Chapter 4

Indexing service for dataspaces

This chapter describes the general architecture of an indexing service for managing and exploiting the multi-level index we proposed.

An indexing service is composed by a set of sub-services managing each layer in the index. These sub-services implement a set of functions including: query answering, resources retrieval and documents persistence. Additionally, the indexing service at each layer interact together to (i) offer query services, (ii) manage the multi-level index, and (iii) optimize queries over the multi-level index.

The following sections present the general architecture of the indexing service detailing the components associated to each layer in order to manage and optimize the execution of operations over its associated indexing structure.

4.1 General architecture

[13] define the minimal components of a dataspace environment (c.f. Figure 4.1) and sets the indexing service as a component giving support capabilities for evaluating queries over the relationships presented among the dataspace's components (consumers, producers, documents, data sources, etc.), thus allowing the consumers to retrieve data from these resources.

Within the context of this project, the indexing service has two main objectives:

1. To answer queries in a prompt way and to reduce the costs related to the global index access. To achieve this, we introduced the notion of *neighborhood* and *cache* as two

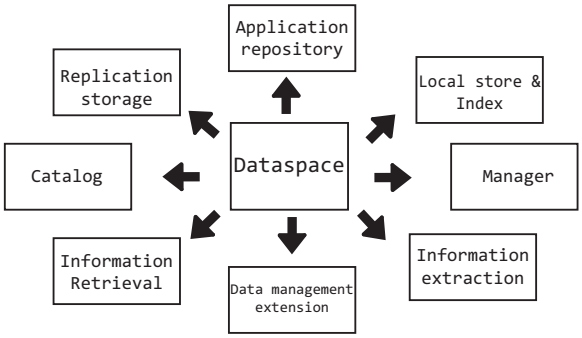

Figure 4.1: Dataspace's components

colaborative elements used to optimize index search. This elements implement a *pay-as-you-go* strategy [26] to refine resources continuou

2. Resolver consultas en tiempo y reducir el acceso al índice global. Para lograrlo hemos introducido la noción de vecindario y caché como dos elementos que colaboran para optimizar búsquedas siguiendo el paradigma Pay-As-You-Go [26] como un refinamiento de resultados incremental que recupera datos de consultas anteriores para mejorar las siguientes. Tanto el vecindario y el caché están compuestos por sub-vecindarios y sub-cachés de cada una de las capas. A reserva de que más adelante se detallen estas nociones en cada capa, de manera general el caché es una versión resumida del índice que se actualiza conforme a relaciones de vecindad entre los datos indexados. La intención es tener pronto acceso a los elementos del índice que son más consultados y que guardan una relación de cercanía entre sí. Hay que notar que el caché no es un caché de datos duros sino que es una parte resumida del índice. Los elementos en el caché resultan tener cierta cercanía (*e.g.,* proximidad física, descripción similar, frecuencia de aparición en los mismos resultados) para hacer al evaluador de consultas más eficiente.

3. To guarantee the validity of results by maintaining up-to-date the indexing structures associated to each layer. To achieve this, we have identified a set of events triggering updating actions along each layer. Events may be presented independently at each layer triggering updating actions over other layers (e.g., a change within a document's content may require to modify the definition of certain concept. Additionally, events may be

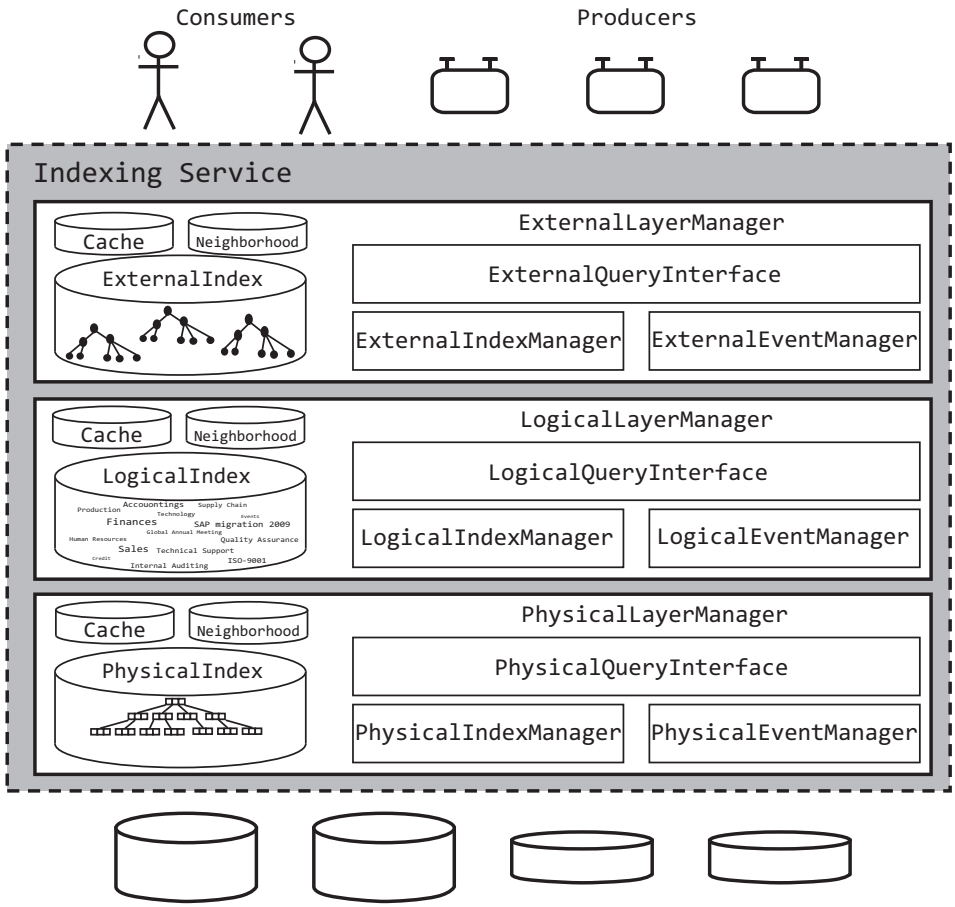

Figure 4.2: Indexing service general architecture

presented outside the index (e.g., several queries over certain concepts may cause to execute update actions over the caches managed at each layer).

As we presented in the previous chapter, we prosed a three-level organization of the dataspace in order to solve the problems associated at each abstraction level (semantics, identification, and location). Each level is indexed with respect to its layer model and must be managed using a layer management service defining and executing a set of query and management functions over the data structures associated to its model.

Figure reffig.IndexingServiceArchitecture presents the general architecture of an indexing service. An indexing service is composed by a set of layer managers, where each manager: (i) specializes an abstract architecture with respect to the model associated to the layer

it manages, (ii) provide query processing capabilities over its indexing structures, and (iii) manages the indexing structures, caches and neighborhood associated to the layer.

The rest of this section describes the main components of the indexing service. First, we will present the structure of the interface exported by the indexing service so that consumers and producers access to its functions(4.1.1). Then, we will describe the abstract architecture of a layer manager (4.1.2). Sections 4.2, 4.3 and 4.4 presents the architecture of the physical, logical, and external layer managers, which specialize the layer manager abstract architecture and adapt the notion of cache and neighborhood according to the characteristics of its associated layer. Finally, Section 4.5 describes how the indexing service is assembled by coordinating each layer manager in order to implement its functions.

4.1.1 Indexing service interface

The indexing service interacts with the consumers and producers of the dataspace through an exported interface and a data model. This interface is composed of two sub-interfaces, each one oriented to the consumers and the producers respectively. The rest of this section describes the data model and interfaces of the indexing service.

Data model. Figure 4.3 presents the data model used by the indexing service satisfying the necessities of the dataspace's consumers and producers. From a consumer perspective, the data model allows to characterize a query (keyword-based or predicate-based) and its associated results. Keyword queries are expressed as a set of terms (`Term`), while predicate queries are expressed as a set of elements (`Filter`) associating attributes and values using binary operations (e.g., $salary > 30,000$, $Autobus$ `is` $Transport$). A query is associated to one or several results partially ordered according to its recall. A result can be defined as a set of documents or terms, where terms may be knowledge domains (e.g., the term `Industrial production` or *Computing*). From a producer perspective, the data model characterizes documents with a name `docName` associated to a `file`, and a unique identifier `GDL` (see Section 3.1.1).

Consumer's interface. Figure 4.4 presents the consumer's interface providing a set of query and document retrieval functions over the multi-level index. The main functions pro-

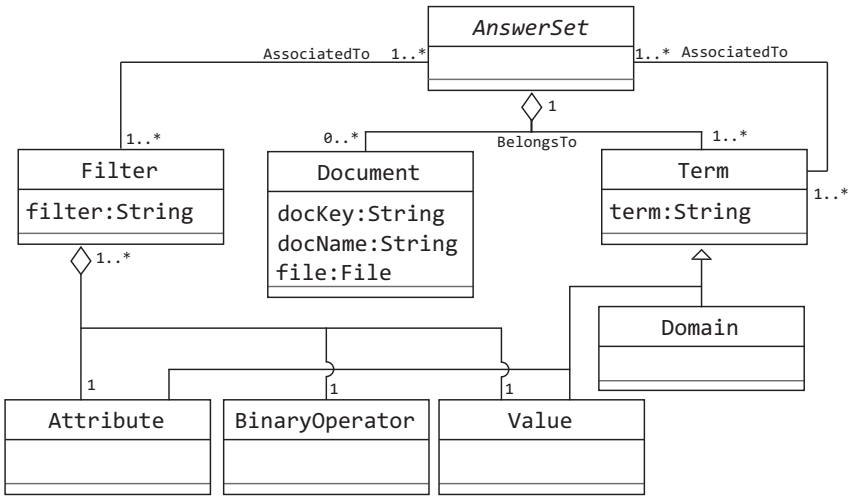

Figure 4.3: Indexing service data model

vided by a consumer's interface are the following:

- `getdomains():AnswerSet.` Retrieves the knowledge domains defined over the dataspace. A domain is defined as a concept that overlaps all the concepts describing it, giving an *overview* of the dataspace's content.

- `subsumption(Concept):AnswerSet.` Retrieves the concepts subsuming a particular concepts executing a *Top-down* search using the taxonomic relations defined over the dataspace (e.g., *Citizen* `is` *Person*). The result is a set of terms and/or documents related to the input concept.

- `expand(Concept,scope):AnswerSet.` Retrieves the concepts that are associated to the input concept using a non-taxonomic relation (e.g., *Citizen* `has` *Nationality*). Retrieved concepts may be associated to other concepts (e.g., *Nacionality* `is` *SimpleNationality*, *Nationality* `is` *DobleNationality*). For this reason, the `scope` is used to determine the amount of non-taxonomic relations to be taken into account given an input concept.

- `applyFilter(AnswerSet, Filter):AnswerSet.` Given an answer set, this operator applies a set of constraints (`Filter`) defined as tuples attribute-value. This operator uses non-taxonomic relations to associate concepts (as terms) with values, thus applying

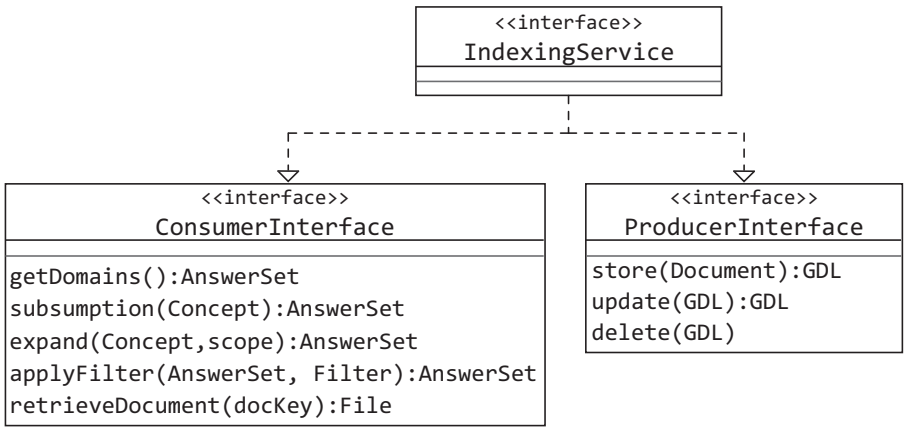

Figure 4.4: Indexing service interface

binary operators (e.g., $Nationality = Mexican$, $Age > 18$). The output of this operator is a subset of the answer set satisfying the constraints defined in `Filter`.

- `retrieveDocument(docKey):File.` Given a document key, this operator retrieves the file containing the document from the storage space where it resides. This function can be used when the key of a document satisfying a consumer's requirement is known.

Producer's interface. Figure 4.4 presents the producer's interface describing the functions (i) to make documents persistent, and (ii) to update and remove them from the storage space. This interface is consider a 'direct access' to the persistence service provided by the physical layer.

While the scope of this project does not involve managing the persistence services as a shared storage space, it is important to specify the functions implemented by the indexing service. This because documents' storage is related to events influencing the behavior of the physical and upper layers.

The main functions described by the producer's interface and associated to the documents' persistence are the following:

- `store(Document):GDL`. Stores a document into the storage space. The output of this function is a key (`GDL`) identifying the document in a unique way within the storage space.

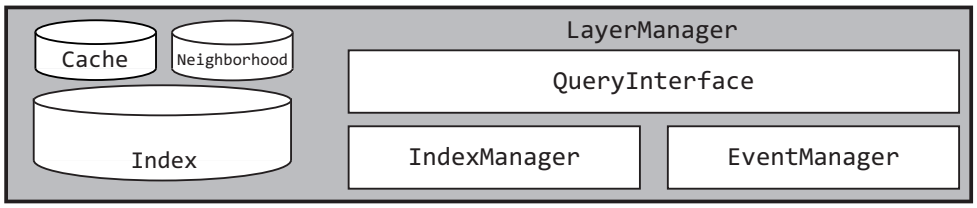

Figure 4.5: Layer manager abstract architecture

- `update(GDL,newDocument):GDL`. Updates a document identified with a `GDL` key. The output is a key whose value can be the same as the input one, or a new key according to the result provided by the persistence service in the physical layer.

- `delete(GDL)`. Removes a document identified with a `GDL` key from the storage space.

4.1.2 Layer manager abstract architecture

An abstract architecture describes the minimal components and functions provided by a layer manager. Additionally, it models the communication protocols among the classes implementing these functions and the minimal messages required to achieve the management and update of the index associated to each layer.

The rest of this section focuses on describing our data model to characterize the notion of cache and neighborhood. Additionally, we present the interface provided by a layer manager service and the functions of each of its components. Finally, we describe a model to characterize the messages interchanged among thse components.

Index management data structures

Data structures used by the manager aim to optimize the execution of queries over the index. As the cache is defined as a summary of the general index, both the index and the cache must be implemented using a same data structure (e.g., B-Tree, TV-Tree, DHT). A cache is updated with respect to the probability a certain element has to be queried. As there are document clusters having documents close to each other according to their content, physical location or as they are the result of a specific query at certain instant (e.g., a requirement associated to the term 'YAGO' involves all documents associated to the terms 'YAGO' and

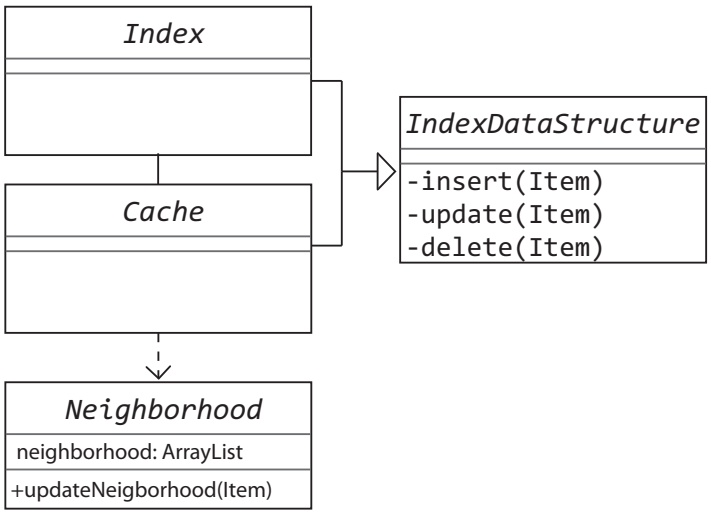

Figure 4.6: Query optimization data structures

'NAGA' [37]). We model this relation as a neighborhood.

Cache. A cache represents a summary of a general index and must be defined using the same data structure as the general index.

Neighborhood. A neighborhood is defined as a list composed by the most significant elements E of the index. Each element E_i has an associated list N of its neighbors. Neighbors are partially ordered with respect to its closeness with E_i.

Layer manager interface

Figure 4.7 details the two minimal functions of a layer manager specified within its interface. According to the managed layer, this interface can be specialized by adding other additional functions.

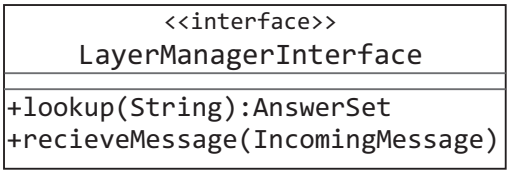

Figure 4.7: Interfaz de servicio administrador de capa

- `lookup(String):AnswerSet.` This function is related to the execution of queries over the index, allowing it to be queried using a string as input. This input is specialized according to the managed layer and the characteristics of the keys identifying the index's components (e.g., set of terms at a logical layer, or a GDL at a physical layer). The output of this function is an answer set.

- `recieve(IncomingMessage).` This function is related to the maintenance of the indexing structure. The input of this function is a notification message sent by the remaining layers to trigger actions in order to update the indexing structure, a cache or a neighborhood. The messages typification is partially described in Figure 4.8. However, messages may be specialized according to each layer and the communication protocols among layers.

Layer manager classes

QueryManager. This class analyzes the queries executed over the index (`lookup(String)`), and executes a search over the cache in order to retrieve the element. If the query is not satisfied by the cache, then the main index is queried. If the query is satisfied either using the cache or the main index, the query manager generates a set of events that are sent to the *Event Manager*. Some of these events are the following:

- The fact that a query over a particular event was received.
- The fact that an element was successfully found in the cache.
- The fact that an element was not found in the cache.
- The fact that an element was successfully found in the index.
- The fact that an element was not found in the index.

EventManager. This class receives local notifications (`LocalMessage`) provided by the `LayerManager`, the `IndexManager`, and the `QueryManager`. Additionally, this class resends messages to the requested layers (`UpgoingMessage`, `DowngoingMessage`) as well as to the `IndexManager`.

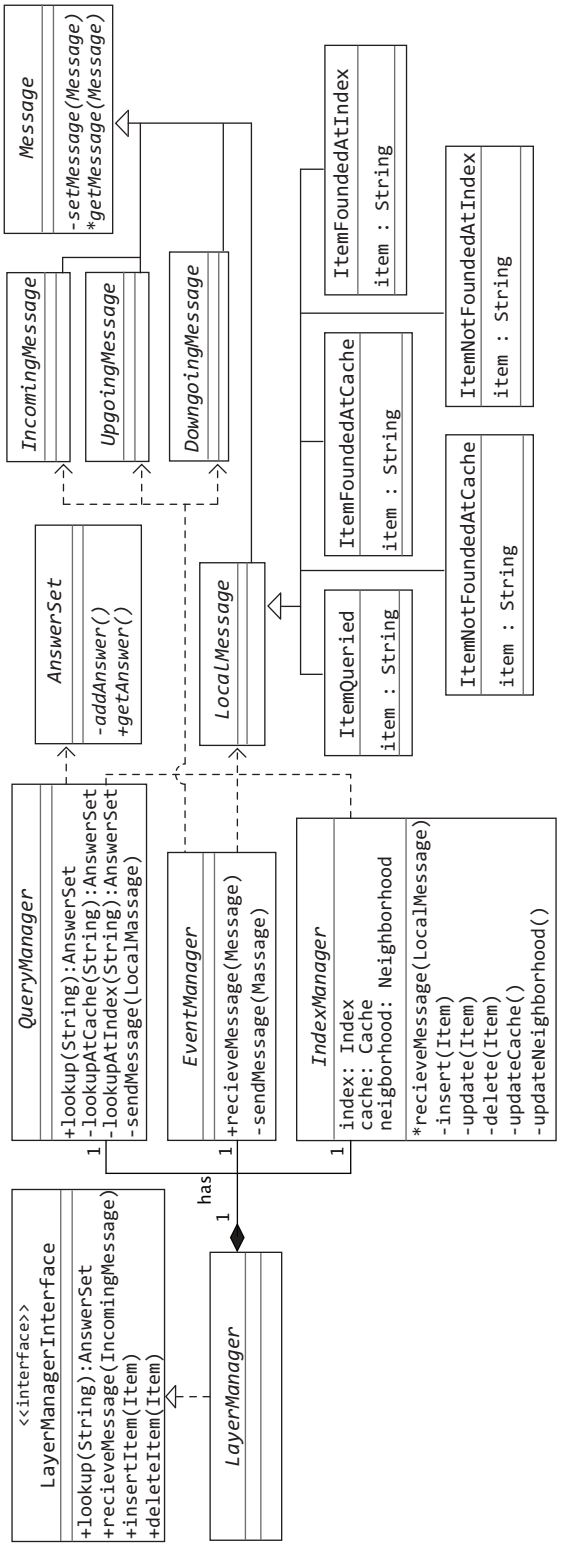

Figure 4.8: Layer manager classes

IndexManager. This class implements the index management operations (`insert()`,`update()`); receives local notifications (`recieveMessage(LocalMessage)`) triggering one or several management operations over the index (`insert(), update(), delete()`), the cache (`updateCache()`), and the neighborhood (`updateNeighborhood()`). Cache update is controlled by a set of policies (selection, size, replacement). These policies are submitted into an experimental process within each layer in order to obtain measures using different indexing structures; applying centralized and distributed architectures to store the index; and combining three policies.

- **Selection policy.** This policy is tightly related to the notion of neighborhood. A selection policy defines the semantic of a neighborhood among the index's elements. For instance, in the physical layer, a selection policy may state that a document and its neighbors (according to a particular query) can be loaded into the cache. Another selection policy may define that documents having a close physical relation are stored in the cache.

- **Size policy.** This policy defines the maximum size of a cache (e.g., the cache size of a mobile device may be limited by its storage capabilities), or the amount of elements stored in the cache (e.g., a 10% of the elements stored in the index).

- **Replacement policy.** This policy defines how the elements to be removed from the cache are going to be selected. For instance, a layer manager may adopt a LRU (*Least Recently Used*) or a MRU (*Most Recently Used*) policy.

Abstract layer manager coordination

The main components of a layer manager interact with each other by sending messages, and together implement the functions offered by a layer manager within its interface (see Figure 4.1.2). The following list establishes the main operations implemented by a layer manager.

LookUp. Figure 4.9 describes how the layer manager's components interact together to complete an element search over the index layer. In principle, it is assumed that a search was requested to the layer manager. The layer manager delegates the action to the query manager, who looks up the element in the cache and notifies the event manager that the

element is being queried. If the cache search is not satisfied, then the query manager executes a query over the index and notifies the event manager that the element was not found in the cache and is been looked up in the index. If the query is not satisfied, then it is notified to the event manager.

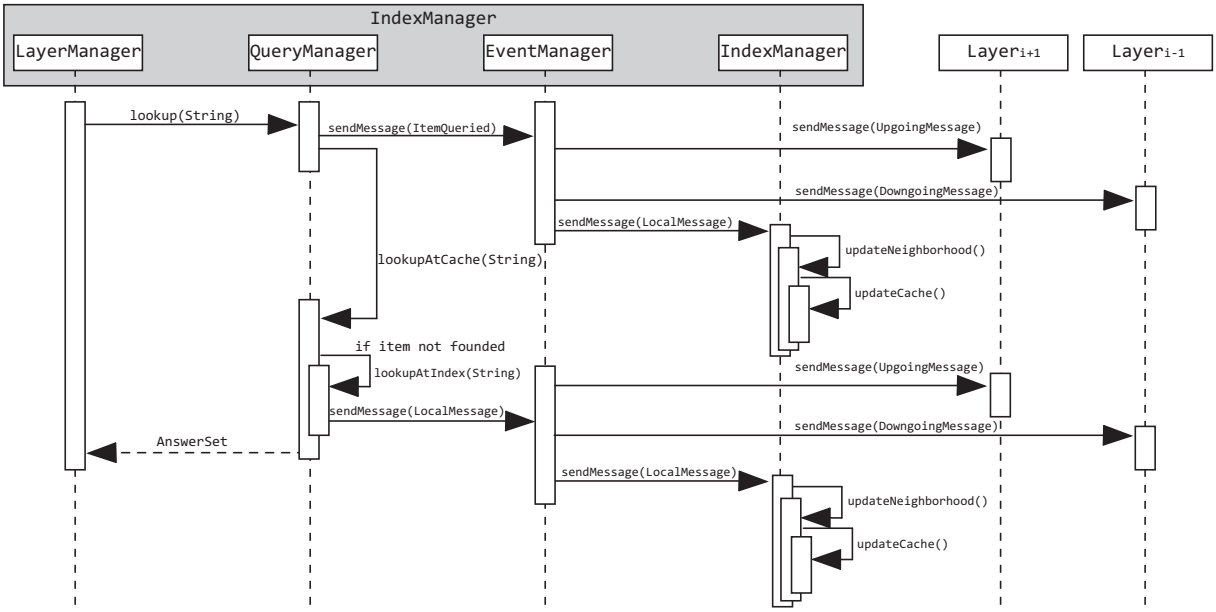

Figure 4.9: Layer manager coordination - Look up

The event manager receives the notifications sent by the query manager stating which elements are being queried and the success or failure of searching them in the cache or the index. This data is sent to the upper layers to execute the pertinent operations, and to the index manager managing the data structures (cache, index, neighborhood).

When the index manager is notified that an element was queried, it updates the data stored in the neighborhood, and execute the pertinent updates over the cache (e.g., the substitution of certain element frequently queried).

Insertion. Figure 4.10 presents the interact among the layer manager, the event manager and the index manager to insert an element into the index.

The layer manager receives a request to insert a document into the index (e.g., a new document is published in the dataspace, so the document must be inserted into the storage

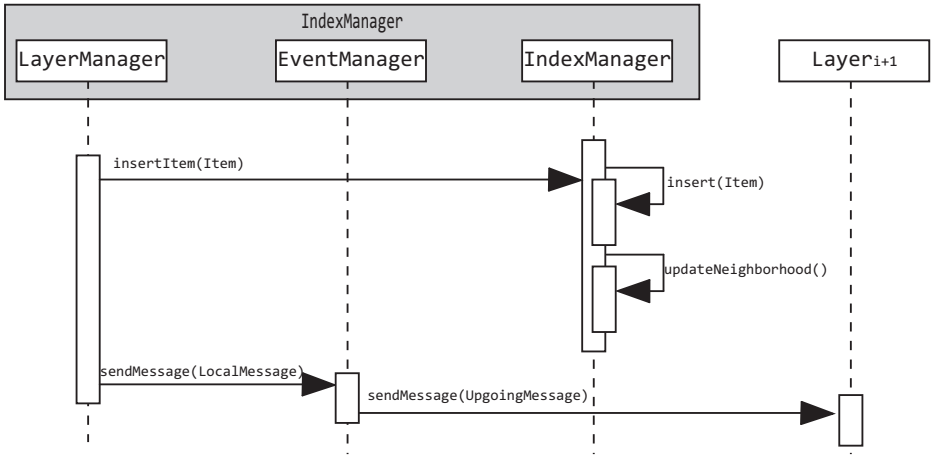

Figure 4.10: Layer manager coordination - Insertion

space, and the resource and the terms describing its content must be incorporated into the term space). The layer manager delegates the insertion operation to the index manager. The index manager inserts the element within the index and updates the neighborhood if it is required (e.g., a new element has a physical closeness to another element). Once the element has been inserted the layer manager notifies the event manager the success of the operation in order to broadcast the message to the pertinent layers.

Removal. Figure 4.11 illustrates how the layer manager's components interact together to remove a particular element from the index.

The layer manager receives the request to remove an element from the index (e.g., a document has been removed from the storage space, a resource has been removed and deleted with their associated annotations). Then, the layer manager delegates this operation to the index manager, who removes the element from the indexing structure and updates the neighborhood if it is required (e.g., a resource having multiple neighbors has been removed).

Message reception. Figure 4.12 describes how the layer manager, the event manager and the index manager interact together by passing local messages to manage pertinently its associated layer. Additionally, these components communicate with the remaining layer using messages. Message notification is oriented to the triggering of Event-Condition-Action rules

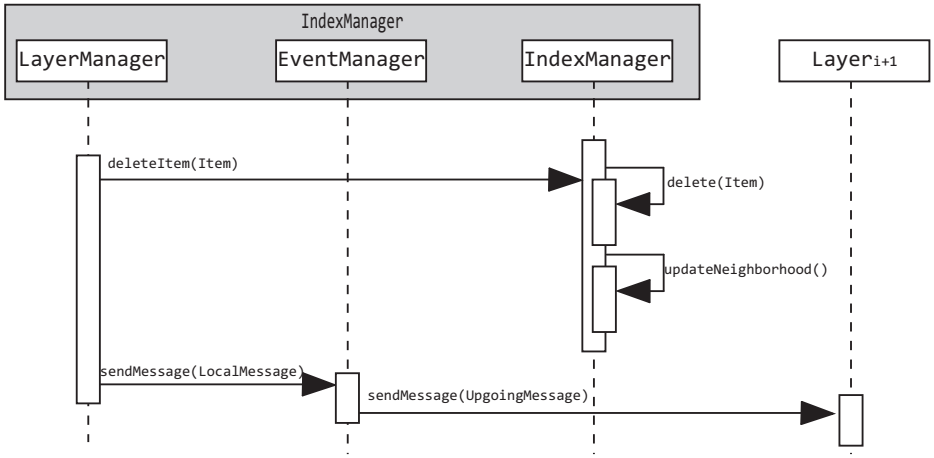

Figure 4.11: Layer manager coordination - Removal

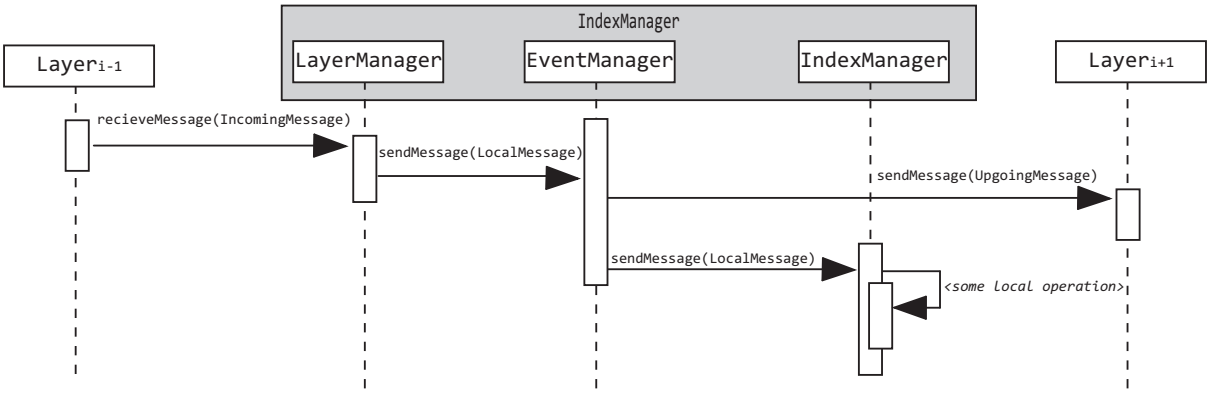

Figure 4.12: Layer manager coordination - Message reception

(ECA) in order to manage the index, the cache, and the neighborhood defined at each layer given a set of events presented over the layers.

The layer manager may receive the messages from the outside or may generate them according to the presence of a local event. These messages are notified to the event manager, who broadcast the message to the other components of the layer manager or to another layers. When the index manager receives a notification, it triggers a maintenance operation over the index, the cache and the neighborhood.

4.2 Physical layer manager

The physical layer manager receives the requests oriented to the persistence of a specific document within the storage space. Given a certain request, a notification message (e.g., a document has arrived, has been updated or has been removed) is sent to the upper layers. The manager implements a function to retrieve documents from the persistence service where it resides. Additionally, the physical layer manager receives, from the persistence services, the notifications stating that a document has modified its physical address. These notifications allow the physical manager to determine when to execute update operations over the indexing structure.

In the physical layer, documents are associated to GDL (Global Document Locator) addresses that locate them globally. These addresses are associated to a segment within the storage space managed by a persitence service.

The following lines describe the implementation of the notion of cache and neighborhood in the physical layer used to optimize the execution of the document retrieval process.

4.2.1 Physical cache and neighborhood

The notion of neighborhood between two GDL $\delta(D_1, D_2)$ is defined as the probability that D_2 is searched along with D_1. Figure 4.13 presents the characterization of a GDL neighborhood using load counters. For each pair of GDL, there exists a load probability formally defined through the formula:

$$\delta(D_1, D_2) = \frac{loads\_count\_D_2\_with\_D_1}{loads\_count\_D_1}$$

Each time D_1 is loaded, the counter $loads\_count\_D_1$ is incremented. When D_2 is loaded along with D_1, the counter $loads\_count\_D_2\_with\_D_1$ is incremented.

4.3 Logical layer manager

The logical layer manager receives the notifications sent by the physical layer (document inclusion, modification, and removal from the storage space). In this layer, we take into account the term space of the dataspace. Terms describe the documents' content using a frequency matrix (3.2). The frequency matrix is built by setting the documents into a term

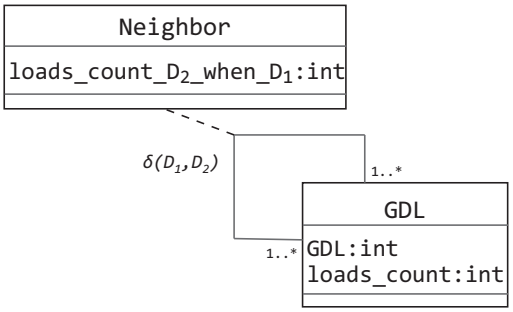

Figure 4.13: GDL neighborhood

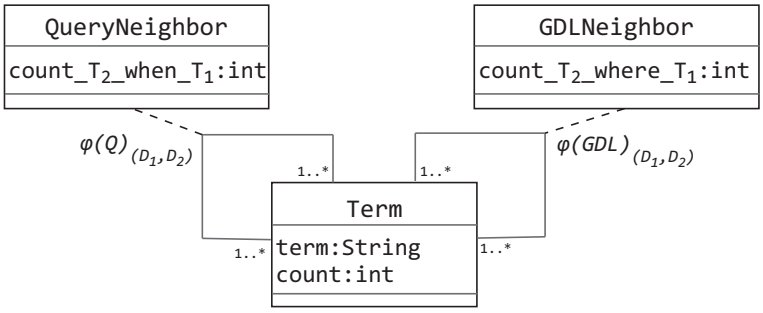

Figure 4.14: Term neighborhood

harvesting process, and associating the retrieved terms to their GDLs. When a document is inserted, modified, or removed, the term space is updated. For this reason, the logical layer must notify this event to the external layer.

The logical layer indexes the term space using a TV-Tree (Telescopic vector) modeling each GDL as a multidimensional resource where each term describing it is considered a dimension. The rest of this section describes the implementation of a logical cache and neighborhood to optimize the execution of queries over this structure.

4.3.1 Logical cache and neighborhood

The notion of neighborhood between two terms $\varphi(T_1, T_2)$ is defined as the probability that T_2 appears along with T_1. In the logical layer, two types of neighborhood are defined (cf. Figure 4.14):

Query occurrence (`QueryNeighbor`) . Let Q_i be a query expressed as a set of terms T. We state that T_1 appears along with T_2 in Q_i if

$$T_1 \wedge T_2 \in Q_i(T)$$

. This way, the neighborhood between T_1 and T_2 is defined as the probability that T_2 occurs along with T_1 within a query Q:

$$\varphi(Q) = \frac{counts\_T_2\_when\_T_1}{counts\_T_1}$$

GDL occurrence (`GDLNeighbor`) . Let GDL_i be a locator associated to the document i described by a set of terms T. We state that terms T_1 and T_2 appear in GDL if

$$T_1 \wedge T_2 \in GDL_i$$

. This way, the neighborhood between T_1 and T_2 is represented as the probability that T_2 occurs along with T_1 in a GDL:

$$\varphi(GDL) = \frac{counts\_T_2\_where\_T_1}{counts\_T_1}$$

4.4 External layer manager

The external layer manager receives the queries expressed by the consumers. This manager provides an integrated view over the concepts evoking the dataspace's content. These concepts are semantically related among each other through taxonomic and non-taxonomic relationships. Taxonomic relationships define a hierarchy among concepts and are of two main types:

- **Subsumption**.A concept specializes another concept (*e.g.*, Man `Subsume` $Person$, $Woman$ `Subsume` $Person$)
- **Coverage**. A concept generalizes a set of concepts (*e.g.*,$Person$ `Cover` $(Man, Woman)$)

Non-taxonomic relations represent a same-level semantic relationship between two concepts and characterize the properties associated to a concept:

- **Equivalence**. A concept is equivalent to another one (*e.g.*, $Human$ `BeEquivalent` $Homosapiens$)
- **Disjointness**. A concept is semantically disjoint to another concept (*e.g.*, $Hombre$ `Disjoint` $Mujer$)
- **Attribute**. A concept is characterized by a specific attribute (*e.g.*,Man `Attribute` Age)

Each concept is associated to a set of terms of the logical index. These associations are defined as a mapping defined by a group of experts or through an inference process.

4.4.1 Concept neighborhood

The notion of neighborhood within the external layer is defined wiht respect to the semantic relationships presented among terms. The neighborhood between two concepts C_1 and C_2 ($\sigma(C_2, C_1)$) is defined as the probability that C_2 appears along with C_1 in an answer A. When a semantic relationship is defined between C_1 and C_2, then $\sigma = 1$. As queries are evaluated, the neighborhood relation increases or decreases. The neighborhood relation between C_1 and C_2 is closer if C_2 along with C_1 belongs to the answer A if

$$(C_1 \wedge C2 \in A) \wedge A \rightarrow Q$$

when A satisfies Q (satisfiability may be determined through user feedback). This way, the neighborhood between C_1 and C_2 is defined as the probability that C_2 co-exists along with C_1 within a query Q:

$$\varphi(Q) = \frac{counts\_C_2\_when\_C_1}{counts\_C_1}$$

4.5 Indexing service coordination

Layer managers interact together in order to provide the main operations of the indexing service (cf. Figura 4.4). Each manager receives requests to trigger a set of operations according to the layer they manage. For instance, the physical layer faces the requests associated to the persistence of documents offered to data producers (`store`, `update`, `delete`)y la recuperación de los documentos (`retrieveDocument`); the external layer faces the requests related to the dataspace's semantic (`getDomains`, `subsumption`, `expand`, `applyFilter`); and the logical layer defines the mapping between the external and physical layers, so the operations it

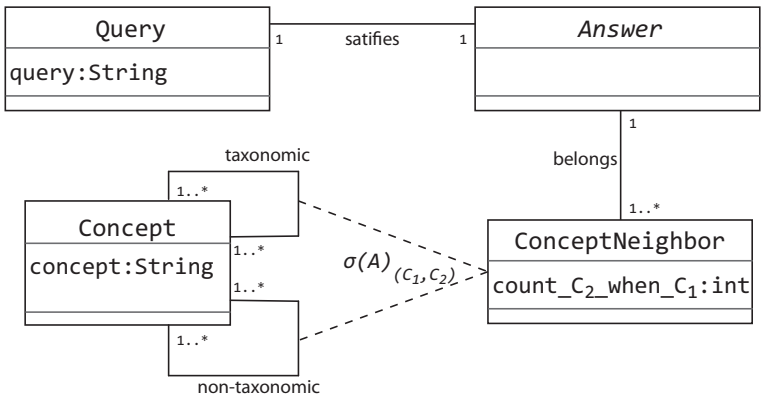

Figure 4.15: Concept neighborhood

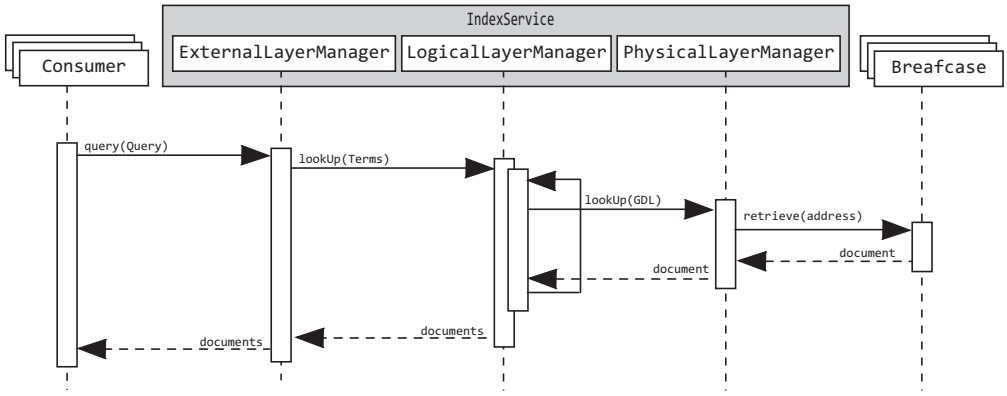

Figure 4.16: Indexing query articulation for query answering

implements are those related to the indexation of the term space associating the concepts in the external layer with the documents in the physical layer.

Figure 4.16 describe the sequence diagram related to the evaluation of a query expressed over the dataspace by exploiting the indexing service. A query is presented by a consumer and may involve the execution of operations over the physical, logical, and external layers of the index. However, there may exists more specialized queries where queries are expressed as a set of terms, similar as search engines; or where queries specify the addresses of the documents to be retrieved. In both cases, queries are evaluated by the pertinent layer manager.

Moreover, consumers may require to enrich their queries over time as the retrieved resources do not totally satisfy its requirements. In this cases, other interface methods within

the indexing services are applied like a `subsumption`, `expand`, `applyFilter` method.

4.6 Conclusions

This chapter presented an architecture for the construction of an indexing service organized into three main levels. Each level characterizes a set of management functions defined with respect to its associated layer model. Each manager functions involve: (i) index layer updating in order to maintain it up-to-date respect to the dataspace's evolution, (ii) query processing over the layer, and (iii) layer communication in order to achieve an effective management of the dataspace's index. All layers managers compose the indexing service of the dataspace. This service allows to identify the concepts describing the dataspace's content, as well as its associated terms describing the resource's content to be retrieved from their persistence services.