed. Beyond this, lines 13-17 provide additional optimizations to queries and associated visualizations. First, extracted business entities can relate to a measure, a dimension or a value of a dimension. Values of dimensions are particularly interesting for building more meaningful queries, with filters for instance. In the sample sentence "our revenue is increasing in France", "France" is recognized as a value of the dimension `Country`. We can use this to propose the filtered query `Revenue per Year [Country = France]`. The application of filters is realized at line 14.

Lines 15 and 16 perform further visualization-related optimizations thanks to standard analysis types (`Comparison`, `Contribution`, `Ranking` and `Trending`, see Section 2.2.1). Line 15 uses extracted analysis types to determine the most significant and influence the selection of an adapted chart. In the previous sample sentence, the verb "increasing" refers to the dimension type `Trending`, which indicates that a trend line would probably be more appropriate than a pie chart. However, we do not elaborate more on the complete chart selection algorithm as we rely on a proprietary technology

Algorithm 2 Text-To-Query: recommendations of dynamically generated structured queries to illustrate a text document.

```
1: Text-To-Query(text)
2: queries ← initialize list
3: sentenceQueries ← null
4: eacSentence ← null
   {Text normalization and entity extraction}
5: entities ← extractEntities(text)
6: eac ← buildEAC(text, entities)
   {Queries are generated in different domain models}
7: for all domain in eac.domains do
8:   for j = 1 to numberOfSentences(text) do
9:     eacSentence ← eac.getSentenceEAC(domain, j)
10:    sentenceQueries ← eacSentence.initQueries()
       {Split into groups of compatible objects}
11:    sentenceQueries ← splitCompatible(sentenceQueries)
       {Query completion and optimization}
12:    sentenceQueries ← eacSentence.representSACs(sentenceQueries)
       {Queries filters and visual optimization}
13:    for all query in sentenceQueries do
14:      eacSentence.setFilters(query)
15:      eacSentence.setAnalysisType(query)
16:      optimizeDimensions(query)
17:    end for
       {Append generated queries}
18:    queries.add(sentenceQueries)
19:  end for
20: end for
21: return queries
```

called Best Chart Recommendation. Finally, in the *optimizeDimensions* method of line 16, we use hierarchies and functional dependencies (when available) to re-order dimensions. For instance, in the query `Revenue per Quarter per Year`, dimensions `Quarter` and `Year` should be swapped. Otherwise, the resulting chart would have axis values ordered by `Quarter` (*Q1 / 2009, Q1 / 2010, Q1 / 2011*, etc.) rather than `Year` (*2009 / Q1, 2009 / Q2, 2009 / Q3*, etc.), which may be misleading.

2.3 Experimental results

In the previous section, we described operations performed at runtime to generate analytical queries related to a text. In the following we describe the architecture of our system, using the *Graph Repository* (GR) and situation

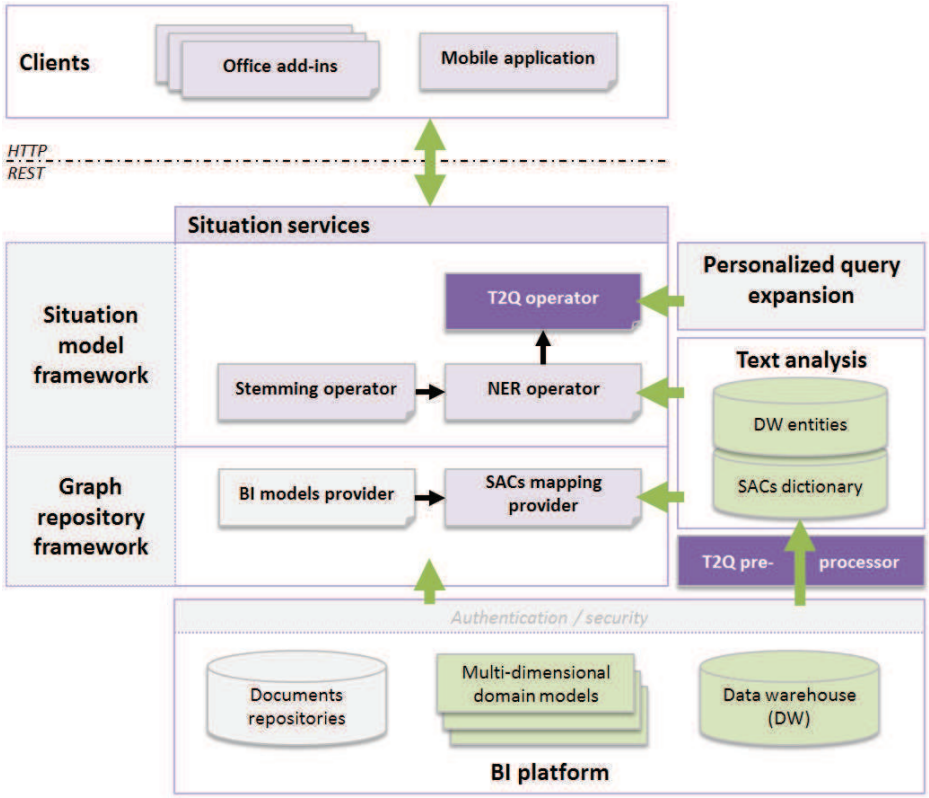

Figure VI.4: Architecture overview for Text-To-Query integrated with the graph repository and situation management frameworks.

management frameworks (see Chapters III and IV).

2.3.1 Architecture overview

Architecture wise, T2Q includes pre-processing and runtime components. Pre-processing ones are responsible for long offline treatments, in particular the generation of dictionaries from available multi-dimensional models of a data warehouse. These dictionaries are refreshed regularly to include data newly loaded in the warehouse. These operations are handled by the component called `T2Q pre-processor` in the architecture overview of Figure VI.4. The mapping between entities of the warehouse and SACs previously described is also performed and controlled offline. In the architecture diagram, the `SACs mapping provider` exposes in the GR this mapping, with gaphs like the one in Figure VI.2.

Then, in order to benefit from personalized components exposed by the GR and provide dynamic recommendations at runtime, we implemented the following *operators*, some of which were also briefly described in Table IV.1 (page 72):

Stemming. Applies stemming to unstructured resources. Stemming consists in text normalization and reduces words to their roots. For instance, the sentence "it is raining cats and dogs" would be reduced to "it be rain cat and dog".

NER. Performs named entity recognition on text resources, extracting entities with various dictionaries. Should they be accessible to the user, extracted entities are mapped to objects of the data warehouse.

T2Q. Combines results of entity extraction with the semantics of a multidimensional domain models and usage statistics (through the query expansion component as described in Section 2.2.2) to suggest meaningful queries.

The question remains as to the activation of these operators. In Chapter IV, Section 2, we presented *activation rules* which can be defined to react to events and trigger operators. Our aim is to react to events of the type `user` *reads* `document`, where `user` is the current user and `document` the text document being read. We illustrate below the rule which can be expressed to trigger the `T2Q` operator:

```
<rule>
  <event>
    USER "http://.../read" "http://.../#UnstructuredDocument", ANY
  </event>
  <condition>
    "http://.../ner" "http://.../hasProcessed" EVENT.object
    AND NOT("http://.../t2q" "http://.../hasProcessed" EVENT.object)
  </condition>
  <action>
    CALL "http://.../t2q" WITH EVENT.object
  </action>
</rule>
```

2.3.2 Clients for supported data acquisition

We reckon that common office applications are a good place to expose recommendations offered by T2Q, as it can uphold a use-case scenario that we call *supported data acquisition*. Let us consider a user, sales manager in a company of the tourism industry, managing resorts around the world. She is working on a presentation to report on her performance in terms of sales. In this situation, T2Q can help her analyze the document she is working on to get immediate and natural access to her corporate data. A video presentation of this scenario is available online[2], but based on an older version of

[2]https://www.sdn.sap.com/irj/boc/index?rid=/webcontent/uuid/10971dfd-2ff1-2b10-6aa6-d58f939d76d9

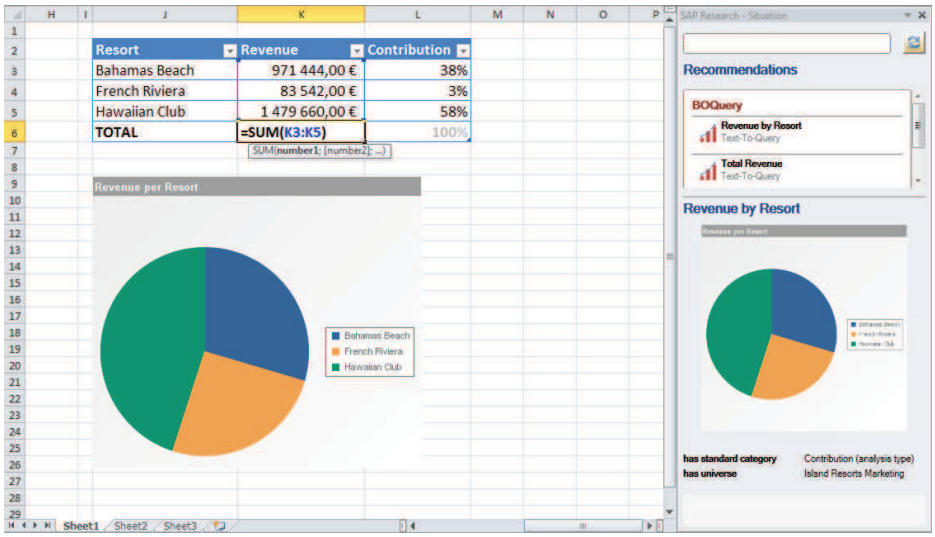

Figure VI.5: Excel add-in for the situation platform, exposing Text-To-Query recommendations. Suggested queries appear in the right panel. Charts and data tables and can be inserted in the spreadsheet.

T2Q.

Figure VI.1 illustrates the PowerPoint integration, letting the user analyze slides she is reading. Similarly, the Excel integration is illustrated in Figure VI.5. Charts and data tables associated to recommended queries can be inserted (by drag and drop) in the spreadsheet to perform further calculations or create more personalized charts. These add-ins are also available for Outlook and Word to further assist the user during document creation and analysis processes.

Add-ins were developed on top of basic REST services exposed by the situation management platform. These services enable authentication, events posting and situation retrieval (see Chapter IV, Section 3.3). In particular, recommendations of T2Q may be retrieved via the corresponding situation dimension.

Similarly, we designed a mobile client application using situation services. Figure VI.6 illustrates this client which allows a user to keep consuming the system's recommendations on the go. The user can select one of these recommendations to get the chart representation and the underlying data table. This first sketch of application is clearly limited but further extensions and improvements could benefit greatly, for instance, from speech recognition capabilities. Using this, the user could refine interactively her analysis until she reaches what she is looking for to eventually share or save it for later.

Figure VI.6: Mobile client for the situation platform, exposing Text-To-Query recommendations. The user can select suggested queries in a list to look at the generated chart and underlying data.

3 Personalization in search

In this section, we present another important experimentation for the concepts that we developed in our work. In particular, we illustrate how the *graph repository* (GR) and situation management frameworks can be used to enable various levels of personalization in a federated search project.

3.1 An open architecture for search

This project for a federated search engine has been initiated in 2011 as part of a broader project inside SAP, called *Business Web*[3]. We now briefly present the open modular architecture that was defined to allow contributions at different levels in the search platform.

3.1.1 Architecture overview

Figure VI.7 depicts the overall architecture that was set-up. In particular, the search platform handles four main types of extension points or plugins:

[3]http://www.sap.com/corporate-en/our-company/innovation/research/business-web/index.epx

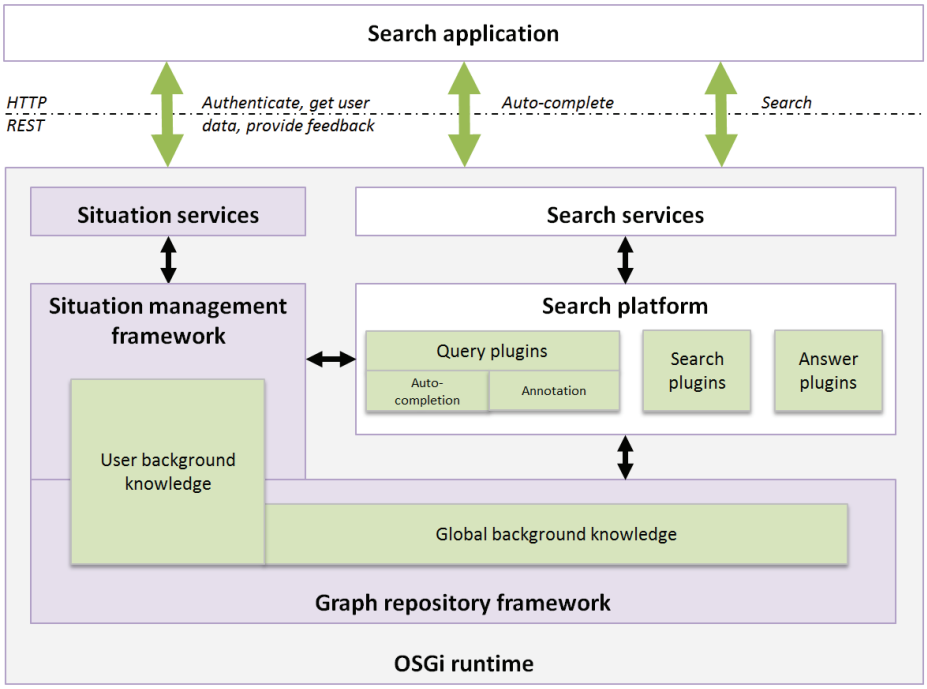

Figure VI.7: Architecture overview for the open search platform, using our *graph repository* and *situation management* frameworks.

Query auto-completion plugins. As users type in their search query, the system suggests possible annotations to complete the request. Annotations describe key entities that could be recognized and anticipated by the system. This process is called auto-completion and is implemented by different plugins, as proposed annotations may originate from different sources.

Query annotation plugins. Annotation plugins are meant to mark users' queries thanks to a certain number of annotations, of possibly varied types. For instance, an annotator is responsible for the capture of measures and dimensions, with techniques similar to those described in Section 2. The output of these plugins is combined to build a complete query tree.

Search plugins. Search plugins are responsible for performing the actual search on different source systems, taking as input the user's query and the query tree representing extracted annotations. Search plugins may target corporate as well as Web content, structured or not. Numerous examples may be considered from the simple one wrapping an existing search engine like Google to more complex corporate search systems.

Answer plugins. Answer plugins are meant to enable multiple represen-

tations of a single result. For instance, this is useful when dealing with results in the form of BI queries, which can be represented either as a data table or as a chart. Answer plugins will not be further discussed here though.

It is worth noting that an OSGi[4] runtime is used to enable a high level of flexibility and real modularity. The GR and situation management frameworks were implemented in this runtime as well (see Chapter IV, which enabled a loosely coupled and progressive integration.

In particular, the OSGi runtime offers a service registry which enables bundles to dynamically declare and consume services. The four types of plugins previously described for the search architecture are dynamically registered to the search platform using this registry. Similarly, the GR is exposed as an OSGi service so that search-related plugins can consume users' repositories and situation graphs to personalize their results. For this purpose, user authentication is performed in the search application thanks to the corresponding situation service and the token is passed through as a parameter of the search request. In the context of the search Web application, session cookies may also be used.

3.1.2 Query parsing and annotations

User queries are parsed and annotated to build query trees. A query tree represents annotations in a hierarchical manner and is produced by aggregating results of the different query annotation plugins available in the search platform.

Briefly, annotations are simple metadata indicating chunks of the user query that correspond to identified entities. Simple annotations usually indicate the entity ID, its name, type, offset (in the query) and length. These entities can be extremely varied and are meant to help with the query analysis process. For instance, in the query "revenue 2008", "revenue" may be used to recognize the measure `Sales revenue` and the term "2008" may be recognized as a value of the dimension `Year`. Entity extraction techniques have been discussed in more details in Section 2 with the *Text-To-Query* system.

3.2 BI queries and charts search

In Section 5 of Chapter V, we presented a personalized query expansion component, based on semantics of multi-dimensional models and usage statistics of BI entities derived from repositories of reports and dashboards. In this section, we illustrate results obtained with this component to implement an

[4]http://www.osgi.org

auto-completion plugin. This plugin iteratively suggests multi-dimensional queries to complete the user's textual query.

3.2.1 BI query auto-completion

As the user types in the search text box, candidate measures and dimensions are proposed to her as auto-completion suggestions. The user can iteratively build queries and trigger their execution to obtain various visualizations (chart or table) of the data. We use the query expansion method described in Section 5.1 of Chapter V with a filtered set of candidate entities, containing only those which names start with characters entered by the user.

First suggestion case. The user starts typing and selects a completed prediction, e.g., the measure `Sales revenue`. Measures and dimensions are suggested based on their occurrences in users' graphs and the user's preferences. Figure VI.8.a) shows measures (from distinct domain models) suggested when the user starts typing "sa": `Sales revenue`, `Avg of savegoal` and `Keeper save goal`. The auto-completion initialization requires that the user roughly knows the names of objects she wants to manipulate, which may be a barrier to adoption. To help her get started and explore available data, suggestions can be surfaced to the user before she even starts typing. For instance, the most commonly used measures and dimensions of various domain models could be suggested to start with.

Iterative query completion. Further suggestions are adapted to measures and dimensions previously selected. In Figure VI.8.b), the user has selected the first suggestion `Sales revenue` and keeps typing "c". The system suggests the two dimensions `City` and `Category`. These two dimensions are compatible and often co-occur with the selected measure.

The personalized query design assistant we discussed was integrated in the search application illustrated in Figure VI.8. Besides, this component could be used more generally in other BI tools to help users build their queries, for instance in a dashboard or report designer.

3.2.2 Related charts search in dashboards and reports

Let us now describe another use case for the GR framework which is used to handle user-specific and global background knowledge. Charts in dashboards and reports usually present very little associated textual metadata. For instance, charts mainly have a title and a description, the latter being often left empty by users.

In this context, the GR framework can serve to develop custom providers and enable semantic-based charts search. In particular, in Chapter V, Figure V.6 illustrates the graph that can be built thanks to a dedicated provider

Figure VI.8: Screenshot of auto-completion in the search text box. (a) First suggestions after characters "sa" and (b) suggestions following the selection of measure *Sales revenue* and character "c".

for the dashboarding solution *SAP Exploration Views*. This provider requires appropriate user credentials and the resulting graph is thus user-specific.

Using this graph, it is for instance possible to issue SPARQL queries and retrieve charts relating to measures and dimensions of the user query. These measures and dimensions can be retrieved as an output of the query tree building process, as previously described. Assuming the user query was annotated with the `Sales revenue` measure, the following SPARQL query selects graphs referencing this entity:

```
SELECT ?chart
WHERE {
  ?chart rdf:type <http://...#Chart> .
  ?chart grepo:references <http://.../SalesRevenue> .
}
```

This example is pretty simple and illustrates the mention of a single measure. More complex queries can be built to enable multiple mentions, for instance with the user query "sales revenue per year per country":

```
SELECT ?chart
WHERE {
```

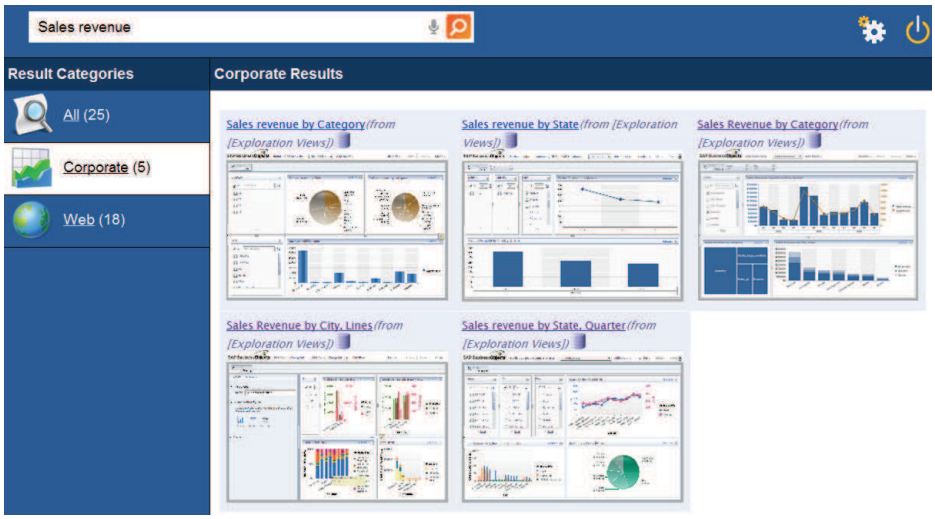

Figure VI.9: Charts search results for the query "Sales revenue" annotated with the measure of the same name, from dashboards available in the user account.

```
  ?chart rdf:type <http://...#Chart> .
  {
    { ?chart <http://...#references> <http://.../SalesRevenue> }
    UNION
    { ?chart <http://...#references> <http://.../Year> }
    UNION
    { ?chart <http://...#references> <http://.../Country> }
}
```

These SPARQL queries do not provide ranking for returned charts. However, it is possible to score them with the user's preferences (see Chapter V, Section 3). Let c denote one of the chart results, and $ref(c)$ the set of entities referenced by c. Let q denote the user query and $ann(q)$ its set of annotations. The chart ranking function may simply be defined as follows:

$$rank_u(c,q) = \frac{|ref(c) \cap ann(q)|}{|ref(c)|} \times Avg_{e \in ref(c)}\left(pref_u(e)\right)$$

3.3 Plugins personalization

The federated search system that we described presents interesting usability challenges for users. In particular, all different types of plugins provide additional information and may influence the interactive behavior of the search application. Consequently, users might get confused as they are not all interested in the same information sources. In this section, we briefly

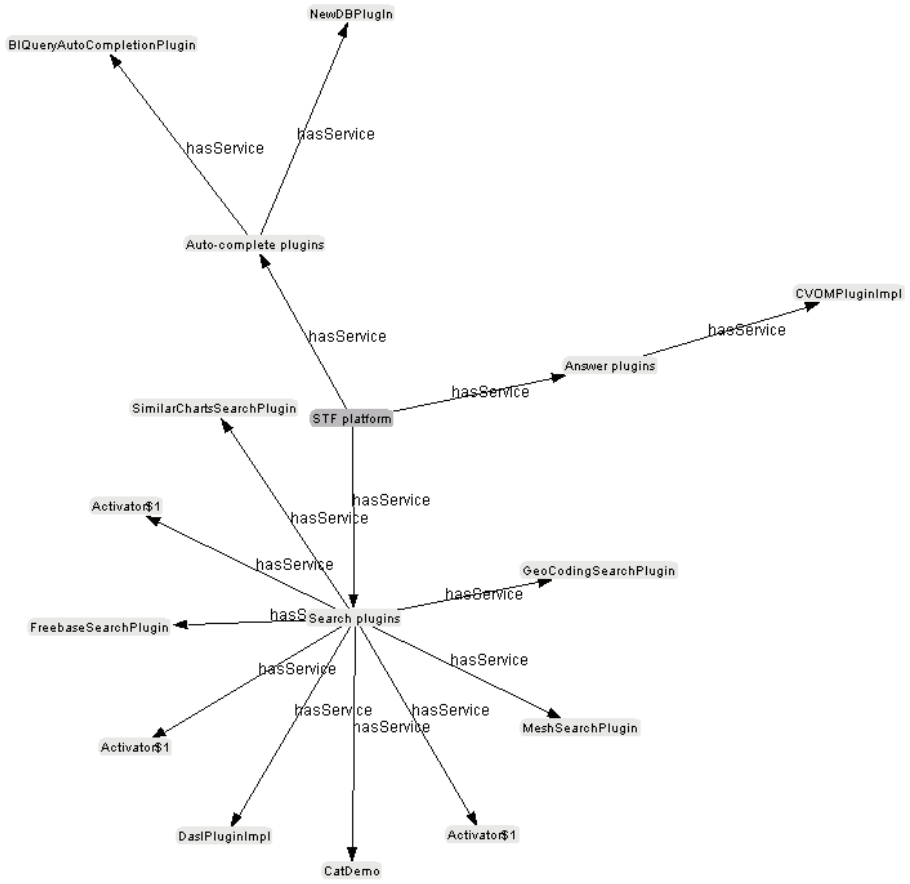

Figure VI.10: Graph of plugins available in the search platform, of different types: search, answer and auto-completion plugins.

introduce custom user preferences to personalize the system's behavior and thus the user's search experience.

3.3.1 Authorizations and ranking

Figure VI.10 illustrates the graph of plugins available in the search platform. On top of these plugins, we define simple preferences in the form of statements like `user` *prefers* `BIQueryAutoCompletionPlugin`. Since statements are defined as extended nodes (see Chapter III, Section 2.1) they enable the definition of custom attributes. From the RDF perspective, this implies to work with reified statements.

We use this capability to add a *score* attribute to preference statements. This allows users to rank their plugins by defining numerical preferences between 0 and 1. Plugins with a preference of 0 could be disabled for this user. This approach can help improve performance for users who require very

specific information, for instance from a single source. Scores determined by the user's preferences can also be used in the results scoring process, for instance by weighting a plugin's contribution in search results.

3.3.2 Other forms of personalization

Let us conclude with other possible illustrations of personalization capabilities enabled by our frameworks.

Personal filters and self-references. We want to be enable the resolution by some query processing plugins of self-references in users' queries. As an example, in the query "revenue of my store", the pronoun "my" refers to the user herself (a sales manager responsible for a given store). This illustration is a simple example but more generally, advanced query interpretation techniques may be developed, e.g., using linguistic patterns [KBA11].

We now simply introduce user preferences that can be defined to indicate that users have personal filters for particular dimensions. In the example above, the user can have a personal filter on the dimension `Store` for the value `New York`. The query processing plugin could access the user's situation model – through the situation provider in the GR framework – and determine if and how the query `Sales revenue by Store` needs to be filtered. Such a personal filter can very simply be represented with statements like `user` *hasPersonalFilter* `New York` and `New York` *isValueOf* `Store`.

User's location and social network. Similarly, search plugins can exploit the *geography* dimension of our situation model to enable location-aware search results. Location is a simple yet extremely valuable context information and is therefore frequently used by various services, in particular with the advent of pervasive computing and GPS-enabled smartphones. A simple experimentation was made using the API provided by *TellMeWhere*[5]. We developed a minimal operator using these services, triggered by an activation rule that reacts to events of the type `user` *asks* `Restaurant` (or petrol station, etc.). Practically, the user's location can be retrieved either from the mobile application of Figure VI.6 (GPS coordinates) or through the Web application of Figures VI.8 and VI.9 (IP address).

Eventually, the user's social network – accessible through the corresponding dimension of the situation model – can be used to simply implement an auto-completion plugin for the user's contacts. For instance, when the user starts typing "ni", she receives suggestions of users like `Nick Cox` .

[5] http://www.tellmewhere.com/

4 Summary and discussion

In this chapter, we illustrated the use of our GR and situation management frameworks in two projects, to enable recommendations and personalization at different levels. First, the Text-To-Query system was presented to recommend structured analytics, dynamically generated to illustrate a textual content. This system was mainly show-cased using add-ins integrated in common office applications, to back a scenario that we call *supported data acquisition*. Then, we described the open and modular architecture of a federated search project which enabled us to bring a significant contribution in terms of personalization and more generally background knowledge management. In particular, we introduced a plugin for BI query auto-completion to ease the user querying process and another one to search for related charts in the user's collection of dashboards (or reports).

First, we would like to discuss the fact that we did not come up with a proper evaluation for recommendations made, be it with Text-To-Query or the auto-completion plugin. The point is that we have been relying on two prototypes, SAP Exploration Views for the dashboarding solution and Social Network Analyzer for users' social network management. Therefore, we did not have access at this time to important repositories of dashboards with enough different users and created dashboards. Second, users of the first solution were not yet tied to users of the second which make it complicated to design a joint dataset for evaluation.

To conclude, we reckon that the GR and situation management frameworks are particularly valuable and proved effective to enable multiple contributions in a distributed project. For instance, preferences can be freely expressed with rich semantics which allow precise, application-specific and adaptive personalization scenarios. Beyond rather static preferences, dynamic aspects enabled by the situation management framework have only partially been exploited and experimented, due to lack of time and resources. However, business events should be further leveraged to enable, for instance, dynamic scoring. This may be achieved by assigning dynamic "boost" factors to nodes of the situation graph, depending on the user's interactions with close resources.

5 Conclusion

In this chapter we elaborated on various experimentations realized thanks to GR and situation management frameworks, previously introduced in Chapters III and IV.

We first presented T2Q, a dedicated RS dynamically generating multidimensional queries related to a text document. Measures, dimensions and their values are extracted from the text thanks to a NER phase. Recognized

entities are combined to form meaningful queries using in particular the query expansion component introduced in Chapter V.

Then, further experimentations were proposed in the context of an open federated search platform. The search platform enables the definition of various plugins to cover tasks as varied as auto-completion, query parsing and search. We presented a certain number of *providers* and *operators* implemented in these frameworks, in this chapter as well as Chapter V, Section 5.2.1. Besides, our work could be integrated to the search platform to provide various capabilities. Most importantly, the GR framework is used for global and user-specific data or knowledge management. Examples of such data include (but are not limited to) users' preferences, which usage for plugins personalization was briefly illustrated in Section 3.3.

T2Q proved extremely interesting to help users reach specific information in a data warehouse, using a first level of content analysis. However, it cannot be denied that text analysis operations performed by T2Q remain quite superficial and as a result, only simple queries can be suggested to the user. There is an ongoing research effort – in the context of the federated search project previously mentioned – to further encourage natural language processing and improve the interpretation of users' queries. This effort is based, among other things, on the pattern-based question answering system proposed by Kuchmann-Beauger et al. [KBA11]. The user's question can be expressed using natural language. Patterns are then used to determine combinations of entities recognized in the question. Patterns that match the question can then be translated into a query expressed in the appropriate language.

To conclude, we reckon that GR and situation management frameworks described in our work present a significant potential to enable diverse personalization scenarios. Spontaneous recommendations and dynamic aspects of these frameworks proved more difficult to illustrate though. Therefore, we reckon that future efforts should first focus on a first phase of re-engineering, meant to take the GR framework to a more stable state. Then, users' global- and application-specific preferences should be more formally defined so they benefit in a more general way, for instance to search plugins. Further, generic techniques inherited from RS could be exposed through the framework, encouraging GR plugins to define or re-use personalization-related indicators. Important examples of recommendation techniques were introduced in Chapter II, Section 2 and a proposed application to BI entities was presented in Chapter V, Sections 3 and 4.

Chapter VII

Conclusion and perspectives

Contents

1 Summary

As the amount and complexity of data generated and maintained by information systems and their users keep increasing, information overload becomes a problem of utmost importance. In response to this issue, personalized systems have emerged and aim at proposing more relevant information and services to their users.

Recommender systems (RS) are an interesting kind of personalized systems, designed to help users explore vasts spaces of available information. If RS have grown extremely popular in a Web context, we believe such systems could also be valuable in corporate environments, dealing with privates resources of an organization. Numerous techniques were developed and are often combined to form hybrid recommendation strategies, as discussed in Chapter II, Section 2. We presented for instance the two most common recommendation techniques, with content-based (CB) and collaborative RS. CB systems stem from information retrieval and information filtering fields, relying on a keyword representation of items and users' preferences in a vector space model. Collaborative filtering (CF) systems do not perform content analysis on items to recommend but rather exploit the so called collective intelligence, e.g., with ratings. CF systems assume that a user is bound to be interested by items that other users with similar tastes liked.

Perosnalized systems usually rely on some model of the user and her preferences to serve better selected and more adapted information. RS are no exception and involve a certain amount of user modeling, centered around

users' interactions with the collection of resources (or items) being considered for recommendation. However, RS often involve an important precomputation phase which may limit their ability to dynamically adapt to a changing user environment. Context-aware (CA) systems represent another important area related to personalization. Remarkably, the approach of CA systems is mainly targeting dynamic adaptation in heterogeneous environments, encouraging model sharing and re-usability accross applications. Context modeling is a key issue in this area and most recent work argue that graph-based systems are better suited for knowledge representation. In particular, Semantic Web technologies like RDF and SPARQL seem well adapted to handle information heterogeneity and enable a high level of expressiveness. Besides, reasoning on context knowledge may also be useful to deal with incomplete information.

Our work focuses on personalization and recommendations related to BI tools. BI analytics rely on an important phase of multi-dimensional modeling to enable an easier exploration of data warehouses by non-expert users. Multi-dimensional models mainly define measures and dimensions to query for available data. Measures are numerical facts that can be aggregated against a certain number of dimensions, e.g., to get the `Revenue` generated by a given `Product` in some `Countries`. In real-life production deployments of BI platforms, multi-dimensional models can grow extremely complex with thousands of measures and dimensions, leaving users with the difficult task of finding the information they are looking for. However, we observe that very few techniques of RS and CA systems have been employed so far to develop personalized BI solutions. Our work thus targets the exploration of syneriges between RS and CA systems to develop personalized data access solutions, suited for users of BI platforms and data warehouses (DW).

Chapter III first introduced the requirements imposed on the user situation monitoring platform that we consider. Model sharing and re-usability across different applications seems particularly important in our case. Indeed, modern BI platforms provide different tools and model sharing could greatly benefit to all these applications, e.g., by reducing the amount of redundant user modeling. Besides, security and privacy are two key aspects to enforce in a user monitoring system, in particular in corporate settings. Privacy is crucial and the system must ensure that user data is not going to be accessible by others. Security on the other hand must also be observed from a system perspective. Indeed, most corporate systems impose access control rules and users may not have access to the same data, leading for instance to different views of the DW. In response to these challenges, we presented a graph-based situation model relying on situation statements, the unit of information in situation modeling. These statements are triples (subject-predicate-object) extended with additional metadata to capture for instance the fact that context knowledge is inherently time dependent. The situation is defined as a collection of statements and takes the form of a

graph centered on the concerned user. On top of this graph view, we defined a certain number of situation dimensions meant to offer a structured and categorized view of lower level information. Examples of such dimensions include (but are not limited to) social and geographical aspects as well as preferences and recommendations.

Graphs involved in the description of a user's situation are managed in our system thanks to a framework called graph repository (GR). This modular framework gives developers the ability to define new information sources with an extensible set of providers. Providers may connect to remote systems thanks to users' credentials, assuming users authorized it in the first place. The security model of the GR thus makes it possible for users to define and customize their own GR. Moreover, Chapter IV further elaborated on dynamic aspects of situation monitoring. Situations are indeed evolving as users interact with various applications and systems. The GR framework is extended with active components like activation rules and operators. In response to business events sent to the platform (or raised internally), activation rules are evaluated and may trigger appropriate operators to update graphs according to the event being interpreted. Rules are defined in the event-condition-action (ECA) framework and can be evaluated thanks to queries on the user's GR. Should graphs be managed using a RDF factory, rules evaluation is performed with a translation to SPARQL queries. Chapter IV introduced these active components and discussed more extensively the GR framework, which proves important to enable a certain adoption and encourage additional developments.

We introduced the GR and situation management frameworks to form a rather generic personalization platform. Later, Chapter V focused on the definition of BI-related components. This chapter presented in particular the semantics of multi-dimensional models which we expose in our framework thanks to a specific provider. Besides, we discussed the use of repositories of BI documents like dashboards and reports to derive additional usage statistics. For instance, measures and dimensions occur in these documents and we exploit this fact to define implicit preferences and a measure of co-occurrence. We defined user preferences related to BI entities according to a quantitative approach, combining explicit and implicit scores. Numeric preferences then allowed us to exploit the traditional Pearon correlation and define users' similarity. In application of these indicators, we discussed a personalized query expansion component, meant to help users iteratively build more complete data queries.

Eventually, we presented in Chapter VI various experimentations realized with our situation management platform. First, we introduced Text-To-Query (T2Q), a system dynamically generating and recommending analytics to illustrate an unstructured text document. T2Q combines semantics of multi-dimensional models and usage statistics – thanks to the query expansion component previously described – to offer meaningful queries and

visualizations. T2Q was show-cased using add-ins for popular office applications. This scenario demonstrated the ability for users to further analyze and enrich documents they are working on, thanks to recommendations of related structured analytics. This supported data acquisition scenario responds to the need for modern BI platforms to enable a smooth transition between structured and unstructured content. To conclude, Chapter VI introduced a federated search project which leverages the GR and situation management frameworks to propose various forms of personalization. The query expansion component was for instance used to offer an interactive query designer through the usual search text box, using an auto-completion approach. Besides, we discussed other possible personalization strategies for various plugins of this federated search project. Federated search involves multiple information sources may lead to confusion for end-users. Thanks to our situation management platform, users may express their preferences for some sources compared to others, leading to less noisy and more personalized results.

2 Discussion and perspectives

First and foremost, we would like to discuss the importance of the GR framework in our contribution. The GR is indeed a key component of our platform, meant to provide graph management capabilities ranging from simple representation to complex querying and reasoning. However, the GR is much more than a simple storage system for graph data since it can also define active components, developed in the form of plugins. The personalization and security model for these plugins enables users to fully customize their own graph repositories, defining what is accessible and how it can be used. This aspect proves crucial to ensure users keep control of what data is collected from various systems by providers. Besides, plugins are more generic than providers and may implement extremely varied capabilites. To sum up, we like to see the GR as a container for both data and applicative components, all of them being fully dedicated to a given user. We reckon this aspect is extremely interesting and presents a drastic change compared to traditional applications which are designed to serve multiple users. On the contrary, every piece of data and every action performed inside a GR instance are entirely focused on the assistance to provide to the concerned user. As a consequence, we believe that the GR framework deserves further efforts to push its development and take it to a more stable state. The industrialization of the GR could for instance base on an important refactoring of the existing code and lean towards a cleaner and simpler architecture. In particular, we argued in Chapter III for the abstraction of the graph storage and querying engine, leading to the definition of specific graphs factories. Factories are used to back graphs data using dedicated triple stores or traditional

relational databases. Even if this abstraction has its interest, it provided little value since even reasonably complex queries require to go down to the appropriate language, be it SPARQL or SQL. The graph storage abstraction thus induced too much complexity in the development and provided limited added value. As a result, we consider that future developments related to the GR should base on a single reference implementation, ideally using a dedicated triple store and SPARQL querying.

Among points that we developed in our work are the dynamic components described in Chapter IV, more specifically activation rules and operators. As previously discussed, activation rules are used to react to business events and trigger appropriate operations and update situation graphs in response. A rule defines conditions that need to be validated before triggering the execution of an operator. Rule validation therefore requires queries to be executed and evaluated on the GR. Assuming a high number of events could have to be processed, this may lead to significant performance problems. For this reason, rules contain a first event filtering part which is used to limit the amount of queries that will actually have to be evaluated and thus limit performance issues. However, we have to admit that we were not able to perform an extensive evaluation regarding this aspect. Besides, the continuous rule evaluation process that we described in Section 1.1 of Chapter IV enables the definition of linear and non-linear evolution scenarios. If linear scenarios are simple to set up and demonstrate, non linear ones induce undeterministic behaviors and are thus more complex to show case in a reproducible way. To conclude on dynamic components, we reckon that they bring a significant contribution to make our frameworks suited for dynamic adaptation. However, to further validate their usage, one would have to conduct an extensive study of performance-related issues and probably optimize rules evaluation in the mean time. On top of that, non linear scenarios should be the focus of future work in this area since they would likely bring more value than simple pre-determined linear scenarios.

The last point that we would like to discuss lies in the balance between the genericity of methods employed and application-specific optimizations. In Chapter V, we introduced a certain number of indicators useful for personalization and recommendations in the context of BI tools. Our work in this regard bases on multi-dimensional models designed for data warehouses, enabling non-expert users to perform ad-hoc query and reporting with key business entities (measures and dimensions). We remain at the business entity level and do not go down to the specific query language. This differs from other work related to query personalization since we do not intend to extend and enrich queries by providing language-specific operators. As a result, our approach undeniably brings less expressivity but it is usable regardless of the actual data warehouse materialization. At the entity level, we defined users' preferences and similarity as numerical indicators which can be used to determine items of interest in various conditions.

The user similarity measure helps define a social contribution in collaborative systems, for instance using other ratings given by users that are similar enough. Although metrics defined in Chapter V aim at being generic and reusable with varied domain models, they are still specific to BI and data warehouses. More generally, this poses the problem of recommendations relying on heavy pre-computations and often made application specific due to advanced optimizations. As part of future work, we would like to extend the work we did to define our collaborative metrics and expose more general computing capabilities through our situation management framework, for instance using the popular Apache Mahout API. However, doing so presents a certain number of challenges, in particular around scalability and data management. Indeed, should developers using our framework define their own metrics, this could lead to important hurdles and impact the performance of the whole platform.

To conclude on perspectives for our work, we would like to see further improvements brought to BI platforms and we reckon that they offer an important potential for personalization and recommendations. Doing so could for instance base on the industrialization and extension of techniques like those described in Chapter V.

Bibliography

[AAH+02] Heikki Ailisto, Petteri Alahuhta, Ville Haataja, V. Kyllönen, and Mikko Lindholm. Structuring context aware applications: Five-layer model and example case. In *Proceedings of the Workshop on Concepts and Models for Ubiquitous Computing*, pages 1–5. Citeseer, 2002.

[ABL09] Sofiane Abbar, M. Bouzeghoub, and S. Lopez. Context-Aware Recommender Systems: A Service Oriented Approach. In *VLDB PersDB Workshop*, pages 1–6, 2009.

[ACK09] Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. Learning string transformations from examples. *PVLDB*, 2(1):514–525, 2009.

[Ala08] Satnam Alag. *Collective Intelligence in Action*. Manning Publications, 2008.

[AS97] Varol Akman and Mehmet Surav. The Use of Situation Theory in Context Modeling. *Computational Intelligence*, 13(3):427–438, August 1997.

[AT01] Gediminas Adomavicius and Alexander Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In Ludger Fiege, Gero Mhl, and Uwe Wilhelm, editors, *Electronic Commerce*, volume 2232 of *Lecture Notes in Computer Science*, pages 180–192. Springer Berlin / Heidelberg, 2001.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[ATZ05] Gediminas Adomavicius, Alexander Tuzhilin, and Rong Zheng. Rql: A query language for recommender systems. *Information Systems Working Papers Series*, 2005.

[BBF+09] Fadila Bentayeb, Omar Boussaid, Cécile Favre, Franck Ravat, and Olivier Teste. Personnalisation dans les entrepôts de données: bilan et perspectives. In *Entrepôts de Données et l'Analyse en ligne (EDA 09), Montpellier, France*, pages 7–22, 2009.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6:161–180, April 2010.

[BCG+08] Manish Bhide, V. Chakravarthy, Ajay Gupta, Himanshu Gupta, Mukesh K. Mohania, Kriti Puniyani, Prasan Roy, Sourashis Roy, and Vibhuti S. Sengar. Enhanced business intelligence using erocs. In *ICDE*, pages 1616–1619. IEEE, 2008.

[BCQ+07] Cristiana Bolchini, Carlo Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. *SIGMOD Record*, 36(4):19–26, 2007.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *IJAHUC*, 2(4):263–277, 2007.

[BFMS06] Erik Behrends, Oliver Fritzen, Wolfgang May, and Franz Schenk. Combining eca rules with process algebras for the semantic web. In Thomas Eiter, Enrico Franconi, Ralph Hodgson, and Susie Stephens, editors, *RuleML*, pages 29–38. IEEE Computer Society, 2006.

[BFPAGS+08] Yolanda Blanco-Fernndez, Jos J. Pazos-Arias, Alberto Gil-Solla, Manuel Ramos-Cabrer, Martn Lpez-Nores, Jorge Garca-Duque, Ana Fernndez-Vilas, Rebeca P. Daz-Redondo, and Jess Bermejo-Muoz. A flexible semantic inference methodology to reason about user preferences in knowledge-based recommender systems. *Knowledge-Based Systems*, 21(4):305 – 320, 2008.

[BGM+05] Ladjel Bellatreche, Arnaud Giacometti, Patrick Marcel, Hassina Mouloudi, and Dominique Laurent. A personalization framework for olap queries. In *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, DOLAP '05, pages 9–18, New York, NY, USA, 2005. ACM.

[BHH+10] Falk Brauer, Michael Huber, Gregor Hackenbroich, Ulf Leser, Felix Naumann, and Wojciech M. Barczynski. Graph-based concept identification and disambiguation for enterprise search. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 171–180, New York, NY, USA, 2010. ACM.

[Bur00] Robin Burke. Knowledge-based recommender systems. In *Encyclopedia of Library and Information Systems*, page 2000. Marcel Dekker, 2000.

[CB07] Sylvain Castagnos and Anne Boyer. Personalized communities in a distributed recommender system. In *Proceedings of the 29th European conference on IR research*, ECIR'07, pages 343–355, Berlin, Heidelberg, 2007. Springer-Verlag.

[CBC08] Iván Cantador, Alejandro Bellogín, and Pablo Castells. Ontology-Based Personalised and Context-Aware Recommendations of News Items. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 562–565, December 2008.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[CDG01] S. Chaudhuri, U. Dayal, and V. Ganti. Database technology for decision support systems. *Computer*, 34(12):48–55, 2001.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(3):197–207, September 2003.

[CFV07] Pablo Castells, Miriam Fernandez, and David Vallet. An adaptation of the vector-space model for ontology-based information retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 19:261–272, 2007.

[CGF07] A.C.M. Costa, R.S.S.G.G. Guizzardi, and J.G.P. Filho. COReS: Context-aware, Ontology-based Recommender system for Service recommendation. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering, CAiSE'07*, 2007.

[CGX09] Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. Mining document collections to facilitate accurate approximate entity matching. *PVLDB*, 2(1):395–406, 2009.

[CHW01] Sonny Chee, Jiawei Han, and Ke Wang. Rectree: An efficient collaborative filtering method. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, volume 2114 of *Lecture Notes in Computer Science*, pages 141–151. Springer Berlin / Heidelberg, 2001.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[CT98] Luca Cabibbo and Riccardo Torlone. Querying multidimensional databases. In *Database Programming Languages*, pages 319–335. Springer, 1998.

[DA99] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC 99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.

[DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166, December 2001.

[Dey01] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[DKDA07] Li Ding, Pranam Kolari, Zhongli Ding, and Sasikanth Avancha. Using ontologies in the semantic web: A survey. In Raj Sharman, Rajiv Kishore, and Ram Ramesh, editors, *Ontologies*, volume 14 of *Integrated Series in Information Systems*, pages 79–113. Springer US, 2007. 10.1007/978-0-387-37022-4_4.

[DSP09] Marina Drosou, Kostas Stefanidis, and Evaggelia Pitoura. Preference-aware publish/subscribe delivery with diversity. *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, page 1, 2009.

[EGCB09] Guillaume Erétéo, Fabien Gandon, Olivier Corby, and Michel Buffa. Semantic Social Network Analysis. In *Web Science*, Athènes, Grèce, 2009.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[EPR08] J.É.Ô. Euzenat, J.É.Ô. Pierson, and Fano Ramparany. Dynamic context management for pervasive applications. *The Knowledge Engineering Review*, 23(1):21–49, 2008.

[FC04] Patrick Fahy and Siobhan Clarke. CASS - Middleware for Mobile Context-Aware Applications. *Workshop on Context Awareness, MobiSys*, 2004.

[Gha02] Rayid Ghani. Building recommender systems using a knowledge base of product semantics. *Hypermedia and Adaptive Web based Systems*, 2002.

[GMN08] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. A framework for recommending olap queries. In *Proceeding of the ACM 11th international workshop on Data warehousing and OLAP*, DOLAP '08, pages 73–80, New York, NY, USA, 2008. ACM.

[GMNS09] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for olap discovery driven analysis. In *Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pages 81–88, New York, NY, USA, 2009. ACM.

[Gor03] Narasimhaiah Gorla. Features to consider in a data warehousing system. *Communications of the ACM*, 46(11):111–115, November 2003.

[GPZ05] Tao Gu, Hung Keng Pung, and Daqing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.

[GR09] Matteo Golfarelli and Stefano Rizzi. Expressing OLAP Preferences. *Scientific and Statistical Database Management*, 5566/2009:83–91, 2009.

[GRB11] Matteo Golfarelli, Stefano Rizzi, and Paolo Biondi. myolap: An approach to express and evaluate olap preferences. *IEEE Transactions on Knowledge and Data Engineering*, 23:1050–1064, 2011.

[GS01] Tom Gross and Marcus Specht. Awareness in context-aware information systems. In *Mensch & Computer*, volume 1, pages 173–182, 2001.

[GWPZ04] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *In Proceedings of Communication Networks and Distributed Systems, Modeling and Simulation conference*, pages 270–275, 2004.

[Hec05a] Dominik Heckmann. Distributed user modeling for situated interaction. In Armin B. Cremers, Rainer Manthey, Peter Martini, and Volker Steinhage, editors, *GI Jahrestagung (1)*, volume 67 of *LNI*, pages 266–270. GI, 2005.

[Hec05b] Dominik Heckmann. Situation modeling and smart context retrieval with semantic web technology and conflict resolution. In Roth-Berghofer et al. [RBSL06], pages 34–47.

[HI04] Karen Henricksen and Jadwiga Indulska. Modelling and using imperfect context information. pages 33–37, 2004.

[HIMB05] Karen Henricksen, Jadwiga Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 846–863, 2005.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing*, Pervasive '02, pages 167–180, London, UK, 2002. Springer-Verlag.

[HKTR04] Jonathan L. Herlocker, Joseph a. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, January 2004.

[HS05] John Horner and Il-Yeol Song. A taxonomy of inaccurate summaries and their management in olap systems. In Lois M. L. Delcambre, Christian Kop, Heinrich C. Mayr, John Mylopoulos, and Oscar Pastor, editors, *ER*, volume 3716 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2005.

[Hul99] David A. Hull. Xerox trec-8 question answering track report. In *TREC*, 1999.

[IS03] Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers*

2003 - Volume 21, ACSW Frontiers '03, pages 143–151, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[JE09] Mohsen Jamali and Martin Ester. *TrustWalker*: a random walk model for combining trust-based and item-based recommendation. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *KDD*, pages 397–406. ACM, 2009.

[JM06] Reed Jacobson and Stacia Misner. *Microsoft SQL Server(TM) 2005 Analysis Services Step by Step*. Microsoft Press, Redmond, WA, USA, 2006.

[JRTZ08] Houssem Jerbi, Franck Ravat, Olivier Teste, and Gilles Zurfluh. Management of context-aware preferences in multidimensional databases. *2008 Third International Conference on Digital Information Management*, pages 669–675, November 2008.

[JRTZ09a] Houssem Jerbi, Franck Ravat, Olivier Teste, and Gilles Zurfluh. Applying recommendation technology in OLAP systems. *Enterprise Information Systems*, pages 220–233, 2009.

[JRTZ09b] Houssem Jerbi, Franck Ravat, Olivier Teste, and Gilles Zurfluh. Preference-based recommendations for olap analysis. In Torben Pedersen, Mukesh Mohania, and A Tjoa, editors, *Data Warehousing and Knowledge Discovery*, volume 5691 of *Lecture Notes in Computer Science*, pages 467–478. Springer Berlin / Heidelberg, 2009.

[KBA11] Nicolas Kuchmann-Beauger and Marie-Aude Aufaure. A natural language interface for data warehouse question answering. In *Proceedings to the 16th International Conference on Applications of Natural Language to Information Systems*, 2011. to be published.

[KK07] Sungrim Kim and Joonhee Kwon. Effective Context-aware Recommendation on the Semantic Web. *Journal of Computer Science and Network Security*, 7(8):154–159, 2007.

[KN10] Natalija Kozmina and Laila Niedrite. Olap personalization with user-describing profiles. In Peter Forbrig, Horst Gnther, Will Aalst, John Mylopoulos, Norman M. Sadeh, Michael J. Shaw, and Clemens Szyperski, editors, *Perspectives in Business Informatics Research*, volume 64 of *Lecture Notes in*

Business Information Processing, pages 188–202. Springer Berlin Heidelberg, 2010.

[Kob07] Alfred Kobsa. Generic user modeling systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2007.

[KPC06] Anders Kofod-Petersen and Jörg Cassens. Using activity theory to model context awareness. In Thomas Roth-Berghofer, Stefan Schulz, and David Leake, editors, *Modeling and Retrieval of Context*, volume 3946 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg, 2006. 10.1007/11740674_1.

[KPVOGM05] Manuele Kirsch-Pinheiro, Marlène Villanova-Oliver, Jérôme Gensel, and Hervé Martin. Context-aware filtering for collaborative web systems: adapting the awareness information to the user's context. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *SAC*, pages 1668–1673. ACM, 2005.

[Lai07] Juhani Laitakari. *Dynamic context monitoring for adaptive and context-aware applications*. PhD thesis, 2007.

[LCCH10] C.-H. Liu, K.-L. Chang, Jason J.-Y. Chen, and S.-C. Hung. Ontology-based context representation and reasoning using owl and swrl. In *CNSR*, pages 215–220. IEEE Computer Society, 2010.

[LD06] Antonis Loizou and Srinandan Dasmahapatra. Recommender systems for the semantic web. In *ECAI 2006 Recommender Systems Workshop*, 2006.

[MA04] Paolo Massa and Paolo Avesani. Trust-aware collaborative filtering for recommender systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, pages 492–508. Springer Berlin / Heidelberg, 2004.

[MB04] Paolo Massa and Bobby Bhattacharjee. Using trust in recommender systems: An experimental analysis. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Trust Management*, volume 2995 of *Lecture Notes in Computer Science*, pages 221–235. Springer Berlin / Heidelberg, 2004.

[MLT09] Jose-Norberto Mazón, Jens Lechtenbörger, and Juan Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12):1452–1469, 2009.

[MMN02] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *in Eighteenth National Conference on Artificial Intelligence*, pages 187–192, 2002.

[MN11] Patrick Marcel and Elsa Negre. A survey of query recommendation techniques for datawarehouse exploration. In *Proceedings of 7th Conference on Data Warehousing and On-Line Analysis (Entrepts de Donnes et Analyse), EDA'11*, 2011.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, July 2008.

[NNT01] Tapio Niemi, Jyrki Nummenmaa, and Peter Thanisch. Constructing OLAP cubes based on queries. In *Proceedings of the 4th ACM international workshop on Data warehousing and OLAP*, pages 9–15, New York, New York, USA, 2001. ACM.

[OA97] P. Öztürk and Agnar Aamodt. Towards a model of context for case-based diagnostic problem solving. In *Context-97; Proceedings of the interdisciplinary conference on modeling and using context*, pages 198–208. Citeseer, 1997.

[OS05] John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, pages 167–174, New York, NY, USA, 2005. ACM.

[PBMW97] Michael Pazzani, Daniel Billsus, S. Michalski, and Janusz Wnek. Learning and revising user profiles: The identification of interesting web sites. In *Machine Learning*, pages 313–331, 1997.

[PdCDL08] E Peis, JMM del Castillo, and JA Delgado-Lopez. Semantic recommender systems. analysis of the state of the topic. *Hipertext*, pages 1–9, 2008.

[PHG00] David M. Pennock, Eric Horvitz, and C. Lee Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings*

of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 729–734. AAAI Press, 2000.

[PJ01] Torben Bach Pedersen and Christian S. Jensen. Multidimensional database technology. *Computer*, 34:40–46, 2001.

[PMT08] Jesus Pardillo, Jose-Norberto Mazón, and Juan Trujillo. Bridging the semantic gap in OLAP models: platform-independent queries. In *Proceeding of the ACM 11th international workshop on Data warehousing and OLAP*, pages 89–96. ACM, 2008.

[RA09] Oscar Romero and Alberto Abelló. A survey of multidimensional modeling methodologies. *IJDWM*, 5(2):1–23, 2009.

[RBSL06] Thomas Roth-Berghofer, Stefan Schulz, and David B. Leake, editors. *Modeling and Retrieval of Context, Second International Workshop, MRC 2005, Edinburgh, UK, July 31 - August 1, 2005, Revised Selected Papers*, volume 3946 of *Lecture Notes in Computer Science*. Springer, 2006.

[RCARM09] Oscar Romero, Diego Calvanese, Alberto Abelló, and Mariano Rodríguez-Muro. Discovering functional dependencies for multidimensional design. In *Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pages 1–8, New York, NY, USA, 2009. ACM.

[Riz11] Stefano Rizzi. New Frontiers in business intelligence: distribution and personalization. In *Advances in Databases and Information Systems*, pages 23–30. Springer, 2011.

[RT09] Franck Ravat and Olivier Teste. Personalization and olap databases. In Stanislaw Kozielski and Robert Wrembel, editors, *New Trends in Data Warehousing and Data Analysis*, volume 3 of *Annals of Information Systems*, pages 1–22. Springer US, 2009. 10.1007/978-0-387-87431-9_4.

[Sal88] Gerald Salton, editor. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven Exploration of OLAP Data Cubes. Technical report, IBM, 1998.

[Sap99] Carsten Sapia. On modeling and predicting query behavior in olap systems. In Stella Gatziu, Manfred A. Jeusfeld, Martin Staudt, and Yannis Vassiliou, editors, *DMDW*, volume 19 of *CEUR Workshop Proceedings*, page 2. CEUR-WS.org, 1999.

[Sar00] Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *Proceedings of the 26th Conference on Very Large DataBases (VLDB), Cairo, Egypt, 2000.*, 2000.

[SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893 – 901, 1999.

[Sch02] Albrecht Schmidt. *Ubiquitous computingcomputing in context*. PhD thesis, 2002.

[Sch05] Sven Schwarz. A context model for personal knowledge management applications. In Roth-Berghofer et al. [RBSL06], pages 18–33.

[Sch06a] Andreas Schmidt. A layered model for user context management with controlled aging and imperfection handling. *Modeling and Retrieval of Context*, pages 86–100, 2006.

[Sch06b] Sven Schwarz. A context model for personal knowledge management applications. In Thomas Roth-Berghofer, Stefan Schulz, and David Leake, editors, *Modeling and Retrieval of Context*, volume 3946 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin / Heidelberg, 2006.

[SDP09] Kostas Stefanidis, Marina Drosou, and E. Pitoura. You May Also Like Results in Relational Databases. *Proc. PersDB, Lyon, France*, 2009.

[SK09] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4:2–4:2, January 2009.

[SKKR00] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender systemsa case study. In *In ACM WebKDD Workshop*, 2000.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[SLP04] Thomas Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*, pages 1–8. Citeseer, 2004.

[SMB07] Ahu Sieg, Bamshad Mobasher, and Robin Burke. Ontological User Profiles as the Context Model in Web Search, 2007.

[SS01] Gayatri Sathe and Sunita Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. In *Proceedings of the 27th VLDB Conference*, Rome, Italy, 2001.

[Tat06] E I Tatli. Context data model for privacy. *Department of Computer Science University of Mannheim*, page 0607, 2006.

[TSM01] Thomas Thalhammer, Michael Schrefl, and Mukesh Mohania. Active data warehouses: complementing olap with analysis rules. *Data amp; Knowledge Engineering*, 39(3):241 – 269, 2001. ¡ce:title¿Data warehousing¡/ce:title¿.

[Vas98] Panos Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 53–62. IEEE, 1998.

[Vas09] Panos Vassiliadis. A survey of ExtracttransformLoad technology. *IJDWM*, 5(3):1–27, 2009.

[WFG92] Roy Want, Veronica Falcao, and Jon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10:91–102, 1992.

[WS07] Wolfgang Woerndl and Johann Schlichter. Introducing context into recommender systems. In *Short Paper, Proc. AAAI 2007 Workshop on Recommender Systems in e-Commerce*, pages 138–140, 2007.

[YL06] S.S. Yau and Junwei Liu. Hierarchical situation modeling and reasoning for pervasive computing. In *Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS*

2006/WCCIA 2006. The Fourth IEEE Workshop on, page 6 pp., april 2006.

[Zie05] Cai-Nicolas Ziegler. Semantic web recommender systems. In Wolfgang Lindner, Marco Mesiti, Can Trker, Yannis Tzitzikas, and Athena Vakali, editors, *Current Trends in Database Technology - EDBT 2004 Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 521–521. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-30192-9_8.

[ZL07] Andreas Zimmermann and Andreas Lorenz. A.: An operational definition of context. *In: CONTEXT*, 2007.

[ZStL04] Cai-nicolas Ziegler, Lars Schmidt-thieme, and Georg Lausen. Exploiting Semantic Product Descriptions for Recommender Systems Categories and Subject Descriptors. In *Proceedings of the ACM SIGIR Semantic Web and Information Retrieval Workshop, SWIR'04*, 2004.

List of Figures

List of Tables