STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Ewa PŁUCIENNIK-PSOTA
Silesian University of Technology, Institute of Computer Science

OBJECT RELATIONAL INTERFACES SURVEY

Summary. Object relational interface, within the meaning of a tool for mapping of a relational database to a set of objects, is an essential element of modern applications co-operating with relational databases. Nowadays at least a few dozen of this type tools exists. For one programming language sometimes there are even a dozen or more to choose from. So the choice is broad. The article presents a review of such tools and proposes basic evaluation criteria for their suitability.

Keywords: impedance mismatch, object relational mapping, object application, relational database

INTERFEJSY OBIEKTOWO RELACYJNE – PRZEGLĄD

Streszczenie. Interfejs obiektowo relacyjny, w rozumieniu narzędzia pozwalającego na mapowanie relacyjnej bazy danych na zbiór obiektów, jest niezbędnym elementem współczesnych aplikacji współpracujących z bazą danych. Obecnie funkcjonuje przynajmniej kilkadziesiąt tego typu narzędzi. Dla danego języka programowania czasami mamy ich do wyboru nawet kilkanaście. Wybór jest więc szeroki. Artykuł przedstawia przegląd takich interfejsów oraz proponuje podstawowe kryteria oceny ich przydatności.

Słowa kluczowe: aplikacja obiektowa, mapowanie obiektowo relacyjne, niezgodność impedancji, relacyjna baza danych

1. Introduction

Nowadays most of applications cooperate with databases. This cooperation mostly takes place at the meeting point of two realms: object and relational. The object realm encompasses applications developed using object programming language (Java, C++, C#, Python, etc.). Relation databases constitute the relation realm. Both of these realms have different paradigms which lead to a very adverse effect called object-relational impedance mismatch[1]. Of course one can say: "why we do not use object databases instead of relational ones? We avoid then this discrepancy". This is true, but for now object databases are not able to threaten relational databases position on the market. Relational model is standardized and well known. Its strength lies first of all in general query language - SQL. Object databases, despite the fact that from early 1990s attempts for creating such language were undertaken [1], do not have general and standard query language. So, as for now, in most of IT projects we have object applications and relational data.

The term object-oriented programming is known from the early 1960s and object programming languages have become widely used in the early 1990s. First successful and popular solution for a cooperation with relational databases was application programming interface Open DataBase Connectivity (ODBC 1992) and Java DataBase Connectivity (JDBC 1996). In this technique SQL query is sent from object application to a relational database, executed and its results are returned usually in a form of a RecordSet – design pattern which has the same structure as SQL query results and can be processed by other system's components [2]. There exists many techniques for embedding SQL in the application code, but none is all-natural and moreover, when programmer does not have the good SQL knowledge, queries can be ineffective [2].

Nowadays object-relational interfaces (within the meaning of a tool for mapping of a relational database to a set of objects - ORM) are the most popular solution for object application and relational database cooperation. They constitute additional layer which mediates between an object application (and its classes that need to be persisted) and relational database access mechanism.

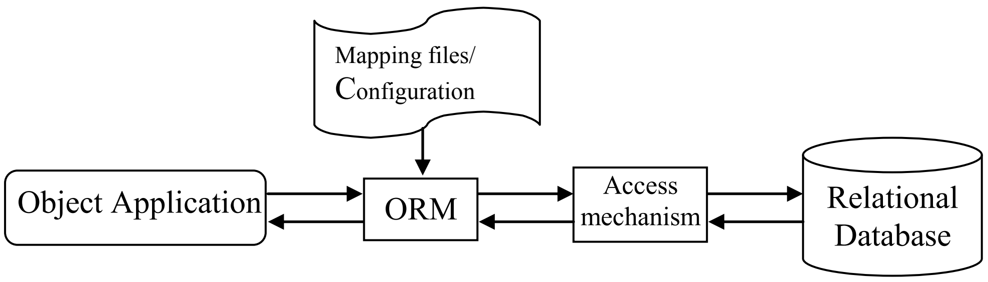

Fig. 1. Scheme of ORM functioning
Rys. 1. Schemat działania ORM

Access mechanism can be ODBC/JDBC or any other like, for example Ruby/DBI (Direct database access layer for Ruby), OCI (Oracle Call Interface), etc. Information about mapping objects into database tables are stored in external XML files or in form of annotations placed

[1] Term "impedance mismatch" comes from electrical engineering and stands for resistance mismatch of source and receiver, which causes loss of power.

directly in classes code. Annotations are shorter and easier to use as programmer has all information about mapping in one class file. On the other hand XML mapping files can be modified without recompiling the code. There are many ORM tools features which should be considered before developer or project manager will decide which will be the best choice. The aim of this article is to propose ORM evaluation criteria which should help to choose ORM meeting the criteria appropriate for a given project or application.

2. History of ORM development

In 1989 The Object People was founded by Carleton University Professors John Pugh, Wilf Lalonde, and Paul White [3]. In the early 90's first object-relational mapper TopLink for SmallTalk emerged. In 1996 TopLink for Java 1.0 was built with the internal code name "Wallace and Grommit" (Grommit was the "Mapping Workbench"). In 1999 TopLink was integrated with a number of J2EE application servers (for example WebLogic and WebSphere) to support EJB (Enterprise Java Beans) container-managed persistence. In 2000 TopLink was sold to WebGain and in 2002 acquired by Oracle [4]. In 2001 arises Hibernate created by Gavin King. The main goal of Hibernate creators was to offer better persistence capabilities and simplicity than offered by EJB 2 [5]. In 2003 Hibernate2 become "de facto" standard for persistence in Java [5]. Hibernate 3 (2005) became an inspiration for EJB 3 [6, 7]. In EJB 2.1 entity beans were heavyweight and dependable of application server and the Java EE runtime. In EJB 3 entity beans became plain old Java lightweight objects (POJO) thanks to the Java Persistence API[2] specification [7]. JPA is now standard for object persistence in Java – specified in JSR 317 (JPA 2.0) [9]. In 2007 Oracle starts cooperation with Eclipse Foundation. Former TopLink developers get involved in Eclipse Persistence Services Project (in short called EclipseLink) - extensible framework that will enable Java developers to interact with relational databases (based on JPA), XML, and Enterprise Information Systems (EIS) [10]. In 2008 Sun, the lead for the Java(TM) Persistence API (JPA) 2.0, has selected the EclipseLink project as the reference implementation [11]. In Oracle TopLink 11g, TopLink Essentials has been replaced with EclipseLink JPA [8].

Now JPA has many implementations – mentioned above including Hibernate 3.5 and higher (latest version 4.1.0 was released at the begging of 2012) [12] and also Open JPA [13] or DataNucleus (formerly JPOX) [14]. OpenJPA was created based on SolarMetric's Kodo product. SolarMetric was purchased by BEA Systems SolarMetric in November of 2005. BEA Systems donated the bulk of the code to the Apache Software Foundation and the result

[2] It is an API for creating, removing and querying across lightweight Java objects and can be used both within a compliant EJB 3.0 Container and a standard Java SE 5 environment [8].

was OpenJPA [15]. JPA implementations among themselves with their own specific annotations. For example, Hibernate *ForeignKey* annotation allows to define foreign key name. List of specific annotation for particular JPA implementation can be found in its documentation.

But Java is not the only programming language. On the Internet one can find some programming language popularity indices[3]. They using different methods of popularity measure: book sales, web searches, line of codes in GNU/Linux distribution, job advertisements, etc. After analyzing such information one can see that, besides Java mostly used programming languages are "C family" (C#, C++, C, Objective-C), Ruby, PHP, Python, etc. When it comes to programming .NET framework should not be forgotten – main Java and its virtual machine competitor. In 2003 the NHibernate (Hibernate for .NET) project was started by Paul Hatcher, Mike Doerfler and Sergei Koshcheyev [16]. NHibernate 3.0 (2010, .NET 3.5) was the first version integrating LINQ (Language INtegrated Query) support. The newest version of NHibernate 3.2.0 was released in 2011.

LINQ, introduced in 2007, is the result of research carried out in Microsoft Research in Cambridge and Redmond. Microsoft's aim was to provide a solution for the object-relational mapping and create simple and universal tool for the interaction between objects and data sources. LINQ eventually become a general-purpose language-integrated querying toolset. LINQ to SQL uses POCO (Plain Old CLR[4] Object) objects to represent application data (the entities) [16]. LINQ to Entities was designed to work with the ADO.NET Entity Framework (ORM framework for the .NET Framework) [17]. It should be noted that the first Microsoft's attempt at object-relational mapping was ObjectSpaces (2001) – a set of data access APIs which allowed to treat data as objects, independent of the underlying data store. This project was abandoned in 2005 [17]. In the same year at Microsoft's Professional Developers Conference (PDC) early versions of the EDM (Entity Data Model) Designer and XML mapping files was presented. EDM's basic elements – "Incremental Approach to an Object - Relational Solution" was patented by Microsoft on March 8, 2007 (U.S. Patent and Trademark Office, patent no. 20070055692). The first Community Technical Preview (CTP) of Entity Framework was released in mid 2006. At the end of August 2007 EF Beta 2 and EDM Designer CTP 1 was released, followed by EF Beta 3 and EDM Designer CTP 2 in early December 2007. The Visual Studio 2008 included updates to EF and the EDM Designer [18]. The newest release of EF (version 4.2) has appeared in 2011.

This is the history of ORM development by two main competitors on IT market. Of course other players did not fall behind. Most existing applications cooperate with database,

[3] For example: The Transparent Language Popularity Index http://lang-index.sourceforge.net/, Programming Language Popularity http://langpop.com/, TIOBE Programming Community Index for January 2012 http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[4] The Common Language Runtime - the virtual machine component of Microsoft's .NET framework.

so most of object programming languages have ORM tools, created since XXI century beginning, own or third party. For example, ActiveRecord ORM is the Ruby implementation of the Active Record pattern (Active Record is an object which wraps a row in table or database view and provides methods for insert, update or delete data [2]). There are also other ORM for Ruby like Rhino or Massive Record for HBase. Phyton programmers can use for example SQLAlchemy, SQLObject or DJango. For application written using scripting languages like PHP or JavaScript developer can choose among LightOrm, Propel, Doctrine, KOHANA, CakePHP for PHP or JazzRecord, Impel, ActiveJS for JavaScript. Interesting proposition for Java and .NET is MyBatis[5] (formerly IBatis) – ORM which, instead relational tables, maps SQL statements results into objects. .NET developers can also choose KyneticORM, OpenAccess ORM or ORM.NET. C++ developers can use, for example ODB or QDjango.

It is impossible to mention all ORM solutions. There exists at least several dozen popular ORMs. There are many ORM tools (sometimes a dozen or so) even for one programming language so choice is wide and should be make thoughtfully.

3. Assessment Criteria

When one needs to use ORM tools a few things should be considered. It is very important to remember that the main goal of using ORM is not application performance improvement but faster and simpler application creation. ORM is an additional application layer and can be considered as an overlay for ODBC/JDBC – using ODBC/JDBC directly will be faster. Of course decline in performance is not desirable so ORM has to deliver some performance improvement mechanisms.

First one is caching mechanism. Most databases offers such mechanism, but still query results have to be transmitted through network – cache resides on database server. We can avoid such transfer using client cache. Some ORMs offer first (L1) and second level (L2) cache for data (for objects and query results). L1 is provided by ORM and it is related with a single session, which can be considered as a logical transaction (single client connection to the database). L2 cache is external, so ORM should provide only interface to use it. Of course it cannot be said that using cache is always a good solution. Before user decides to use it, he or she needs to analyze queries frequency and types and also data mutability in the production environment. For environment where queries are not repeatable, improper cache configuration can lead not only to performance decline but also to errors caused by out-of-date data.

[5] http://www.mybatis.org/

Second mechanism which can improve performance is possibility to use native SQL queries – let us remember that ORM generates SQL queries (mostly even series of SQL queries per one user request) and user do not have much influence on this process. Experienced SQL programmer can write some unusual or complicated queries better than any artificial generator. Using native SQL with ORM is like using an inline assembler in high level programming language – sometimes it is necessary to achieve better performance. So ORM should offer some method to run native queries. Of course if such method does not exists (which is unlikely) user can always use direct database access mechanism, for example JDBC. Very good solution is a possibility to store such queries in external files. They are then much easier to manage, especially for database administrators. It has to be remembered that using native queries with some constructions peculiar to a specific database, the application becomes less portable (if it comes to changing the database server). So it is better to avoid such queries or store them in external files where they can be modified without having to recompile whole application or module.

Another way to improve performance is the lazy loading mechanism. It allows an entity or collection of entities associated with some other entity to be loaded when they are directly requested. For example, if we have *Department* entity with *employees* property of list of *Employer* type and we request *Department* list we do not need load *employees* data until they are explicitly referenced. In opposition to lazy loading we have eager loading. Default retrieving mode in ORMs is mostly lazy. Decision which mode will be better depends on a concrete operation and its degree of interaction with user. When an operation is of batch type (e.g. script execution), eager mode would be appropriate. If operation requires interaction with the user, for example the user explores the list of products and for chosen products want to see some details, lazy loading should perform better than eager one. Lazy loading uses less memory but increases database server traffic [18].

When it comes to performance, it has to be stated that each ORM generates SQL queries in its own way. So if developer can choose from few or more ORM tools with similar functionality it is good idea to compare their performance in environment close to production system (with some inserting, selecting, updating, joining queries typical for a particular application) to check how fast SQL queries are processed by database server and how much memory is used for these operations.

When application cooperates with database it is obvious that it processes not only single records but also group of records meeting some criteria. So ORM should provide some mechanism to retrieve such records. It can be SQL-like object query language (for example Java Persistence Query Language or Doctrine Query Language) which syntax is usually not checked during compilation or some other way to query with mostly compile-time syntax-checking (for example Criteria API for Hibernate or *find* method for Active Record). It is always better to use ORM query language instead of pure SQL if it comes to application portability amongst (between) different database servers. It have to be stated that there are no standard for ORM query language but most of these languages are similar (due to similarity to SQL). Some standardization is JPQL for JPA.

As was mentioned above goal of using ORM is simpler and faster application creation. ORM should have tools for generating entities classes and XML mapping files on the basis of existing database (reverse engineering) or UML class diagrams. Also basic code generator for manipulating entities would be useful. But it has to be remembered that generated code or configuration always needs developer's review. Sometimes some changes are needed or even necessary and it is better to make them instantly. Interesting example of easy way to use ORM is Ruby Active Record which is based (as whole Ruby framework) on "convention over configuration" principle. For developer this principle means quick and simple start up without spending time on configuration [19]. For example, Ruby convention assumes that database table name is the pluralized lowercase name of the class defined in Active Record program with separating underscores if class name includes multiple words that begin with capital letter [19]. So developer must know all convention's assumptions and if he wants to break the convention it will involve additional work. Active Record does not need any XML configuration file(s) or annotations, so as opposite to other ORMs stays in accordance with DRY (Don't Repeat Yourself) rule – avoids multiple representation of information until developer sticks to convention [19, 20, 21].

Of course very important question is if chosen ORM is able to cooperate with a particular database. Most of ORMs offer enough wide range of adapters/dialects/data providers for most popular databases. Sometimes developer can use adapter offered directly by database producer or third-party one. For example, IBM DB2 offers own adapters for SQLAlchemy, Active Record, EF. EF offers own data provider only for Microsoft SQL Server, list of third-party providers encompasses, for example MySQL, Oracle, SQLite [6]. Eventually developer can create his own adapter but probably it won't be necessary.

What makes the biggest difference between realm of relation and world of objects is inheritance. When we need to persist some classes hierarchy we can use one of mapping inheritance patterns: Single Table Inheritance (STI), Class Table Inheritance (CTI) and Concrete Table Inheritance (CoTI) [2]. Since in relational database inheritance does not exist, mapping classes hierarchy on table(s) is always bound up with some inconvenience. In STI all hierarchy classes are persisted in single table which causes data redundancy and requires a dis-

[6] Microsoft MSDN Data Developer Center > Learn > ADO.NET > ADO.NET Data Providers, http://msdn.microsoft.com/en-us/data/dd363565

criminator column. With CTI, where all hierarchy classes have own tables, retrieving single object requires relatively many joins in SQL query. CoTI, where each concrete class has its own table with all attributes including inherited ones, avoids joins but is troublesome when changes in parent classes are needed [2]. ORM should offer these patterns or their variations. Possibility to decide if superclass will be persisted is/(might be) very useful. For example, JPA defines *@MappedSuperclass* annotation – class designated with this annotation has no table, but its attributes are persisted in subclasses tables. To sum up, it seems that it is better to avoid inheritance in persistent classes especially if application works with legacy database(s).

Table 1

Comparison of chosen ORM tools

Feature	ORM				
	Hibernate	Entity Framework	Active Record	MyBatis	Doctrine
Cache	L1, L2	L1, L2	L1, L2	L1, L2	L1, L2
for objects	yes	yes	yes	yes	yes
for queries	yes	yes	yes	yes	yes
Native SQL execution method	*createSQLQuery*	*ExecuteStoreQuery, ExecuteStoreCommand*	*find_by_sql*	yes	*Doctrine_RawSql*
Queries stored in external files	yes	no	no	yes	no
Lazy/eager loading	yes/yes	yes/yes	yes/yes	yes/yes	yes/yes
Inheritance mapping	all patterns[7]	all patterns	STI	STI	all patterns
Query language	HQL	LINQ	no	N/A	DQL
Query with compile-time syntax-checking	Criteria API	LINQ	no	N/A	no
Generating tools (own or external)	yes	yes	no	yes	yes
Operating system	any[8]	Windows	any	any	any
Language /framework	Java	.NET	Ruby	Java .NET	PHP
Open source	yes	no	yes	yes	yes

For some developers it is very important if a given ORM is open source (Open source ORM-s are very important to some developers). They obtain much more control over its be-

[7] Mentioned above inheritance patterns or their variations.
[8] Windows or Linux.

haviour or can fix errors themselves. For example, in Hibernate full join does not work[9] although *org.hibernate.sql.JoinType* includes FULL_JOIN option[10] – solution is a slight modification of *JoinProcessor* class. A question of licence and operating system is also vital.

A good documentation is also very important. Tutorials and developer forums – the more popular ORM the more materials, discussions, tests can be found on the Internet. If ORM is widely used by programmers it probably means that it is worthwhile.

Last, but not least criteria is programming language or development framework used to built the application. It narrows down the list of (possible)available ORMs. If the desired ORM is not on this list, developer or project manager can think about including an add-on module in the application in a language proper for this ORM.

Table 1 presents a few basic features of selected ORMs. Table was created on the basis of technical documentation of presented tools.

As one can see there are no big differences in functionality, but some can be substantial depending of what is expected from a given ORM. It has to mentioned that although Ruby's Active Record does not have own query language it offers methods, for example *find*, *first*, *select*, for query database [19]. If it comes to generating tools, on the Internet one can find very useful Ruby codes for reverse engineering etc.

4. Summary

Of course criteria defined above do not exhaust list of ORM desirable features like transaction management, locking, versioning, dynamic and named queries etc. There are more or less advanced ORM-s in terms of functionality. ORM choice should be sensible and adequate to particular project's needs. ORM always constitutes an additional layer in application. Sometimes Data Access Object component or smaller and simpler ORM could be sufficient for application proper functioning [21].

It should be mentioned there are more and more ORMs which operate not only with relational databases but also with NoSQL and object databases. To name a few Versant JPA for Versant Object Database, DataNucleus which supports NoSQL (for example Mongo DB and Google's BigTable) and object databases (db4o and NeoDatis ODB) in addition to relational ones, or Hibernate Object/Grid Mapper for NoSQL databases. This kind of ORMs can be very helpful in case of IT projects co-operating with hybrid data storage and data migration.

When discussing object-relational impedance mismatch issue it is impossible to omit object features of relational databases. For the first time some of this kind of features were de-

[9] https://hibernate.onjira.com/browse/HHH-2664.
[10] http://docs.jboss.org/hibernate/orm/4.0/javadocs/org/hibernate/sql/JoinType.html.

fined in SQL:1999 standard. Relational databases creators equipped their products with possibility to define complex types and methods, inheritance or object view of relational table, etc. IBM DB2, Oracle, Microsoft SQL Server support object technology with varying degree and most popular open source, object-relational database is PostgreSQL. As for now ORMs are the mainstream solution of object-relational impedance mismatch problem.

BIBLIOGRAPHY

1. Lausen G., Vossen G.: Models and Languages of Object-Oriented Databases. Addison-Wesley, 1997.
2. Fowler M. et al.: Patterns of Enterprise Application Architecture. Addison-Wesley, 2003.
3. The original Object People, https://sites.google.com/a/objectpeople.com/objectpeople-com/About [online, access 2012-01-15].
4. Smith D.: A Brief History of TopLink, http://www.oracle.com/technetwork/topics/history-of-toplink-101111.html [online, access 2012-01-15].
5. History-Hibernate-JBoss Community, http://www.hibernate.org/about/history [online, access 2012-01-15].
6. Bauer Ch., King G.: Hibernate in Action. Manning Publications, 2005.
7. Burke B., Monson-Haefel R.: Enterprise JavaBeans 3.0, 5th edition. O'Reilly Media, 2006.
8. Oracle TopLink JPA, http://www.oracle.com/technetwork/middleware/toplink/index-085257.html [online, access 2012-01-29].
9. JSR-000317 Java Persistence 2.0 – Final Release, http://jcp.org/aboutJava/community-process/final/jsr317/index.html [online, access 2012-01-15].
10. EclipseLink Project, EPS Creation Review, http://www.eclipse.org/projects/project.php?id=rt.eclipselink [online, access 2012-01-15].
11. http://www.eclipse.org/org/press-release/20080317_Eclipselink.php [online, access 2012-01-29].
12. Minter D., Linwood J.: Beginning Hibernate. Second Edition, Apress 2010.
13. Apache OpenJPA, http://openjpa.apache.org/ [online, access 2012-02-02].
14. DataNucleus, http://www.datanucleus.org/ [online, access 2012-02-02].
15. Apache OpenJPA, http://openjpa.apache.org/faq.html#FAQ-Whatisthehistoryof OpenJPA%253F [online, access 2012-02-02].
16. Kuaté P. H., Harris T., Bauer Ch., King G.: NHibernate in Action. Manning Publications, 2009.

17. Marguerie F., Eichert S., Wooley J.: LINQ in Action. Manning Publications, 2008.
18. Jennings R.: Professional ADO.NET 3.5 with LINQ and the Entity Framework. Wiley Publishing, 2009.
19. Marshall K., Pytel Ch., Yurek J.: Pro Active Record. Databases with Ruby and Rails. Apress, 2007.
20. Hunt A., Thomas D.: The Pragmatic Programmer. From Journeyman to Master. Adison-Wesley, 1999.
21. Ford N.:The Productive Programmer. O'Reilly Media, 2008.

Wpłynęło do Redakcji 31 stycznia 2012 r.

Omówienie

We współczesnym świecie większość nietrywialnych aplikacji współpracuje z bazą danych. Współpraca ta odbywa się na styku dwóch światów: obiektowego i relacyjnego. Świat obiektowy to świat obiektowych języków programowania, takich jak Java, C++, C#, Python itd. Świat relacji to świat relacyjnych baz danych. Oba te światy opierają się na różnych paradygmatach, co prowadzi do niekorzystnego zjawiska, zwanego niezgodnością impedancji obiektowo relacyjnej. Można stwierdzić, że najprostszym sposobem na uniknięcie problemów jest zastosowanie obiektowej bazy danych. Jednak na chwilę obecną obiektowe bazy danych nie są w stanie zagrozić rynkowej pozycji baz relacyjnych.

Obecnie najpopularniejszym sposobem współpracy obiektowej aplikacji z relacyjną bazą danych jest interfejs obiektowo relacyjny, w rozumieniu narzędzia pozwalającego na mapowanie relacyjnej bazy danych na zbiór obiektów (ORM). Ich historia zaczęła się we wczesnych latach 90. Obecnie funkcjonuje przynajmniej kilkadziesiąt tego typu narzędzi. Dla danego języka programowania czasami mamy do wyboru nawet kilkanaście możliwości. Wybór jest więc szeroki i powinien być dokonywany rozważnie.

ORM stanowi dodatkową warstwę pośredniczącą między aplikacją a bazą danych, wykorzystującą natywny mechanizm dostępu do bazy danych, jak np. JDBC, oraz informacje o sposobie odwzorowania relacji na obiekty zapisane w plikach XML bądź w postaci adnotacji umieszczanych bezpośrednio w kodzie. Jako dodatkowa warstwa, ORM może wpłynąć ujemnie na szybkość działania aplikacji. Jego głównym zadaniem jest uproszczenie tworzenia aplikacji, a nie zwiększenie wydajności. Jednym z podstawowych kryteriów wyboru narzędzia ORM powinny więc być mechanizmy pozwalające na zwiększenie szybkości jego działania, takie jak np. możliwość korzystania z pamięci podręcznej bądź natywnych zapytań do bazy danych. W artykule zaproponowano również kilka innych podstawowych cech, pozwalających ocenić przydatność danego narzędzia. W tabeli 1 zaprezentowano zestawienie tych cech dla kilku wybranych narzędzi ORM. Zwrócono również uwagę na narzędzia pozwalające odwzorowywać obiekty aplikacji do baz NoSQL lub obiektowych.

Address

Ewa PŁUCIENNIK-PSOTA: Silesian University of Technology, Institute of Computer Science, Akademicka 16, 44-100 Gliwice, Poland, Ewa.Pluciennik-Psota@polsl.pl.