STUDIA INFORMATICA 2010
Volume 31 Number 2A (89)

Krzysztof GOCZYŁA, Piotr PIOTROWSKI
Politechnika Gdańska, Katedra Inżynierii Oprogramowania

APPLICATION OF KNOWLEDGE VIEWS

Summary. This paper introduces the Knowledge View concept and its application in software engineering. The work presented is aimed at bringing knowledge engineering and Semantic Web technologies closer to software engineers and programmers. It is done by disguising knowledge bases as relational or object databases and applying patterns known to software engineers.

Keywords: knowledge base, ontology

ZASTOSOWANIE WIDOKÓW NA BAZĘ WIEDZY

Streszczenie. W artykule przedstawiono koncepcję widoków na bazę wiedzy i jej zastosowanie w inżynierii systemów. Praca ma na celu przybliżenie inżynierii wiedzy i technologii Semantic Web inżynierowi oprogramowania oraz programiście. Cel ten osiągany jest poprzez upodobnienie baz wiedzy do relacyjnych lub obiektowych baz danych oraz poprzez stosowanie wzorców znanych w inżynierii oprogramowania.

Słowa kluczowe: baza wiedzy, ontologia, inżynieria systemów

1. Introduction

There is much talk about Semantic Web, Semantic Web technologies, knowledge and knowledge engineering. However, the adoption of these technologies and ideas in the IT industry is still not satisfactory. One of the reasons is that software engineers are often not familiar with Semantic Web technologies. Even if they are familiar with them, using those technologies seems risky – some of the Semantic Web standards are actively developed, for example OWL [14], while others seem to be abandoned like SWRL [4], which has the status of a submission for over five years now, even though it is partially implemented by some tools like Pellet [9] or KAON2 [6]. Moreover there are problems with interoperability with

existing information systems. Certainly for the Semantic Web technologies to be widely adopted by the IT industry it is necessary to address those issues.

2. Problems with Semantic Web technologies

Let us take a closer look at the mentioned problems. The first one is that programmers and software engineers are not familiar with Semantic Web technologies. This issue can be addressed in a variety of ways. One of them is to provide simple, easy to learn APIs to knowledge bases. Making those APIs similar to widely known and commonly used technologies can help greatly in the process of acceptance.

Another problem is that Semantic Web standards are still relatively young and tools are often immature and not robust enough to be exploited in commercial applications. This poses a risk when using such technologies among industry. Establishing standards is an inherently slow process and cannot be hurried. Technologies mature if they are used and feedback from users is taken into account while creating new versions and functionalities. This is also a lengthy process. However, to lower the risk of using an immature technology one can provide interfaces compatible with existing technologies (e.g. with relational databases), so that old and new technologies (e.g. OWL-based knowledge bases) would become interchangeable. This approach allows a gradual replacement of older technologies with newer Semantic Web ones. Moreover, if there is a problem with new solutions, developers can always step back to the verified ones.

The third problem addressed here is the compatibility problem. RDF [7, 2] is the format of choice for the Semantic Web community and many Semantic Web tools are based on it. However, RDF is not the dominant format for storing information in the Internet. Therefore there is a need for Semantic Web tools to take advantage of the existing information stores, like relational database systems, XML databases, etc. Moreover many of existing information sources will never be converted to RDF, partly because it might be costly, but also because RDF is obviously not suitable for all purposes. Therefore, Semantic Web tools, even if they use RDF internally, need to be able to interface to legacy systems that use various information formats and use them as sources of information.

3. Knowledge enabled application architecture

Nowadays the dominant application architecture is a three layer one (see Fig. 1.).

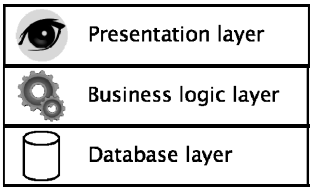

Fig. 1. The traditional application architecture
Rys. 1. Klasyczna architektura aplikacji

To seamlessly introduce semantic tools to applications without the need for software engineers to relearn everything they got familiar with so far, a similar architecture is proposed for knowledge enabled applications (see Fig. 2.).

From the business logic programmer's point of view the proposed architecture does not differ much from the traditional one and if we assume that the knowledge layer provides similar or even identical interfaces as the database layer, then it does not differ at all from what the programmer is familiar with. Moreover such an architecture allows replacing the database layer with knowledge layer in existing applications with little or no modifications. This can also work the other way round, that is if the tools used in the knowledge layer does not fulfil requirements, they can be easily replaced with traditional, database tools.

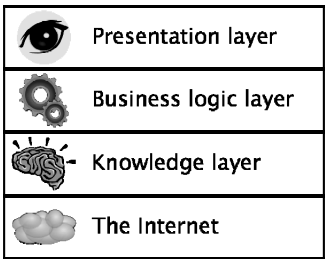

Fig. 2. The proposed application architecture
Rys. 2. Proponowana architektura aplikacji

The knowledge layer gathers information from various kinds of sources scattered throughout the Internet, performs reasoning over the gathered information, according to particular needs, and delivers inferred facts to the business logic. The information sources can be: databases, knowledge bases, web services, web pages, etc. In practice not only the resources found on the Internet can become information sources for the knowledge layer, but virtually anything that can be encoded in the form of unary and binary predicates. This proposition can also take advantage of the Service Oriented Architecture (SOA). The knowledge layer can use services as information sources and be itself exposed as a service or be embedded as a component of some service.

4. The Knowledge View concept

The knowledge layer is where the Knowledge Views are placed. Their architecture (see Fig. 3.) consists of three layers. The bottom layer are adaptors. They wrap underlying information sources and provide a common interface. This interface is used in all the tools of the Knowledge Views. The interface consists of the following methods: getConcepts, getRoles, getAttributes, getConceptInstances, getRoleInstances, getAttributeInstances. Additionally there is an optional set of methods for adding assertions to an information source. The getXInstances methods implement the "query by example" technique. The interface is as simple as possible to make it easy for a developer to write new adaptors for new information sources as needed. Anything that implements the interface is seen as an ABox of a knowledge base. This is the first manifestation of the view concept – ABoxes are virtual assertion stores or views.

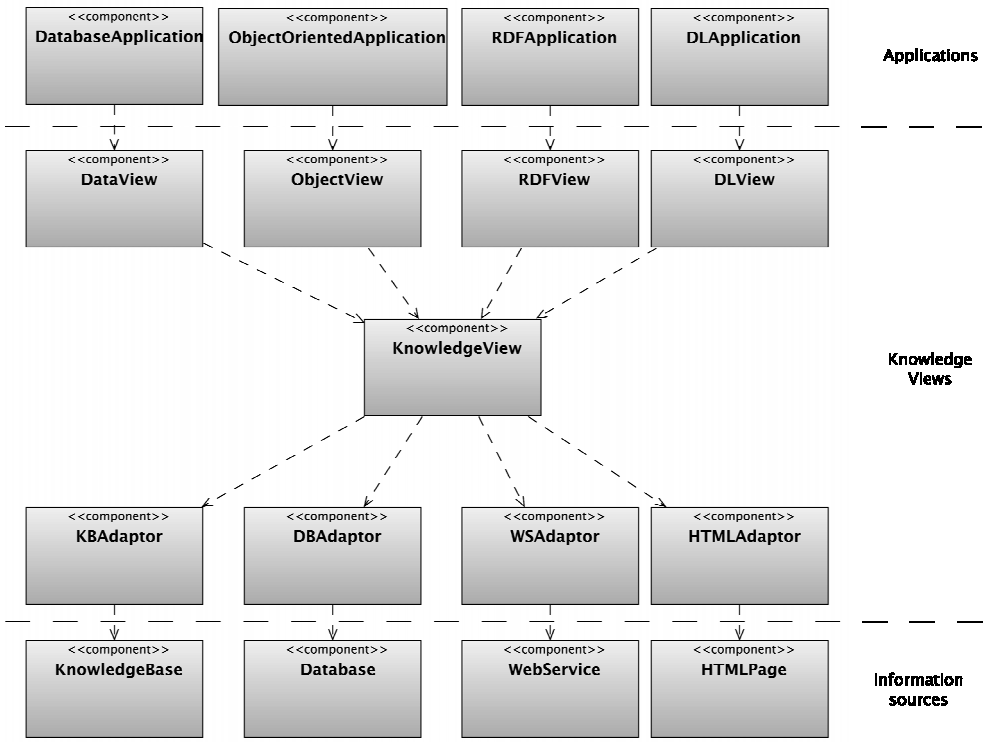

Fig. 3. The Knowledge Views' architecture
Rys. 3. Architektura widoków na bazę wiedzy

The middle layer in the Knowledge Views' architecture, called the Knowledge View, is a set of tools. Most of the tools implement the ABox interface. The tools provide means to merge ABoxes, manipulate data of an ABox, including changing values or data schema (terminology), and reason over ABoxes using description logic or Horn logic. There is a possibility to add new tools to enhance the Knowledge View's capabilities. This layer has

been modeled in the vision of typical Unix tools where each tool is simple, performs only one function, and a combination of tools allows a user to obtain complex results.

The third layer implements mappings of an ontology (DL) model to other models, like relational or object model. This layer provides interfaces for the business logic layer. Specific components provide different interfaces:

- Data View: SQL,
- Object View: OQL,
- RDF View: SPARQL,
- DL View: DIG.

This layer enables the business layer to see the knowledge layer as a relational database, object database, RDF store, etc. More views can be added if necessary.

5. Example

The proposed solution is illustrated below by an example. In this example there are two information sources. The first one is a malware database. It contains information about malware and their requirements. The second information source contains a network layout with information about what software is installed on which computer. Together those sources can be used to infer which computers are endangered by which malware – in this way an infection spread can be simulated. This example focuses on Knowledge Views' functioning rather than what is modeled therefore the process of spreading malware is much simplified.

5.1. Ontology

The ontology used in this example (called the "infection ontology") is not a simple file loaded to a knowledge base, but rather a set of modules (see Fig. 4.) that interact with each other to produce the illusion of a knowledge base with a single ontology loaded. The ontology modules are components well known from software engineering. Each provides a service through a public interface and has a private implementation.

The two information sources are seen as ABoxes (the malware ABox and the network ABox). Additionally some custom code that queries other ABoxes is also seen as an ABox (the run info ABox). The network ABox combined with the network rule set is seen as the network ontology. The malware, network and run info ABoxes together are seen as a single ABox (the infection ABox). Finally, the infection ABox combined with the infection TBox and the infection rule set is seen as a single ontology (the whole infection ontology) that is used by an application. One of the main advantages of configuring an ontology using some

components rather than creating one monolithic entity is the ability to change individual components or even the whole configuration without changing the resulting ontology or rather the resulting view of the ontology. It is also worth noting that the information sources are used directly without the need of rewriting their content, which is not always feasible. Moreover this approach supports semantic modularity, which can help to comprehend the created ontology to be used properly.

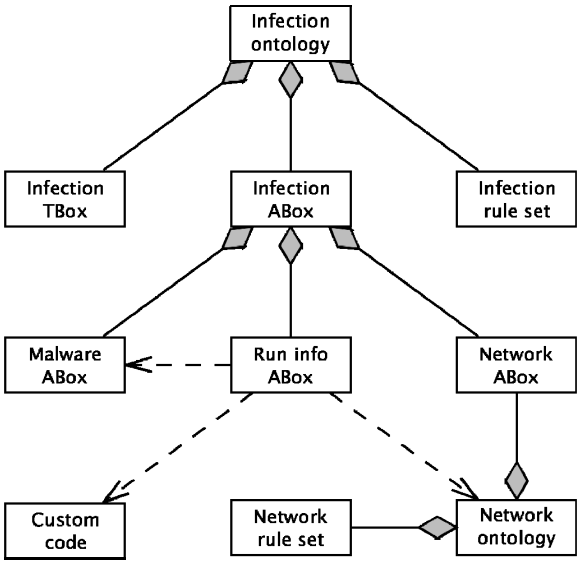

Fig. 4. The infection ontology
Rys. 4. Ontologia infekcji komputerowych

5.1.1. Terminology

The terminology (infection TBox) is provided to the system via an OWL file. It defines the following concepts: *Computer*, *Malware*, *Software*, and the following roles and attributes with appropriate ranges and domains:

- *canBrowse* (domain: *Computer*; range: *Computer*),
- *canDownloadFrom* (domain: *Computer*; range: *Computer*),
- *canRunOn* (domain: *Malware*; range: *Computer*),
- *canUploadTo* (domain: *Computer*; range: *Computer*),
- *description* (domain: *Malware*; range: *string*; functional),
- *endangeredBy* (domain: *Computer*; range: *Malware*),
- *hasInstalled* (domain: *Computer*; range: *Software*),
- *hasPayload* (domain: *Malware*; range: *Malware*),
- *ip* (domain: *Computer*; range: *string*; functional),
- *name* (domain: *Malware*; range *string*; functional),
- *requires* (domain: *Malware*; range: *Software*),
- *riskLevel* (domain: *Malware*; range: *int*; functional),

- *sourceOf* (domain: *Computer*; range: *Malware*).

In this example the terminology is used to define the data schema visible by the user rather than to perform reasoning. However, even such a simple terminology can be used to infer some new facts, not explicitly given. For example there is no need to state explicitly which individuals belong to the *Software* concept – this can be inferred based on the range of *hasInstalled* and *requires* roles.

5.1.2. Rules

In this example the reasoning process is mainly driven by the infection rule set:

$$
\begin{aligned}
&sourceOf(\text{internet}, m) \leftarrow Malware(m)\\
&endangeredBy(c, m) \leftarrow canRunOn(m, c) \wedge sourceOf(c, m)\\
&endangeredBy(c, m) \leftarrow canRunOn(m, c) \wedge canDownloadFrom(c, s)\\
&\qquad\qquad \wedge sourceOf(s, m)\\
&endangeredBy(c, m) \leftarrow canRunOn(m, c) \wedge canBrowse(c, s)\\
&\qquad\qquad \wedge sourceOf(s, m)\\
&sourceOf(s, m) \leftarrow hasPayload(n, m) \wedge endangeredBy(s, n)\\
&sourceOf(s, m) \leftarrow canUploadTo(c, s) \wedge sourceOf(c, m)
\end{aligned}
\tag{1}
$$

The rules specify which computers are endangered by malware and which can be used to distribute malware. The rules define how the malware propagates throughout the network. They can be easily extended to support other means of malware propagation, for example transmission via USB flash drives.

5.1.3. ABox

The third part of the infection ontology is the infection ABox. This ABox by itself only merges the three underlying ABoxes: the malware ABox, the network ABox and the run info ABox. It is introduced only to combine what is below it into a single ABox. That way the upper layer does not have to explicitly handle multiple sources, which would unnecessarily complicate the matters.

The malware ABox provides information about malware. In particular it provides assertions for: *Malware*, *description*, *hasPayload*, *name*, *requires* and *riskLevel*. This ABox is backed by a relational database with a schema conforming to the ERD in Fig. 5.

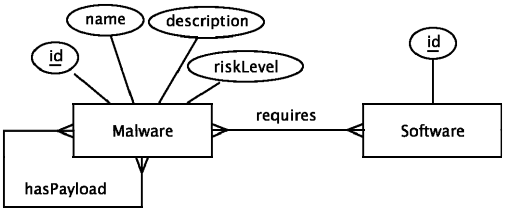

Fig. 5. Malware database ERD
Rys. 5. ERD bazy danych szkodliwych programów

The network ABox provides information about computers and connections between them. This ABox provides assertions for: *Computer*, *canBrowse*, *canDownloadFrom*, *canUpload-*

To, *hasInstalled* and *ip*. This ABox is backed by a relational database with a schema conforming to the ERD in Fig. 6.

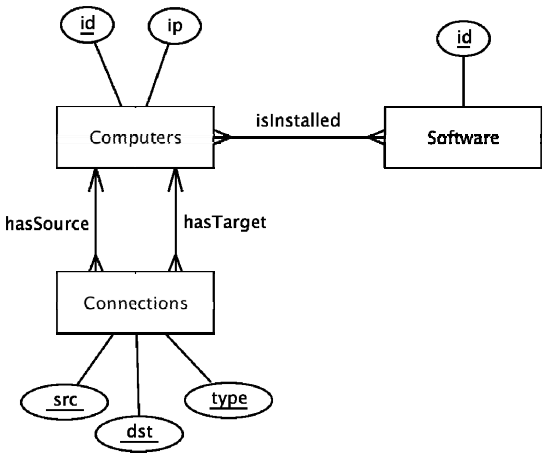

Fig. 6. Network database ERD
Rys. 6. ERD bazy danych sieci komputerowej

The third ABox, run info ABox, unlike the previous two, is not materialized in a database. Instead it is a piece of software that can handle queries for instances of the *canRunOn* role. A *Malware canRunOn* a *Computer* if all the requirements of the *Malware* are met by the software installed on the *Computer*. The requirements are retrieved from the malware ABox, while the software installed on a particular *Computer* are retrieved from the network ontology. The Knowledge View's reasoning capabilities are not used to confront the two retrieved sets, because the reasoning is performed under open world assumption (OWA). As a consequence of this assumption, during the reasoning process, the requirements of a given *Malware* are interpreted as known requirements with the possibility that there are also some unknown requirements. This makes it impossible to determine whether all requirements are met. The custom code, on the other hand, can query the malware ABox for requirements and assume that the results contain all requirements. Then they can be confronted with the software installed on the *Computer*.

The requirements may not always match the installed software: for example some malware might require Windows operating system, while some computer might have Windows Vista installed. Windows Vista is still a Windows operating system, therefore the requirement is met. However, the information that Windows Vista is Windows has to be given explicitly. Moreover Windows Vista Service Pack 2 is both Windows Vista and Windows operating system. To handle this case the network ontology is created (see Fig. 4.). This ontology combines the network ABox with the network rule set that consists of the following recursive rule:

$$hasInstalled(c, g) \leftarrow hasInstalled(c, s) \wedge \text{is}(s, g) \tag{2}$$

The instances of the *is* role act as a bridge between the network ABox and the malware ABox. These instances are stored in the network ABox to avoid creating yet another ABox for a single role, although such a configuration is also possible.

5.2. Application

An application does not need to use the ABox interface directly. If it had, then using the Knowledge Views would be similar to using libraries like Jena [5], OWL API [3, 13] or RDF2Go [12]. Each of these libraries provide some API for manipulating RDF or OWL [8] files. These APIs, even though similar, are incompatible with each other – there is no standard that regulates this. Some of the libraries do provide interface for issuing SPARQL [11], however SPARQL gives only read only access. In the Knowledge Views' case the application can use any of the views: Data View, Object View, RDF View, DL View depending on the application profile and programmer's knowledge. Unlike the mentioned libraries those views provide interfaces that are standardised (SQL, OQL, SPARQL) and it is the application developer who chooses which to use.

In this example let us focus on the Object View interface. It is similar to Java Persistence API, therefore it should be easy to learn for any enterprise Java programmer. It is also possible to use Data View with Java Persistence API on top of it, which would increase the compatibility at the cost of an additional layer. As the first step the programmer has to define classes corresponding to concepts in the source ontology. Those classes have fields that correspond to roles and attributes from the ontology. The code of a sample class corresponding to the *Computer* concept follows:

```
@Concept("Computer")
public class ComputerInfo {
    private String uri;
    private String ip;
    private Collection<Malware> endangeredBy;

    public String getUri() {
        return uri;
    }
    @Uri
    public void setUri(String uri) {
        this.uri = uri;
    }
    public String getIp() {
        return ip;
    }
    @Attribute("ip")
    public void setIp(String ip) {
        this.ip = ip;
    }
    public Collection<Malware> getEndangeredBy() {
        return endangeredBy;
    }
    @Role(uri="endangeredBy", range=Malware.class)
    public void setEndangeredBy(
            Collection<Malware> endangeredBy) {
```

```
            this.endangeredBy = endangeredBy;
        }
    }
```

The `@Concept`, `@Role` and `@Attribute` annotations define mappings between the ontology and the object model. The `@Uri` marks the setter for a field representing an individual's id. The `Malware` class referenced from this field also has to be similarly annotated. Automatic generation of classes from terminology is also possible.

To use the annotated classes in an application and query the ontology an `ObjectViewManager` has to be created:

```
ObjectViewManager ovm = new ObjectViewManager(infectionOntology,
        new Class<?>[] {ComputerInfo.class});
```

Next the `ObjectViewManager` is used to issue queries. To retrieve information about a computer with a given IP, the following code is required:

```
ComputerInfo ci = ovm.executeQuery(
    "SELECT c FROM ComputerInfo c WHERE c.ip = '192.168.1.123'",
    ComputerInfo.class);
```

6. Summary

The presented example shows how the Knowledge Views handle the stated problems with Semantic Web technologies. The ease of use is ensured by providing interfaces the programmers are familiar with – knowledge engineers create knowledge bases, but software engineers use them. Compatibility with existing information sources is ensured by a layer of wrappers. The third problem of lowering the risk of introducing Semantic Web technologies by providing standard interfaces is only mentioned. Additionally the example presents an approach to creating ontologies that treats them as software modules. Such an approach brings knowledge engineering closer to software engineering. This may cause software engineers to more readily use solutions they understand. Moreover, as a side effect, this solutions allows “mixing” different kinds of reasoning in addition to “mixing” technologies.

Initial versions of Data Views and Object Views have already been used as parts of the Knowledge Management System (KMS) [1] developed for the PIPS (Personalised Information Platform for Life and Health Services [10]) project funded by the European Commission under a Framework Programme 6 call. The purpose of KMS was to manage knowledge from a knowledge base to support decision making health care business processes. The Data and Object Views were the intermediate layers between the knowledge base and a web portal. The approach turned out to be very useful so we decided to proceed with further theoretical investigation and implementation of other adaptors and views. The

current and further development of both the concept and the software, influenced by the experience gained from this project, will be reported soon.

BIBLIOGRAPHY

1. Goczyła K., Waloszek W., Zawadzka T., Zawadzki M.: Inference mechanisms for knowledge management system in e-health environment. Software engineering: evolution and emerging technologies. IOS Press 2005, pp. 418÷423.
2. Hayes P.: RDF Semantics. W3C 2004, http://www.w3.org/TR/rdf-mt/.
3. Horridge M., Bechhofer S., Noppens O.: Igniting the OWL 1.1 Touch Paper: The OWL API. 3rd OWL Experienced and Directions Workshop, Innsbruck 2007.
4. Horrocks I., Patel-Schneider P., Boley H., Tabet S., Grosof B., Dean M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C 2004, http://www.w3.org/Submission/SWRL/.
5. Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/.
6. KAON2, http://kaon2.semanticweb.org/.
7. Manola F., Miller E.: RDF Primer. W3C 2004, http://www.w3.org/TR/rdf-primer/.
8. McGuinness D., van Harmelen F.: OWL Web Ontology Language Overview. W3C 2004, http://www.w3.org/TR/owl-features/.
9. Pellet: The Open Source OWL DL Reasoner, http://clarkparsia.com/pellet.
10. Personalised information platform for life and heath services, http://www.pips.eu.org/.
11. Prud'hommeaux E., Seaborne A.: SPARQL Query Language for RDF. W3C 2008, http://www.w3.org/TR/rdf-sparql-query/.
12. RDF2Go, http://rdf2go.semweb4j.org/.
13. The OWL API, http://owlapi.sourceforge.net/.
14. W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C 2009.

Recenzent: Dr inż. Sławomir Niedbała
Dr hab. Tadeusz Pankowski, prof. Uniwersytetu im. Adama Mickiewicza

Wpłynęło do Redakcji 27 stycznia 2010 r.

Omówienie

W artykule przedstawiono koncepcję widoków na bazę wiedzy, sugerowaną rolę takich widoków w typowej aplikacji wraz z propozycją architektury (patrz rys. 2.) aplikacji korzystającej z bazy wiedzy poprzez widoki na bazę wiedzy. Zaprezentowana jest architektura widoków na bazę wiedzy (patrz rys. 3.) ze wskazaniem możliwości rozszerzania o dodatkowe elementy. Trzon artykułu stanowi szczegółowe omówienie przykładowego wykorzystania widoków na bazę wiedzy. Pokazane jest, jak tworzona jest ontologia, która wykorzystuje możliwości modularyzacji widoków oraz jak korzysta się z już gotowej bazy wiedzy zasłoniętej obiektowym widokiem na bazę wiedzy. Szczególna uwaga zwrócona jest na łatwość wykorzystania widoków oraz ich kompatybilność z istniejącymi źródłami informacji. Ponadto pokazano, w jaki sposób widoki na bazę wiedzy łączą inżynierię wiedzy z inżynierią oprogramowania poprzez wykorzystanie w inżynierii wiedzy sprawdzonych wzorców z inżynierii oprogramowania. Umożliwiło to między innymi łączenie różnych metod wnioskowania, np. wnioskowanie w świecie otwartym z wnioskowaniem w świecie zamkniętym, w jednej modularnej ontologii.

Addresses

Krzysztof GOCZYŁA: Politechnika Gdańska, Katedra Inżynierii Oprogramowania, ul. Narutowicza 11/12, 80-233 Gdańsk, Polska, kris@eti.pg.gda.pl .

Piotr PIOTROWSKI: Politechnika Gdańska, Katedra Inżynierii Oprogramowania, ul. Narutowicza 11/23, 80-233 Gdańsk, Polska, piopio@eti.pg.gda.pl .