Towards the Rapid Application Development Based on Predefined Frameworks

Dongjin Yu
School of Computer, Hangzhou Dianzi University, Hangzhou, China
Email: yudj@hdu.edu.cn

***Abstract*—The development of large or medium-sized domain application systems usually involves intensive workforce due to its complexity. Reuse of existing components, especially those architectural ones, could dramatically reduce the production cost and improve the quality. However, the problems related with making and adapting reusable components among different systems often inhibit the introduction of reuse. Fortunately, domain-oriented application systems, especially those data-centric ones, usually share similar behaviors no matter what they server for. This paper extracts the common behaviors existing in different domains and introduces the templates based application framework, called RADF. RADF provides application skeletons and confines domain specific coding in predefined templates of classes and configuration files. The proprietary behavior of domain specific applications could be realized via simply filling codes in these templates. RADF not only consolidates the programming paradigm and provides the supporting classes for default behaviors expected in different domains, but also allows manually extending and reassembling these supporting classes. Four cases of RADF-based development have proved that RADF helps rapid application development with significantly reduced number of manually-coded source lines.**

***Index Terms*—application systems, framework, rapid development, templates, domains, software reuse**

I. INTRODUCTION

The domain-oriented application software, or simply application, is designed to process data and support a specific organizational function or process, such as inventory management, payroll or market analysis. The development of application has accounted for the major part of global software development. With the increased severity of market competition, more and more companies nowadays demand for lower software production and maintenance costs, faster delivery of systems and increased software quality. Reuse-based software engineering is a software engineering strategy where the development process is geared to reusing existing software. For reuse-based approach, software components have to be discovered in library, understood and, sometimes, adapted to work in a new environment. However, due to the business diversities, well defined components with high reusability are always hard to be created, especially when reusing components across domains.

Rapid Application Development (RAD) is another approach to software development aimed at fast delivery of applications. It often involves the use of database programming and development support tools such as screen and report generators. RAD is most suitable for generating of repeated routine codes, but not for the production of more complicated structural or business logic codes.

The domain-oriented application software usually contains a great number of lines of codes, many of which merely handle the data operations such as registering, querying and updating. Moreover, the interaction process is almost similar, no matter which line of business it deals with. For instance, data should be located before they are ready to be modified, and could not be allowed to be deleted unless it is confirmed.

Software architecture involves the structure and organization by which system components and subsystems interact to form systems and the properties of systems that can best be designed and analyzed at the system level [1]. Good software architecture is critically important for successful software development. It has a profound influence on all technical decisions [2]. The Rapid Application Development Framework, or simply RADF, presented in this paper aims to make application development more efficiently and effectively by means of architectural reuse. With the help of predefined architectural skeletons and coding templates, RADF consolidates the programming paradigm, but allows manually coding to satisfy domain specific requirements. If the development is confined within the structure and conventions specified by RADF, the programming process is nothing more than filling out templates of 7 supporting class and 3 configuration files. In this way, RADF ensures the consistency of application framework and the quality of application itself. More importantly, RADF dramatically reduces manually-coded source lines, which eventually increases the software development productivity.

The rest of the paper is organized in the following manner. Section 2 analyzes the common behaviors usually occurred in data-centric application systems. Section 3 introduces the framework of RADF in detail, especially its unique features compared with Struts. The programming paradigm and related process model when using RADF are illustrated in section 4 and 5 respectively. Section 6 presents successfully implemented cases. After the evaluation of RADF discussed in Section 7, Section 8 offers related works. Finally, the last section provides concluding remarks and future research directions.

doi:10.4304/jsw.6.9.1795-1804

II. FREQUENT OCCURRED BEHAVIORAL SKELETONS OF APPLICAION SYSTEMS

The application system that provides business services is usually composed of a number of separate programs which run on different nodes. Although they are designed to help people perform diverse types of works, application systems, especially those data-centric systems, share similar behaviors. In other words, no matter which lines of business they server for, the operation processes fall inevitably in data manipulation such as record creating, retrieving, updating and deleting iteratively, although the business logics behind could be totally different.

RADF summarizes 17 pieces of skeletons of behavioral scenarios which usually occur under different contexts of business domains (Table 1). In practice, most domain-oriented business logics are expected to be accomplished inside the edit-like or update-like scenarios by changing data status or triggering stored procedures. Instead of realizing specific behavioral scenarios, RADF provides programming templates for these scenarios. When developing real application systems, codes could be filled in these templates to fulfill specific domain requirements.

TABLE I. FREQUENT OCCURRED BEHAVIORAL SKELETONS SUMMARIZED FROM DIFFERENT APPLICATION SYSTEMS

ID	Behavioral Skeleton Name	Explanation
CRTMID	Create with manual ID	Add single record with manually selected ID. The ID duplicity should be avoided.
CRTAID	Create with auto-ID	Add single record with auto generated ID.
CRTBMI	Create in batch with manual IDs	Add multiple records with manually selected IDs. The ID duplicity should be avoided.
CRTBAI	Create in batch with auto-IDs	Add multiple records with auto generated IDs.
CRTMND	Create Master and Detail	Add single master record and its related detail records.
LCTSRI	Locate Single Record with ID	Locate the only record matched with the given ID.
FLTREC	Filter Records	Search and present overview list of one or more records matched with given conditions.
QRYREC	Query Records	Filter records and present the detail of specific one when required.
UPDREC	Update Records	Filter records and modify/delete the specific one when required.
EDTREC	Edit Records	Filter records and present the detail of specific one for modifying when required.
DELREC	Delete Records	Filter records and delete the specific one when required.
EDTBCH	Edit in Batch	Filter records and edit together.
DELBCH	Delete in Batch	Filter records and delete multiple chosen records together.
EDTSEQ	Edit in Sequence	Locate one record and edit its fields step by step (page by page).
SUMREC	Summarize Records	List the required fields with aggregation in groups and whole.
EDTMND	Master and Detail Edit	Locate the master and edit its detail.
QRYMND	Master and Detail Query	Locate the master and present the master and all its detail.

The behavioral skeletons given in Table 1 could be further described with UML activity diagrams. Fig. 1 gives the examples of FLTREC, QRYREC, UPDREC and CRTMID, where the swim lanes are used to separate manual actions or system actions.

Under certain circumstances, some scenarios are invoked as sub activities of others. For example, QRYREC and UPDREC invoke FLTREC, denoted by rake symbols in Fig. 1.

III. RADF: THE RAPID APPLICATION DEVELOPMENT FRAMEWORK

Generally speaking, the application framework is implemented as a set of concrete and abstract classes that are specialized and instantiated to create an application [3]. RADF, as one of well organized application frameworks, provides the general structure that would form the basis of a family of applications. Moreover, different with traditional frameworks such as Struts (http://struts.apache.org/) and Spring (http://www.springsource.org/), RADF goes one more step forward. It presents the programming templates and coding conventions. In other words, RADF consists of frozen spots and hot spots.

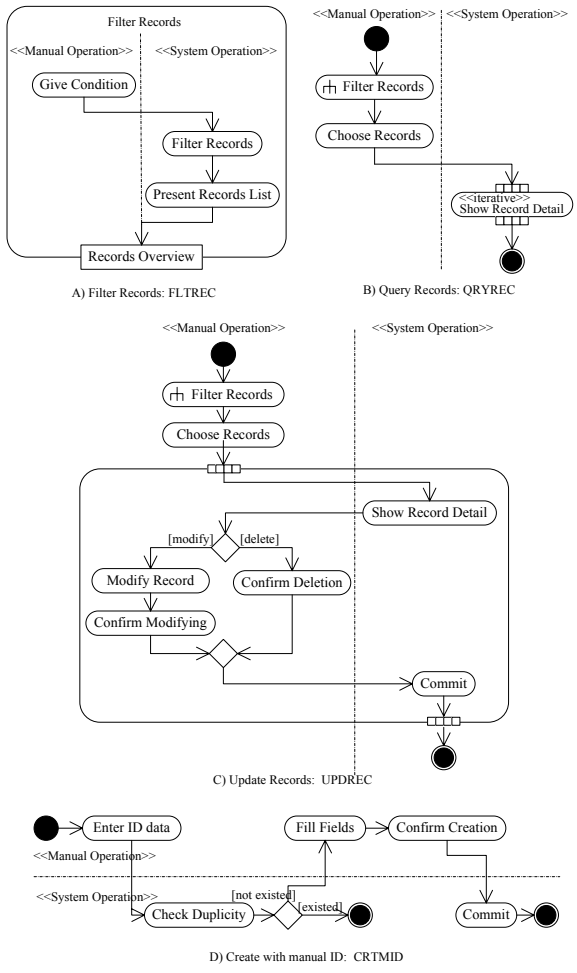

Figure 1. Requirements of FLTREC, QRYREC, UPDREC and CRTMID illustrated in UML activity diagrams.

The frozen spots define the basic components and the relationships between them, which remain unchanged (frozen) in any instantiation of the application framework. On the other hand, hot spots represent those parts of the software framework that relates to individual software systems. Hot spots are designed to be specific and can be adapted to the needs of the domains.

RADF lies between standard JavaEE/J2EE platforms and specific application systems (Fig. 2). More specifically, RADF runs directly on JavaEE/J2EE platforms, but realizes the application skeleton and fulfills the general application requirements.

Different with some existing code generators, development based on RADF allows manually coding or even extending the framework to address domain problems, which gives programmers much more flexibilities. In addition, development based on RADF does far more than traditional component based software development. By consolidating the architecture through predefined templates, RADF guarantees the quality of applications to be developed.

Similar to many traditional frameworks, RADF adapts the well-known Model-View-Controller structure. However, RADF has some different features (Fig. 3).

A. The View Layer

The client, or the view layer, could either run in Web browsers or take the form of desktop application. For desktop clients, user interactions could be issued through button clicks or menu selections, whereas for Web applications, they appear as GET and POST HTTP requests. Both types of clients simply communicate with RADF via HTTP. For desktop clients, however, the requests and responses would be encapsulated in SOAP-formatted messages. Thus, the message parsers are indispensable on both client side and server side for desktop applications.

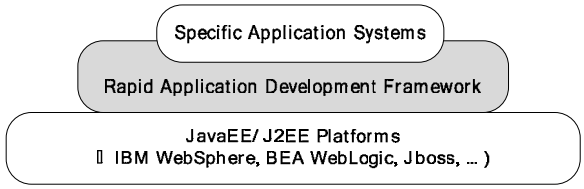

Figure 2. JavaEE platforms, RADF, and specific application systems.

RADF offers general-purposes JSP tag library, which helps reusing GUI components. Typical tags include Pager Tag rendering selected records page by page, Operator Tag showing current operator, Code List Tag giving the name list mapped to different codes, and etc. These tags are frequent reused among different applications and make their graphic interfaces look similar.

Besides, RADF provides the invocation chain for all filters that perform filtering tasks on the request to a resource. Filters that have been realized in RADF are Authentication Filters, Logging Filters, Encryption Filters and Auditing Filters.

B. The Model Layer

The business logics are encapsulated in Business Process Objects, or simply BPOs, which are reflected by the embedded service locator in RADF. Moreover, BPOs could be arranged in the tree-like calling chain, where fine grained objects are assembled to accomplish coarse grained logic. RADF has two types of BPOs, i.e. general BPOs and BSImp's.

General BPOs deal with indivisible logic, whereas BSImp's assemble related general BPOs to fulfill complicated logic. In general, BSImp's implement Façade interfaces, invoked by action methods. Transaction integrity is guaranteed inside one single method of BPOs. However, recursive transactions or transactions across multiple requests are not allowed.

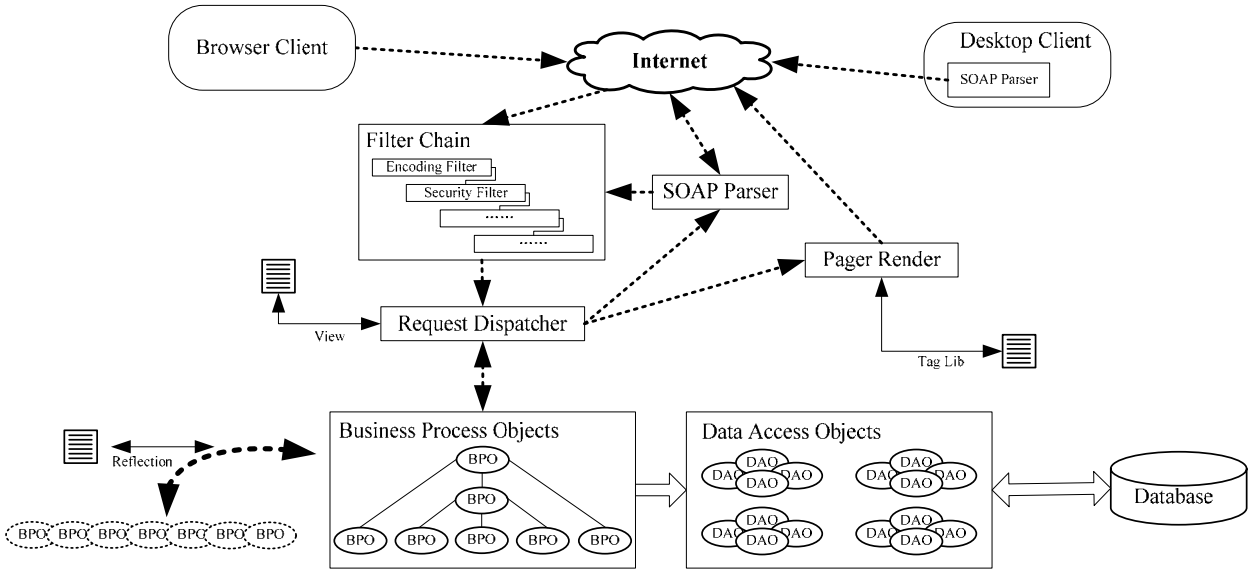

Figure 3. RADF: the Rapid Application Development Framework.

Data access logics are fulfilled by predefined Data Access Objects, or simply DAOs. Different with BPOs, DAOs are entity-oriented, which roughly means that each entity corresponds one DAO Supporting class containing data inserting, updating, deleting and querying methods. Moreover, DAO Supporting classes could be further extended to support complicated data access logic, such as querying records from multiple tables.

SQL statements used for data manipulation could be configured in property files, where delimiters are introduced to represent placeholders of query parameters. Under simplest circumstances, RADF-based development means no more than configure the SQL templates and fill the required parameters.

C. The Control Layer

The controller in RADF translates interactions with the view into actions to be performed by the model, which may include activating business processes or changing the state of the model. Based on the user interactions and the results of the model actions, the controller responds by selecting an appropriate view.

In RADF, the control layer interacts with the model layer by calling methods of façade, the interface of BPO. The invoking parameters are packaged in the object of RequestMessage while the results are packaged in the object of ResponseMessage. RequestMessage includes the head part preserving session information such as current operator, and the body part preserving entity information in hashed map. Meanwhile, the head part of ResponseMessage is mainly used to preserve the calling status and error messages (if any). It is not necessary for developers to resolve ResponseMessage, which is automatically done by RADF. Like RequestMessage, the information in the body part of ResponseMessage, as the returned result, is preserved in the form of hashed map. Therefore, through pre-defined RequestMessage and ResponseMessage, RADF consolidates the interface between the Actions and BPOs.

D. Start-up of RADF

When RADF-based applications boot up, global parameters, business services and database connections are automatically loaded and configured in Servlet contexts. Later when necessary, they are fetched directly from running contexts, instead of persistent files or databases. RADF allows hot deployments of global code-name pairs and parameters, meaning updated values would propagate to Servlet contexts immediately when administrators have done the modification.

Besides, concrete BPOs are located and loaded when RADF-based applications start up. In this way, BPOs are ready to be used in the memory if requested in future.

IV. PROGRAMMING PARADIGMS IN RADF

The development process, especially for development of data-centric applications, could be significantly eased if RADF is employed. RADF-based development only requires creating 7 classes and configuring existent 3 global configuration files no matter what specific domain requirements are. Moreover, both the classes and the configuration files have their predefined templates which stereotype and thus simplify the development of application.

The classes involved during the process of RADF-based development are shown in Fig. 4.

1) ActionForm: The POJO class which represents an HTML form that the user interacts with over one or more pages. String-formatted properties should be provided to hold states of the form with getters and setters to access them. ActionForms can be stored in either the session or request scope.

2) BEO: The class which stores fields of business entities and IDs mapped to actual SQL statements for database manipulations. Different with ActionForm, fields of BEO could have variable types such as Date, Double, or Integer.

3) Façade: The interface which defines all the business methods to be implemented. BSImp or BPO implementing Façade are reflected and loaded while booting up the application.

4) BSImp: The business service class which accomplishes domain functions. Under certain circumstances, BSImp would assemble several BPOs to accomplish more complicated functions.

5) BPO: The business service class which deals with atomic domain functions. BPO is optional and its behavior could be modeled together with BSImp if the related domain function is simple enough.

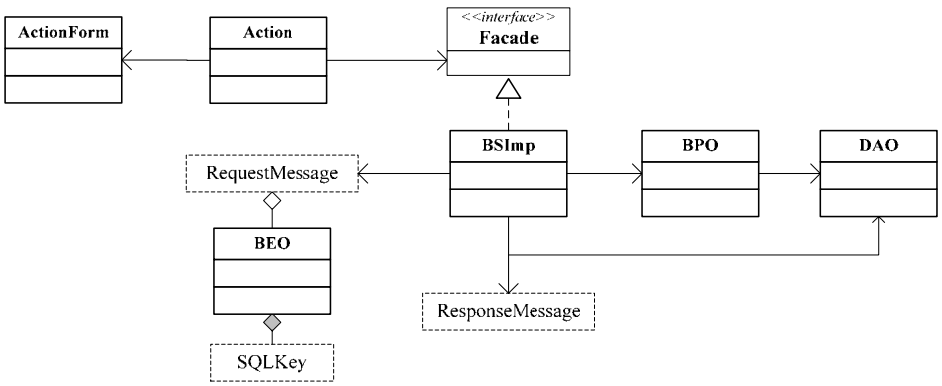

Figure 4. The classes involved when programming based on RADF. Dotted rectangles represent RADF-predefined classes, while solid rectangles represent need-to-create classes according to domain contexts.

6) Action: The control class which processes a request, via its execute method, and return an ActionForward object that identifies where control should be forwarded (e.g. a JSP, Tile definition, Velocity template, or another Action) to provide the appropriate response. Action populates BEOs with ActionForm, wraps BEO into RequestMessage, calls the required method in BSImp with RequestMessage and gets the returning message in ResponseMessage.

7) DAO: The class which provides an abstract interface to some type of database or persistence mechanism, related to the basic data manipulation such as adding, deletion, modifying and querying.

RADF normalizes the package structure of applications. Thus, RADF-based applications could be divided into several sub-applications according to different sub-domains, and each sub-application contains 7 packages of classes with names complying with predefined conventions (Table 2).

Besides above 7 classes, some configuration should be made in following 3 existent configuration files.

1) Action mapping file: The XML-formatted file which defines how each request URI should be mapped to an appropriate Action, similar to struts-config.xml in Struts.

2) SQL statements file: The property file which maps the SQL statement ID with actual SQL statement itself. With the given ID and arguments, the whole statement could be located, rendered and submitted.

3) Service mapping file: The XML-formatted file which pairs interfaces of façades with corresponding concrete classes. If necessary, concrete classes could be simply replaced with newer versions if the latter meets with interfaces of façade.

V. PROCESS MODELS

Two types of process model could be adopted during RADF-based development: Moderate Model and Fast Model. Which process model is actually more suitable depends on the team structure, and to which extent team members are familiar with RADF.

A. Moderate Process Model

For the Moderate Process Model, the priority is given to the simple and error-prone functions units existed in application bottom. A steady bottom layer will greatly reduce subsequent testing time. General speaking, the Moderate model could be introduced if the majority of team members are not familiar with RADF or have limited development experience.

Step 1: Design the conceptual data model and write BEO for each business entity.

Step 2: For each BEO, write following five fixed data accessing methods in the related DAO: doCreate(), doDelete(), doUpdate(), getAllRecords(), and doFind().

Step 3: Write BPO for each entity class and connect BPO with related DAO.

Step 4: Define Façade interface and all action classes.

Step 5: Write BSImp classes which implement Façades, and modify BPO if necessary.

Step 6: Implement Action classes.

Step 7: Elaborate front pages and connect them with relative action methods.

Among above steps, the first three are the most important ones since the well-programmed BEOs and BPOs would lead to fewer errors afterwards. Meanwhile, Step 5 and Step 6 are comparatively complicated. Step 7, however, involves large amount of workload, although it is not difficult.

Generally speaking, it is quite easy to control each step in Moderate model. Moreover, the application bottom could be solidly built without much deviation. However, Moderate model spends much more time, compared with the Fast Model illustrated later.

B. Fast Process Model

Fast Process Model focuses on parallel operations in order to reduce potentially duplicated activities. Nevertheless, it challenges both team leader and team members. Fast Process Model is most suitable if the majority of team members is familiar with RADF or has rich development experience.

In Fast Process Model, the project team should be divided into two groups. The members from first group are expected to be quite familiar with business domain or good at designing, while the others in second group could be less experienced. The number of members in first group would probably be double that of in second group.

Step 1: The first group writes Action classes according to the page documents, and defines necessary Façades. At the same time, the second group writes the entity classes. Entity classes should be finished earlier than Action classes, because entity classes are expected to be used in Action classes.

Step 2: The first group defines BSImp skeletons (not fully implemented), while the second group writes the five database access methods as doCreate(), doDelete(), doUpdate(), getAllRecords(), and doFind(), and then defines BPO skeletons.

TABLE II. CONVENTIONS OF PACKAGE STRUCTURES IN RADF

Package	Classes in Package
org.radf.apps.[sub-application-name].form	POJO class with names of [XXX]Form
org.radf.apps.[sub-application-name].action	Extends org.radf.plat.util.action.ActionSupport, with names of [XXX]Action
org.radf.apps.[sub-application-name].facade	Extends org.radf.plat.util.FacadeSupport, with names of [XXX]Façade
org.radf.apps.[sub-application-name].imp	Extends org.radf.plat.util.imp.IMPSupport, with names of [XXX]IMP
org.radf.apps.[sub-application-name].bpo	Extends org.radf.plat.util.bpo.BPOSupport, with names of [XXX]BPO
org.radf.apps.[sub-application-name].dao	Extends org.radf.plat.util.dao.DAOSupport, with names of [XXX]DAO
org.radf.apps.[sub-application-name].entity	Extends org.radf.plat.util.entity.EntitySupport, with names of [XXX]Entity

Step 3: The first group connects front pages with Action methods for the desired control flows. Meanwhile, the second group elaborates BSImp's and BPOs as needed.

In Fast Process Model, the two groups advance from different sides toward BSImp and BPO, which are the core of application. The flexible allocation of human resources according to individual's ability and experience improves the team's efficiency, because the experienced members are allocated to complete complicated tasks and the others are allocated to complete tasks with heavy workload but less difficulty.

VI. CASE STUDIES

Multiple templates, such as action templates, BPO templates, façade templates, entity templates and SQL templates are involved during the process of RADF-based development. Codes are added in the templates to meet with domain requirements. The following is the programming case for manipulating Configuration Items in the development of ITIL-compliant Maintenance Platform (ITILMP) based on RADF. The management of configuration is one of the key functions in ITILMP. According to Information Technology Infrastructure Library, Configuration Management controls not only the aspects covered by Asset Management, but also identifies the nature and importance of relationships between assets.

In ITILMP, Records of Configuration Items are stored in table TBLITSMCI containing fields such as ID, name, type, location, version, description, status, and supplier. The class of Tblitsmci, as the entity class corresponding to table of TBLITSMCI, keeps instances of configuration items and SQL keys mapped to actual SQL statements.

The interface of CIFacade, as one of the façades in the application, encapsulates all necessary methods dealing with manipulation of Configuration Items. The following code fragment gives the definition of CIFacade, created by filling underlined codes in the façade template. Here, the non-underlined codes are from the Façade template

```
public interface CIFacade {
    public ResponseMessage printCI(
        RequestMessage request);
    public ResponseMessage saveCI(
        RequestMessage request);
    public ResponseMessage modifyCI(
        RequestMessage request);
    public ResponseMessage deleteCI(
        RequestMessage request);
}
```

The class of CIImp realizes CIFacade to provide the actual implementation. As an illustrative example, the codes for saving one configuration item record are listed as the following. Here, tblitsmci_select and tblitsmci_insert are mapped to actual SQL statements of querying and adding records in SQL property file respectively. Again, only those underlined codes should be filled according to the BPO template.

```
public ResponseMessage saveCI(RequestMessage
reqenv) throws AppException {
    ResponseMessage resenv = new
ResponseMessage();
    HashMap map = (HashMap) reqenv.getBody();
    //get business entity object
    Tblitsmci tblitsmci = (Tblitsmci) map.get("beo");
     Connection con = DBUtil.getConnection();
    DBUtil.beginTrans(con);
     if (null==tblitsmci.getCiid()
        || "".equals(tblitsmci.getCiid())) {
        throw new AppException(
            "Null ID Not Allowed!");
    }
    //set SQL statement ID of record querying
     tblitsmci.setFileKey("tblitsmci_select");
     if (getCount(con, tblitsmci, 0) > 0) {
        throw new AppException("Same ID Existed!");
    }
    //set SQL statement ID of record adding
    tblitsmci.setFileKey("tblitsmci_insert");
    store(con, tblitsmci, null, 0);
    DBUtil.commit(con);
    HashMap retmap = new HashMap();
    retmap.put("ciid", tblitsmci.getCiid());
    retmap.put("workString",
        "Inserted one CI record!");
    resenv.setBody(retmap);
    return resenv;
}
```

Similar to action classes in Struts, the class of CIAction parses the form bean, delegates the control flow to the appropriate methods in CIImp and determines the next page. Below is the code fragment pertinent to saving request, with the non-underlined codes from the Action template.

```
public ActionForward saveNewCI(ActionMapping
mapping, ActionForm form, HttpServletRequest request,
HttpServletResponse res) throws Exception {
    CIForm cf = (CIForm)form;
    Tblitsmci tblitsmci = new Tblitsmci();
    //transfer properties  from ActionForm to BEO
    ClassHelper.copyProperties(cf, tblitsmci);
    //locate the service
    CIFacade facade
        = (CIFacade) getService("CIFacade");
    RequestMessage requestMessage
        = new RequestMessage();
    EventResponse returnValue = new
EventResponse();
    HashMap<String, Object> mapRequest
        = new HashMap<String, Object>( );
    mapRequest.put("beo", tblitsmci);
    requestMessage.setBody(mapRequest);
    //invoke method in the BSImp
    ResponseMessage resEnv
        = facade.saveCI(requestMessage);
    returnValue = processRevt(resEnv);
    if (returnValue.isSucessFlag()){
        super.saveSuccessfulMsg(request,
            "Configuration Item added successfully!");
        //determines the next successful page
```

```
        return mapping.findForward("newci");
    }else {
        //determines the next failure page
        return mapping.findForward("backspace");
    }
}
```

The SQL statements of *tblitsmci_select* and *tblitsmci_insert* in SQL property file are as following. Here, both : and ! are delimiters to represent placeholders of query parameters.

```
tblitsmci_select
    = select * from tblitsmci where ciid=:ciid!
tblitsmci_insert
    = insert into tblitsmci (ciid, ciname, citype,
        cilocation, cidescription, cistate, civersion,
        cisupplier) values (:ciid!, :ciname!, :citype!,
        :cilocation!, :cidescription!, :cistate!,
        :civersion!, :cisupplier!)
```

VII. EVALUATION OF RADF-BASED DEVELOPMENT

RADF has been successfully employed to reconstruct several applications formally based on Struts. In addition to ITIL-compliant Maintenance Platform (ITILMP) introduced in previous section, other recent examples include Labor Market Management Information System (LMMIS), Audiphones Manufacturing Management Information System (AMMIS) and Labor Market Decision Support System (LMDSS). LMMIS, ITILMP and AMMIS are data-centric applications which mainly focus on data manipulation, while LMDSS is model-centric and therefore involves intensive computation. The survey was conducted to compare the number of classes and the number of lines of code needed to be written with Struts 1.1 or with RADF. It is generally accepted that an application with fewer number of classes or number of lines of code indicates that it is relatively easier to be built, and vice verse. Fig 5 gives the numbers of classes and lines of codes of 4 applications based on Struts 1.1 and RADF. Automatically generated POJO, such as form beans and entity classes, are not counted in.

More accurate comparisons could be further made among manually coded KLOC (thousands of lines of code), manually filled KLOC, and auto generated KLOC respectively. If two manually filled lines are assumed to be equal to one manually coded line plus one auto generated line when considering programming workloads, the actual cut-off rates of source lines could be calculated as following.

$$CR = M_a / (M_m + M_a)$$
$$= (M_g + 0.5M_f) / (M_c + M_f + M_g)$$

Here, M_m stands for the number of *equivalent* manually coded lines, M_a for *equivalent* auto generated lines, M_c for manually coded lines, M_f for manually filled lines, and M_g for auto generated lines.

The survey finds that total lines of codes of RADF-based development are not greatly reduced compared with that of Struts-based development. However, if following RADF, approximately 27.10% codes of data-centric applications could be generated automatically, although the difference is not so obvious for model-centric applications such as LMDSS. Table 3 gives the cut-off rates of above 4 applications based on RADF.

VIII. RELATED WORKS

Software reuse enables developers to leverage past accomplishments and facilitates significant improvements in software productivity and quality [4]. Software reuse has been practiced since programming began. Active areas of reuse research in the past twenty years include reuse libraries, domain engineering methods and tools, reuse design, design patterns, domain specific software architecture, generators, and measurement [5].

An important approach to reuse and one tightly coupled to the domain engineering process is generative reuse [6]. Generative reuse is done by encoding domain knowledge and relevant system building knowledge into a domain specific application generator. New systems in the domain are created by writing specifications for them in a domain specific specification language. The generator then translates the specification into code for the new system. In this sense, RADF is roughly for generative software reuse. However, it is more like a framework than a tool, and it allows manual intervention and thus extends flexibility.

As for frameworks, there exist hundreds or even thousands employed in domain application development. As many applications run on JavaEE/J2EE platform and function from web sites, following discussion mainly focus on Java Web frameworks.

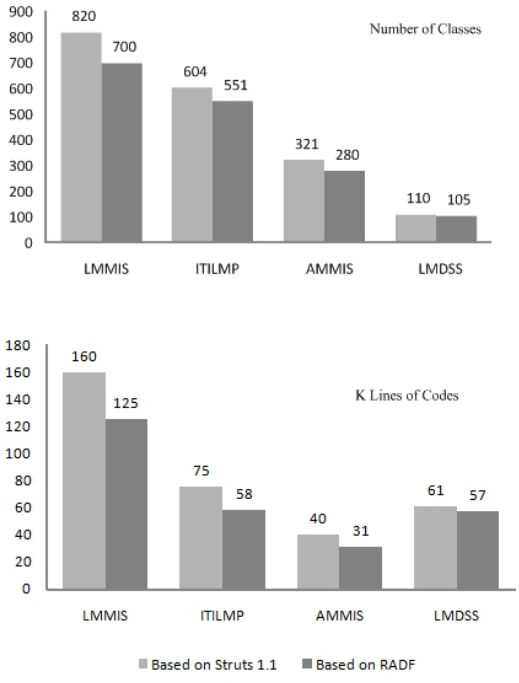

Figure 5. The numbers of classes and lines of codes of 4 applications based on Struts 1.1 and RADF.

TABLE III. NUMBER OF MANUALLY-CODED AND AUTO-GENERATED SOURCE LINES OF APPLICATIONS BASED ON RADF.

App.	Manually Coded KLOC (Mc)	Manually Filled KLOC (Mf)	Auto Generated KLOC (Mg)	Cut-off Rate
ITILMP	38.96	5.81	13.40	28.03%
LMMIS	90.31	16.19	18.59	21.33%
AMMIS	18.70	4.23	7.65	31.94%
[Average]	***49.32***	***8.74***	***13.21***	***27.10%***
LMDSS	50.07	2.76	4.42	10.13%

Shan and Hua classified various existing Java Web application frameworks into 5 categories as Request-based Framework, Component-based Framework, Hybrid Framework, Meta Framework and RIA-based Framework [7]. According to the taxonomy, the Request-based Framework, like Struts or Beehive, uses controllers and actions that directly handle incoming requests, whereas the Component-based Framework, like JSF and Tapestry, abstracts the internals of the request handling and encapsulates the logic into reusable components. The Hybrid Framework combines the features of both request-based and component-based frameworks. The Meta Framework, or the framework of frameworks, has a set of core interfaces for common services and a highly extensible backbone for integrating components and services. Typical Meta Frameworks include Spring and Keel. Finally, the RIA-based Framework uses client-side container models, like Direct Web Remoting, or Echo2.

Almost all these Java Web frameworks, no matter which category they belong to, have their presentation layers and model layers. As for the presentation layer, many researchers focus on the fast development of web application. Page-centric Web development usually structures an application into individual scripts, each responsible for processing a user request and generating a response. This imposes a go-to hardwiring of the control flow because each page must know what comes next. Another issue with Web presentation frameworks is the limited support they provide for composing multiple parts on the same page. The Seaside framework (www.seaside.st) [8] provides a uniform, pure object-oriented view for Web applications. By exploiting Smalltalk's dynamic nature and reflective capabilities, Seaside offers ways to have multiple control flows active simultaneously on the same page. WISBuilder [9], based on Generative Programming paradigms, is another worth-mentioned framework which facilitates the development of Web-based Information Systems. Systems based on WISBuilder could be specified by two XML-based languages: WSML language that uses XSL to transform the views, navigation and user's access into HTML, and WAML language that uses XSL to transform the applications specification into HTML, embedding PHP and JavaScript instructions for executing specific tasks.

Interactive web forms can be created with Asynchronous Javascript and XML (Ajax) programming. However, Ajax programming is complex in a way that the model-view-controller (MVC) code is not clearly separated. A variety of frameworks have been developed to simplify Ajax programming. Google Web Toolkit, ASP.NET Ajax, Yahoo UI Library, Dojo, Prototype, and DWR are among the popular list. These frameworks typically provide the JavaScript engines and server libraries to save programming efforts in coding sophisticated web UIs and Ajax messaging. In addition to above popular Ajax frameworks, there still exist some other web form frameworks without Ajax programming. For instance, Webformer, presented in [10], focuses on RAD of web forms using its Web Form Application Language that provides MVC specification for web forms, and the Webformer Compiler that generates the runtimes of the web forms from their models. In [11], a web framework is provided to support multimodal web applications.

As for the model layer, a domain model, represented by specific domain classes, is usually needed to describe the core data and their behavior. When a model is well designed, a developer can then focus more rapidly on views of the software, since they are what users care about the most. In [12], Michał Lentner and Kazimierz Subieta introduce an object-oriented environment for rapid database application development, called ODRA. Based on the declarative high level object-oriented language SBQL (Stack-Based Query Language), ODRA provides functionality common to the variety of popular technologies in a single universal, easy-to-learn application programming environment. Domain Model Lite, described in [13], is another framework that facilitates the definition and the use of graphical domain models in a restricted way, which minimizes the number of decisions that a domain model designer must make.

Framework-based software development has been proven a useful technique to develop an application. However, the development of a large application framework itself is considered complex due to its large size and the vague requirements. Methodologies or patterns such as Inversion of Control, Aspect Oriented Programming are frequently used when developing application frameworks. High performance is another key quality that general application frameworks care most. In order to balance flexibility and performance qualities, which are in trade-off, many projects need to analyze the performance of the employed application framework. Budi Kurniawan and Jingling Xue compare and evaluate the ease of application development and the performance of the three design models (Model 2, Struts, and JSF) by building three versions of an online store application using each of the three design models, respectively [14]. They find that it is most rapid to build Web applications using JSF. Model 2 applications are the least rapid but give the best performance. Struts applications sit in the middle of the other two design models in both

comparisons. Many methods could be employed to evaluate the performance of framework. For instance, in [15], Yonghwan Lee et al. show how much the performance of framework is affected by changing schemes of the framework at the cost of flexibility.

The main objective of application framework is to promote the reuse of both design and code in the development of new applications. The use of a framework will significantly decrease the amount of time taken for developing new applications. However, new framework users often find that the documentation provided along with a framework is usually not very effective for new users. In order to come up with new solutions to new problems posed by the framework users, Hajar Mat Jani et al. propose documentation approach which reuses previously documented cases [16]. This requires the documentation system to be capable of understanding and learning from past experiences.

Finally, many specific application frameworks have been developed in certain domains. For instance, Weidong Liao and Benjamin J. Koonse present the JMEI (Java Mathematical Engine Interface) which is a layered Java application framework for providing mathematical computing services for the Internet/Web [17]. Fortunata, as a wiki-based framework, could facilitate the management and publication of semantic data in web-based applications [18]. Other examples include application frameworks used to construct manufacturing systems [19], cooperative design systems [20], availability management system [21], decision support systems [22], and so on.

IX. CONCLUSIONS

The ultimate purpose of domain engineering and systematic software reuse is to improve the quality of the products and services that a company provides and, thereby, maximize profits [5]. Framework-based application development has been proven a useful technique to develop the domain oriented applications. RADF provides application skeletons and confines domain specific coding in predefined templates of classes and configuration files. The proprietary behavior of domain-oriented applications could be therefore realized via simply filling out these templates. RADF not only consolidates the programming paradigm and provides the supporting classes for default behaviors expected in different domain-oriented applications, but also allows manually extending and reassembling these supporting classes. In this way, RADF helps the rapid development of application systems with possible consistent qualities.

In 2006, Mary Shaw and Paul Clements pointed that previous 15 years or so had been software architecture's golden age [23]. RADF references many other well-known frameworks flourishing in this period. However, we are still considering upgrading RADF. Future work mainly focuses on the following three issues.

1) Promotion of performance.

RADF relies heavily on reflections of classes when instantiating BSImp's. Current version of RADF reflects all BSImp's when it boots up. Theoretically, reflections would decrease the performance. For large applications, however, it would take intolerable time to get RADF-based applications started. To make things worse, the reflected classes would then keep enormous space all the time. To promote the performance of RADF, we are currently considering to instantiate BSImp's in multiple batches.

2) Migration to Service-oriented Architecture

Service-oriented computing promotes the idea of assembling application components into a network of services that can be loosely coupled to create flexible, dynamic business processes and agile applications that span organizations and computing platforms. We are now upgrading RADF to support the interactions of SCA-compliant services with the help of Apache Tuscany.

3) Current development of applications based on RADF requires filling codes in generated templates of sources. It is likely that programmers would carelessly misuse the templates, since both filled codes and generated codes are mixed together. Accordingly, the RADF plug-in in Eclipse IDE is now being designed in order to facilitate the RADF-based development process.

ACKNOWLEDGMENT

The work is supported by the Foundation of Key Science and Technology Projects (No. 2008C11099-1), the Natural Science Foundation (No. Y6090312) of Zhejiang province of China, and the Science Foundation of the Hangzhou Dianzi University (No. KYS055608069).

Many have contributed to the work demonstrated in this paper. The requirement and initial idea was developed by Jianwei Shao in Zhejiang Institute of Computing Technology. Pei Zhang and Rongyong Ruan helped construct the final version of RDAF. Jiangchen Qiu presented many valuable suggestions gathered from several development cases based on RADF. Special thanks are dedicated to all above mentioned ones and also the reviewers of the paper.

REFERENCES

[1] Philippe Kruchten, Henk Obbink, and Judith Stafford, "The Past, Present, and Future of Software Architecture", *IEEE Software*, pp. 22-30, March/April 2006.

[2] Joseph F. Maranzano, Sandra A. Rozsypal, Gus H. Zimmerman, Guy W. Warnken, Patricia E. Wirth, and David M. Weiss, "Architecture Reviews: Practice and Experience", *IEEE Software*, pp. 34-43, March/April 2005.

[3] Ian Sommerville, "Software Engineering, Eighth Edition. Pearson Education Limited", Pearson Education Limited, pp. 427, 2007.

[4] Richard W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems", *IEEE Transactions on Software Engineering*, Vol. 31, No. 6, pp. 495-510, June 2005.

[5] William B. Frakes and Kyo Kang, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, Vol. 31, No. 7, pp. 529-536, July 2005.

[6] Polster Franz J., "Reuse of Software Through Generation of Partial Systems", *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 3, pp. 402-416, 1986.

[7] Tony C. Shan and Winnie W. Huam, "Taxonomy of Java Web Application Frameworks", *Proc. of IEEE International Conference on e-Business Engineering*, IEEE Computer Society, pp. 378-385, 2006.
[8] Stéphane Ducasse, Adrian Lienhard and Lukas Renggli, "Seaside: A Flexible Environment for Building Dynamic Web Applications", *IEEE Software*, pp. 56-63, September/October 2007.
[9] Angel Israel Ortiz-Cornejo, Heriberto Cuayahuitl and Carlos Perez-Corona, "WISBuilder: A Framework for Facilitating Development of Web-Based Information Systems", *Proceedings of the 16th IEEE International Conference on Electronics, Communications and Computers*, pp. 46-51, 2006.
[10] David W.L. Cheung, Thomas Y.T. Lee, and Patrick K.C. Yee, "Webformer: A Rapid Application Development Toolkit for Writing Ajax Web Form Applications", In: T. Janowski and H. Mohanty (Eds.) *ICDCIT 2007, LNCS 4882*, Springer-Verlag Berlin Heidelberg, pp. 248-253, 2007.
[11] Polymenakos Lazaros C., Soldatos John K., "Multimodal web applications: Design issues and an implementation framework", *International Journal of Web Engineering and Technology*, Vol. 2, No. 1, pp. 97-116, 2005.
[12] Michał Lentner and Kazimierz Subieta, "ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development", In: Y. Ioannidis, B. Novikov, and B. Rachev (Eds.) *ADBIS 2007, LNCS 4690*, Springer-Verlag Berlin Heidelberg, pp. 130-140, 2007.
[13] Dzenan Ridjanovic, "Rapid Development of Web Applications with Web Components", In: K. Elleithy (ed.) *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, Springer-Verlag Berlin Heidelberg, pp. 99-103, 2007.
[14] Budi Kurniawan and Jingling Xue, "A Comparative Study of Web Application Design Models Using the Java Technologies". In: J.X. Yu, X. Lin, H. Lu, and Y. Zhang (Eds.) *APWeb 2004, LNCS 3007*, Springer-Verlag Berlin Heidelberg, pp. 711-721, 2004.
[15] Yonghwan Lee, Junaid Ahsenali Chaudhry, Dugki Min, Seungkyu Park, and Duckwon Chung, "A Comparative Performance Analysis of CBD Application Framework with JBean Application Framework for Improved Distributed Application Development", *Proc. of Intelligent Pervasive Computing 2007*, pp76-79, 2007.
[16] Hajar Mat Jani and Sai Peck Lee, "Applying Case Reuse and Rule-Based Reasoning (RBR) in Object-Oriented Application Framework Documentation: Analysis and Design", *Proc. of 2008 Conference on Human System Interactions*, pp. 597-602, 2008.
[17] Weidong Liao and Benjamin J. Koonse, "A Layered Java Application Framework for Supplying Mathematical Computing Power to the Distributed Environment", *Proc. of 31st IEEE Software Engineering Workshop*, pp. 279-283, 2007.
[18] Mariano Rico, David Camacho, and Óscar Corcho. "A contribution-based framework for the creation of semantically-enabled web applications", *Information Sciences*, Volume 180, Issue 10, pp. 1850-1864, 2010.
[19] Sai Peck Lee, Siew Khim Thin, and Hong Song Liu, "Object-oriented Manufacturing Application Framework", *Proc. of 34th International Conference on Technology of Object-Oriented Languages and Systems*, pp. 253-262, 2000.
[20] Yan Cao, Hua Chen, Lina Yang, and Yanli Yang, "Distributed Cooperative Design System Development Based on XML and J2EE Platform", *Proc. of International Symposium on Information Engineering and Electronic Commerce*, pp. 622-625, 2009.
[21] Chae Heung Seok, Cui Jian Feng, Park Jin Wook, Park Jae Geol, and Lee Woo Jin, "An object-oriented framework approach to flexible availability management for developing distributed applications", *Journal of Information Science and Engineering*, Vol. 25, No. 4, pp. 1021-1039, 2009.
[22] Efremov Roman, and Insua David Ríos, "An experimental study of a web-based framework for group decision support with applications to participatory budget elaboration", *International Journal of Technology, Policy and Management*, Vol. 7, No. 2, pp. 167-177, 2007.
[23] Mary Shaw, and Paul Clements, "The Golden Age of Software Architecture", *IEEE Software*, pp. 31-39, March/April 2006.

AUTHOR BIOGRAPHY

Dongjin Yu is currently a professor at Hangzhou Dianzi University, China. He received his BS and MS in Computer Applications from Zhejiang University, and PhD in Management from Zhejiang Gongshang University. His current research efforts focus on information systems and software engineering, especially the novel approaches to constructing large enterprise systems effectively and efficiently by emerging advanced information technologies. The concern of his research closely relates with real applications of e-government and e-business. He has led a number of government funded projects, including the Rapid Application Development Framework, OLAP Middleware, and Service-based Decision Support Systems. He is also the vice director of Institute of Intelligent and Software Technology of Hangzhou Dianzi University.