SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2017)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2472

JDAS: a software development framework for multidatabases

Guoqi Xie[1,2,*,†], Yuekun Chen[1,2], Yan Liu[1,2], Chunnian Fan[3], Renfa Li[1,2] and Keqin Li[1,4]

[1]*College of Computer Science and Electronic Engineering, Hunan University, China*
[2]*Key Laboratory for Embedded and Network Computing of Hunan Province, China*
[3]*Nanjing University of Information Science and Technology, China*
[4]*Department of Computer Science, State University of New York, New Paltz, New York, USA*

SUMMARY

Modern software development for services computing and cloud computing software systems is no longer based on a single database but on existing multidatabases and this convergence needs new software architecture and framework design. Most current popular frameworks are not designed for multidatabases, and many practical problems in development arise. This study designs and implements a software development framework called Java data access service (JDAS) for multidatabases using the object-oriented programming language Java. The JDAS framework solves related problems that arise when other frameworks are employed in practical software development with multidatabases by presenting and introducing design methods. JDAS consists of the modules of the database abstract, object relational mapping, connection pools management, configuration management, data access service, and inversion of control. Results and case study reveal that the JDAS framework effectively reduces development complexity and improves development efficiency of the software systems with multidatabases.

Received 29 May 2016; Revised 22 October 2016; Accepted 23 November 2016

KEY WORDS: design methods; development framework; Java; multidatabases; object-oriented programming

1. INTRODUCTION

1.1. Background

Services computing and cloud computing are two mainly co-existing paradigms that are widely demanded at present software systems [1]. The services computing paradigm aims at the development of medium-sized and large distributed inter-organizational systems by assembling it as a system of service providers and consumers, known as a service-oriented computing application [2]. In turn, the cloud computing paradigm aims to provide a cost-effective, scalable and dynamic provision of IT resources (processing, storage, and networking) for elastic demands [3, 4]. Most works focus on the problem of satisfying the one or multiple quality of service (QoS) requirements for users while keeping satisfactory performance of the software systems [5, 6]. However, both services computing and cloud computing software systems rely on large amounts of data processing to provide efficient services and provision [7]. A typical mature data storage method is the database storage [8, 9]. With the merging, sharing and integration of different software systems, multidatabases have become the most important storage of current services computing and cloud computing software systems [10]. For example, office automation, e-commerce, and e-government

*Correspondence to: Guoqi Xie, College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, 410082, China.
†E-mail: xgqman@hnu.edu.cn

have grown rapidly with the expansion of the business requirements of enterprizes and governments by integrating multidatabases, such as Oracle, MySQL, Sybase, and Microsoft SQL Server. Therefore, modern software development for services computing and cloud computing software systems is no longer based on a single database but on existing multidatabases and this convergence needs new software architecture and framework design.

1.2. Motivation

Different developers who use various frameworks have developed many software systems with the extensive growth of data in services computing and cloud computing software systems. Data integration, exchange, and sharing among different databases are common in current software development practice [11, 12]. However, many popular frameworks are designed merely for single database rather than multidatabases. As a result, low development efficiency is caused in practice.

First, large-scale software systems are usually developed by different developers of various companies in different phases. These developers often choose their own framework combination that they are familiar with (e.g., Spring [13] and Hibernate [14]), preferred database products (e.g., Oracle, MySQL, Sybase, and Microsoft SQL Server), and connection pool technologies (e.g., C3P0 and DBCP [15]) to develop required functions. As a result, the system integrates various bytecode files of different frameworks. Some frameworks may be incompatible, and the system easily results in the potential risk of fault propagation. Moreover, further development and maintenance have increasingly become difficult because of the increasing complexity of the system.

Second, data should be divided and stored according to purpose, security, and performance in large-scale information systems. For example, databases usually include basic, core business, solution, sharing, public service, and other databases. However, the current mainstream frameworks (e.g., Spring and Hibernate) merely focus on the software development with single database. When these frameworks are applied to the software systems with multidatabases, the data integration, exchange, and sharing of different databases need to handle. However, completing these functions require many bloated and inflexible codes to be written. These codes cannot even be understood by developers themselves. As a result, many problems (e.g., poor software quality, lower development efficiency, and higher development complexity) arise when these frameworks are used.

The root of the issues earlier is that current many software development frameworks are oriented to a single database, and cannot be adapted to multidatabases environments of services computing and cloud computing software systems. Therefore, it is necessary to introduce new design methods to implement an efficient software development framework for multidatabases, thereby to improve development efficiency in practice.

1.3. Our contributions

To solve the issues earlier in software development practice with multidatabases, an effective software development framework called Java data access service (JDAS) for multidatabases implemented by the object-oriented programming (OOP) language Java is presented in this study. Our contributions contain two parts (i.e., data related part and service related part) and are described as follows:

(1) In data related part, we implement the following modules in JDAS: (i) we implement the module of database abstraction (DA) to simplify the diversity of table-level multidatabases by abstracting them as common interface; (ii) we implement the module of object relational mapping (ORM) to optimize compound primary key, timestamp, and primary and foreign keys association; (iii) we implement the module of connection pools management to seamlessly integrate any connection pool technologies into JDAS by using parameterized factory method pattern; and (iv) we implement the module of configuration management to configure the mapping between database interfaces and connection pool instances, and to configure the remote data access, and the local data access for databases
(2) In service related part, we implement the following modules in JDAS: (i) we implement the module of data access service (DAS) to merge the business logic and data access layers by

DOI: 10.1002/spe

using template method pattern; a DAS object is the combination of the business logic and data access objects (DAOs) and it extends many methods and can execute create, retrieve, update, and delete (CRUD) operations without adding any additional codes; (ii) we implement the module of IoC based on the dependency lookup, and any DAS object can be obtained with the same method in any program (e.g., JSPs, Servlets, Actions, and Components).

In summary, JDAS skillfully combines the data and service on the basis of the aforementioned presented modules. Developers write only a few of codes to complete the basic functions by JDAS in the software development with multidatabases.

The rest of this paper is organized as follows. Section 2 presents related frameworks. Section 3 designs and implements the JDAS framework. Section 4 verifies the performance of the JDAS framework. Section 5 concludes this study.

2. RELATED FRAMEWORKS

Table I lists important acronyms and abbreviations that are used in this study.

Java is a widely used object-oriented programming language with the characteristics open-source and cross-platform [16, 17]. A typical software framework contains presentation layer, business logic layer, and data access layer [18]. This study focuses on the data access and business logic layers.

2.1. Data access layer frameworks

Persistence is the core component in the data access layer. ORM is a method of mapping objects in the memory and data in databases [19]. The object does not bind by any special restriction except for the getter/setter methods. Thus, the object is usually defined as a plan old Java object (POJO) in Java. Many persistence technologies have been implemented with Java, and the most popular ORM framework is Java persistence API (JPA) [20], which was released in 2006, and Hibernate is an important implementation of JPA. Hibernate (or JPA) has become crucial in software development with database, but it still faces the following problems in practical development with multidatabases.

(1) Hibernate does not effectively handle data exchange and sharing in multidatabases because it needs to build a Hibernate instance for each database. Too many instances would consume more resources in large-scale multidatabases.
(2) In the same database, various table design solutions often come from different designers in various development phases. For example, some designers employ program-controlled table design solution where primary and foreign keys associations are not reflected in tables. In other words, the primary foreign keys association is only indicated by the database design documentation provided by the designers. Programmers write code by hand to control the association when writing programs. However, a new designer may use Hibernate in the development of

Table I. Important important acronyms and abbreviations that are used in this study.

JDAS	Java data access service
QoS	quality of service
OOP	object-oriented programming
DA	database abstraction
ORM	object relational mapping
DAS	data access service
DAO	data access object
CRUD	create, retrieve, update, and delete
POJO	plan old Java object
JPA	Java persistence API
HQL	Hibernate query language
IoC	Inversion of control
JNDI	Java naming and directory interface

DOI: 10.1002/spe

software systems with multidatabases. If he or she considers the primary foreign keys association using Hibernate, then unpredictable structural change of tables caused by Hibernate would occur.

(3) Compound primary keys are usually used for some complex tables. However, when developers use Hibernate, they need to write an additional compound primary key class according to the strict specification of Hibernate. This specification is too complex and time consuming in practical development.

(4) Timestamp is widely used in the data exchange of multidatabases. When one record of a table is added or modified, the timestamp field can denote the relative order with 16 hexadecimals in some databases (e.g., Sybase and Microsoft SQL Server) or indicate the modifying time of the record in other databases (e.g., Oracle and MySQL). However, Hibernate does not directly support this timestamp mechanism, such that it would render data exchange very complicated.

(5) Hibernate query language (HQL) is the persistence language of Hibernate, and this language has lower query performance than JDBC because of the multilayer encapsulations of the SQL language. Furthermore, the Hibernate N+1 SELECT's problem often happen. Hence, Hibernate is not suitable for performance-critical systems aimed at data query and analysis.

2.2. Business logic layer frameworks

Inversion of control (IoC) means that the creation and maintenance of an object is attributed to the container rather than the program [21]. Objects can be obtained using the dependency lookup or the dependency injection in IoC. Spring is a very mature and widely used lightweight IoC open-source framework in Java technology systems [13, 22], but it also faces many problems in practical development with multidatabases.

(1) Excessive interface-oriented design can easily result in the low efficiency of system development. If each Java class uses the interface-oriented dependency injection, problems of interface abuse, and too many classes would occur. These problems increase maintenance cost. For example, if a function needs to be added in the presentation layer, the business logic and data access layers also need to be amended accordingly.

(2) In practical development with multidatabases, some databases have the same table names and table structures. For example, database $business\_db\_01$ is used to store the data of district A, and databases $business\_db\_02$ and $business\_db\_03$ are used to store the data of districts B and C, respectively. Another situation is that $business\_db\_01$ stores the primary data, and $business\_db\_02$ and $business\_db\_03$ store the backup data of $business\_db\_01$. The table structures are completely equal in these databases. This database set is referred as table-level multidatabases. However, Spring does not effectively handle these table-level multidatabases. For example, assume that three databases exist (e.g., $business\_db\_01$, $business\_db\_02$, and $business\_db\_03$), and each has an equal table called $AuditPOJO$. If developers want to operate the $AuditPOJO$ of each database, then they need to create three business logic beans (i.e., $AuditPOJO01ServiceBean$, $AuditPOJO02ServiceBean$, and $AuditPOJO03ServiceBean$). All these beans implement the interface $AuditPOJOService$. However, if developers do not specify one concrete business logic bean (or its name) to be injected into the $AuditPOJOService$, then the system throws an exception. In other words, the advantage of the dependency injection of the Spring framework no longer exists and even becomes a hindrance in the software development with table-level multidatabases.

(3) Some programs (e.g., JSP, Servlet, and Filter) cannot be managed by the Spring framework. Thus, the dependency injection is invalid in these programs. Hence, the business logic beans are obtained only by the dependency lookup that specifies the object name. As a result, developers need use different methods (i.e., dependency injection and dependency lookup together) to obtain the same business logic beans in the same Java platform. These inconsistent approaches increase programming complexity and lower code readability.

(4) Many connection pool technologies are developed for various platforms and databases. However, Spring manages only use the specified connection pool technologies (e.g., C3P0 and

DOI: 10.1002/spe

DBCP) and does not directly support many third-party connection pool technologies. For example, the UniEAP pool developed by the company NeuSoft [23] cannot be supported by Spring.

3. JDAS FRAMEWORK

Figure 1 describes the functional modules of the JDAS framework and shows the relationships between the database layer (including multidatabases and multiconnection pools) and the presentation layer (including JSPs, Servlets, and Actions). JDAS consists of the modules of the DA, ORM, connection pools management, configuration management, DAS, and IoC. In general, the first four belong to the data part, and the latter two belong to the service part.

3.1. Database abstraction

DA is the basis for simplifying the software development with multidatabases. JDAS provides a maker interface *Database*, which does not employ any methods or fields, and identifies only the semantics of the DA. For a concrete database, developers should define an interface that extends the *Database* interface. The advantage is that developers simply define a common DA interface for table-level multidatabases. For example, developers define two DA interfaces, *NetworkDA* and *BusinessDA*, as shown in Figure 2. The DA interface

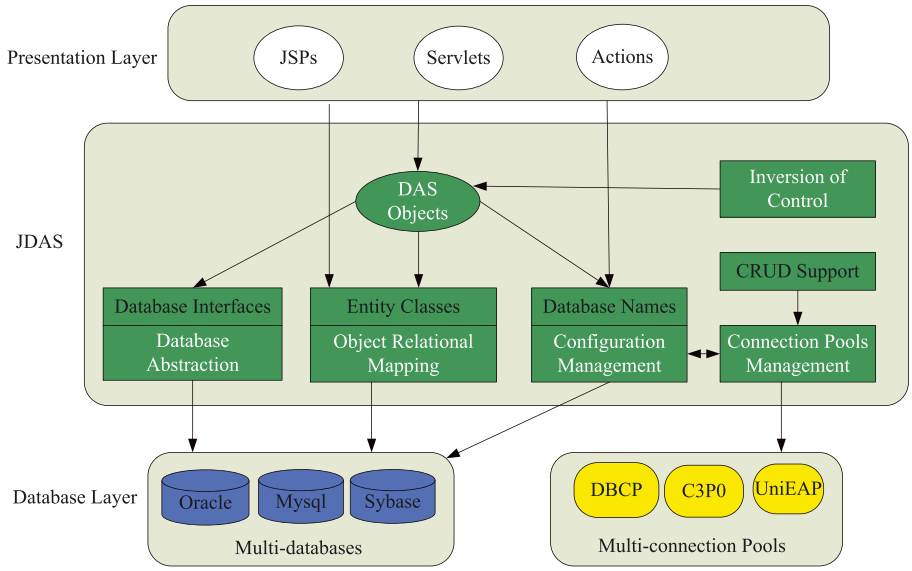

Figure 1. Modules of JDAS. [Color figure can be viewed at wileyonlinelibrary.com]

```
public interface Database {
}
```

```
public interface NetworkDA extends Database{
}
```

```
public interface BusinessDA extends Database{
}
```

Figure 2. Example of database abstract. [Color figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

NetworkDA is the database abstract of the database *network_db*, whereas the DA interface *BusinessDA* is the database abstract of the table-level multidatabases *business_db*_01, *business_db*_02, and *business_db*_03. The main objective of DA is to simplify the diversity of the table-level multidatabases. More concrete details of DA will be discussed together with other modules.

3.2. Object relational mapping

JPA supports the use of XML-based ORM [19]. Excessive XML files lower the readability of the codes because developers need to separately watch the codes and the XML files. Annotation has been the main mapping and configuration and the annotation-based ORM can address the problem stated earlier. Hence, JDAS also uses the annotation-based ORM and makes the following important improvements relative to Hibernate (JPA) ORM.

(1) The mapping of the compound primary key is improved. A compound primary key consists of two or more attributes that uniquely identify a record in Hibernate. The problem is that all the compound primary key fields must be defined in an embedded class and cannot intuitively reflect the correspondence of the fields between POJOs and tables. To address the problems stated earlier, JDAS first discards the embedded class and introduces the annotation *@Pk* to mark each primary key field in the POJO *CustomerPk* directly, as shown in Figure 3. In contrast to Hibernate ORM, JDAS ORM reduces complexity and improves the intuition of the compound primary key.
(2) The annotation *@Stamp* is used. Timestamp data type is very useful for the data exchange of multidatabases as mentioned earlier. However, Hibernate ORM does not directly support timestamp. JDAS introduces the annotation *@Stamp* to mark the field timestamp. As shown in Figure 3, the field *tstamp* is a byte[] data type, and this field is marked with the anno-

```
@Table("Customer")
public class Customer {

        @Pk
        private String name;

        @Pk
        private Long ssn;

        private String address;

        @Temp
        private String info;

        @Stamp
        private byte[] tstamp;

        ...
}
```

Figure 3. Annotation-based ORM of compound key in JDAS. [Color figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

tation $@Stamp$. When the CRUD operations are executed, JDAS processes the annotation $@Stamp$.

(3) The primary and foreign keys association is disregarded. Some designers prefer to directly define such association in databases and annotate it in POJOs, whereas others do not prefer the association in databases and use the program to control it instead. To prevent the unpredictable structural changes and risks, the JDAS ORM disregards the association in POJOs and directly controls this association in programs.

(4) The ORM and the operations are separated. Hibernate ORM is usually employed to map and execute CRUD operations with HQL. However, Hibernate ORM should create one instance for each database in advance. However, this approach increases the complexity of the development. To solve the problems stated earlier, JDAS ORM is only for mapping and not for the operations. The concrete CRUD operations are responsible for the DAS module, as discussed in Section 4.5.

3.3. Connection pools management

Connection pools can reduce the waiting time of a connection and resource overhead when CRUD operations are executed [24]. Many connection pool technologies (e.g., DBCP and C3P0) implement the $javax.sql.Datasource$ interface, and some containers (e.g., JBoss) provide connection pool technologies based on the Java naming and directory interface (JNDI) lookup. These two types described earlier can be supported by the Spring framework. However, partial connection pool technologies developed by some companies (e.g., the UniEAP pool developed by the company Neusoft [23]) do not implement $javax.sql.Datasource$. These technologies cannot be supported by the Spring framework. To clear the underlying problem, a industrial practical example is illustrated as follows.

As shown in Figure 4, a project was developed by the UniEAP framework and the UniEAP pool. This project uses the database Sybase, which is performance-matched with the UniEAP pool. Later, other developers took over this project for further development. However, they are not familiar with the UniEAP framework. If they continue to use the UniEAP framework to develop this project, then the development period cannot be controlled. If they consider using the Spring framework to develop, then many additional configuration and Jar files are introduced, which easily results in a conflict with the UniEAP framework. Moreover, the Spring framework cannot directly support the UniEAP pool, which cannot be replaced by other connection pool technologies because the UniEAP pool, Sybase database, and UniEAP framework are significantly performance-matched. If they replace the UniEAP Pool with the DBCP Pool, both the Sybase database and the UniEAP framework become unstable. Hence, the Spring framework is not suitable for the further development of this project.

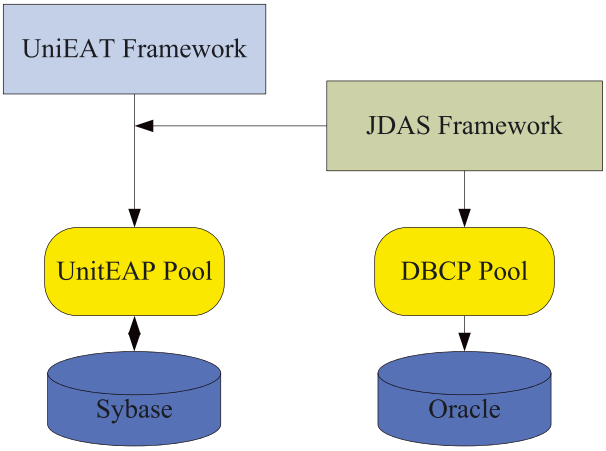

Figure 4. Example of connection pools management. [Color figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

JDAS can solve the problems stated earlier. JDAS uses a mechanism to make the framework compatible with the UniEAP pool by implementing a maintainable and compatible connection pools management solution using the parameterized factory method pattern [25, 26].

The parameterized factory method pattern is the extension of the factory method pattern. As shown in Figure 5, a factory method was first defined in the factory class (i.e., the $getConnection()$ method in the $PoolManager$ class), and allow the $getConnection()$ method to generate various connection objects with different input parameters (i.e., the $databaseName$ and the $poolName$), which generate an instance of the $ConnectionPool$ class. Hence, the $getConnection()$ method is a parameterized factory method. The parameterized factory method pattern is different from the general factory method pattern because the factory implements the factory method (e.g., the $getConnection()$ method). Hence, the factory class is no longer an abstract class of the initial factory method pattern. For example, to make the JDAS framework and the UniEAP pool compatible, developers do the following tasks: (i) define a class $UniEAPPool$ that implements the $ConnectionPool$ interface (Note that the concrete implemented details are omitted in this study); (ii) obtain the UniEAP pool by invoking the parameterized factory method $getConnection()$ of the factory class $PoolManager$ with different input parameters (e.g., the $databaseName$ and the $poolName$). These parameters are configured in a specified configuration file. More details of the configuration file can be seen in Section 4.4.

In summary, only if all the connection pool classes implement the $ConnectionPool$ interface, then they can be managed by JDAS without changing the existing codes. Concrete operations, such

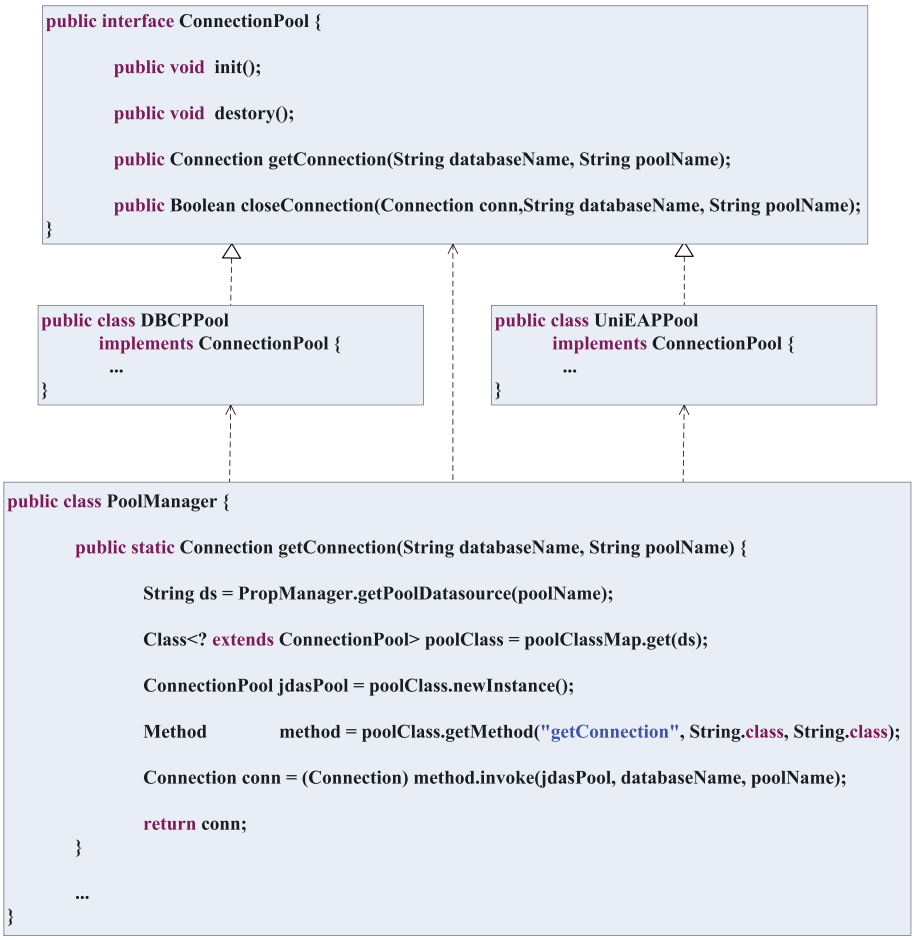

Figure 5. Connection pools management implemented by parameterized factory method pattern. [Color figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

as opening connection, obtaining the connection, and closing connection, are implemented by the connection pool classes themselves.

3.4. Configuration management

Although only a common database interface is required for table-level multidatabases as mentioned in Section 4.1, each database should still configure its own connection pool classes and instances. In other words, if n databases exist, developers should provide n connection pool instances to connect the databases and JDAS. Hence, developers should configure the mapping relationship between the database interfaces and the connection pool instances, and following information was configured in the file '$jdas.properties$'.

(1) Aliases of the connection polls. Section 4.3 defines the concrete connection pool classes for all the connection poll technologies using the fully qualified name of a concrete connection pool class. For convenience, an alias was defined for each connection pool class (e.g., the alias of the $DBCPPool$ class is $dbcp$, and that of the $UniEAPPool$ class is eap), as shown Figure 6.
(2) Names of the databases. For table-level multidatabases, merely defining the common database interface is not enough because each database has its own property values (e.g., driver class, connection uniform resource locator (URL), username, password, etc.). These property values are organized in their individual dependent files. For example, three property files exist ($business\_db\_01.properties$, $business\_db\_02.properties$, and $business\_db\_03.properties$) for the table-level multidatabases $business\_db\_01$, $business\_db\_02$, and $business\_db\_03$ to save the individual property values. Consequently, the database $business\_db\_01$ and the connection poll $unieap$ are mapped with the form of [$business\_db\_01$, $unieap$, $business\_db\_01.properties$] (Figure 6). That is, the information of database $business\_db\_01$ is organized in the $business\_db\_01.properties$, which is configured with the connection poll $unieap$.
(3) Local and remote databases. JDAS provides two database access modes, namely, the local database access and remote data access. Developers configure the database $network\_db$ as the local database (e.g., $network\_db\_local$ = [$network\_db, dbcp, network\_db.properties$] of Figure 6). For table-level multidatabases, only one database is configured with the local database, and the remaining databases should be configured with the remote databases. For example, assuming that the database $business\_db\_01$ is configured with the local database, databases $business\_db\_02$ and $business\_db\_03$ should be configured with the remote databases. In other words, $business\_db\_01$ is the default database of the DA interface $BusinessDA$ that can be directly accessed. If developers want to access the remote database $business\_db\_02$, they should use the '$business\_db\_02$' as as an input parameter to access. concrete access details can be seen Sections 4.5 and 4.6.

3.5. Data access service

In the data access layer, the data access objects implement the corresponding DAO interfaces to execute the CRUD operations. In the business logic layer, the business logic objects invoke the data access objects based on oriented-interface programming. However, the design earlier generates a large number of data access and business logic objects. These objects consume more memory resource and increase the maintenance cost as mentioned earlier. To address the problems stated

```
pool_alias = [eap,org.jdasframework.core.pool.impl.UniEAPPool],[dbcp,org.jdasframework.core.pool.impl.DBCPPool]

NetworkDA_local = [network_db,dbcp,network_db.properties]

BusinessDA_local  = [business_db_01,unieap,business_db_01.properties]
BusinessDA_remote = [business_db_02,unieap,business_db_02.properties],[business_db_03,unieap,business_db_03.properties]
```

Figure 6. Example of configuration management. [Color figure can be viewed at wileyonlinelibrary.com]

DOI: 10.1002/spe

Figure 7. Data access service implemented by template method pattern. [Color figure can be viewed at wileyonlinelibrary.com]

earlier, JDAS merges the data access and business logic layers. The core objects in JDAS are called the DAS objects. Figure 7 shows the concrete design of the DAS objects.

First, the generic interface $DAO < T >$ was defined to specify the common data access methods. Note that type parameter T represents a POJP. Second, the generic abstract class $JdbcDAOSupport < D, T >$ was defined and it is a data access operation support class that implements the $DAO < T >$ interface, where D represents the interface of a database and T inherits the T in $DAO < T >$. $JdbcDAOSupport < D, T >$ is the common implementation of relational databases. Note that the concrete SQL operations for each method in the $JdbcDAOSupport < D, T >$ are transferred to the class $JdbcDAOSupportCRUD < T >$. The $JdbcDAOSupport < D, T >$ contains many CRUD operations (only two methods are listed for simplicity), but it is designed as an abstract class. Hence, the $JdbcDAOSupport < D, T >$ cannot create any instance. This design is reasonable because D and T are unknown. The $JdbcDAOSupport < D, T >$ class should be extended by concrete DAS classes. For example, the DAS class $ApplicationPOJOServiceBean$ extends the $JdbcDAOSupport < NetworkDA, ApplicationPOJO >$. That is, the $ApplicationPOJOServiceBean$ class inherits all the CRUD methods of the $JdbcDAOSupport < NetworkDA, ApplictationPOJO >$. Hence, $ApplicationPOJOServiceBean$ can directly operate the POJO $ApplicationPOJO$ of the DA interface $NetworkDA$ without adding or modifying any methods. Similarly, $AuditPOJOServiceBean$ should extend the $JdbcDAOSupport < BusinessDA, AuditPOJO >$ class, and can directly operate the POJO $AuditPOJO$ of the DA interface $BusinessDA$. Certainly, both the $ApplictationPOJOServiceBean$ and the $AuditPOJOServiceBean$ can also add new business logic methods if necessary.

DOI: 10.1002/spe

The design of the DAS object is the typical template method pattern [27], which is a behavioral design pattern. This pattern defines the program skeleton of an algorithm in the template method. This template method defers some steps to its subclasses and allows certain steps of the algorithm to be redefined without changing the algorithm's structure. The $JdbcDAOSupport < D, T >$ is the program (CRUD) skeleton, and the methods in CRUD are the template methods. On the one hand, the $JdbcDAOSupport < D, T >$ constrains the behavior of its subclasses; that is, these subclasses presented as the data access objects. On the other hand, $JdbcDAOSupport < D, T >$ allows these subclasses to redefine and add certain methods; that is, these subclasses are also denoted as the business logic objects. Hence, $JdbcDAOSupport < D, T >$ achieves reusable DAS with the generic and the refection of Java. Note that JDAS combines the data access and business logic layers, but this choice does violate the principles of abstraction and separation of concerns because these two functions still relative separated.

The $save()$ and $saveByPoolName()$ methods are used to explain the CRUD operations. The DA interface $NetworkDA$ is only the abstract of the database $network\_db$, which is configured with the local database. The $ApplictationPOJOServiceBean$ invokes the $save()$ method to save $ApplictationPOJO$ objects directly. However, the DA interface $BusinessDA$ is the abstract of the table-level multidatabases $business\_db\_01$, $business\_db\_02$, and $business\_db\_03$. Except for that the $business\_db\_01$ database is configured with the local database, the other databases should invoke the $saveByPoolName()$ method to save $AuditPOJO$ objects by inputting the database name because developers can only configure a database as the local database for the DA interface $BusinessDA$, and the other databases should be configured with remote databases. Moreover, JDAS has higher development efficiency than the Spring framework because JDAS defines only one class $AuditPOJOServiceBean$, whereas the Spring framework needs to define three classes $AuditPOJO01ServiceBean$, $AuditPOJO02ServiceBean$, $AuditPOJO03ServiceBean$ to access the POJO $AuditPOJO$ in the three databases.

3.6. Inversion of control

As the DAS objects are the core components of the JDAS framework, managing and obtaining the DAS objects constitute a major problem. As mentioned earlier, many Java components (e.g., JSP, Servlet, and Filter) cannot be handled by the Spring container. In other words, the dependency injection is invalid for the components. In the Web context, if developers want to obtain one Spring Bean instance in Servlets, Filters, or JSPs, developers should determine the $ServletContext$ in advance. Furthermore, different web components use various methods to obtain the $ServletContext$. For example, the $ServletContext$ is obtained by the $ServletConfig.getServletContext()$ method in Servlets, the $FilterConfig.getServletContext()$ method in Filters, and the $pageContext.getServletContext()$ method in JSPs.

To address the problems earlier, JDAS uses the dependency lookup approach to obtain the DAS objects in any program of the same Java platform. This approach is implemented as follows: (i) scan all the DAS classes, which extends the $JdbcDAOSupport < D, T >$ class; (ii) use the reflection technology to generate one $XxxServiceBean$ instance for each $XxxServiceBean$ class when it is started; (iii) organize the $< XxxServiceBean.class, XxxServiceBeaninstance >$ in its context; (iv) use the $Service.get(XxxServiceBean.class)$ method to obtain the $XxxServiceBean$ instance. The greatest advantage of this approach is that any DAS object defined in the program can be obtained with the unified $Service.get(XxxServiceBean.class)$ method in any programs. Moreover, JDAS not only supports CRUD operations, but also supports directly SQL operations and commands for complex queries. Hence, JDAS is more convenient and deterministic than the Spring framework in obtaining the $XxxServiceBean$ instance.

3.7. Calling process

On the basis of the previously presented modules, developers write only a few of codes to complete the basic functions in the software development with multidatabases. The calling process of the example was shown in Figure 8: (i) define two DA interfaces $NetworkDA$ and $BusinessDA$. (ii)

DOI: 10.1002/spe

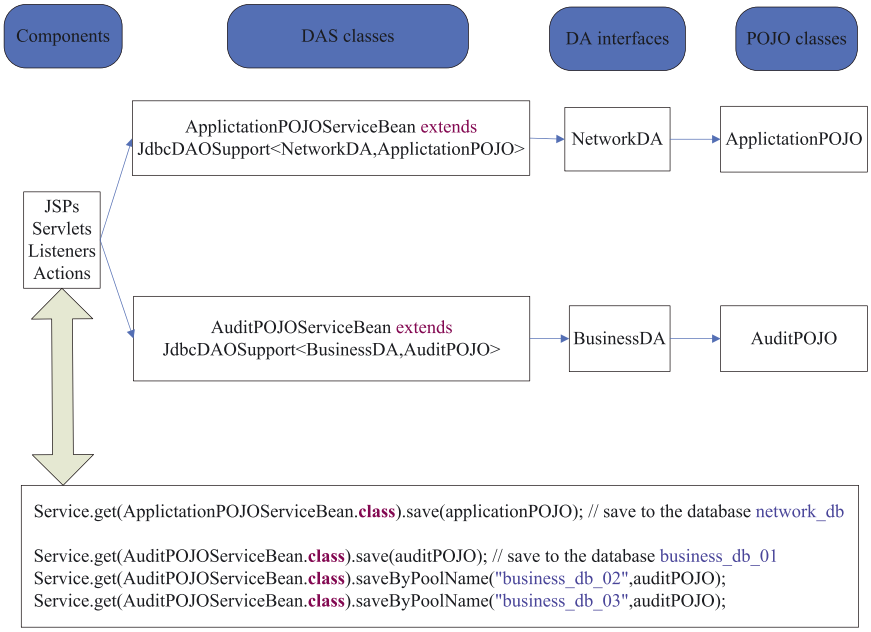

Figure 8. Example of calling process using JDAS. [Color figure can be viewed at wileyonlinelibrary.com]

define two POJO classes $ApplictationPOJO$ and $AuditPOJO$; and (iii) define two DAS classes $ApplictationPOJOSericeBean$ and $AuditPOJOServiceBean$.

Developers can call the business methods of the DAS classes in JSPs, Servlets, and Actionss, and others. For example, the $save()$ method of the $AuditPOJOServiceBean$ is called to save the $auditPOJO$ to the local database $business\_db\_01$. The $saveByPoolName()$ method of the $AuditPOJOServiceBean$ is called to save the POJO $auditPOJO$ to the remote databases $business\_db\_02$ and $business\_db\_03$, respectively.

4. EXPERIMENTS AND CASE STUDY

This study collects the experimental data of a practical multidatabases web project to analyze the code effect. The goal of the framework is to reduce development complexity and improve development efficiency. Thus, this study mainly evaluates the code effect. Code effect refers to the number of codes, configuration files, and libraries, and it would affect the development complexity and efficiency. This study selects the frameworks JDBC, Hibernate, MyBatis, and Spring that are free (i.e., open-source), relatively popular, and do not require a Java enterprise edition (JEE) for comparison. The project runs on JDK 1.7. 0, Windows 7 Professional, and Pentium dual-core processor (2.0 GHZ/3.0 GB RAM).

4.1. Code impact experiments

Modern software development usually integrates IoC frameworks (e.g., Spring) and persistence frameworks (e.g., Hibernate and MyBatis) to improve development efficiency and reduce development complexity. JDAS itself is a framework integrated with persistence and IoC. To better reflect the code effect of practical development process, other persistence frameworks are integrated with the Spring framework in experiments.

Experiment 1. This experiment compares the total code lines (including the lines of Java code and those of the configuration code) of different frameworks (e.g., Spring and JDBC, Spring and MyBatis, Spring and Hibernate, and JDAS). All frameworks implement the business operations of the POJO $ApplicationPOJO$. The configuration files include XML and property files.

DOI: 10.1002/spe

Table II. Lines of code of a single database.

Framework	Lines of Java code	Lines of configuration code	Total lines of code
JDBC&Spring	1,440	25	1,465
MyBatis&Spring	732	712	1,444
Hibernate&Spring	656	156	812
JDAS	115	20	135

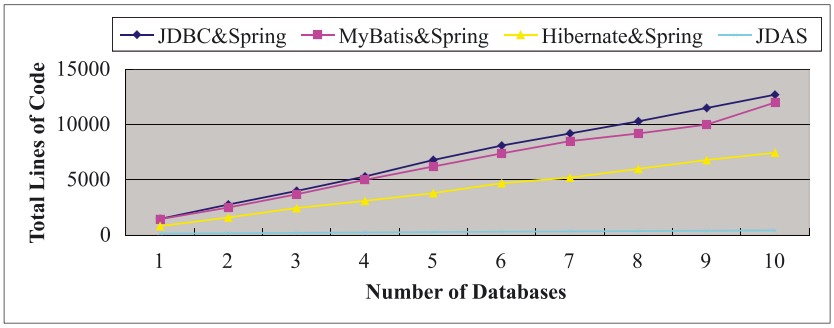

Figure 9. Total code lines of the table-level multidatabase. [Color figure can be viewed at wileyonlinelibrary.com]

Table II shows the lines of code for different frameworks of a single database Sybase 12.5. The combination of JDBC and Spring has the most number of Java code lines among all the frameworks. The total code lines for MyBatis and Spring are not significantly reduced compared with those of JDBC and Spring because the combination still requires developers to write SQL to deal with the CRUD operations, although the SQL code is freed from the Java code for MyBatis. JDAS has the least Java code lines and configuration codes. The total of the code lines for JDAS is only 135, which is significantly less than the lines of all the other frameworks (i.e., only approximately 1/10 of JDBC and Spring, 1/10 of MyBatis and Spring, and 1/5 of Hibernate and Spring). The reason for this behavior can be explained as follows: (i) JDAS merges the database access of the object and business logic layers into the DAS layer; (ii) The DAS objects automatically inherit many complete CRUD methods, which can be directly invoked by many programs.

The significant advantage of employing the JDAS framework is that it is completely applicable to multidatabases. Hence, this experiment achieves at most 10 table-level multidatabases. Figure 9 shows the total code lines for varying number of the databases. As can be seen that, with an increase in databases, the code line number for JDAS remains very small. However, the code line number for other frameworks almost increases linearly with the number of databases.

Experiment 2. Merely considering the number of the codes does not fully reflect the size of the application. In some cases, especially in hardware-constrained environments, the size of executable codes (i.e., byte codes, libraries, etc.) is not a negligible problem because the library size in bytes affects the system's startup and memory's usage. Hence, this experiment compares the size in bytes (including the size in bytes of the business logic classes, configuration of files, core libraries, and external libraries) of different frameworks (Spring and JDBC, Spring and MyBatis, Spring and Hibernate, and JDAS).

Table III shows the size in bytes of a single database. As observed, the size in bytes of the business classes for JDAS is significantly less than that of other frameworks (i.e., only approximately 1/10 of JDBC and Spring, 1/6 of MyBatis and Spring, and 1/5 of Hibernate and Spring). Except for the database driver file and library files, JDBC and JDAS do not require other third-party libraries. The core library file of JDAS is only 300 KB, which is significantly less than that of Hibernate and MyBatis.

Figure 10 indicates the total size in bytes for varying number of databases. As can be seen that, with an increase in databases, the total size in bytes for JDAS is relative stable, whereas the total size

DOI: 10.1002/spe

Table III. Size in bytes of a single database.

Framework	Business classes	Configuration files	Required libraries	Core files
JDBC&Spring	20,900	100	0	50
MyBatis&Spring	13,300	11,800	420	620
Hibernate&Spring	10,100	5,400	4,400	4,400
JDAS	2,000	90	0	300

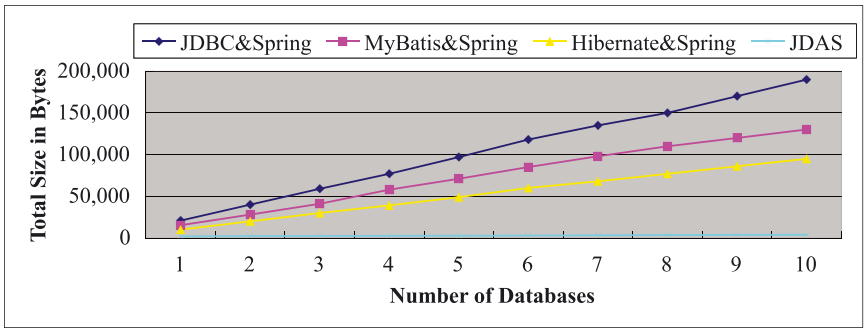

Figure 10. Total size in bytes of table-level multidatabase. [Color figure can be viewed at wileyonlinelibrary.com]

in bytes for the other frameworks almost increases linearly with the number of databases. Hence, JDAS has higher development efficiency than the other popular frameworks in developing table-level multidatabases.

4.2. Case study

In order to provide more effective and convenient services to enterprizes and citizens, e-governments have grown rapidly in the past years. E-government software systems are large-scale multidatabases systems that integrate, exchange, and share massive data among different databases. We demonstrated the applicability and the possible benefits of our framework with a case study derived from a real sub-project called online business application and approval (OBAA) of the e-government project deployed in Java environments and developed by JDAS.

In OBAA, users submit their applications from the Internet to the Intranet. Then staffs in the Intranet approve the applications and make the final decision return to the user in the Internet. The OBAA sub-project involves multidatabases because the external part (for users) and internal system (for staffs) are relatively independent due to security considerations, but they should share partial data because of the interaction. As shown in Figure 11, the sub-project involves three DAs: $NetworkDA$ ($network\_db$), $BusinessDA$ ($business_db$_01,..., $business_db$_02,..., $business_db$_10), and $WorkflowDA$ ($workflow\_db$).

This main details and process of the OBAA sub-project are as follows:

(1) A user fills the POJO $ApplicationPOJO$ information and persists it into the database $network\_db$ in the external system.
(2) A staff receives the POJO $ApplicationPOJO$ from $network\_db$ and modifies it in the internal system.
(3) If the $ApplicationPOJO$ is eligible after a series of approvals, then it is converted to the $AuditPOJO$ and stored into one of the databases ($business_db$_01, $business_db$_02,..., $business_db$_10) of the DA interface $BusinessDA$ according to the region of the $ApplicationPOJO$ that it belongs to.
(4) If the $ApplicationPOJO$ is not eligible, it is sent back to the users. Moreover, the database $workflow\_db$ records the audit information (workflow processing) of the $ApplicationPOJO$, which can be access by both users and staffs.

DOI: 10.1002/spe

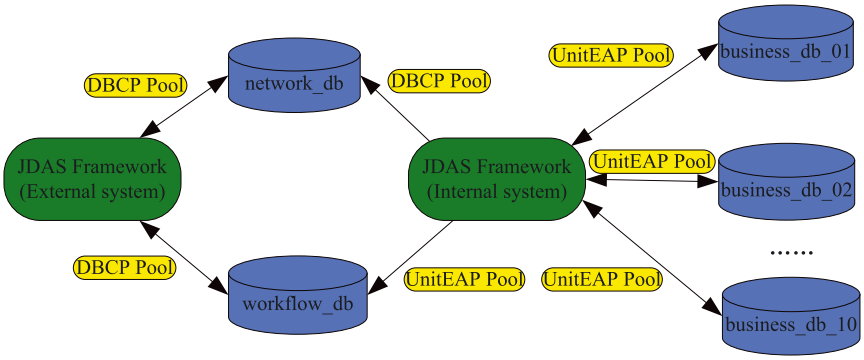

Figure 11. Case study of a industrial practical multidatabases web project. [Color figure can be viewed at wileyonlinelibrary.com]

Note that the connection pool DBCP is used for the DA interface $NetworkDA$ in both the external and internal systems. DBCP is also used for the database $workflow\_db$ in the external systems. The connection pool UniEAP is used for the databases $business\_db\_01$, $business\_db\_02$,..., $business\_db\_10$ and the database $workflow\_db$ in the internal system. The reason is that the internal system was first developed with the UniEAP framework. In other words, the internal system is further developed based on the UniEAP framework, whereas the external system is a newly developed system with JDAS.

Many design problems encountered in this project can be solved easily by JDAS, which simplifies the design and development of the project. The main completed work and design methods of JDAS are as follows:

(1) DA. 12 databases exist in the internal system. JDAS defines DA interfaces $NetworkDA$, $BusinessDA$, and $WorkflowDA$. Databases $business\_db\_01$, $business\_db\_02$,..., and $business\_db\_10$ can be accessed by configuring the local and the remote based.
(2) ORM. Some tables exist in the primary and foreign keys association, whereas some tables do not exist such association; meanwhile, other some tables use the compound keys. For consistency, JDAS does not consider the association and uses an intuitive compound primary key mapping to simplify the design and reduce the risk. Moreover, a timestamp attribute was added for each table to facility data exchange without affecting the stable runs of the previous functions.
(3) Connection pool integration. The internal system was developed by earlier developers with the UniEAP framework and pool. JDAS can directly manage the UniEAP pool, such that the compatibility of the system can be met and the risk of further development can be minimized.
(4) DAS. DAS can complete many CRUD operations for multidatabases using the minimum codes. Moreover, DAS can be directly invoked in all the original programs developed by earlier developers (e.g., JSP, Servlet, Listener, and any Java code). In other words, JDAS can be easily integrated into any other framework without affecting the original functions.

5. CONCLUSION

This study develops an efficient software development framework JDAS for software systems with multidatabases, such as services computing and cloud computing. The JDAS framework is designed to and addresses many practical problems in the software development with multidatabases by presenting and implementing many novel modules. Results reveal that the JDAS framework effectively reduces development complexity and improves the development efficiency over the popular frameworks in software development with multidatabases. We applied this framework to an e-governmental case study, and showed it can effectively simplify the design and development of the project. At present, JDAS is mainly for relational multidatabases. In the future, we will extend JDAS to including heterogeneous no-SQL multidatabases. JDAS will also include the models transac-

DOI: 10.1002/spe

tion management of multidatabases, cache management, security management, and aspect-oriented programming (AOP).

ACKNOWLEDGEMENTS

The authors would like to express their gratitude to the anonymous reviewers for their constructive comments which have helped to improve the quality of the paper. This work was partially supported by the National Key Research and Development Plan of China under Grant No. 2016YFB0200405 and 2012AA01A301-01, the Natural Science Foundation of China under Grant Nos. 61672217, 61173036, 61432005, 61370095, 61300037, 61370097, 61502405, and 61502162, the China Postdoctoral Science Foundation under Grant No. 2016M592422, the Priority Academic Program Development of Jiangsu Higer Education Institutions (PAPD), and the Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology (CICAEET).

REFERENCES

1. Reyes-Delgado PY, Mora M, Duran-Limon HA, Rodríguez-martínez LC, O'Connor RV, Mendoza-gonzalez R. The strengths and weaknesses of software architecture design in the rup, msf, mbase and rup-soa methodologies: A conceptual review. *Computer Standards & Interfaces* 2016; **47**:24–41.
2. Zhu Y, Huang Z, Zhou H. Modeling and verification of web services composition based on model transformation. *Software: Practice and Experience* 2016. DOI:10.1002/spe.2434.
3. Quinton C, Romero D, Duchien L. Saloon: a platform for selecting and configuring cloud environments. *Software: Practice and Experience* 2016; **46**(1):55–78.
4. Xia Z, Wang X, Zhang L, Qin Z, Sun X, Ren K. A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing. *IEEE Transactions on Information Forensics and Security* 2016; **11**(11): 2594–2608.
5. Dastjerdi AV, Garg SK, Rana OF, Buyya R. Cloudpick: a framework for qos-aware and ontology-based service deployment across clouds. *Software: Practice and Experience* 2014; **45**(2):197–231.
6. Zeng L, Xu S, Wang Y, Kent KB, Bremner D, Xu C. Toward cost-effective replica placements in cloud storage systems with qos-awareness. *Software: Practice and Experience* 2016. DOI: 10.1002/spe.2441.
7. Xia Z, Wang X, Sun X, Wang Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 2015; **27**(2):340–352.
8. Fu Z, Sun X, Liu Q, Zhou L, Shu J. Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing. *IEICE Transactions on Communications* 2015; **98**(1):190–200.
9. Ren Y, Shen J, Wang J, Han J, Lee S. Mutual verifiable provable data auditing in public cloud storage. *Journal of Internet Technology* 2015; **16**(2):317–323.
10. Rui HRP, Ortin F. Modularizing application and database evolution c an aspect-oriented framework for orthogonal persistence. *Software: Practice and Experience* 2016. DOI:10.1002/spe.2415.
11. Wu H, Li DX. Integration of distributed enterprise applications: a survey. *IEEE Transactions on Industrial Informatics* 2014; **10**(1):35–42.
12. Bernstein PA, Haas LM. Information integration in the enterprise. *Communications of the ACM* 2008; **51**(9):72–79.
13. Johnson R, Hoeller J, Arendsen A, Risberg T, Kopylenko D. Professional java development with the spring framework. *APC,* Paris, 2007:195–237.
14. Bauer C, King G. Hibernate in action (in action series). In *Guide to Web Development with Java*. Manning Publications Co.: Greenwich, London, 2004; 137–184.
15. Hohenstein U, Jaeger MC, Bluemel M. Improving connection pooling persistence systems. *International Conference on Intensive Applications and Services, Intensive 2009*, Valencia, Spain, 2009; 71–77.
16. Balland E, Moreau PE, Reilles A. Effective strategic programming for java developers. *Software: Practice and Experience* 2014; **44**(2):129C162.
17. Portillo-Dominguez AO, Perry P, Magoni D, Wang M, Murphy J. Trini: an adaptive load balancing strategy based on garbage collection for clustered java systems. *Software: Practice and Experience* 2016; **46**(12):1705–1733.
18. Maenhaut PJ, Moens H, Ongenae V, Turck FD. Migrating legacy software to the cloud: approach and verification by means of two medical software use cases. *Software: Practice and Experience* 2016; **46**(1):31C54.
19. Torres A, Galante R, Pimenta MS. Towards a uml profile for model-driven object-relational mapping. *in Xxiii Brazilian Symposium on Software Engineering*, Fortaleza, 2009; 94–103.
20. Linskey PC, Prud'hommeaux M. An in-depth look at the architecture of an object/relational mapper. *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ACM, 2007; 889–894.
21. Fowler M. Inversion of control containers and the dependency injection pattern: Beijing, 2004. Available at: http://martinfowler.com/articles/injection.html, 2004 [last accessed 23 January 2004].
22. Arthur J, Azadegan S. Spring framework for rapid open source j2ee web application development: a case study. *International Conference on Software Engineering, Artificial Intelligence, NETWORKING and Parallel/distributed Computing, 2005 and First Acis International Workshop on Self-Assembling Wireless Networks, Snpd/sawn*, Towson, 2005; 90–95.

23. Cao Y, Wang L, Xu H. Research on business platform software product development process based on knowledge integration–neusoft business platform software product unieap for example. *2010 International Conference on Management and Service Science, MASS 2010*, IEEE, Wuhan, 2010; 1–4.
24. Basanta-Val P, Garcia-Valls M, Estevez-Ayres I. A dual programming model for distributed real-time java. *IEEE Transactions on Industrial Informatics* 2011; **7**(4):750–758.
25. Stencel K, Wegrzynowicz P. Detection of diverse design pattern variants. *2008 15th Asia-Pacific Software Engineering Conference*, IEEE, Beijing, 2008; 25–32.
26. Ellis B, Stylos J, Myers B. The factory pattern in api design: A usability evaluation. *Proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society, 2007; 302–312.
27. Gamma E. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education: India, 1995.

DOI: 10.1002/spe