EDRM: A Unified Approach for Enterprise Data Resource Management

Chen Weiwen and Ma Shilong

National Key Lab of Software Development Environment, Beihang University, Beijing, China

Summary

Data resource always plays the key role in modern enterprise applications. In the paper, we first give a detailed introduction of the existing data resource management approaches. Via the analysis and comparison of the existing apprpaches, a new unified framework for enterprise data resource management is proposed. The framework can encapsulate the heterogeneous data resources in a universal form based on the data persistence technology, and thus provide a unified data access interface for the enterprise applications. The architecture and the main modules of the framework are detailedly introduced in the paper. And an application example is included in the end to show the usability of the framework.

Key words:

Data resource management, data persistence, heterogeneous data.

1. Introduction

Data always play the central roles in modern enterprise applications, in most of which frequent data storing and accessing actions are usually involved. That's all because of the development of Internet, which increases the shared information and makes it necessary for enterprise applications to interchange data with heterogeneous data sources located in distributed network nodes. Statistics show that, in the development of today's enterprise applications, 70% developing time is spent in creating and maintaining the data access mechanisms. Then how to manage the distributed and heterogeneous enterprise data resources and provide a unified and high-efficiency data access mechanism has become one of the challenges confronted with the enterprise application developers.

In early days of the information system development, the data access details were always embedded in the business logic modules, in which SQL (Structure Query Language) codes were usually included. This mechanism worked well for the quick demo development, but it caused the business process logic tightly coupled with the data storage methods and reduced the maintainability and extensibility of the systems. Meanwhile, as encapsulated in the concrete applications, the data access codes could rarely be reused.

With the development of software engineering and data access technology, we began to employ specified application modules to accomplish the data access process.

This strategy encapsulated the data storage method–related access logic into the separate modules , prevented the business process logic from being coupled with the data resources, and improved the maintainability of the systems. However, similar to the early days' approach, data access details, such as SQL codes, are still embedded in the application modules, and the data access modules are difficult to be reused in the future.

To solve the problems mentioned above, a persistence layer-based data access approach has been introduced recently both in the industry and research fields. In the approach, a data persistence layer focusing on the data access logic is built between the data source layer and the business application layer. The data persistence layer provides certain persistence mechanism that can automatically transform the operations on data objects in memory to the data access codes, which meets the requirement of the object-oriented system developments. Compared with the two traditional strategies, as the data persistence layer codes are not coupled with any concrete applications, the method not only reduces the coupling between business logic and data resource but also provides greater flexibility and reusability.

With the analysis above, the paper proposes a new Enterprise Data Resource Management Framework (EDRM) based on the theory of data persistence layer, and applies it to a practical space-equipments auto-testing system. The EDRM framework has following features: first, EDRM can integrate and manage the distributed and heterogeneous data resources, relational databases and XML files included; Via a general data persistence method (GDP) it provides a series of unified persistence operation interfaces for all kinds of data sources, and solves the problem within several existing persistence frameworks. Thus, EDRM hides the storage structure of data resources for enterprise application development. Second, EDRM provides lots of data access interfaces which can be customized as services. Then, the business application modules can subscribe the data access services on demand, which increases the agility of data access.

Manuscript received January 5, 2007
Manuscript revised January 25, 2007

2. The EDRM Framework

2.1 Architecture Overview

The architecture of EDRM comprises three layers: data resource integration layer, data persistence layer, and data access service provider layer, which is shown in Figure1.From the figure, we can see that the data resource integration layer is the lowest layer which integrates different kinds of data resources into the EDRM. The data persistence layer, right above the data resource integration layer, is the core layer of the EDRM, which encapsulates the data persistence details completely, and, with the idea of GDP, provides the unified data access methods for all kinds of data resources. The data access service provider layer located on top of the EDRM interacts with the enterprise applications directly, which provides agile data access services. Then, we will give a detailed explanation of these layers.

2.1.1 Data Resource Integration Layer

This layer is mainly used to register and then connect the data resources including relational database and XML files into the EDRM framework. It is composed of the resource register module and resource connector module.

The resource register module is in charge of the data resource registry. As shown in Figure 1, there is a data resource registry manager build within this module, it can standardize the data resource being registered into EDFM, and add them to the resource directory.

The resource connector module is mainly designed to locate and connect the data resources. It consists of data resource manager and data resource locater. With database connection technology [5] and XML document location technology [6], data resource manager creates the interfaces used for interacting with the data resources. Data resource locater can locate the data resources by the registry id and then acquire the connection handler.

2.1.2 Data Persistence Layer

This layer mainly takes charge of the data persistence process, in which the data objects in memory are stored into the data storage medium, such as relational database and XML document. From Figure1, we can see that the data persistence layer mainly consists of four modules: the data persistence module, the data access module, the transaction manager module and the log module.

The data persistence module implements a general persistence mechanism. With the metadata mapping method, the mechanism unifies the low layer data resources including relational database and XML document into data objects for the enterprise application module. The data persistence module exposes convenient persistence operation interfaces and can automatically generate concrete data access codes for the interface invoke.

The data access module provides the data access classes for data object mapping. The module builds the corresponding data access class for each data object and encapsulates the persistence operation interfaces in the classes. Then the data access layer employs the data access classes to provide various data access services.

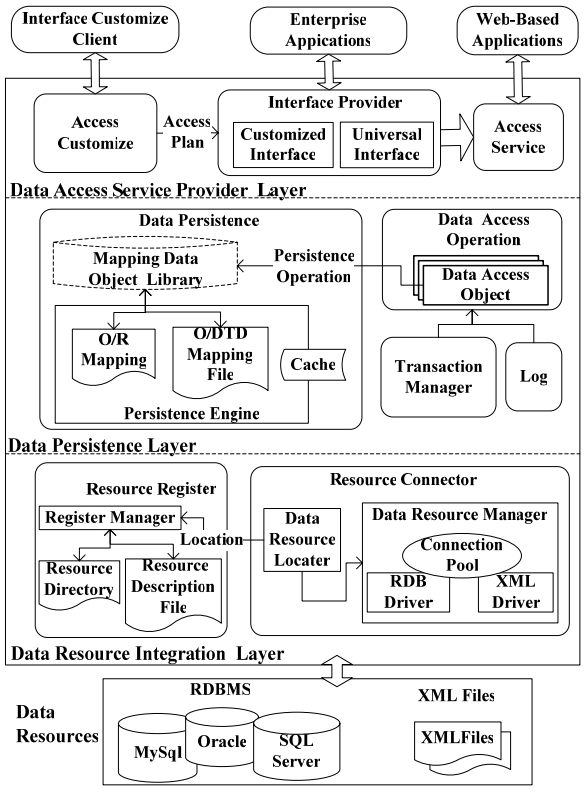

Fig. 1 EDRM architecture overview

The transaction manager module deals with the data access-related transaction issues in the enterprise-level distributed computing environment. This module adopts the two-phase commit protocol to realize the distributed transaction management, in which a transaction coordinator is introduced to ensure the integrity and consistence of data access operations. Meanwhile, the module also provides the interfaces for transaction isolation level configuration and improves the transaction management agility.

The log module is used to record and manage the data access logs. A set of specified log recording programs are included in the module to transform the messages generated by the data access operations to logs and output them into the log record files according to the predefined log levels. Then users can trace the access process details via the files.

2.1.3 Data Access Service Provider Layer

This layer provides diversified data access interfaces for enterprise applications, such as the universal data access interface, customized data access interface and web service-based data access. As shown in Figure 1, the layer is composed of access customize module, interface provider module and access service module.

The access customize module is responsible for data access plans customizing. Through parsing the access plan configuration files submitted by clients, this module can build the concrete data access plans, which are then used by the interface provider module to generate the customized data access interfaces.

The interface provider module provides data access interfaces for enterprise applications. There are always two types of interfaces. The first type is the universal data access interface, which provides the basic operations on the mapping data objects in a uniform manner, including add, delete, update, query, and so on. The second type is the customized data access interface, which is generated from the access plan script provided by the access customize module and intent to satisfy the users' personalized access requirement.

The access service module provides the web service-based data access interface. The module encapsulates the data access APIs into web services and publishes them into the UDDI registry for enterprise application to invoke.

2.2 Work Process

Via effectively integrating and managing the low layer data resources, EDRM provides simple and agile data access services for the upper enterprise applications. The temporal logic of the work process is shown in figure 2.

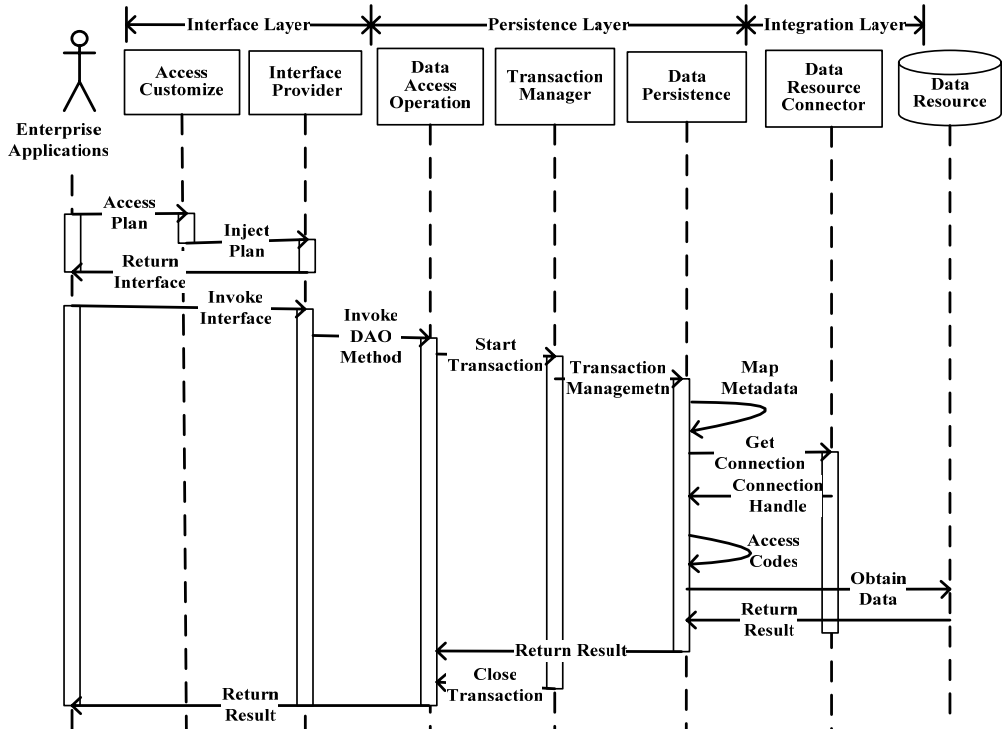

Fig. 2 Temporal logic of the work process

Step1 : The enterprise application submits the data access plan to the EDRM via the interface customize client. With the plan, the access customize module generates the data access scheme and injects it into the interface provider module. Then the interface provider module returns the customized data access interface to enterprise applications.

Step2 : The enterprise applications then access data with the customized interface returned in Step1, which employ the data access objects (DAO) provided by the data access module to complete the data access and storage process. The DAO encapsulates the data resource persistence method implemented within the data persistence module, which can obtain the data source connection handle according to the metadata mapping file, and then automatically generate the detailed data access codes to realize the access operation on physical data sources.

Step3 : The EDRM returned the result data of the access operation as object set to the enterprise applications upwards layer by layer.

During the whole process, the data access and storage details are shielded from the enterprise applications, which

can access data via EDRM framework without knowing the structure and location of the target data resources.

3. Analysis of Core Modules

3.1 Data Persistence Module

The data persistence module is contained in the data persistence layer as the main function module. This module implements a General Data Persistence method (GDP) which unifies the low layer data resources including relational database and XML document into data objects for the enterprise applications to use, and thus enables the EDRM to shield the storage structure of the physical data resources. As the detailed business logics are not involved, the GDP is a universal method and can be reused in various systems. The GDP method extends the persistence mechanism of Object-Relation Mapping (ORM) [1] and implements the persistence from in-memory data objects to relational databases and XML documents. The Figure gives the working details of the data persistence module.

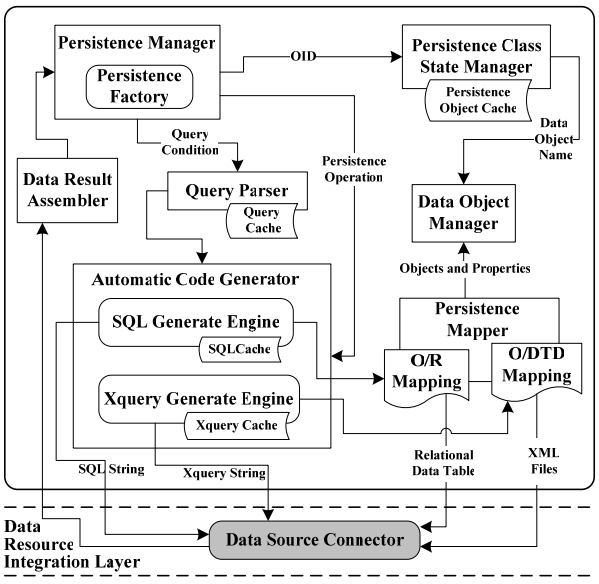

Fig. 3 Working details of the data persistence module

From figure 3, we can see that the data persistence module consists of a persistence mapper, a data object manager, a persistence manager, a persistence class state manager, an automatic code generator, a query parser, and a data result assembler, the six parts of which collaborates and together implements the GDP mechanism.

(1) Persistence mapper is the kernel part of the GDP mechanism, which is used to implement the mapping from data relations and XML schemas to objects. As shown in figure 3, the mapper maintains the mapping metadata by the mapping from data Relations to Objects (O/R) and from XML DTD files to Objects (O/DTD mapping), in which the O/R mapping maintains the mapping metadata between the in-memory data objects and relational data tables and the O/DTD mapping maintains the mapping metadata between in-memory data objects and XML schema files. The mapping metadata is consistent with the O/R and O/DTD mapping rules defined in the GDP mechanisms, which are shown in table 1 and table 2.

Table 1 O/R mapping rule

Class	Relational table
class A {String b; C c; }	Table A: Column b, Column c_fk
class C {String d; String e; }	Table C: Column d, Column e Column c_pk

Table 1 shows the mapping rules from relational data tables to in-memory data objects in the GDP mechanism. The relational data tables are mapped to classes, in which the columns of data tables are mapped to scalar properties of the classes and the primary keys and foreign keys are mapped to the pointers and reference properties. For example, data table A is mapped to the class A, in which the column b is mapped to the scalar property b and column c is mapped to the reference property about class c, as column is the foreign key of table A and primary key of table C.

Table 2 O/DTD mapping rules

Class	DTD
class A{ String b; C c; }	<!ELEMENT A (B, C)> <!ELEMENT B (#PCDATA)> <!ELEMENT C (D, E)>
class C {String d; String e; }	<!ELEMENT D (#PCDATA)> <!ELEMENT E (#PCDATA)>

Table 2 shows the mapping rules from XML schema documents to the in-memory data objects, in which the complex element types are mapped to classes and simple element types are mapped to the properties of the classes. The schema of the xml documents are defined with DTD (Document Type Definition) [8] documents. For example, the element A and C are complex type elements and mapped to class A and class C. And the element D and E are simple type elements and are mapped to properties d and e of class C.

(2) The persistence manager provides external interfaces for data objects persistence operations. The manager is designed according to the factory pattern and

constructs the persistence factory object for each data source. Then the persistence manager class (PMC) can be obtained from the factory. The PMC class maintains a data source connection handle from the properties and provides the persistence operations on the in-memory data objects mapped by the properties, such as add, delete, update, query, and so on.

(3) The persistence class state manager is used for maintaining of the data objects mapping state information. The in-memory data objects can be in either of the two states: persistent or dissociative. The objects in persistent state are stored in the persistent objects cache and assigned the unique Object Identifications (OID), which are associated with a unique record of the data sources. The objects in dissociative state are not associated with data records. When the operations on data objects are carried out, the OIDs are first used to look up the objects in persistent objects cache. If the objects are found, they are directly obtained from the cache without interacting with the physical data sources, which reduces the data source access frequency. When the properties of the data objects are changed in the cache, the persistence class state manager can automatically update the corresponding records in the data sources, which guarantees the consistence between data objects and the data source records.

(4) The automatic code generator is used to translate the persistence operations on data objects to the detailed implementation codes of concrete data access. The generator contains two engines that are SQL auto-generation engine and XQuery [10] auto-generation engine. The SQL auto-generation engine can generate the standard SQL codes for relational data source access based on the O/R mapping metadata in the persistence mapper. The XQuery auto-generation engine can generate the XQuery codes for XML data source access by parsing the O/DTD mapping metadata in the persistence mapper. The automatic code generator can chose the proper engine to generate the detailed data access codes according to the types of mapping data sources. Moreover, both of the two engines are designed with caches and can generate basic codes in cache at the initialization phase of the program, which can significantly improve the data operation efficiency.

(5) The query parser is employed to parse the query conditions for persistent data objects. The parser can split the query conditions according to the involved data sources and parses one multi-source query string into several single-source query strings. Then the parsed query strings are passed to the automatic code generator to generate the SQL codes or XQuery codes for the concrete data sources.

(6) The data result assembler is used to rebuild and return the data result as object sets. The assembler first constructs the mapping data object sets based on the mapping metadata and then assigns the class properties based on the result data.

3.2 Interface provider module

The interface provider module is included in the data access interface layer and used for providing data access interfaces for the enterprise applications. The module provides both the static universal data access interface and the dynamic customizable data access interface. Then the enterprise applications can customize the interfaces on demand and thus get more agile data access services via the EDRM framework.

There are mainly four function classes in the interface provider module, which are the access interface façade class, the data access class, the access customizing class and the access schema class. The function classes work together to provide the function of the data access interface module. For example, the access interface façade class is mainly in charge of providing various data access interfaces, which needs the data access function provided by the data access class to implement the concrete data access process, and the access customizing class is used to implement the customizable data access operations, which employs the user data access plans provided by the access schema class to process the corresponding data. The tight coupling between the classes caused by the inter-dependencies reduces the flexibility of data access interface customizing, so the Inversion Of Control (IOC) [11] pattern is introduce to build the interface provider module.

The IOC pattern can help to reduce the coupling between two collaborative classes. In the pattern, an external configuration file is employed to describe the dependency between two collaborative classes (e.g. Class A and B). When the instance of one class (e.g. Class A) is initialized, the collaborative class (e.g. Class B) is injected based on the external configuration file, which implements the dynamic configuration of the dependency between classes. With the IOC pattern, the interface provider module implements the loose coupling between function classes based on the configuration file, and thus achieves the flexibility of data access customizing. The detailed architecture of the module is shown in figure 4.

The access interface façade class and the access customizing class in the module provides the universal

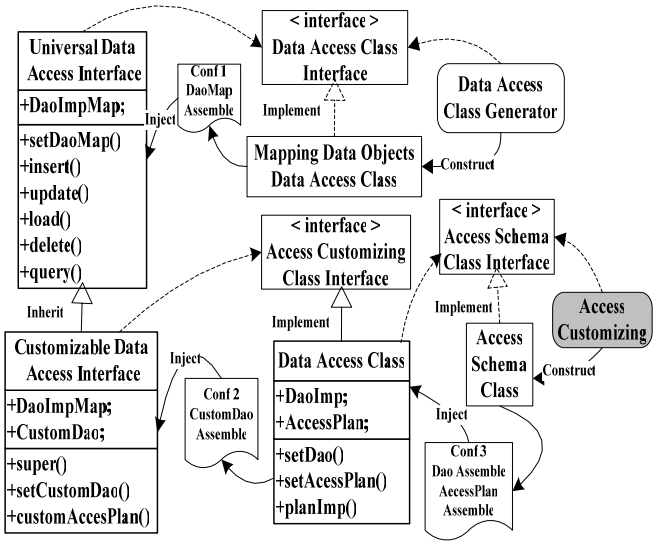

Fig. 4 Interface provider module architecture

data access interface and customizable data access interface for the enterprise applications respectively.

(1) The universal data access interface is provided by the collaboration of data access interface façade class and data access class, in which the dependency between them is described in the configuration file Conf1.When the instance of the access interface façade class is created, the interface provider module will inject the data access class as the property DaoImpMap based on the data access assembling method DaoMap in the Conf1 file. Then when the enterprise applications invoke the interface method of the access interface façade class, the concrete data access process is implemented by the data access class.

(2) The customizable data access interface is dynamically generated according to the user-defined access plan, which is based on the collaboration of the access interface façade, the access customizing class and the access schema class. As shown in figure 4, based on the data access class assembling schema Dao and the access plan assembling schema AccessPlan in the configuration file Conf3, the interface provider module first injects the user-defined data access plan into the instance of data access class as a property with the property assigning methods SetDao and setAccessPlan, which enables the instance can carried out the data access operations according to user requirement. Then, based on the configuration file Conf2, the module injects the instance of data access class into the access interface façade class. Thus the enterprise applications can achieve the customized data access operations by calling the customizing interface of the access interface façade class.

4 EDRM Application Example

The Space Equipment Auto-Testing System (SEATS) is an integrated information system for one large corporation, in which a great deal of data need to be collected from several data sources including the testing database, the device database, the parameter database, and so on. Then the collected data are provided for various applications to use, such as testing preparation, statistics query application, report generation, and so on. The data sources involved in the SEATS are distributed and heterogeneous, which includes both the relational databases, such as Oracle, MySQL and SQL Server, and the XML document sets. To effectively manage the data resources, EDRM Framework is employed to build the data platform for SEATS system.

The data platform of SEATS system is developed using Microsoft .NET framework, the main user interface of which is shown in figure 5. The area A labeled in the figure shows the data sources registered into the data platform, each of which comprises several mapping data classes. For example, the testing database consists of planet class, subsystem class, electrics testing class, research class, work project class, testing task class and so on. The area B labeled in the figure 5 indicates the basic information of the data sources, including the location URL, the source type, the user name and password, the data adapter and so on. In addition, the associating relationship between the data classes is also shown as a mapping picture. The area C marked in the figure 5 gives the class definition for each mapping data classes, which detailedly describes the attributes and member methods. With the operations on these data classes, the business applications based on the data platform can achieve the physical data access and storage without knowing any details about the scheme and location of the target data source. The area D labeled in the figure 5 lists the universal data access service interfaces and their invoke methods, which implement the basic data access operations for the business application modules, such as add, delete, update, query and so on. The area E marked in the figure 5 lists the user-customized data access services and their invoke information, which are customized by the application developers on demand via the interface customize clients. For example, in figure 5, the service named getTestProjectbyPRE is customized by the testing execution application to satisfy the requirement of getting the testing project information by the satellite type, the research phase name and the electrics testing phase name.

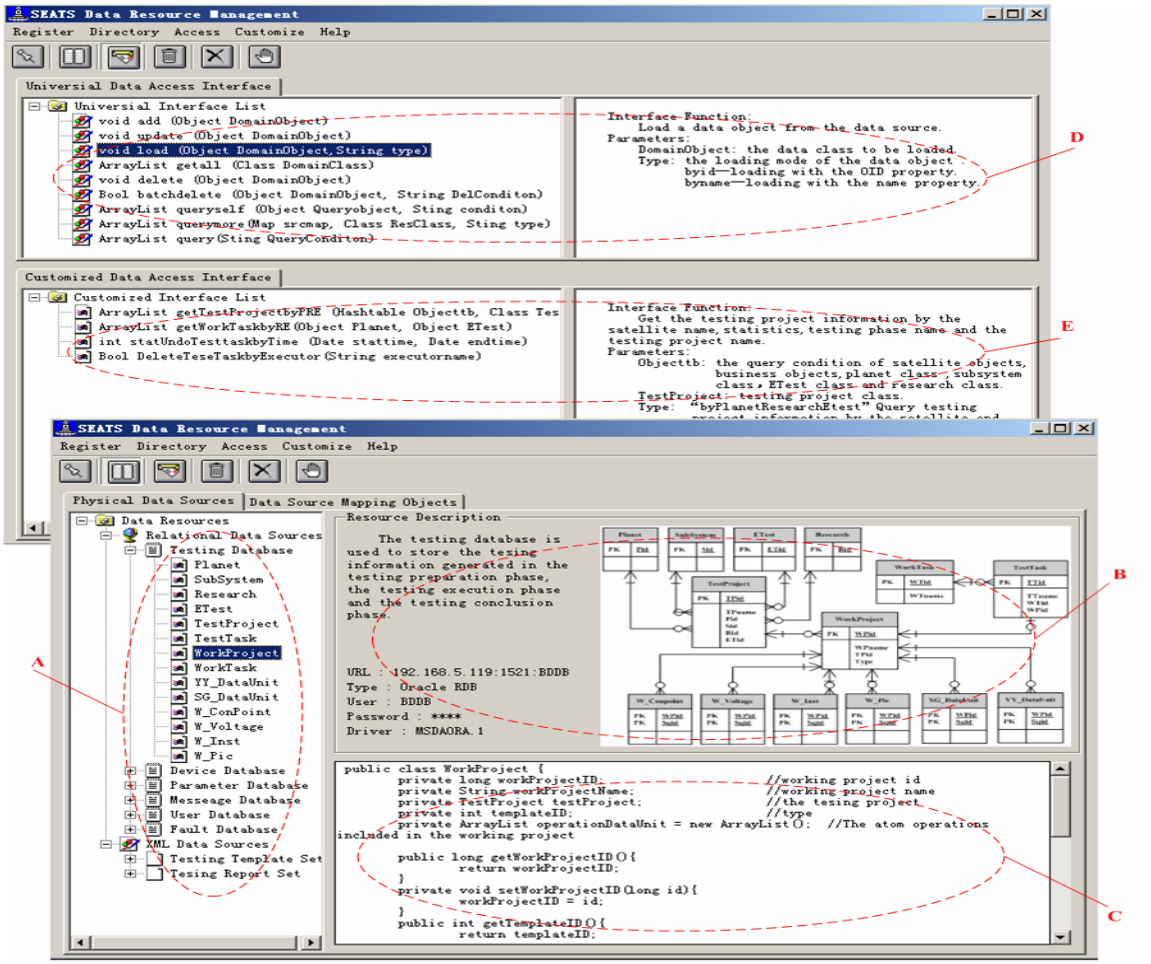

Fig.5 User interfaces of SEATS platform

Nowadays, the data platform of SEATS has been put into the practical use for several months, which shows the good performance of the EDRM-based data platform. By shielding the distributed and heterogeneous data resources and providing the function of data access service customization, the data platform well satisfies the requirement of data access and storage in the space equipment testing process.

5 Conclusion

Based on the idea of data persistence mechanism, the paper introduces a new enterprise data resource management framework (EDRM). The framework implements the general data persistence method (GDP) and data access service customization, which can encapsulate the data access logics and manage the distributed and heterogeneous data resources in a uniform way. The EDRM framework reduces the coupling between the business process logic and data storage scheme, and then provides the transparent and agile data access method. The EDRM framework has been successfully applied into the space equipment testing system and can be conveniently transplanted into other enterprise applications.

In the future research, we will further optimize the cache strategy in the EDRM framework to reduce the access efficiency lost caused by the metadata mapping process. Then the EDRM framework will better satisfy the need of mass data access in various enterprise systems.

References

[1] Jin Qiangyong, Li Guanyu, Zhang Jun. Development and Present Situation of Heterogeneous Data Integration Tech nology. Computer Engineering and Applications. 2002, 38(11): 112-114.

[2] Xiao Aihua, Wang Shilin. A Common Way Implements Data Access Object Pattern for J2EE Application. Computer Applications and Software. 2005, 22(09): 136-138.

[3] Scott W. Ambler. The Design of a Robust Persistence Layer for Relational Databases. http://www.ambysoft.com/persistenceLayer.pdf, 2000.
[4] Hibernate Reference 2.1.6. http://www.hibernate.org/hib_docs/reference/zh-cn/pdf/hibernate_reference.pdf, 2005.
[5] Zhu X M, Liu W D, Lin W M. Study of Database Connection Middleware. Computer Engineering and Applications. 2003, 39(20):176-178
[6] Anders Berglund and Scott Boag. XML Path Language (XPath)2.0. http://www.w3.org/TR/xpath20/, 2002.
[7] Scott W. Ambler. Mapping Object to Relational Database. http://www.Ambysoft.com/mappingObjects.pdf, 2000.
[8] He Yingjie,Wangshan. Mapping DTD to Relational Schema: An Approach to Preserving Data Dependency. Journal of Computer Research and Development.2004, 41(05): 868-873.
[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented software. Beijing: China Machine Press, 2002.
[10] Sun Dengfeng, Yu Xiaofeng. Research on XML Query Language. Computer Engineering. 2003, 29(13): 4-6.
[11] Martin Fowler. Inversion of Control Containers and the Dependency Injection Pattern. http://martinfowler.com/articles/in-jection.html, 2004

Chen Weiwen is a Master degree Candidate of the Beihang University (BUAA). She received the B.S. degree in Computer Science and Technology from North Eastern University in 2004. During 2004 to 2007, she studied in the National Key Lab of Software Development Environment of BUAA. Her research interests include massive information process and distributed data management.

Ma Shilong is now a professor of computer science at Beihang University. His research interests include grid computing, service-oriented computing, massive information process and so on.