A Pattern-based Tool for Learning Design Patterns

Christian Paz-Trillo , Renata Wassermann , Fabio Kon

[1]Computer Science Department
Institute of Mathematics and Statistics
University of São Paulo, Brazil
Rua do Matão 1010, São Paulo, SP 05508-090

{cpaz, renata, kon}@ime.usp.br

Abstract. *Looking for information in long videos can be a time-consuming experience. The* ONAIR *system was developed in order to allow the users to find the information they want through queries in written natural language. We implemented the vector space model for information retrieval and used ontologies to improve the retrieval. In this paper we describe the system architecture, which makes strong use of patterns. We then describe an application of the system to the domain of Object-Oriented patterns in a course lectured by Joseph Yoder at the University of São Paulo.*

1. Introduction

The ONAIR(Ontology-Aided Information Retrieval) system [Paz-Trillo et al., 2004] was originally developed to allow users to query a video collection using written natural language. Our intention is to save the user from the time-consuming experience of manually browsing the entire collection to satisfy his/her information needs. The system includes a retrieval engine based in the vector space model [Salton et al., 1975]. It also implements a query expansion mechanism using an ontology [Gruber, 1993], containing the main concepts of the domain related to the interviews. In adition to the terms in the original user query, the expanded query includes some related terms. These terms are obtained from the video collection and their selection depends on an ontology-based similarity measure.

The system was first used with an interview with Ana Teixeira, a Brazilian Artist. The interview falls in the Contemporary Art domain and the language used is Brazilian Portuguese. We designed this system to be adapted to other subjects by changing the underlying ontology and video collection and some other configurations available from an administrative interface.

The system architecture is complex and uses some open-source libraries to achieve the expected functionality. We used some design patterns to reduce the complexity and to facilitate the maintenance and readability of the system.

In this paper, we describe an application of ONAIR to allow students learning about Object Oriented Programming (OOP) and Design Patterns [Gamma et al., 1995] taking advantage of a filmed course lectured by Joseph Yoder at the University of São Paulo, in 2004. The course is almost twelve hours long and it is not expected (neither realistic) to have students watching it from beginning to end. The idea is to help them to find the part of the course which best answers their questions.

In Section 2 we explain the two main processes of ONAIR: indexing and retrieval. Section 3 shows ONAIR's architecture, highlighting the use of patterns. We describe our proposal to use ONAIR with the filmed course in Section 4. In that section we also justify the use of an ontology to improve the retrieval in this domain. In Section 5, we describe the results we expect to get when using ONAIR with the course.

2. The ONAIR System

ONAIR is in essence an information retrieval system for retrieving documents from a collection [van Rijsbergen, 1979, Baeza-Yates and Ribeiro-Neto, 1999]. And, as such, it has two main processes: indexing and retrieval. The indexing process takes a collection of documents and other information (a domain ontology and keywords associated to each document) and generates the structures needed to allow the retrieval process to use it to respond to user queries. In this section, we describe both processes and in the next section we show how we used patterns to keep the architecture simple.

2.1. Indexing Process

The indexing process is responsible for the creation of the structures that the retrieval process will use. The input of this process consists in:

- **A Video Collection**: In the case of ONAIR, the collection consists in a set of short clips, manually extracted from a long interview or lecture. Each clip is associated with: (1) a set of manually assigned keywords, usually selected by the domain specialist; (2) a set of resources (images) that can be shown during the video in a specified period and, optionally; (3) a transcription of the speech.
- **A Domain Ontology**: An ontology is *"a specification of a conceptualization"* [Gruber, 1993]. It describes the domain concepts and the relationships between them. The ontology forms the system vocabulary, so it should contain the main concepts that are referred to in the collection, as well as the terms that will be possibly used in user queries. It is used in the retrieval process to expand the original user queries.
  OWL [McGuinness and van Harmelen, 2004][1] format is a recently adopted standard to represent ontologies. We used Protégé [Gennari et al., 2003, Knublauch et al., 2004] ontology editor to build the ontologies used by ONAIR. Protégé is an open-source tool with an easy-to-use interface which facilitates the creation of the ontology by the domain expert.

The indexing process is executed through an administrative application that allows a system administrator to register the video clips, the ontology and establish other configuration values. This process produces the three following components:

- **XML Configuration File**: Contains the associations between the clips and their transcriptions, keywords and resources, and some general configuration data.
- **Inverted Index**: An inverted index is an structure that stores the frequency and the occurrences of the terms in a collection [Jackson and Moulinier, 2002], in ONAIR case, the video collection. It also stores the weight of terms in a document,

[1]OWL: Web Ontology Language

to avoid computing it in the retrieval phase. The terms are saved after their affixes are removed by a stemming process. In ONAIR's first application we used the RSLP[2] algorithm proposed by [Orengo and Huyck, 2001]. Stopwords, i.e., common words such as articles or prepositions, are not included in this index because they do not help to determine the relevance of the document to a query.
  An index using the text in video transcriptions and a second one using just the keywords registered by the domain expert can be generated, depending on the information availability.
- **Ontology Data Structure**: OWL is a very expressive language, but in this application we did not use all of its representational power. Given that Ontology engines, like Jena[3][McBride, 2001], are prepared to deal with much of OWL expressiveness, we designed a simplified data structure that simulates the ontology behavior used by the retrieval process. This allowed us to improve the performance of the system significatively in terms of processing time[4].

The indexing process is a computationally expensive process but it needs to be executed only when the video collection or the ontology are modified.

2.2. Retrieval Process

Once the indexing process is executed, the configuration file, the inverted index and the ontology data structure are used by the retrieval process, implemented by a visualizer application. Figure 1 shows a screenshot of ONAIR using the original tested domain about the interview with Ana Teixeira.

This application allows the user to enter a query in natural language and shows to the user a list containing the videos that better answer the query ordered by relevance. It also allows the user to watch sequentially the listed videos or manually select one of them.

The retrieval process is shown in Figure 2 and its subprocesses are described here:

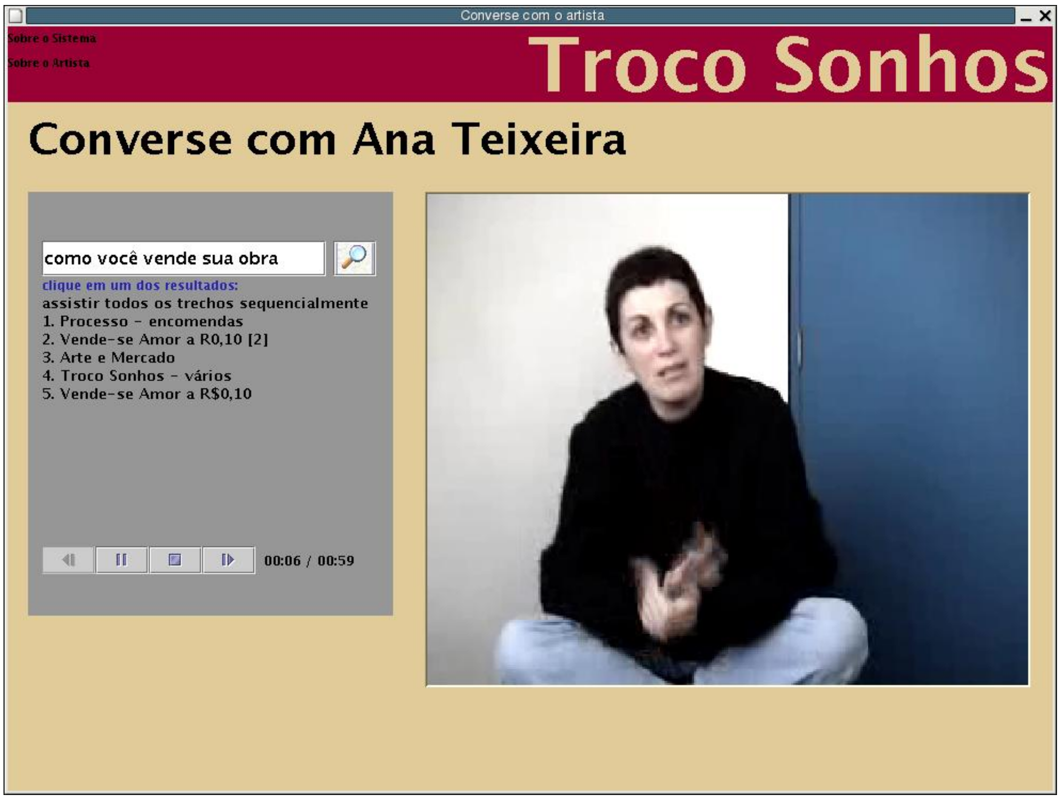
- **Pre-processor**: In this subprocess, a misspelling detection is applied to the user query. The Jazzy API[5] is used with a general dictionary (we used a Brazilian Portuguese dictionary, br.ispell [Ueda, 2002]) and a domain dictionary automatically extracted from the domain ontology, during the indexing process. Suggestions are presented with the results of the query, so the user can manually reformulate his/her query.
  Besides the misspelling detection, the stemming process is applied to the query. Finally, weights are assigned to the terms in the query, based on their presence or absence in the ontology and their frequency in the collection. For example, in the query *"Give me an example of a Wrapper implementation"*, *"me"*, *"an"* and *"of"* are stopwords, *"give"* receives a weight of 1, *"example"* and *"implementation"* are present in the ontology and receive a weight of 1.5 and *"Wrapper"* receives

[2]RSLP: Portuguese Language Suffix Remover

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

[4] Despite of this fact, we kept the implementation using Jena, because ontology engines are in constant development and possibly, in future versions, its performance will allow us to use it.

[5]Jazzy is an open-source Java API for misspelling correction and it is available at `http://jazzy.sourceforge.net/`



como você vende sua obra

clique em um dos resultados:
assistir todos os trechos sequencialmente
1. Processo - encomendas
2. Vende-se Amor a R0,10 [2]
3. Arte e Mercado
4. Troco Sonhos - vários
5. Vende-se Amor a R$0,10

00:06 / 00:59

Figure 1: ONAIR visualizer application screenshot showing part of the interview with Ana Teixeira.

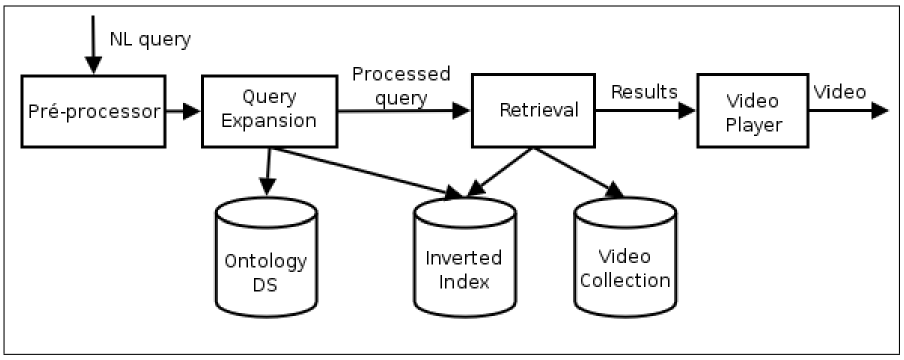


Figure 2: Retrieval process.

a weight of 2 because besides being in the ontology it has low frequency in the collection.

- **Query Expansion**: For each term in the collection, its similarity to the query is computed. This similarity is a weighted average of the similarity between the term and each query term. A synonym of a term, expressed as an equivalent class in OWL, has the maximum similarity value: 1.
  We used an approach that combines the use of the class hierarchy in the ontology, the terms frequency in the collection and the relationships in the ontology to compute the similarity [Lin, 1998]. The r most similar terms to the query are added to the original query, where r is a system parameter.
  In the former example, the expansion mechanism would add the terms *"Adapter"*

and *"Decorator"* to the query because they are very related to *"wrapper"* in the ontology (Figure 6).
- **Retrieval**: In this stage, the query is compared with the video clips contents (transcriptions or keywords) and the system retrieves those who best answer the query. We used the vector space model [Salton et al., 1975, Jackson and Moulinier, 2002] for retrieval. Both indices explained in the previous section can be used in conjunction and combined by a factor specified in the XML configuration file.
- **Video Player**: The system presents the list of videos ordered by relevance to the query. The user can then select which videos to play or watch all of them in a sequential manner. We implemented the video player using the Java Media Framework[6], which offers basic video functionalities such as play, stop, and pause.

3. A pattern-based architecture

ONAIR consists of two main applications: an administrative tool, that exposes the indexing process functionalities and a visualizer, for the user retrieval. They share a lot of structures and some processes implement and allow switching between different strategies. All this behavior suggested us the use of patterns, and we will show the patterns that helped us the most to enhance system design.

To explain the patterns used, we first describe the overall system architecture. In Figure 3 we present a simplified package diagram representing it. The packages in the system are:

- **admin**: Encloses the administrative application.
- **model**: Contains all classes needed to represent the videos, keywords and resources. It also contains a subsystem that allows to serialize and read it in XML format.
- **invertedindex**: Implements the data structure that stores the inverted index and contains a subsystem that generates it based on the model.
- **ontology**: Encapsulates the ontology behavior used in the application and implements the ontology data structure.
- **stemmer**: Implements a Java version of the RSLP algorithm.
- **retrievalengines**: Offers an interface to a retrieval engine and includes two implementations: a simple keyword-based engine that matches exact terms in the query with the video keywords and a ranked retrieval engine that uses the vector space model and query expansion as explained before.
- **visualizer**: Encloses the visualizer application.
- **extern packages**: Jazzy, JMF and Jena are the external APIs used by the system.
- **spellchecker**: Offers a Façade [Gamma et al., 1995] to the Jazzy API for spell checking.

The architecture uses the following design patterns from the GoF book [Gamma et al., 1995].

3.1. Façade

The Facade pattern provides a unified interface to a set of interfaces of a subsystem [Gamma et al., 1995]. We used this pattern to encapsulate the Jazzy and Jena external packages (Figure 4). The

[6]http://java.sun.com/products/java-media/jmf/

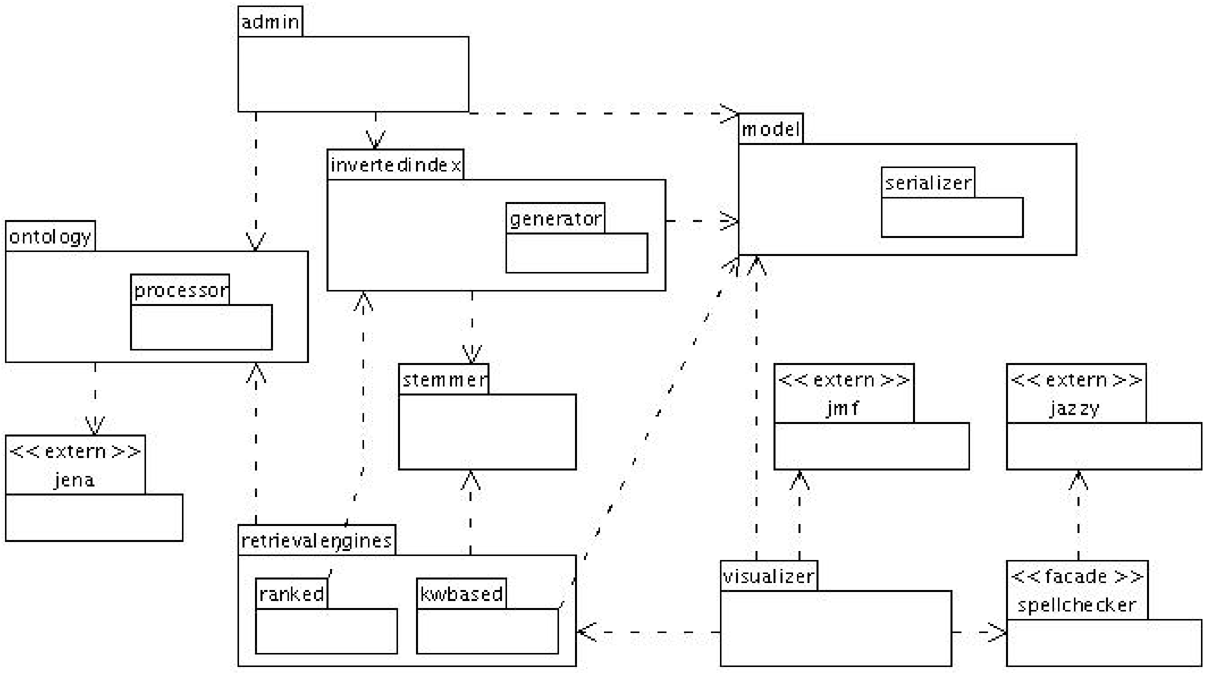


Figure 3: ONAIR arquitecture.

model package also offers a `Model` class that exposes all of its functionality to the other packages.

3.2. Strategy

The Strategy pattern allows to define a family of algorithms [Gamma et al., 1995]. We used it to switch between options of retrieval. The **retrievalengines** package contains the `Engine` and `Query` interfaces specifying that any `Engine` implementation must respond to a $retrieve$ method, receiving a query string and a boolean specifying whether expansion should be done or not. The `Query` interface enforces a $load$ method, which loads a string query into a `Query` object and an $expand$ method, that expands the query to add related terms, for instance. If an engine does not implement expansion, it should throw an `UnsupportedExpansionException`.

Two strategies were originally implemented: (1) a simple keyword based retrieval (**kwbased** package) and (2) a ranked retrieval, based on the vector space model (**ranked** package).

3.3. Template Method

The Template Method defines the skeleton of an algorithm deferring some steps to subclasses [Gamma et al., 1995]. The similarity between two terms is computed by weighting two factors: one based on the class hierarchy and the terms frequency and the other based in the density of properties relating the terms. Depending whether we use the inference engine or the ontology data structure, some computations differ.

We used the Template Method pattern in the abstract class `OntologyWrapper`, which is responsible for computing similarity (Figure 4). It delegates frequency computing, common parent classes, and property extraction to subclasses (`OntologyWrapperJena` and `OntologyWrapperED`).

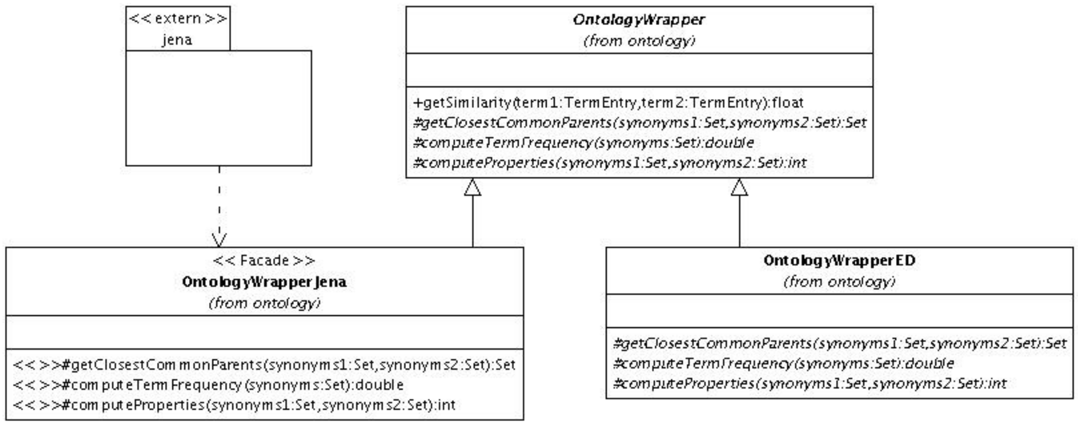

<< extern >>
jena
OntologyWrapper
(from ontology)
+getSimilarity(term1:TermEntry,term2:TermEntry):float
#getClosestCommonParents(synonyms1:Set,synonyms2:Set):Set
#computeTermFrequency(synonyms:Set):double
#computeProperties(synonyms1:Set,synonyms2:Set):int
<< Facade >>
OntologyWrapperJena
(from ontology)
<< >>#getClosestCommonParents(synonyms1:Set,synonyms2:Set):Set
<< >>#computeTermFrequency(synonyms:Set):double
<< >>#computeProperties(synonyms1:Set,synonyms2:Set):int
OntologyWrapperED
(from ontology)
#getClosestCommonParents(synonyms1:Set,synonyms2:Set):Set
#computeTermFrequency(synonyms:Set):double
#computeProperties(synonyms1:Set,synonyms2:Set):int

Figure 4: Ontology package showing the use of the Template Method Pattern.

3.4. Mediator

In the **model** package, we needed to represent the relations between video clips and keywords, and between videos and resources. These relations are complex and we tried to keep them separated, so we used the Mediator pattern [Gamma et al., 1995]. In Figure 5, we show how we modeled this interaction.

3.5. Singleton

Both in the administrative application and in the visualizer application, the class `Model` has only one instance, representing the current model being modified or queried. Given that, we used the Singleton pattern [Gamma et al., 1995]. This pattern also helps allowing `Model` to be accessed from anywhere in the system.

4. Using ONAIR with the OOP and Patterns Course

In 2003, the Institute of Mathematics and Statistics at University of São Paulo offered an advanced course on object-oriented design and development presented by Joseph Yoder, software consultant from Refactory Inc [7].

4.1. Description

The course is almost twelve hours long and covers three main topics: (1) Architecture and Design of Adaptive Object Models; (2) Design Patterns (Java and C# edition) and; (3) Refactoring Principles. This course was filmed, digitalized and made available in the Web to the community interested in these themes (`http://eclipse.ime.usp.br/cursos/OO/yoder.html`).

Simply making the videos available does not make attractive for students to exploit their contents. It is not expected (neither realistic) to have students watching it from beginning to end. We expect ONAIR will make it easier for the community to explore the videos, allowing to take more advantage of this course.

[7]`http://www.refactory.com`

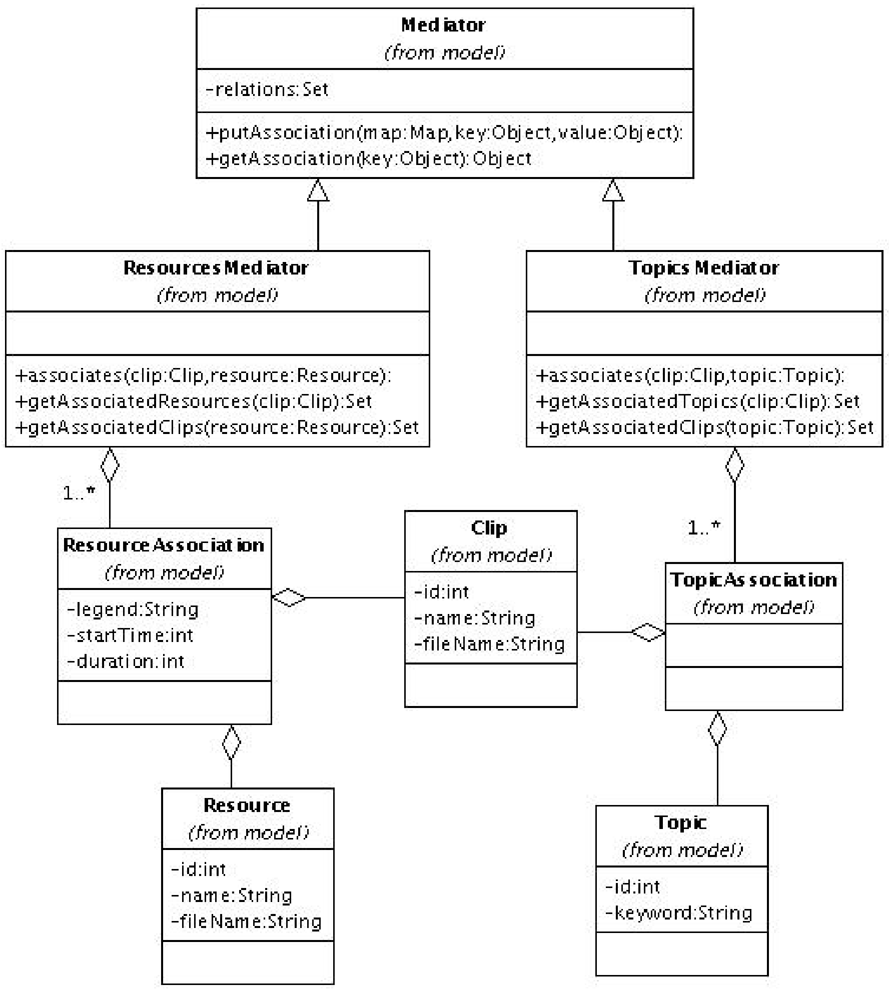


Figure 5: model package showing the use of the Mediator Pattern.

4.2. Ontology

To be able to use ONAIR in a domain we need to fragment the video into small clips, to select keywords for each clip and/or to transcript the speech. In addition, to exploit intelligent retrieval in ONAIR , we need to develop an ontology to describe the course domain.

In Figure 6 we show a sample fragment of an ontology that could help ONAIR to improve retrieval. It relates to a small subset of the design patterns [Gamma et al., 1995] described in the course. The ontology represents domain knowledge and is composed of a class hierarchy, class instances, and their relationships. For example, a `Design Pattern` can be a `Creational Pattern`, a `Behavioral Pattern` or a `Structural Pattern`. `Façade` is a structural pattern, and it is functionally related to `Mediator`. `owl:sameAs` is the OWL construction that represents class equivalence, so it joins `Wrapper` to `Adapter` and `Decorator` given that it is an alias of both of them.

For instance, this ontology makes the system capable of relating the term *“wrapper”* to the term *“decorator”*, so a query like: *“How can I implement a wrapper?”*, would retrieve some video fragments containing the term *“decorator”*, as they are probably rel-

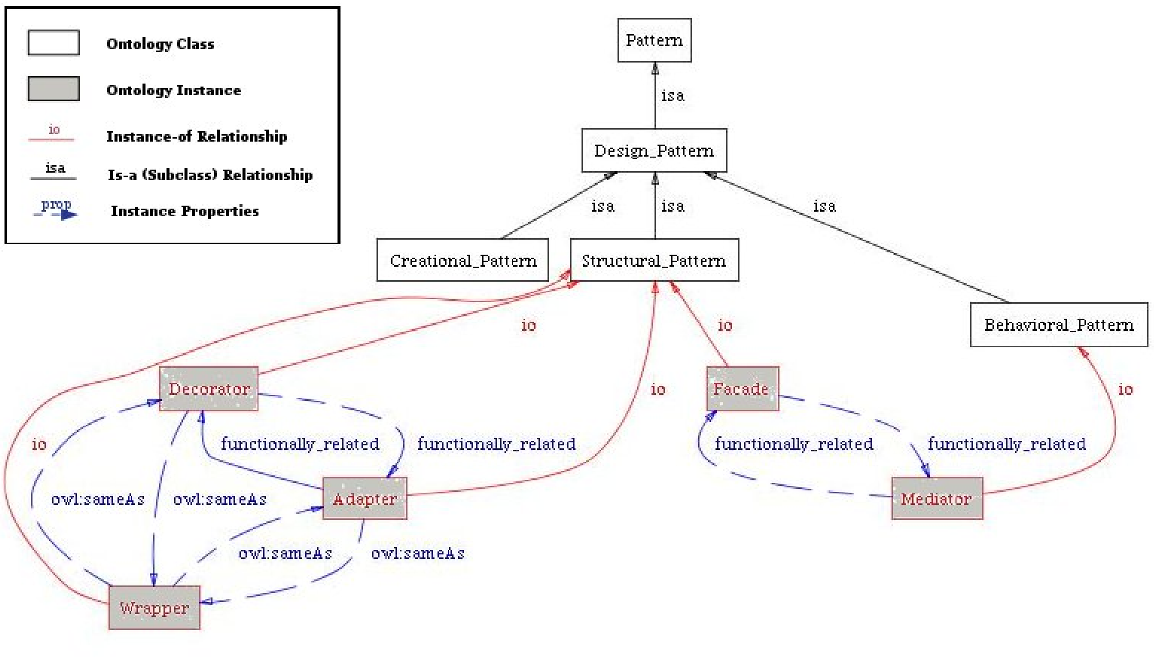


Figure 6: Ontology sample to improve retrieval.

evant for the query.

5. Expected Results

This is an ongoing work, we describe in this section the current state, the expected results, and future work.

We are currently making the transcriptions of the videos, as it is going to bring two main advantages: (1) it will be useful to subtitle the videos and (2) it allows us to identify the domain information that is required to be expressed in the ontology. Video subtitling is important to help non-native English speakers to take advantage of the course.

Developing the ontology about patterns will be useful for other knowledge-based systems, such as tutoring systems. [de Oliveira et al., 2004] proposed an Ontology for the specification of pattern systems (ONTO-PATTERN), this work serves as a conceptual base for our ontology, but as the approach is different it does not cover the domain the way the course does.

Finally, we plan to make the system available on the Web, which will allow us to evaluate the relevance of results and system usability. Also it will confirm the system flexibility to work in a different domain and a different language. ONAIR 's source code is currently available at `http://www.ime.usp.br/~cpaz/ontologies/arte`.

Acknowledgements: ONAIR development was supported by CAPES. Renata Wassermann is partially supported by the CNPq grant PQ 304486/2004-3. This work has been supported by CNPq project 55.0222/2003-0.

We would like to thank the Art Historician Paula Braga for his expert support through ONAIR 's development.

References

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley Longman.

[de Oliveira et al., 2004] de Oliveira, I. R., de Sousa Neto, R., and Girardi, R. (2004). Uma ontologia para a especificação de sistemas de padrões. In *Proceedings of the Fourth Latin American Conference on Pattern Languages of Programming*, Ceará, Brazil.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, R. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Adisson Wesley.

[Gennari et al., 2003] Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., Noy, N., and S.Tu (2003). The evolution of Protégé–2000: An environment for knowledge–based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220.

[Jackson and Moulinier, 2002] Jackson, P. and Moulinier, I. (2002). *Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization*. John Benjamins Publishing Co.

[Knublauch et al., 2004] Knublauch, H., Musen, M., and Rector, A. (2004). Editing description logics ontologies with the Protégé OWL plugin. In *International Workshop on Description Logics*, Whistler, BC, Canada.

[Lin, 1998] Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, pages 296–304, San Francisco, USA. Morgan Kaufmann Publishers Inc., 1998.

[McBride, 2001] McBride, B. (2001). Jena: Implementing the rdf model and syntax specification. In *Proceedings of the Second International Workshop on the Semantic Web*, Hong Kong, China.

[McGuinness and van Harmelen, 2004] McGuinness, D. and van Harmelen, F. (2004). OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[Orengo and Huyck, 2001] Orengo, V. and Huyck, C. (2001). A stemming algorithm for the Portuguese language. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval(SPIRE) 2001*, pages 186–193. An implementation of the algorithm in C is available at: `http://www.cs.mdx.ac.uk/research/PhDArea/rslp/RSLP.htm`.

[Paz-Trillo et al., 2004] Paz-Trillo, C., Wassermann, R., and Braga, P. (2004). Using ontologies to retrieve video information. In Freitas, F., Stíckenschmidt, H., and Volz, R., editors, *Proceedings of the Workshop on Ontologies and their Applications*, pages 55–66, São Luís Maranhão, Brazil.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620.

[Ueda, 2002] Ueda, R. (2002). Ispell Dictionary for Brazilian Portuguese: br.ispell. Available at `http://www.ime.usp.br/˜ueda/br.ispell/`.

[van Rijsbergen, 1979] van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworths, 2nd edition.