MANAGING WEB SERVICES COMMUNITIES: A CACHE FOR QUERIES OPTIMISATION

Hela Limam[1] and Jalel Akaichi[2]

WDW-SOIE, Institut Supérieur de Gestion de Tunis, 41, Avenue de la Liberté,Cité Bouchoucha, Le Bardo 2000, Tunisia

[1]hela.limam@isg.rnu.tn

[2]jalel.akaichi@isg.rnu.tn

ABSTRACT

With the advance of Web Services technologies and the mergence of Web Services into the information space, tremendous opportunities for empowering users and organizations appear in various application domains including electronic commerce, travel, intelligence information gathering and analysis, health care, digital government, etc. However, the technology to organize, search, integrate these Web Services has not kept pace with the rapid growth of the available information space. The number of Web Services to be integrated may be large and continuously changing.

The ubiquitous need for Web Services integration across heterogeneous information sources pushed Web Services providers to join each others into Web Services Communities.

Different approaches related to the specification, management and querying of a community were proposed. Whoever, current approaches for managing and querying Web Services Communities' lack of a general formal and clear design. Hence, we propose a framewok which enables the community management then allows community members to share reusable fragments of kow-how in order to optimize query processing among communities .To demonstrate the viability of our approach, we propose a health care community infrastructure for supporting a community management and querying as well as a prototype application that utilizes communities.

KEYWORDS

Communities, management, query optimisation

1. INTRODUCTION

With the emergence of Service Oriented Architectures (SOA) in the information space as a model for homogeneous distribution and composition of components. Web services are new to developers and appear to be the way to achieve and develop an SOA [22].

They offer tremendous opportunities for empowering users and organizations in various application domains including electronic commerce, travel, intelligence information gathering and analysis, digital government.

In fact, Ample definitions of Web services exist [1, 2, 3] .All of them agree the Web services are software components providing a specific functionality on the internet [4]. The increasing numbers of Web services available on the Web made information sources become component that we can use, re-use and match to enhance Internet application to be integrated.

However, technology to organize, search, integrate, and evolve these Web services has not kept pace with the rapid growth of the available information space. The prominent need to face the hard competition between enterprises populating the Web, and the ubiquitous need for information integration across heterogeneous Web portals are factors that pushed Web Services providers to join each other's into Web services communities.

DOI : 10.5121/ijwsc.2010.1101

Gathering Web services into communities facilitates the discovery and selection of Web services. Providers develop several Web services that could offer similar functionalities like hotel booking and car rental. Hence the community infrastructure enhances Web services availability and improves the collaboration between communities' members by providing a centralized access across distributed Web services. However, the organization into communities raises new issues going from communities' management to communities querying.

In fact, communities introduce management problem resulting from the lack of a generic tool for community building and the absence of interoperability among different community support platforms. Community members often want to query, monitor and discover information about various entities and relationships not only in their communities but also in other communities.

A critical challenge therefore, is to design a system able to manage communities taking into account many tasks such as discovering and updating communities then building relationships between them. Our solution is to design and to implement a system able to manage Web Services Communities while addressing all of these issues.

Moreover, these communities are built with the purpose to be queried transparently and easily by users, which aim to satisfy their informational needs in a satisfactory time and in a pertinent retrieval.

A user query may involve the access of a number of distributed communities, which may imply the access of a number of members. This multiple access to distributed data sources to retrieve data requested by a user can be expensive and may consume too much time that may generate costs and consequently an important waiting time that can imply an important loss of customers.

Because performance is a crucial issue in query processing, the idea of semantic cache appears very promising and can be applied to solve the above problems and to improve the efficiency of the query processing. The semantic caching system exploits the idea of saving and reusing cached query results to answer new queries based on identified links to involved communities and members. The approach uses semantic information to describe cached data items with the goal of exploiting semantic locality to improve query response time by caching and maintaining query rewriting.

In this work, we first investigate the challenges imposed by the problem of communities' management and present our approach for tackling it. Second, as communities are built with the goal to be queried we have to address the problem of queries resolution among communities. The problem is to formalize queries using description logic, then to choose and to adopt the algorithms for queries rewriting and for queries resolution that satisfy the users queries constraints.

Third, as the query resolution process over many distributed communities is expensive, adopting a mechanism capable of caching previous computed views results for answering future queries would be beneficial for improving the query performance. This can be achieved by using a semantic cache.

Throughout this work, we adopt a Health Care Community as a running example. It is composed of Web Services related to the Health Care domain. We show how to create and to manage it. Then we demonstrate how it is used to process users queries. We apply the cache maintenance process to it in order to update stored queries results.

The remainder of this work is organized as follows. Section 2 reviews the related issues and solutions under the umbrella of Web Services Communities' management, the query resolution and the semantic caching in Web environment. Then in section 3, we present the system requirements for Web Services Communities' management and querying. Section 4 details issues related the system architecture. Section 5 addresses the issue of the Web services communities' management. Furthermore, Section 6 addresses the problem of querying communities. Next, section 7 describes the implementation of our management, querying and semantic cache system. Finally, in section 8 we conclude the work, summarize our results and give the direction of our future works.

2. THE STATE OF THE ART

Two major areas of related research are managing and querying Web Services Communities. In Fact, managing Web Services Communities includes integrating Web Services into Communities then managing communities. Whereas, querying Web Services Communities include essentially the choice of the querying strategy and the query optimization. In the following we draw a survey on the researches covering communities' management and querying.

2.1. Communities management

A community is typically a group of people living together or united by shared interests, cultural, religious or political reasons. When it comes to Web services, communities help gather Web services that provide a common functionality, thus simplifying the access to Web services via a unique communication endpoint, which is the access point to the community.

Several research works propose to use communities for easing the management and access to Web services [1, 2, 3, 4, 5,6].

Benatallah et al. define [1] community as a collection of Web services with a common functionality, although these Web services have distinct non-functional properties like different providers and different QoS parameters [1].

In [5], authors propose an approach that supports the concepts, architecture, operation and deployment of Web service communities. The notion of community serves as an intermediary layer to bind to Web services. A community gathers several slave Web services that provide the same functionality. The community is accessed via a unique master Web service. Users bind to the master Web service that transparently calls a slave in the community. . This work details the management tasks a master Web service is responsible for. Such tasks include among other things registering new Web services into the community, tracking bad Web services, and removing ineffective Web services from the community. A master Web service represents the community and handles users' requests with slave Web services with the help of a specific protocol.

Benatallah et al. propose a solution with SELF-SERV [1] for gathering functionally-similar Web services into communities that are accessed via a common interface. Several mediators establish correspondences between the community interface and Web services that implement the functionality of the community.

Benslimane et al. [2], also group Web service into communities. The community is accessed via an interface implemented as an abstract Web service that describes the provided functionality in an abstract fashion and a set a concrete Web services that implement the functionality. A generic driver called Open Software Connectivity (OSC) handles the interactions between clients and the community.

Medjahed proposes a community-based architecture for semantic Web services in [4]. In this work, communities gather services from the same domain of interest and publish the functionalities offered by Web services as generic operations. Community ontology is used as a general template for describing semantic Web services and communities. A major advantage of

this work is the peer-to-peer community management solution that addresses the problems of centralized approaches.

In the context of eE-catalog communities, WS-CatalogNet [7, 8] offers a set of integrated tools that allow for creating communities, registering catalog members, creating peer relationships between communities, querying individual communities and routing queries among communities. It is a hybrid of peer-to-peer and web services technologies.

A community [1] is a container of e-catalogs of a specific domain that cater for similar customer needs (e.g. e-marketplaces for hardware, vertical portals organized on a special business topic, etc). A community is associated to an ontology that provides a description of a specific domain of interests. E-catalogs can register themselves into a community as members by exporting (all or part of) their descriptions. Moreover, to achieve interoperability across similar domains, communities themselves can be linked together as peers based on inter-ontology relationships.

The general architecture of WS-CatalogNet [7] contains three main components: Community Manager, Member Manager and Cooperative Query Manager .The Community Manager is used to create communities and build peer relationships between communities. The Member Manager supports registering individual e-catalogs into communities. The e-catalogs are either already Web Services, or converted to Web Services through the Member Manager. The cooperative Query Manager processes the query which requires locating e-catalogs capable of answering the query. The latter component allows us to introduce the second research field: Querying Web Services communities which will be detailed in the following section.

2.2. Querying communities

In fact, satisfying users' queries is in the heart of organizing Web services into communities. However processing queries among communities requires selecting appropriate Web services .In this context several approaches were proposed for Web services selection

In [9] authors propose a novel approach for querying and automatically composing Data providing services. The proposed approach largely draws from the experiences and lessons learned in the areas of service composition, ontology, and answering queries over views. First, it introduces a model for the description of Data Providing (DP) services and specification of service-oriented queries. DP services are modeled as RDF views over mediated (domain) ontology. Each RDF view contains concepts and relations from the mediated ontology to capture the semantic relationships between input and output parameters. Second, a query rewriting algorithm is proposed for processing queries over Data Providing services. The query mediator automatically transforms a user's query (during the query rewriting stage) into a composition of DP services.

In Taher et al.'s work [6], Web service selection is performed according to a set of QoS criteria (speed, reliability, reputation, etc.). The community is also in charge of administrative tasks such as addition and suppression of services to and from the community. Web service substitution is also addressed in this work and consists of replacing a non-functioning or non-responding Web service with a functionally equivalent one, in order to find an alternative way to enable a composition in case of exception.

In [3] authors propose a community-based approach for web service selection where super-agents with more capabilities serve as community managers. They maintain communities and build community-based reputation for a service based on the opinions from all community members that have similar interests and judgement criteria. The community-based reputation is useful for consumer agents in selecting satisfactory services when they do not have much personal experience with the services. A practical reward mechanism is also introduced to create incentives for super-agents to contribute their resources and provide truthful community-based reputation information, as strong support for the approach.

In WSCatalogNet [14], authors use the concept of e-catalog communities and peer relationships among them to facilitate the querying of a potentially large number of dynamic e-catalogs. E-Catalogs communities are essentially containers of related e-catalogs. A flexible query matching

algorithm that exploits both community descriptions and peer relationships is used to find e-catalogs that best match a user query. The user query is formulated using a description of a given community.
A query matching algorithm is developed and adopted by the authors [12]; Best Query Rewriting (BQR) [13]. This algorithm identifies which part of the query can be answered by local members of the community and which part of the query cannot (hence, needs help of peers).
Because of the variety of e-catalogs offering similar information and the large number of available e-catalogs, it is important to provide appropriate support to first select those e-catalogs that are relevant to a given query. In WS-CatalogNet [14], a query is posed over community ontology. Relevant e-catalog selection is formalized as a new instance of the query rewriting problem, where a query Q over a community C is reformulated in terms of Q local: local queries (i.e., the part of Q that can be answered by e-catalogs registered with C), Q rest: remaining parts of Q that cannot be answered by C. Q rest part will be forwarded to the other communities known to C via the peer relationships. The expected answer from forwarding is the identification of external (i.e., registered to other community) e-catalogs who can resolve the query. Once the relevant e-catalogs are identified, the community sends the query to the e-catalogs and assembles the results. The system provides a formalization of query rewriting in the context of category hierarchy based ontologies and propose a hypergraph-based algorithm to effectively compute best rewritings of a given request.
What set us apart from these approaches and what makes the originality and the main contribution of our work is that we propose a general design of Web services communities and we propose an optimisation for query processing by adding a cache mechanism.

2.3. Queries Optimization

In Web services communities systems that access distributed Web services providers, an efficient query processing requires an advanced caching mechanism to reduce the query response time.
Data caching in general and semantic caching in particular have been intensively studied in the context of data and information retrieval systems. These works comprise classical client-server databases [17, 18], heterogeneous multi-database environments [15], as well as Web databases.
The idea of semantic regions was introduced by Dar et al. [17]. Semantic regions are defined dynamically based on the processed queries which are restricted on selection conditions on single relations. Constraint formulas describing the regions are conjunctive representations of the used selection predicates. Thereby, the regions are disjoint.
The semantic query cache (SQC) approach for mediator systems over structured sources is presented in [15]. The authors discuss most aspects of semantic caching: determining when answers are in cache, finding answers in cache, semantic overlapping and semantic independence and semantic remainder in a theoretical manner. Our approach also reflects most addressed aspects, but is appropriate to web services communities.
Keller and Basu describe in [18] an approach for semantic caching that examines and maintains the contents of the cache. Therefore, the predicate descriptions of executed queries are stored on the client as well as on the server. Queries can include selections, projections and joins over one or more relations but results have to comprise all keys that have been referenced in the query.
Semantic caching in Web mediator systems is proposed in [16]. The authors also introduce a technique that allows the generation of cache hits by using of additional semantic background information. In contrast to our approach, the cache is not located in the mediator access component but in the wrappers.

3. THE SYSTEM REQUIREMENTS

In the area of research related to the design of Web services communities, current approaches lack of a clear design for communities' management which may alter the system evolving and maintenance. Nevertheless, the number of autonomous and heterogeneous Web Services to be integrated may grow larger and continuously changing. Consequently, effectively integrating and querying them is a delicate and time-consuming task, which introduces two main problems: A design problem resulting from the lack of a generic tool for community management and a query optimization problem since we are facing more and more exigent consumers.
A critical challenge therefore, is to design a system able to manage Communities while addressing all the management issues going from the communities building to the communities update to the query processing among communities. All of the above problems constitute the system requirements and are detailed in the following.

3.1 Communities management

The main requirements related to communities' management can be resumed by creating and updating Web services communities then building relationships between them. These requirements are detailed in the following, and illustrated through an UML use case diagram in figure 1.
Community Creation: The Community manager creates a community by grouping Web Services related to the same domain then he defines its schema to provide a description of the field to which the community belongs without referring the Web Services providers.
Community Update: As communities evolve in the Web environment characterized by its dynamism, changes can frequently affect communities. Hence, communities should be permanently updated. The community update takes in general two forms, deletion or modification.

- Community deletion: The community that does not contain any Web Service is deleted, for this purpose the Community Manager has to identify a community that users constantly leave without performing any further action. The Deletion is manual or automatic.
- Community modification: It consists on adding or updating members and consequently adding or updating Web Services.

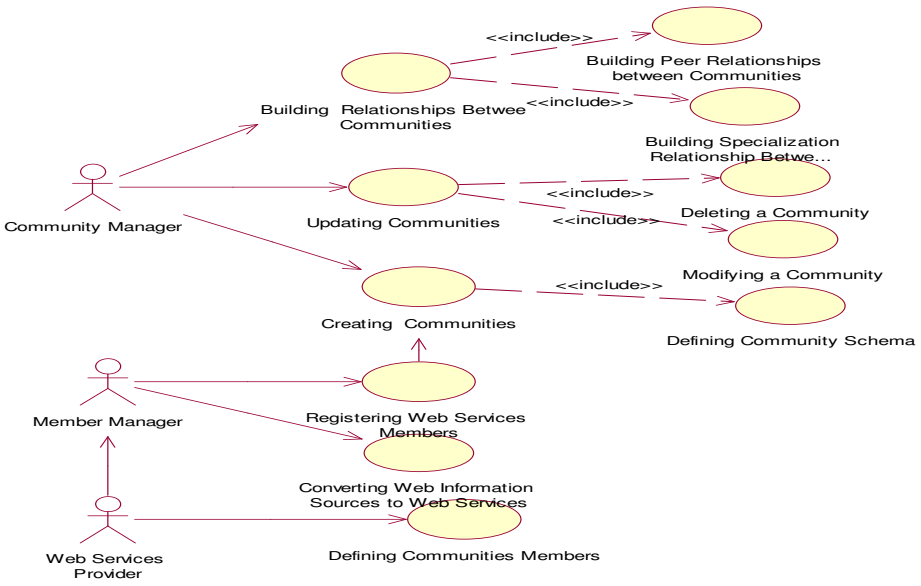

Figure 1. UML use case diagram of communities' management.

3.2 Querying Web services communities

The main requirements of the query processing among communities are: the identification of relevant communities that contribute in the generation of the answer, the collection of answers from communities and the delivering of answers to users .The final and the important step is the optimization of the query processing through the use of a cache in order to store collected results for a future reuse. In the following, we state the different steps of the query processing:

- Formulating a query: The users may view and access information about the community in general, but they are not able to navigate through the system's functionality. Their main role is to request a service.
- Processing the query between communities: The Query Solver migrates between communities in order to process the query formulated by the user according to the following steps:
- Searching into the cache: The Query Solver consults the cache in order to find a part of or the whole query answer. If the answer doesn't exist inside cached queries the latter identifies concerned communities.
- Identifying concerned communities: The Query Solver identifies the combination of members whose query capabilities, when put together, satisfy all constraints expressed in the query. The members can be local (belonging to the community), or external (belonging to the community peers).
- Rewriting the query for concerned communities: The Query Solver divides the query into sub-queries to the identified communities.
- Routing queries among communities: The sub-queries are sent to the identified communities.

- Identifying concerned members: The Query Solver has to identify communities' members concerned by the query. Then it rewrites the query for concerned members. Finally, it delivers the results to the user.
- Collecting results: The Queries Collector collects the queries answers in order to reuse these results for similar requests and to store the links towards communities requested.
- Updating results: The Cache Maintainer updates the stored queries results acc0rding to changes that occur in communities.

The system requirements related to querying Web Services Communities introduced above are illustrated through a UML Use Case diagram in figure 2.

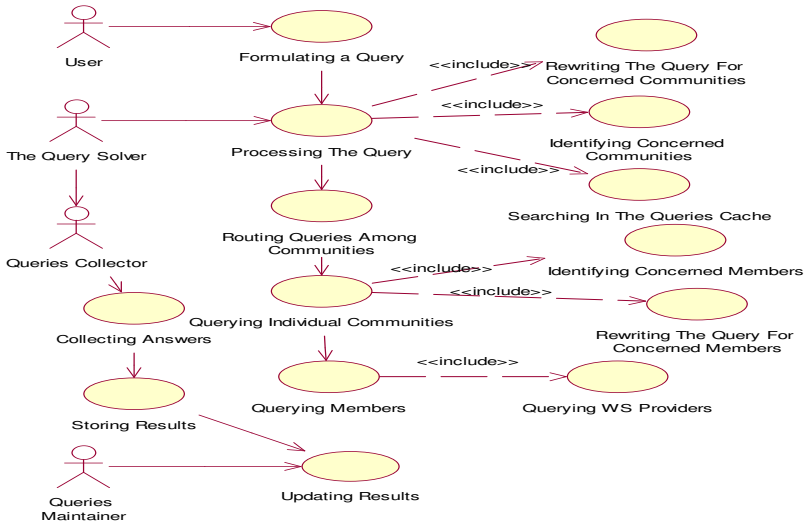

Figure 2. UML use case diagram of query processing

4. THE SYSTEM ARCHITECTURE

In order to meet the system requirements described above, we describe in these part functionalities and architectural issues in the design of a Web Services Communities system. Our system can be seen as a mediator used to process users' queries and to identify relevant knowledge. When the query is received, the mediator analyzes the query, locates relevant sources, and presents the answer that is merged, assembled or redefined. Furthermore, it may store fully or in part, the result of queries.

The mediator is in charge of community creation, community update, community members' participation and the research of the best provider for a requested service. The mediator architecture includes five main components as shown in figure 3. The Community Knowledge Base (CKB) responsible of the communities' management, The User Queries Knowledge Base (UQKB) where user queries are stored and a Query Solver (QS) which main role is to extract and route sub queries destined to specific communities. The Cache saves the queries results for an immediate and future use of these queries. In addition, the cache maintainer (CM) is responsible for the coherence and the availability of the cache.

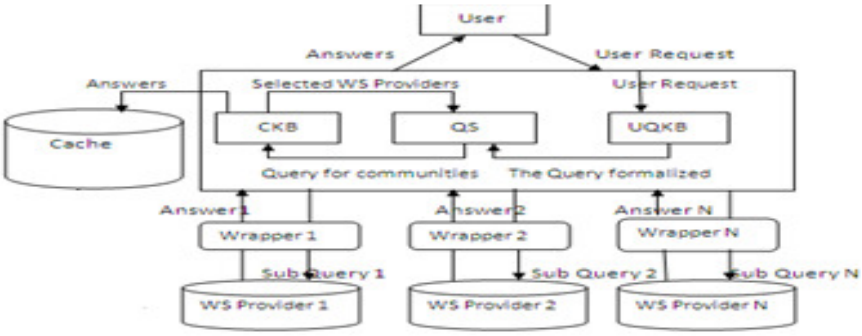

Figure 3. The system architecture

Web Services providers join the mediator via Wrappers which convert them to Web Services, then Web Services are integrated into communities. Communities are managed in the mediator thanks to the Community Knowledge Base (CKB). The CKB is used in order to provide context for queries. It represents all the entities that exist on the system, these entities are objects that become source of knowledge for the users of the Mediator .The CKB support the community creation and management, and the building of relationships between communities. It contains descriptions of communities using the structure (CId, CName, WSCList), where Cid describes the community identifier, CName represents the community name, and WSCList symbolizes the Web Services list from which communities are created according to the community schema generated by the CKB. Moreover, we include into the CKB peer constraints and specialization relationships between communities.
The goal of our mediator is to satisfy user's queries. The achievement of this goal while using an optimized manner requires the following components:

- The User Queries Knowledge Base (UQKB): It offers a space to a user in which he is able to retrieve information stored in the system, communicate or interact. The UQKB performs the task of the description and the formalization of the query.
- The Query Solver (QS) :The main functionality assumed by the QS consists on processing the query ad answering it which requires locating Web Services capable of giving an answer to the query by routing the latter among communities then querying individual community .The result of this step is finding the combination of members satisfying the query constraints
- The Semantic Cache (SC): The SC is indirectly accessed, through the query. Links between communities are stored in the cache; however, they are stored in a format very flexible with regard to changes and amendments of the community schema.
- The Cache Maintainer (CM): The CM is in charge of managing the semantic cache space when changes occur in communities by replacing them by their peers

5. WEB SERVICES COMMUNITIES MANAGEMENT

5.1 Communities Creation

A community is a container of Web Services of a specific domain that cater for similar customers needs (e.g. Health Care Community). A community is associated to an ontology (community schema) that provides description of a specific domain of interests. The community schema is described in terms of categories for the domain it represents. A category is described

by a set of attributes. Categories within ontology may be inter-related via the specialization relationship.

The communities' creation process starts when a community Administrator asks for creating a community and specifies its name. Hence, the Community Manager prepares a community structure of the schema inside which Web Services will be integrated progressively. For example, to create a Health Care Community, the Community Manager specifies that sub communities are Examination, Medical institutions, Patients, Pathologies and Doctors. Thereafter, the Community Manager feeds the empty structure with Web Services by locating related Web Services and specifying its category and for each category, a set of attributes. For example, the community Hospitals may have a category Medical Institutions, which is described using attributes such as Id Institution, Name, and Localization (see figure 4).

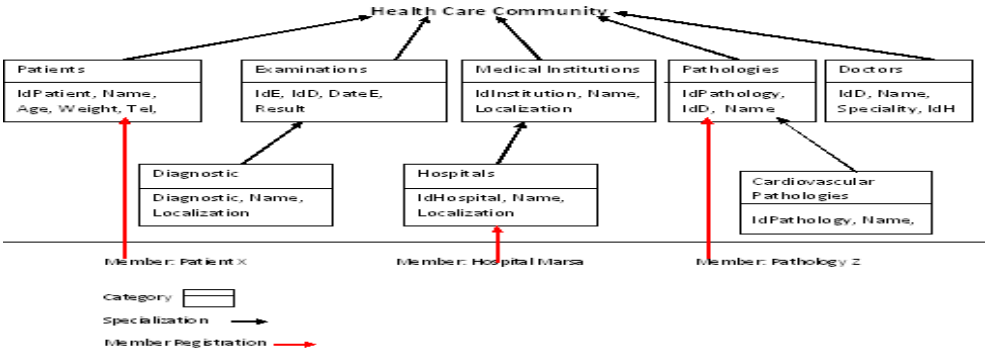

Figure 4. Health Care community, its categories and members

When the community structure is well defined, Web services register themselves in communities following a Local As View (LAV) [19] approach for integrating a community schema with the descriptions of its members as shown in figure 5 . The descriptions of community members are expressed in terms of the community schema; thereby our system follows LAV approach as it allows integrating a potentially large number of Web Services. Clearly, a Global As View integration approach [19], where the development of a community schema requires the understanding of both structure and semantics of all descriptions of community members, is hardly applicable in Web Services environments because of the dynamic nature and size of the Web. Furthermore, the addition of a new member to the community requires only the provision of the exported description of the member. It does not require any changes to the community schema.

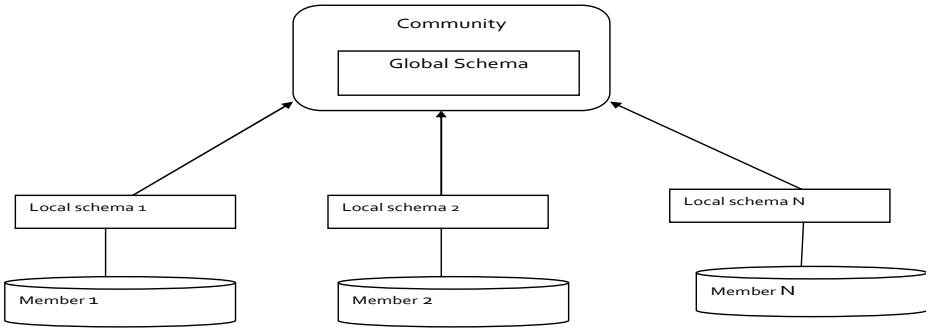

Figure 5. The integration approach

To be accessible through a community, the members register themselves in communities. When registering, they specify the categories and attributes of the community schema they support. This form of information is the members' definition. The members' subscription process is illustrated through the UML Class Diagram in figure 6.

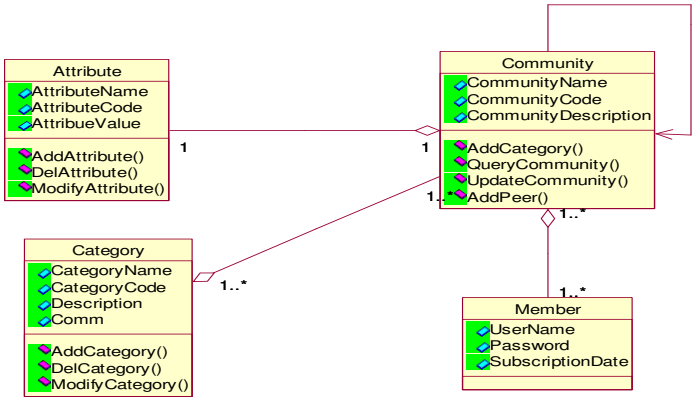

Figure 6. UML class diagram of members' subscription

In fact, the member subscription takes in general two forms depending on the Web providers. If Web Information sources are Web Services, they directly apply for subscription in communities. They indicate through the Member Manager which category in the community schema they belong to, then for each category chosen they specify the attributes they support as shown in figure 7. Else, they are converted to web services in order to be accessible through a community as given in figure 8.

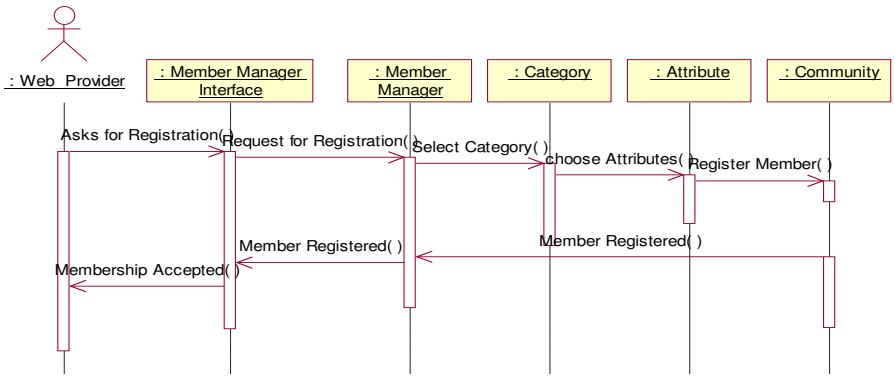

Figure 7. Scenario 1, member's subscription

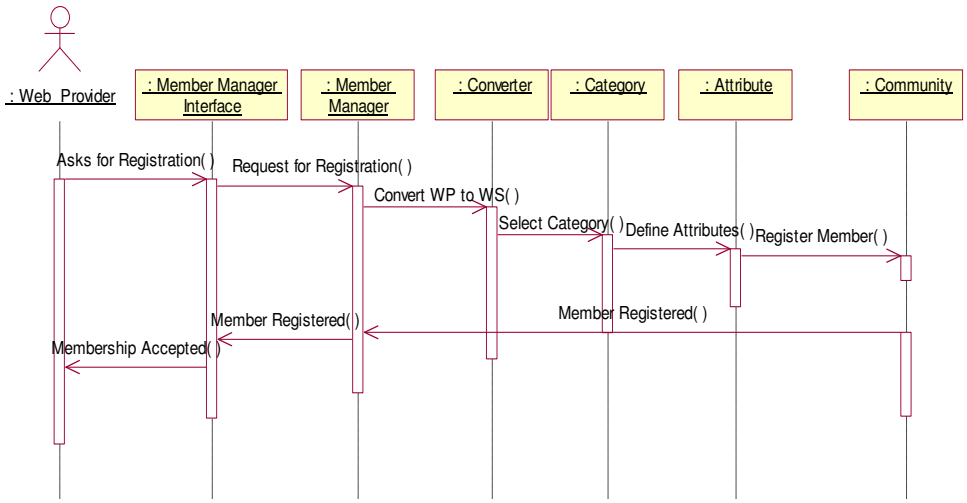

Figure 8. Scenario 2, members' subscription

We detail in the following a community creation process:

Web Information Sources and their included relations are described through an identifier and a set of attributes as given in table 1 . By a subscription process, the Web Information Source becomes a member of the community .A community therefore is a container of Web Services of the same domain (e.g. Health care community). The community is defined through descriptive attributes. Community (name, C-id, attributesList) as given in table 4.
In the following, we give an example of Health Care application. Each Web information source has its own schema and content.

IS	Description
S1	Patient(IdP, Name, Age, Tel, IdH) Hospital(IdH, Name, Localization) Pathology(IdPathology, IdD, Name)
S2	Patient(IdP, Name, Age, Tel, IdH, Med_Resp) Pathology(IdPathology, IdD, Name) Examination(IdE, IdD, DateE, Result)
S3	Patient(IdP, Name, Age, Tel) Doctor(IdD, Name, Speciality, Hospital, IdS) Hospital(IdH, Name, Localization) Examination(IdE, IdD, DateE, Result)

Table 1. Web information sources schemas

(ws1, {s1, s2})	ws1 is built from s1 and s2
(ws2, {s1, s2})	ws2 is built from s1 and s2
(ws3, {s3})	Ws3 is built from s3

Table 2. Relation between Web services and Web information sources

(C1, {ws1, ws2})	C1 is built from ws1 and ws2
(C2, {ws1, ws2, ws3})	C2 is built from ws1, ws2 and ws3
(C3, {ws1, ws3})	C3 is built from ws1 and ws3
(C4, {ws2, ws3})	C4 is built from ws2 and ws3

Table 3. Relation between Web services and communities

C1	Patient(Idp, Name, Age, Tel, IdH, IdD)
C2	Pathology(IdPathology, IdD, Name)
C3	Hospital(IdH, Name, Localization)
C4	Examination(IdE, IdD, DateE, Result)
C5	Doctor(IdD, Name, Specialty, IdH)

Table 4. Health Care community schema

5.2. Communities update

As Web Services Communities evolve in a dynamic environment, changes can occur and affect as well as their schemas as their contents. For these reasons an update process in to be applied permanently. In fact, the update takes in general two main forms: deletion or modification.

The Community Manager deletes the community, which does not contain any Web Service. As it cannot satisfy requests, users constantly leave it without performing any further action. When the Community Administrator asks for community deletion, the Community Manager counts the Web Services in the community. If the latter does not contain any Web Service, the community manager has to delete it. Deleting a community is removing its category and its associated attributes from the CKB as illustrated in figure 9.

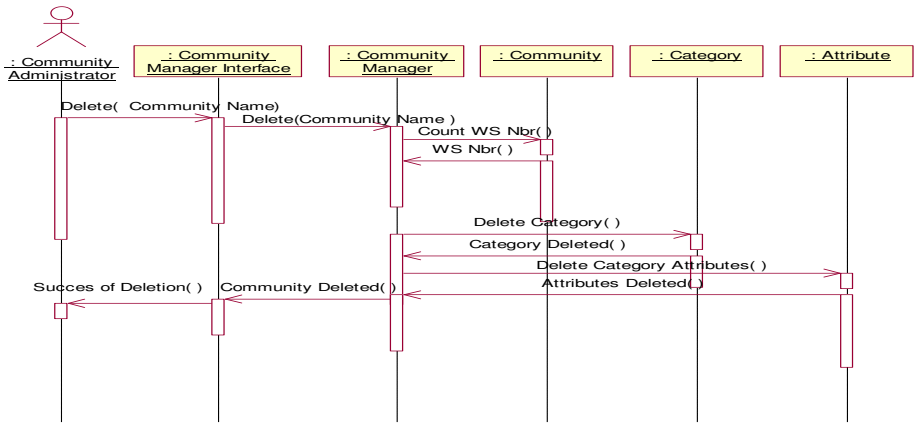

Figure 9. UML sequence diagram of community deletion

Community modification consists on adding or updating members or adding new Web Services

5.3. Relationships between communities

Communities are built with the goal of processing queries which requires establishing relationships between them in order to cooperate to satisfy a user request. However, relationships between communities fall into two types:

- SubCommunityOf relationship: represent specialization between two communities' domains (for example, Hospital is a subcommunity of Medical Institutions). We assume each community has at least one supercommunity but can have no subcommunity.
- PeerCommunityOf: When a user cannot find (or is not satisfied with) information from a community, she or he can refer to other communities that the community considers as its peers (for example, Patient is a peer community of Examination). Communities can also forward queries to each other via a PeerCommunityOf relationship. In fact, Peer relationship is based on the similarity between categories or attributes of different communities. We present in the following a peer relationship building based on the similarity between attributes of different communities.

Peer relationship constraint between two communities C1 and C2 states that communities C1 and C2 can be joined by a peer relationship. Peer constraint between communities is based on similarity between attributes related to different communities and can be defined as follow:

PC	**Peer constraints**
PC1	C1.Patient.IdH=C3.Hospital.IdH
PC2	C1.Patient.IdD=C5.Doctorl.IdD
PC3	C1.Patology.IdD= C5.Doctorl.IdD
PC4	C1.Examination.IdD= C5.Doctorl.IdD
PC5	C1.Hospital.IdH=C5.Doctorl.IdH

Table 5. Peer Constraints

6. QUERYING WEB SERVICES COMMUNITIES

Accessing information sources to retrieve data requested by a user is one of the main goals of building Web Services Communities. However, due to the distributed and dynamic environment in which communities evolve, answering a query could be an expansive process.

Because performance is a crucial issue in Web Services Communities' querying the semantic caching is often beneficial and can improve system performance and scalability. The idea consists on caching the links towards communities that contributed to the generation of the query answer.

The approach uses semantic information to describe cached data items with the goal of exploiting semantic locality to improve query response time. The support of a semantic cache implies several issues concerning the management of the semantic cache space. The first issue is about how the cached query answers are organized. The logical query descriptors of the corresponding data content are the enabling mechanism for the management of a semantic cache.

The idea of applying the semantic caching approach to Web Services Communities for maintaining a cache of queries answers appears very promising since it can dramatically reduce demand on the network as well as latency seen by the user.

In the following part, we detail the cooperative query processing among communities then we introduce the use of the semantic cache for queries optimization. Hence, functionalities of the UQKB, the QS and the semantic cache are to be dealt in this chapter.

In fact, a semantic caching system needs to provide support for the integration of two essential parts: A fundamental technique for tackling the query containment and rewriting problem, which will be detailed in this chapter.

6.1. The Query processing between communities

6.1.1. The query expression

To provide formal expression, necessary for precise characterization of queries over Web Services Communities, we propose to use the description logic [20] designed to represent ontology descriptions in terms of classes (unary predicates) and attributes (binary predicates). Description logics are a family of logics that were developed for modeling complex hierarchical structures and to provide a specialized reasoning engine to do inferences on these structures.
Class descriptions are denoted by expressions formed by means of the following constructors:

- The class conjunction (Π), e.g., the description Patient accommodations Medical Institutions denotes the class of Medical Institutions which are instances of the classes Patient accommodations (e.g., a Clinic).
- The universal attribute quantification (∀R.C), e.g., the description ∀ PatientWeight.Float denotes that the data type of the attribute PatientWeight is Float.
- The existential attribute quantification (∃R), e.g., the description ∃PatientAge denotes the class of Patients having at least one value for the attribute PatientAge.

We consider a simplified example. Assume a query:
Select Patients from Health Care Community where Age=20, Weight=70 and Pathology Name = {Valve Disease}

The query expression in the description logic:

Q≡∃PatientAge Π ∀PatientAge.Float Π ∃PatientWeight Π ∀PatientWeight.Float Π ∃ Pathology∀Pathololgy.String

6.1.2. The query processing technique

Processing the query requires locating communities' members that are capable of answering the query. These members could be local members of the community or the members of the peers.
The cooperative query processing technique consists of two steps:

- Identify combination of members whose query capabilities, when put together, satisfy as much as possible the constraints expressed in the query.
- Answer the query by sending it to the selected combination of members.

The first step uses a query rewriting algorithm, (BQR) [21], which allows identifying the part of the query can be answered by local members of the community and the part of the query that cannot by be answered by local members and hence can be forwarded to peer communities. The algorithm takes as input the community definition, members' definitions and the query (all in their class descriptions format) and produces the following outputs:

- Qlocal: the part of the query Q that can be answered by the community's local members. It also gives the combination of the local members that can answer all (or part of) the query. For each selected local member, we compute the part of the query to be sent to the member.
- Qrest: the part of the query that cannot be answered by the local members. This part of the query will be forwarded to peers. It should be noted that the expected answers of the forwarding is the combination of the external members that are capable of answering the part of the query.

6.1.3 Description of the query processing

The following scenario of the figure 11 is used to describe the query processing between communities:

- The user exploits a community to express queries; he submits the query to the current community. SubmitQuery (userId, QueryQId) where userId is the user identifier, QueryQ could be a global query, which uses the community attributes, or a source query, which concerns one community member.
- The Query Solver takes as input the UserId, the QueryQ and the community definition C (userId, QueryQId, C).It identifies Qlocal (userId, Qlocal, C). The part of the query that can be answered by the community's local members and rewrites Qrest the others parts that cannot be answered by the local members (userId, Qrest, C). The community will collaborate with peers to identify any external members who can answer this part of the query.
- The Qrest is forwarded to peers (userId, QtosubC) if Q is a global query, else it is forwarded to the identified community members (CM) if Q is a source query (userId,QtosubCM).
- Answers are collected by sub community , community then sent to the User. The Query Collector sends the UserId, The query and the answer (userId, QueryQ, Qanswer) to the Queries Cache where queries are stored with their answers for a future reuse.

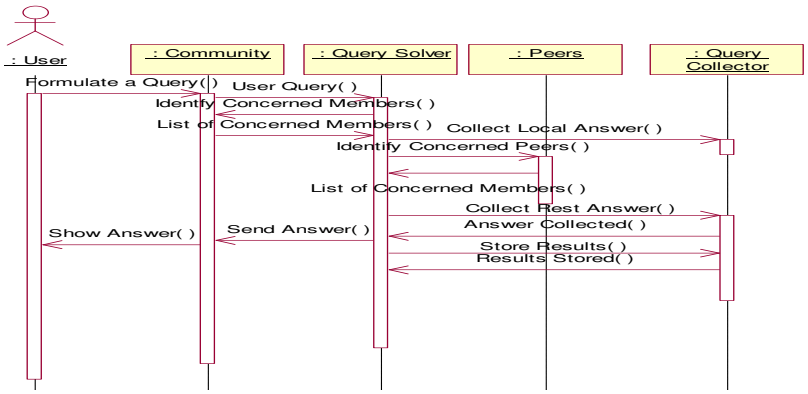

Figure 10. UML Sequence Diagram of query processing

To illustrate our querying process we consider the query:
Q≡∃PatientAge Π ∀PatientAge.Float Π ∃PatientWeight Π ∀PatientWeight.Float Π ∃ Pathology∀Pathololgy.String

We can see that the community Patient provides an answer for the part of the query ∃PatientAge Π ∀PatientAge.Float Π ∃PatientWeight Π ∀PatientWeight.Float but no member definition provides an Answer for the part Pathology∀Pathololgy.String. Hence, The Query Solver forwards it to the Pathology Community, which is a peer community to the Patient Community that can provide the missing information.

6.2. The query optimization

As shown above, our system accesses distributed Web Services Communities in order to satisfy a user query .However an efficient query processing requires an advanced semantic caching mechanism to reduce the query response time.
The idea of semantic caching [31] uses semantic information to describe cached data items with the goal of exploiting semantic locality to improve query response time. That is, semantic caching suggests to cache users' queries and correspondingly to organize the answers (instead of pages or tuples) by their query descriptions. Based on these query descriptions, a semantic caching system determines whether a given input query Q is logically contained or overlapping with a cached query V.
When a query is posed at a community with a semantic cache, the query is split into two pieces:

- A probe query, which retrieves the portion of the answer available in the local cache
- A remainder query, which retrieves the missing portion of answer from communities. If the remainder query is not null (i.e., the query asks for answer that is not cached), the remainder query is sent to the others communities and processed there.

If the remainder query is empty, the probe query serves the answer from the cache content and communities are not contacted. If the remainder query is not empty, its evaluation proceeds in a regular way. The answer to the user query is built up from the answers to the probe and remainder queries: Q=Prob (Q) ∪ Rem (Q).
For example, if the cache contains the region R=``Patient AND Pathology", and query is Q=``Patient", the probe query coincides with R: Prob (Q) = ``Patient AND Pathology'', while the remainder query is Rem (Q) =``Patient AND NOT Pathology".
Once the cache detects some regions relevant to the query and constructs the probe query Probe (Q), the formula Q AND NOT Probe (Q) defines the remainder query Rem (Q). However, the following containment holds: Rem (Q)⊃ AND NOT Prob (Q).
In what follows we consider three operational cases processed by the semantic cache; each case refers to a particular relationship between a user query Q and regions in the cache. Below we list and describe all the cases. To process these cases the semantic cache takes in input cache with semantic regions and query Q and produces as output answer to Q and updated cache. It has to verify the query Q against all region descriptors in the cache:

- Equivalence: The cache contains a region R which formula is equivalent to the query formula: Q≡ R.
- Query containment: The cache contains one or more regions, which formulas R1, R2, Rm≥ 1contains the query formula: Q ⊂ R.
- Region containment: The cache contains regions R1, R2, Rm≥1 which formulas are contained in the query formula: Ri ⊂ Q.

7. IMPLEMENTATION

To evaluate our approach, we propose a prototype that is a Web Service based environment for building Web services communities. The prototype uses XML for request and response specification between users and Web services communities, JDK 1.4 for operation performance, and Eclipse 3.1 as an integrated development environment.
The cache is designed to store result data, which is received as XML documents, and the corresponding queries, which are the semantic description of those results. If a new query arrives it has to be matched against the cached queries and possibly a (partial) result has to be extracted from the cache's physical storage .
In order to realize the assumed behavior a simple and effective way of storing XML data persistently and fail-safe is needed. One way of storing is the usage of a native XML database solution which is Apache's XINDICE in this work. The open source database XINDICE stores

XML documents into containers called collections. These collections are organized hierarchically, comparable to the organization of folders in a file system. The cache is placed below the community schema level, which means the cached entries are collected regarding to the queried concepts. All entries corresponding to a concept are stored in a collection named after that concept's name. The actual data is stored as it is received from the sources in a sub-collection "results", the describing data, the query string decomposed to the single sub-goals and a reference to the corresponding result document, are stored in another sub-collection called "entries" . During query matching the XML encoded entries are read from the database and the match type for the currently handled query is determined. If an entry matches, a part of or the whole result document is added to the global result data. If only a part of the cached result is needed, i.e. if the two queries overlap in some way, the part corresponding to the processed query has to be extracted. In order to retrieve the required data we simply have to execute the current query against the data of the entry.

8 CONCLUSION AND FUTURE WORKS

In our work we tried to draw general design for Web communities. We intend to help project managers and system developers, as well as everyone that is involved in the design, implementation or adoption of Web services communities' solution. Thus, the design approach proposed can be exploited in the design phase of any Web service community as a vector of design goals that should be addressed by the designers.

Therefore, we trust that our proposed mediator can contribute towards a more automated query resolution and optimization of query answering approaches, both at the conceptual and the implementation level. Our work provides complementary contributions to related work on Web Communities Building and Querying. We focus on providing support for achieving effective and efficient access Queries treatments results.

In summary, in this work, we presented design approach for modeling and integrating Web communities. We proposed queries cache that helps answering frequent queries. We also illustrated the viability of the proposed approach through a case study for a Health Care System. However, queries maintenance system involved in our approach will be proposed and detailed in our future works since it is complicated and addresses many issues. Ongoing Works include case studies to evaluate our mediator in a distributed environment. We also plan to extend the proposed approach by adding to our mediator the functionality of the automated discovery of communities' members.

To improve the query processing efficiency, a component for query refinement can be added to our system in order to process queries based on users' preferences.

Moreover, an important issue needs to be dealt with a semantic caching system in general is the policy of admission into the cache and eviction from it. When a query is executed, we must decide what part of the query result to add to the cache. If the cache is full, we must also decide which of the currently cached results to evict from the cache.

REFERENCES

[1] B.Benatallah, Q. Z. Sheng. & M.Dumas, (2003) "The self-serv environment for web services composition", IEEE *Internet Computing*, Vol. 7, No. 1, pp40-48.

[2] D.Benslimane, Z. Maamar, Y. Taher, M. Lahkim, M.-C. Fauvet & M. Mrissa, (2007) "A multi-layer and multi-perspective approach to compose web services", IEEE *Computer Society In AINA*, pp 31–37.

[3] B. Medjahed . & Y. Atif, (2007) "Context-based matching for web service composition", *Distributed Parallel Databases*, Vol. 21, No. 1, pp5-37.

[4] B. Medjahed . & A. Bouguettaya (2005) " A dynamic foundational architecture for semantic web services", *Distributed and Parallel Databases*, Vol. 17, No. 2, pp179-206.

[5] S. Sattanathan, P. Thiran, Z. Maamar . & D. Benslimane, (2007) " Engineering communities of web services", iiWAS *Austrian Computer Society*, Vol.229, pp57-66.

[6] Y. Taher, D. Benslimane, M.-C. Fauvet & Z. Maamar, (2006) " Towards an approach for web services Substitution", *IOS Press*, pp 166–173.

[7] K. Baina, B. Benatallah, H. Paik, F. Toumani, C. Rey, A. Rutkowska. & B. Harianto, (2004) "WS-CatalogNet: An Infrastructure for Creating, Peering, and Querying e-Catalog Communities" , *In Proc. of the 30th VLDB Conference.*

[8] H. Paik, B. Benatallah. & F. Toumani, (2004) "WS-CatalogNet: Building Peer-to-Peer e-Catalog*", In Proc. of 6th International Conference on Flexible Query Answering Systems.*

[9] D. Benslimane, M. Barhamgi. & M. Medjahed, (2010) "A Query Rewriting Approach for Web Service Composition", IEEE *transactions on service computing*, Vol.3.

[10] Y.Wang, J. Zhang. & J.Vassileva, (2010) "Effective Web Service Selection via Communities Formed by Super-Agents", IEEE / WIC / ACM *International Conferences on Web intelligence.*

[11] H.-Y. Paik, B. Benatallah. & F. Toumani, (2005) "Towards Self-Organizing Service Communities", IEEE *Transactions on Sys. Man and Cyb* , Vol. 35, No. 3, pp408-419.

[12] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey. & F. Toumani, (2004) "On Automating Web Services Discovery". In .VLDB Journal.

[13] M. J. Carey, M. J. Franklin, M. Livny E. & J. Shekita, (1991) "Data Caching Tradeoffs in Client-Server" DBMS Architectures, In Proc. SIGMOD *Conference*, pp 357-366.

[14] B. Benatallah, M.-S. Hacid, H.-Y. Paik, C. Rey. & F. Toumani, (2003) " Peering and Querying e-Catalo Communities", Technical Report UNSW-CSE-TR-0319, CSE, UNSW.

[15] P. Godfrey. & J. Gryz, (1999) "Answering Queries by Semantic Caches. In Database and Expert Systems Applications", 10th Int. Conf., DEXA, Vol 1677 of LNCS, pp 485- 498.

[16] D. Lee W. & W. Chu (2001) "Towards Intelligent Semantic Caching forWeb Sources" Journal Of Intelligent Information Systems, Vol. 17, No. 1, pp23-45.

[17] S. Dar,M, J. Franklin, B. Jonsson, D. Srivastava . & M. Tan, (1996) "Semantic Data Caching and Replacement" In VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, pp 330–341.

[18] A. M. Keller . & J. Basu, (1996) " A Predicate-based Caching Scheme for Client-Server Database Architectures." VLDB Journal: Very Large Data Bases, Vol. 5, No. 1, pp35-47.

[19] M. Lenzerini, (2002) "Data Integration: A Theoretical Perspective" In L. Popa (Ed.),PODS'02, pp. 233–246.

[20] C. Beeri, A. Levy. & M.-C. Rousset, (1997)" Rewriting Queries Using Views in Description Logics". In. L. Yuan (Ed.), PODS'97, pp. 99–108.

[21] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava . & M. Tan, (1996) "Semantic Data Caching and Replacement" In Proc. 22nd VLDB Conference, pp 330-341.

[22] J. Fonseca, Z. Abdelouahab, D. Lopes. & S. Labidi, (2009) "A Security Framework for SOA Applications in Mobile Environment ", International Journal of Network Security & Its Applications (IJNSA), Vol.1, No.3, pp 92-107.

Authors

Hela Limam received the master degree in Computer Sciences applied to management Institute of management in 2007.She is actually a PhD student in computer sciences in the high institute of management of Tunis