In the past few years, the use of ontologies for creating more intelligent and effective applications has increased considerably. This growth is due to the fact that ontologies attempt to provide semantics to the data consumed by machines so that they can reason about this data. However, developing complex ontology-based applications is still difficult and time-consuming because the existing tools do not provide a simple and unified environment for developers. Most of these tools only provide data manipulation using RDF triples, complicating the development of applications that need to work with the object-oriented paradigm. Furthermore, tools that provide instances manipulation via object orientation do not support features such as manipulating ontologies, reasoning over rules or querying data with SPARQL. In this context, this work proposes a framework and a tool for supporting the efficient development of ontology-based applications through the integration of existing technologies. Furthermore, we also define a methodology to use this tool efficiently. In order to evaluate the benefits of our work, a controlled experiment with eight developers (unfamiliar with ontologies) was performed to compare the proposed tool, JOINT, with another one, Jastor/Jena, frequently used by the community. The results suggest that our tool helps novice developers to create ontology-based applications faster and with fewer errors in the code. In addition, a real educational application with 10 ontologies, more than 200 ontology concepts (classes) and more than a million triples is already using the proposed tool successfully.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Recently, ontologies have obtained quite a lot of attention in the computer science community. The term “ontology,” which has origins in philosophy, becomes a useful word in computer science for a new approach to knowledge representation about real-world entities. Ontologies offer a shared understanding of a particular domain and a formalization that allows its data to be interpretable by machines (Hepp, Leenheer, Moor, & Sure, 2007; Isotani, Mizoguchi, Inaba, & Ikeda, 2010). As a result, ontologies are not only applied as bases for the Semantic Web, but also in other areas of computing research and industry. For example, e-commerce applications use ontologies for parametric searches and heterogeneous systems integration (Das, Wu, & McGuinness, 2002). Another industry segment is the media system that has used this approach to do real-time data inference, delivering up-to-date content for its users (Kiryakov et al., 2010). In addition, several other fields use ontologies, such as medicine (Bard & Rhee, 2004), mobile devices (Cheyer & Gruber, 2010) and adaptive education (Bittencourt, Costa, Silva, & Soares, 2009; Bittencourt et al., 2006; Bittencourt & Costa, 2011; Isotani et al., 2013).

This dissemination is a consequence of the growing number of tools and software libraries that allow the development of ontology-based applications. Currently, more than 170 tools are listed at the semanticweb.org, a list that has been growing considerably in the past few years. Despite the high number of tools, not all of them aim to support the development of applications for the Semantic Web. Furthermore, the tools that offer this type of support do not provide it through a simple and unified environment; that is, they fail to offer common functionalities when developing applications based on ontologies, such as managing and querying ontologies, reasoning over rules, manipulating instances via an object oriented paradigm (in contrast to the manipulation of instances via triple RDF (Resource Description Framework), among others (Holanda, Bittencourt, Isotani, Elias, and Bandeira, 2012).

In this context, this work proposes a toolkit called JOINT that provides simplified development of ontologies through the object-oriented model. Moreover, the toolkit provides an integration of existing technologies and techniques to create a unified environment for developers of applications based on ontologies. JOINT provides services such as operations on ontologies, manipulating...
instances, SPARQL\(^1\) (Simple Protocol and RDF Query Language) queries, data inference over SWRL\(^2\) (Semantic Web Rule Language) rules, and so on. The proposed tool was evaluated through a controlled experiment, in which developers compared it with another tool, referred to as Jastor/Jena, freely available for the community. In addition, a real world educational application with 10 ontologies, more than 200 ontology concepts (classes) and 1 million triples is already using the proposed platform successfully.

This paper is organized as follows. Section 2 outlines the characteristics of each type of ontology programming. Section 3 describes the proposed tool/toolkit. Section 4 presents the performed experiment, evaluating the proposed work. Section 5 describes the real world educational application built with the proposed system and methodology. Section 6 presents some conclusions and future works.

2. Ontology programming

Manipulation of instances is an important step in the process of ontology-based application development. Currently, two main approaches have been used by ontology management systems: RDF triples and object oriented development. In the following subsections, the main distinctions and benefits of the aforementioned approaches are detailed.

2.1. RDF triples development

Most current APIs (Application Programming Interface) to implement ontology-based applications are still working with the development based on RDF triples (subject, predicate and object). Thus, application developers need to be aware of how ontology works in the RDF layer, in order to manipulate the data through each triple in application code.

When the developer desires to add a resource (subject), with several properties inherent to it, to an ontology, several lines of code representing each triple of the resource will be necessary. Each triple captures a single value of each property. Similarly, if the application removes this resource, several triples should be removed as well. For example, several lines of code are needed to create an instance Alice of the entity Person with the datatype property name “Alice” using the Sesame API, which works with the development based on RDF triples. Fig. 1 shows how to add this resource in the Sesame repository.

```java
... ValueFactory f = myRepository.getValueFactory();

// create some resources and literals to make statements out of URI alice = f.createURI("http://example.org/people/alice/"); URI name = f.createURI("http://example.org/ontology/name"); URI person = f.createURI("http://example.org/ontology/Person"); Literal alicesName = f.createLiteral("Alice");

RepositoryConnection con = myRepository.getConnection();

// add a Person Person alice = new Person();
alice.setName("Alice");

// add a Person to the repository con.addObject(alice);
...
```

Fig. 1. Example of a code from Sesame API.

2.2. Object-oriented development

Instead of RDF triples, object-oriented applications manipulate data at the object level, including the set of attributes and values that characterize each object. In this sense, a tool is necessary to “map” the object’s operations to the RDF triples infrastructure that works underneath. Some tools were created for handling instances in ontologies based on this paradigm. As a result, developers do not need to have a deep knowledge of the ontology representation language. An object in the code represents an instance in the ontology; their attributes are mapped to the properties of the instances and RDF classes become Classes in the programming language. Therefore, to add a resource to the ontology, the developer just needs to add the object.

In comparison with the Sesame example (Fig. 1), Fig. 2 shows the same resource added to the Sesame repository using Alibaba API, which allows development based on the object-oriented paradigm.

```java
...
ObjectConnection con = repository.getConnection();

// create a Person Person alice = new Person();
alice.setName("Alice");

// add a Person to the repository con.addObject(alice);
...
```

Fig. 2. Code example from Alibaba API.

3. Joint

Currently, there are several tools that manipulate ontology through the paradigm of object orientation (e.g. Jastor Szekely & Betz, 2009 and Elmo (Mika, 2007) instead of RDF triples. However, these tools provide only the manipulation of instances, which is only one step of the ontology-based development process. Therefore, when working with these tools, developers need to search for other tools to build semantic applications with the usual features (e.g. querying ontologies, handling instances, performing rules, etc.). To fill the gap of appropriate development tools to build and maintain ontology-based applications, this paper proposes the toolkit JOINT (Java Ontology Integrated Toolkit), which integrates several features that facilitate the development process of application-based ontologies.

Some steps of this process are quite common, either by characteristics inherent to the area of ontologies or the nature of application development in general, as shown in Fig. 3.

The following details both the steps and how the proposed system supports each:

- **Modeling**: Before starting the development of an application based on ontologies, it is necessary to first create these ontologies. There are several methodologies for how to build ontologies in the literature such as Ontology Development 101 (Noy & Mcguinness, 2001), AFM: Activity-First Method in Hozo, METHONTOLOGY (Fernandez, Gomez-Perez, & Juristo, 1997) and On-To-Knowledge Methodology (Sure & Studer, 2002). For practical reasons, the entire process of ontology engineering should be done in an ontology editor, such as Protégé Ontology Editor\(^3\) (Noy et al., 2001) or Hozo Ontology Editor. JOINT does not

\(^{1}\) SPARQL. Query Language is W3C recommendation since January 2008 – http://www.w3.org/TR/rdf-sparql-query/.

\(^{2}\) More information at: http://www.w3.org/Submission/SWRL/.

\(^{3}\) http://www.hozo.jp/.
support ontology modeling, but there is a JOINT plugin for Protégé to perform other services (e.g. generate Java code from ontologies).

- **Environment configuration**: The second step of the process is to configure the environment for development of the application. For instance, it is necessary to set up the appropriate tool for storing ontologies as RDF triples. The JOINT environment uses Sesame API (Buschmann, Henney, & Schmidt, 2007) and thereby has Sesame as a triple store tool by default. However, developers can use other triple store tools compatible with Sesame, such as OWLIM (Kiryakov, Ognyanov, & Manov, 2005) and Virtuoso. Next, developers and project managers must choose which development IDE will be used. Finally, developers need to set up libraries for the project, the JOINT Java library, in this case.

- **Application development/maintenance**: The first two steps refer to modeling and configuration managing of the application. Development begins in the third step, using JOINT to generate the ontology’s codes, which will then be added to the system design. Finally, implement all the features and functionality of the application, based on the generated model.

- **Application evolution**: While Requirements Management (RM) has been a hot topic in both research and industry, and there are many RM tools (Zainol & Mansoor, 2008; Zainol & Mansoor, 2011), those described in the literature do not support all phases of RM (Gumus & Ertas, 2004; Nielsen, 2006). However, the challenges lie in managing the changing requirements of deployed applications. That is, what is the impact of changing or adding a requirement to the application after it is built up and running? At least three aspects need to be considered: (1) The application model; (2) the database structure and usage; (3) and the application code. In order to deal with them, JOINT supports the management of functional requirements from the modeling layer to the system layer.

### 3.1. Systems requirements

The proposed tool JOINT has been developed to deal with the following list of requirements:

- **Repository handling**: The first service needed for ontology-based development that the tool provides is the remote management of (Sesame) repositories. This management is composed of creation and delete of repositories in a (Sesame) server;

- **Ontology persistence service**: The tool also provides users with ontology persistence service through repositories. OWL or RDF files can be added to a repository, which stores all data in binary files. Besides adding ontologies, this service allows users to delete and retrieve a given ontology from the repositories;

- **Instance handling**: The main service provided by the tool is the manipulation of ontology instances present in a repository. Note that this manipulation is done through the object orientation paradigm. This service is composed of the methods for creating, retrieving and removing instances from a repository;

- **Java code generation**: To improve instance manipulation following the object orientation paradigm, the automatic generation of Java code from ontologies is required. This feature allows developers to “map” ontologies in Java code, creating classes that represent entities and concepts in these ontologies. The final result of this feature is a Java library.

- **SPARQL queries**: When developing any application, it is very common to query the data stored by it. And it is no different for applications based on ontologies, for which developers need a service that can query RDF graphs. The proposed tool provides a service that offers queries on ontologies based on the SPARQL language;

- **Backup repositories**: For some industrial applications, developers often need to keep backups of a repository. This feature allows the creation of backup files, recovering these files into an empty repository and their copies between different repositories;

- **Ontology verification and validation**: A service that allows consistency checking of an ontology. In other words, this feature verifies if an ontology does not contain contradictions to ensure a unique interpretation for each concept;

- **Reasoning over rules**: This service allows the performing of SWRL rules present in the repository, creating new information through reasoning over these rules.

### 3.2. Architecture overview

The JOINT architecture is based on the layers pattern (Buschmann et al., 2007) where each layer uses only the services of the layer below (Fig. 4). JOINT provides three layers (modules) for users (mainly developers and knowledge engineers): (1) an API, for ontology-based application developers to implement functionalities; (2) a Desktop interface, for knowledge engineers unfamiliar with programming; and (3) plugins on (and for) external tools, to optimize the work of both users.

These layers can either operate on OWL files, or operate on the triple store tool. The following subsections will describe each one of these layers and their functionality.

#### 3.2.1. API layer

The proposed API was built on top of the Sesame API, as shown in Fig. 5. The methods required to connect the repository and data

---

manipulation (through RDF triples) are in this layer. Above the Sesame API, there are four components: operations on ontologies, reasoning modules, operations on repositories, and the KAO pattern. The “Ontology Operations” component is composed of functionalities performed on the ontologies (the ontology itself, not the instances). Among its features are the insertion and deletion of an ontology, the Java code generation from one or more OWL files and the ontology validation functions. The “Repository Operations” component brings together relevant services for manipulating Sesame repositories. This module allows the developer to operate in remote repositories, such operations can be: creating or removing a repository, clearing a repository (deleting all data present on it) and creating backup copies of a given repository. The next component to be detailed is the “Reasoner Module”. This component aims to infer new data in a repository by executing SWRL rules present on it. Once the user runs the reasoner, it will check if there are rules in the repository; if any, it will check for new data to be inferred from the execution of these discovered rules. As shown in Fig. 5, the last three discussed components are under the “Repository Facade” component, which is designed to provide unified access to these three components through the Facade design pattern (Gamma, Helm, Johnson, & Vlissides, 2004).

Last but not least comes the “Knowledge Access Object” (KAO) component, which is a persistent pattern similar to the Data Access Object (DAO), with the difference that the KAO not only works with data, but works with information from the ontologies. The KAO pattern aims to provide an abstraction of the persistence mechanism used, providing some specific operations (such as creation, retrieval and removal of instances, among others) without exposing details about connections to the repository. Thus, this pattern can isolate the persistence layer of the business layer. The KAO component has an abstract class called AbstractKAO, which is responsible for all the abstraction of the KAO pattern. It provides operations for manipulating instances (creating, removing and retrieving) and performing queries. The query methods of the abstract class are protected, that is, only a class that extends AbstractKAO can use these methods. This is intentional, so that the KAO pattern works properly.

Fig. 6 exemplifies the KAO pattern from the perspective of a user who wants to manipulate instances in a given ontology A. This user must create a concrete class (called OntologyAKAO) that extends AbstractKAO. This class has concrete methods of creation, retrieval and removal of instances inherited from the abstract class. If the user wants to make a query, s/he performs the query inside the class OntologyAKAO, thereby isolating the persistence component of the other application components.

3.2.2. Desktop GUI
JOINT provides a friendly desktop interface (developed in java) where users can do operations with repositories and ontologies, already described in Section 3.1. This interface is very important for non-programmer users. To start using this tool, users need to connect with a Sesame server. Then, they can create a repository or select an existing one. Thereafter, other services can be done, such as adding an ontology to the repository, clearing the repository and running SPARQL queries, among others. Fig. 7 shows the JOINT desktop interface with the feature “Backup of Repository”.

3.2.3. Plugins
JOINT also provides several plugins. For instance, in the NetBeans IDE development there is a plugin that has the same operations of the Desktop GUI. The purpose of building this plugins is to optimize the development time. Therefore, programmers can do all the operations on a single tool (NetBeans IDE with JOINT library and Plugin). After the plugin is installed on the IDE, a button appears in the upper left corner (Fig. 8) that, when clicked, requests the server address to start the application. The plugin appears in the right position, but the user can drag it to any position. Another toolkit plugin is the JOINT Protégé Ontology Editor Plugin (developed in Java with Protégé API). Once set up, the plugin can be accessed in Window > Tabs > JOINT.

4. Experiment
This section presents an experiment that analyzed the proposed tool and provided both quantitative and qualitative evaluations. The experiment also did a comparison between our tool (JOINT) and another tool available in the literature (Jestor/Jena). This comparison aimed to verify if the obtained results satisfy the initial proposal; in order words, if the proposed tool increased the efficiency of program’s development. The experiment was based on
the work of Wohlin and colleagues (Wohlin et al., 2000) and it included four steps: defining the problem, planning and then running the experiment, and analyzing the results.

4.1. Definition

The first step was to determine the problem to analyze. This work designed and executed a comparative analysis between the cost to implement a semantic application using the proposed tool and the cost to implement the same application using another tool named Jastor. The experiment was completed in 2012 within a university context. The experiment had eight participants. According to Nielsen (2006), five subjects are enough to validate the usability and quality of a tool/interface and its functionalities.

4.2. Planning

After defining the scenario, the next step was planning, which was divided into three main activities: (i) creation of control groups; (ii) specification of the application to be developed; and (iii) specification of the evaluation metrics. The experiment also evaluates the proposed tool and the Jastor tool together with Jena. The choice of these tools was based on two reasons. First, Jena is one of the most popular APIs for manipulating ontologies in Java. Second, Jastor is the main tool for working with ontologies in the object level with Jena. Thus, there was a need to integrate the two tools (Jastor and Jena) for this experiment.

The first planning activity is the creation of control groups. This experiment had eight undergraduate students of computer science and computer engineering courses. Several lessons were taught in order to equalize the knowledge of participants, thus increasing the experiment control. Some of the topics covered in these lessons included: (i) object-oriented programming and Java language; (ii) ontologies modeling using the Protégé environment; (iii) queries on ontologies using SPARQL language; and (iv) inference on ontologies using SWRL. It is important to highlight that none of the participants had built an ontology-based application previously.

The experiment environment was divided into four control groups, wherein each group had a pair of students. The first group of partners, named F5, used machines with an Intel Core i5 CPU processor and 4 GB of RAM. The second group was named F3, and its pair used machines with an Intel Core i3 CPU processor and 4 GB of RAM. Both group F5 and F3 used the proposed tool.

On the other hand, groups J5 and J3 used the Jastor/Jena tool and their machines were the same as F5 and F3, respectively. All machines in the experiment performed the operating system Windows 7 Professional 64-bits. Table 1 summarizes the experiment environment.

After creating the groups, the second activity was specification of the application to be developed by the participants. During the experiment, each group developed the same ontology-based application using its allocated tool. This application uses an adaptation of the Family.swrl.owl ontology, which is included in Protégé ontology libraries and aims to demonstrate the use of SWRL rules in ontologies about family relationships. The same Java project was given to all groups and contained the main method (Main) with features not implemented. The groups’ goal was to run the Main method. Therefore, the groups must implement the necessary code and infrastructure using the allocated tool. The Main method has common steps in the development of ontology-based applications, such as: (i) adding the ontology to the repository/database; (ii) manipulating instances of ontology; (iii) running the SWRL rules; and (iv) executing SPARQL queries.

The last activity of the experiment planning was the specification of evaluation metrics. As previously mentioned, the experiment focuses on the efficiency of the participants’ application development process. Therefore, some quantitative metrics were defined, such as: (i) development time, measured in hours; (ii) number of lines of actual implemented code. The commented lines and the blank lines were not counted in this case; (iii) performance, measured in milliseconds; (iv) memory usage, measured in Megabytes; and (v) number of errors encountered during development, which were Java exceptions thrown when a method of the tools was run incorrectly.

In order to have a subjective evaluation of the proposed tool, a questionnaire concerning each developer’s experience was given at the end of the experiment. The quantitative and subjective results were used to compare the tools. The questions were:

1. What did you think about the tool documentation?
2. What did you think about the tool setup?
3. What was the complexity level of the addition of the ontology into repository?
4. What was the complexity level of the manipulation of instances?
5. What was the complexity level of running the SWRL rules?
6. What was the complexity level of running the SPARQL queries?
7. What did you think about the tool overall?

4.3. Running

The experiment was carried out over a week. On the first day, the experiment was introduced to the participants. During this time, a general explanation was presented about the activities of the week, and this moment was the first contact between the participants and the concept of development of ontology-based applications. After the explanation, the pairs were formed and the tools were allocated to each pair (see Table 1). The links that present the documentation

<table>
<thead>
<tr>
<th>Group</th>
<th>Machine</th>
<th>Allocated Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Intel Core i5 and 4 GB of RAM</td>
<td>Proposed Tool</td>
</tr>
<tr>
<td>F3</td>
<td>Intel Core i3 and 4 GB of RAM</td>
<td>Proposed Tool</td>
</tr>
<tr>
<td>J5</td>
<td>Intel Core i5 and 4 GB of RAM</td>
<td>Jastor/Jena Tool</td>
</tr>
<tr>
<td>J3</td>
<td>Intel Core i3 and 4 GB of RAM</td>
<td>Jastor/Jena Tool</td>
</tr>
</tbody>
</table>

5 Available at: http://protege.cim3.net/file/pub/ontologies/family.swrl.owl/
family.swrl.owl.
of the tools were also provided. The first data was collected during the experiment. Each pair reported the development time and number of errors encountered. When a team could run the Main method without errors, their code was evaluated. After evaluation, the number of implemented code lines was collected. This count was done manually. On the other hand, the JConsole tool was used to collect the data related to performance and memory usage. For more valuable information about the two target tools, please see Table 2, which presents data related to quantitative metrics.

4.4. Analysis of results

In regard to development time, both pairs using the proposed tool completed the task much faster than the Jastor/Jena tool. On the one hand, the F3 pair finished the experiment in 6 h and the F5 pair in 7 h, both of which used the JOINT tool. On the other hand, the J3 pair finished in 18 h and J5 in 15 h, both of which used the Jastor/Jena tool. The development time is one of the most important factors to evaluate the efficiency of ontology-based application development. Ultimately, the proposed tool helped the developers in this issue.

The third column of Table 2 presents the number of code lines for each team. Although, not much different was found between the control groups and the experimental groups, the experimental group presented a more optimized code. This is due to the high level of abstraction JOINT provides. Then, two factors (performance and memory, column 4 and 5 of Table 2) were analyzed. Although, these factors are not directly related to efficiency in development, they are very important in the performance of the built application. Performance and memory usage were measured when the Main function of the program was running. The performance of the experimental groups was better if compared to the control groups. In other words, the application developed using the proposed tool presented a higher performance, particularly when using a less capable machine. The best performance was the F5 team (2.5 s) and the worst was J3 team (4.1 s). In regard to memory usage, the experimental groups were better because the applications developed with JOINT had significantly smaller memory usage if compared to the application of the two pairs who did not use it. The number of errors metric can be considered one of the reasons for the difference in development time, as the groups who used the proposed tool had four times less errors than the pairs who used Jastor/Jena.

Regarding qualitative analysis, the Table 3 presents the results of the questionnaire. The participants gave a standard grade on each question, where 1 was the lowest and 5 was the highest. For each question, 1 meant terrible and 5 meant great. According to the participants the proposed tool is quite easy to use while Jastor/Jena is still too complex for them.

5. Real world application

This section shows a real application using the proposed tool. The “UFAL Línguas” is an educational system for teaching languages. The system’s main goal is to personalize student learning, focusing on adapting the content according to the student’s knowledge. The educational system provides intelligent customization of learning experience, supports the activities developed by teachers and provides real-time involvement of all participants (teacher, student, coordinator, director and parents) in the process of teaching and learning. Nowadays, the system is available on the Web. Fig. 9 shows the system’s homepage. The development of the application was based on the common steps shown in Fig. 3. The following subsections detail the process.

5.1. Modeling

The first step is ontology modeling. Each ontology represents an important part of the system. The following provides a short description about these ontologies:

- **Domain ontology**: Describes the characteristics of the domain to be learned. It defines the educational resources and their dependence through a multidimensional knowledge vision. The ontology represents the required resources for a particular curriculum and the available resources that can be used for the application.
- **Pedagogical ontology**: Responsible for defining how the interaction can be conducted. It defines the visualization sequence of the educational resources.
- **Learner ontology**: Models information about the student interaction with the system, such as the average time of responses and the number of attempts to solve a given problem, among others.
- **User ontology**: Describes the user model. It is responsible for saving user information, such as name, phone number, and email, among others.
- **School ontology**: Represents information about the school organization. It defines the class, grades and courses provided by the school.
- **Teacher ontology**: Describes the teacher model and its interactions with the system, such as created activities, student evaluations, related courses and so on.

To illustrate, Fig. 10 shows a piece of the domain ontology representing educational resources. A resource could be an activity, content, evaluation resource (form, problem-based evaluation, etc.), question or a problem. Fig. 10 highlights the problems, mainly matching, multiple-choice or true/false problems. Each type contains specific options, such as match column options.

---

Table 2
Quantitative Data of the experiment.

<table>
<thead>
<tr>
<th>Group</th>
<th>Development time</th>
<th>Code Lines</th>
<th>Performance</th>
<th>Memory Usage</th>
<th># of Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>7 h</td>
<td>72</td>
<td>2584 ms</td>
<td>15.4 MB</td>
<td>3</td>
</tr>
<tr>
<td>F3</td>
<td>6 h</td>
<td>81</td>
<td>3757 ms</td>
<td>16.2 MB</td>
<td>1</td>
</tr>
<tr>
<td>J5</td>
<td>15 h</td>
<td>89</td>
<td>4070 ms</td>
<td>61.5 MB</td>
<td>11</td>
</tr>
<tr>
<td>J3</td>
<td>18 h</td>
<td>84</td>
<td>4144 ms</td>
<td>52.2 MB</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3
Qualitative data of the experiment.

<table>
<thead>
<tr>
<th>Question</th>
<th>Tool</th>
<th>Grade 1</th>
<th>Grade 2</th>
<th>Grade 3</th>
<th>Grade 4</th>
<th>Grade 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>Proposed</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Question 2</td>
<td>Jastor/Jena</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Question 3</td>
<td>Proposed</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Question 4</td>
<td>Jastor/Jena</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Question 5</td>
<td>Proposed</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Question 6</td>
<td>Jastor/Jena</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Question 7</td>
<td>Proposed</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

---

7 Available at: http://openjdk.java.net/tools/svc/jconsole/.

8 http://www.ufal.educa.nees.br.
multiple choice options and true/false options. This topic will be discussed further in the section on application evolution.

5.2. Environment configuration

The second step of the process is to configure the environment for development of the application. The application was developed using the OWLIM Lite version and was used along with the Sesame server, providing access to the OWLIM repository, remotely. Next, all ontologies were added to this repository. The system was developed using a NetBeans IDE, version 7.1. The JOINT Java library was added to the required libraries in the Java web project.

5.3. Application development/maintenance

The third step of the process is application development. To facilitate the development process, JOINT generates the codes of the ontologies, as described in Section 5.1, and generates a Java library. Fig. 11 shows a part of code generated by JOINT, which focuses the application’s resources.

The generated code represents the concepts and relationships described in the ontologies. Besides the methods for representing relationships, the standard methods GET and SET can be used to access the objects’ properties. Since the code was generated, the logic and functionalities were created based on this model. A persistence layer, using JOINT API and the KAO Pattern for each ontology used, was built during the development process. The application was then created and deployed. Fig. 12 shows an example of a content based on video and Fig. 13 shows a screen of the resolution of a multiple-choice problem. Today, the application is in use by hundreds of high school users. A SPARQL query performed on the repository shows that the number of triples has already surpassed the one million mark, totaling 1,095,168 triples.

5.4. Application evolution

The UFAL Línguas system was conceptualized by many stakeholders (professors, educators, developers, and experts) and it is still in the improvement stage. To illustrate how JOINT helps developers on RM tasks, we will describe an evolution scenario related to the requirement of problem types. Fig. 10 presents the various problem types that developers had to implement during the initial stages of the UFAL Línguas system. However, during the deployment of UFAL Línguas, the stakeholders included a new kind of problem: the fill-in-the-blank problem. Although JOINT does not support the modeling phase, it provides a plugin integrated with Protégé editor that helps domain engineering to easily make this change. This plugin provides automatic code generation and can automatically add this information in the domain ontology (resource) as shown in Fig. 14. Although this feature provides many
benefits, the system cannot work properly without the final implementation of the new problem type by the application developers.

JOINT also provides automatic configuration of Java projects on Netbeans. Furthermore Netbeans and Protégé plugins can be configured to work together. Fig. 15 presents an automatically generated code within a Netbeans project based on the changes made in the ontology. It is important to note that JOINT plugins help developers during the main phases of the development lifecycle. Thus, the proposed toolkit decreases development costs. Moreover, developers and domain engineers can work within an integrated environment via JOINT.

6. Conclusions and future works

This paper proposed a toolkit, called JOINT, to unify several features, such as API, Desktop GUI and plugins, necessary for the development of applications based on ontologies. JOINT was presented through descriptions of its architecture and services. To check its usefulness, an experiment was conducted to evaluate it, focusing on the efficiency of programmer development in comparison to another pair of tools. The experiment compared JOINT with Jastor and Jena tools. We used both quantitative (development time, lines of code, performance, memory and number of errors) and qualitative (interview based on questionnaire) measures to verify the benefits of JOINT. The results show that novice programmers that used JOINT can develop ontology-based applications faster and with fewer errors in the code. Furthermore, the applications based on JOINT had better performance, running faster and using less memory if compared with the same application developed using Jastor/Jena tools. The results of the questionnaire also reviews that participants that used JOINT did not have difficulties to use it while participants that used Jastor/Jena found quite complex to work with them.

Finally, as a proof of concept, we presented a real educational system developed using the JOINT environment. This system is called UFAL Línguas and supports language learning. Currently it is being used by hundreds of users, it has more than one million triples in its database and more than 200 concepts in its ontologies. Other published works also used JOINT to support their ontology-based development. Some examples are (da Silva et al., 2011; Ferreira et al., 2011; Holanda et al., 2012; Ataide, Brito, Pedro, Costa, & Bittencourt, 2011).

In Future work we will conduct experiments extending the evaluation to other tools, such as OWL2Java and ProtegeOWLAPI and with a greater number of developers. The SWRL algorithm will also be improved aiming a better performance and further features that can be added to the presented tool.

Acknowledgement

CNPq, W3C Brasil and Nic.Br provided support for this research. We also thank MeuTutor Soluções Educacionais for technical support of our work.

References


