Proceedings for XX Winter Course

On the Way to the SOA-GTDS Framework: Modernization of the GTDS Architecture

Nuhad Shaabani, Prof. Dr. Christoph Meinel

Hasso-Plattner-Institute, University of Potsdam, Germany

***Abstract*— The GTDS is a system for use as a clinical cancer register and in the follow-up care of cancer patients. Much has been implemented in this system in the way of specialized medical knowledge, making it a valuable and comprehensive system for the documentation of cancer data. From the technical side, however, GTDS is not easily adaptable, user-unfriendly, dependent on a particular older platform, and therefore not very efficient when viewed in the light of present-day requirements. In this article, a modern architecture for the re-design of GTDS is conceptually introduced. One of the most important properties of this architecture is the service layer, which can be considered a milestone in the identification of web service in the SOA-GTDS framework.**

***Index Terms* – GTDS, SOA-GTDS, Web Service, Quality Control, Software Architecture**

I. Introduction

The Gießener Tumor Documentation System (GTDS) [1] is a system that was implemented with Oracle Forms technology. Many of the Oracle Form-based applications, however, are candidates for redesign. There are a variety of reasons for this, from the expiry of client/server architecture, to new requirements on the user interface, or a new strategic alignment of SOA. Therefore, the need to modernize GTDS architecture exists in terms of maintenance costs, ergonomics and future security. This last aspect is in one respect important because the system is the most widespread clinical tumor documentation system in Germany, used in approximately 60 clinics and tumor centers. In another respect, the modernized GTDS architecture can be exploited as the foundation or milestone toward the implementation of the SOA-GTDS framework [2].

Manuscript received January 27, 2012.

N. Shaabani is a PhD student at Hasso-Plattner-Institute at Potsdam University, Germany (e-mail: Nuhad.Shaabani@hpi.uni-potsdam.de)

C. Meinel is the CEO of Hasso-Plattner-Institute for IT-Systems Engineering and full professor for computer science of Potsdam University, Germany (e-mail: christoph.meinel@hpi.uni-potsdam.de)

Current GTDS Architecture and Oracle Forms

a. What is Oracle Forms?

If the Gießener Tumor Documentation System is viewed from a technical perspective, Oracle Forms may be described as a development and runtime environment on the basis of procedural SQL-expansion PL/SQL for OLPT-based (Online Transaction Processing) applications. Since the beginning of the 90's, the development of screen dialogs or dialog windows, on the basis of Windows or OSF Motif, is possible with Oracle Forms. Here, the run-time environment provides a powerful basis function that supports the fast execution of data-driven applications. Since the end of the 90's, the architecture of Oracle Forms applications has been a classic client/server architecture with Java applet-based presentation layer (see Fig. 1). Forms applications can now be installed directly in the "Oracle Application Server." The representation of the Forms mask is carried out via a Java applet in the user's browser [4].

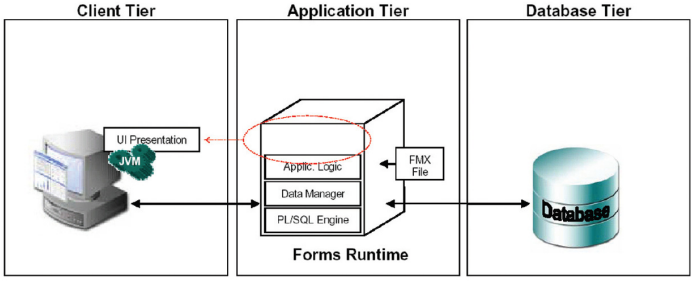


Fig. 1: The client/server application architecture of Oracle Forms

b. Reasons for the Modernization of the GTDS Architecture

Vendor Lock-In: The classic client/server architecture was no longer supported by Oracle. Support for the last client/server version (6i) expired on 31 January 2008 [3].

Maintenance and Extensibility: Oracle Forms is naturally not object-oriented. It does not have logical layer separation, and a separation of presentation and business logic is only possible in a limited scope. Furthermore, in Oracle Forms projects, areas such as test automation or continuous integration are markedly more difficult to implement than, for example, in Java application development. These are a few reasons why the middle-term and long-term maintenance costs for such applications are often relatively high [4, 5].

Usability: There have been few changes to the principle elements of the user interface in recent years and versions. For example, there is not an up-to-date table display in which the user can enlarge column width or change column order. Also, the size of a window where a Forms mask is represented is inflexible and does not adapt to a client's altered screen resolution. Numerous users find the standard Forms interfaces to be antiquated, which means they often fail to conform to expectations [4, 5].

Fig. 2: Multi-tier architecture for the GTDS

II. Requirements for the New Design of the GTDS Architecture

In Forms applications there does not exist a clear separation between business functions and cross-section functionality (infrastructure of the application). The new re-design of the GTDS architecture must offer the following components:

Configuration: Standardized storage and management of configuration data.

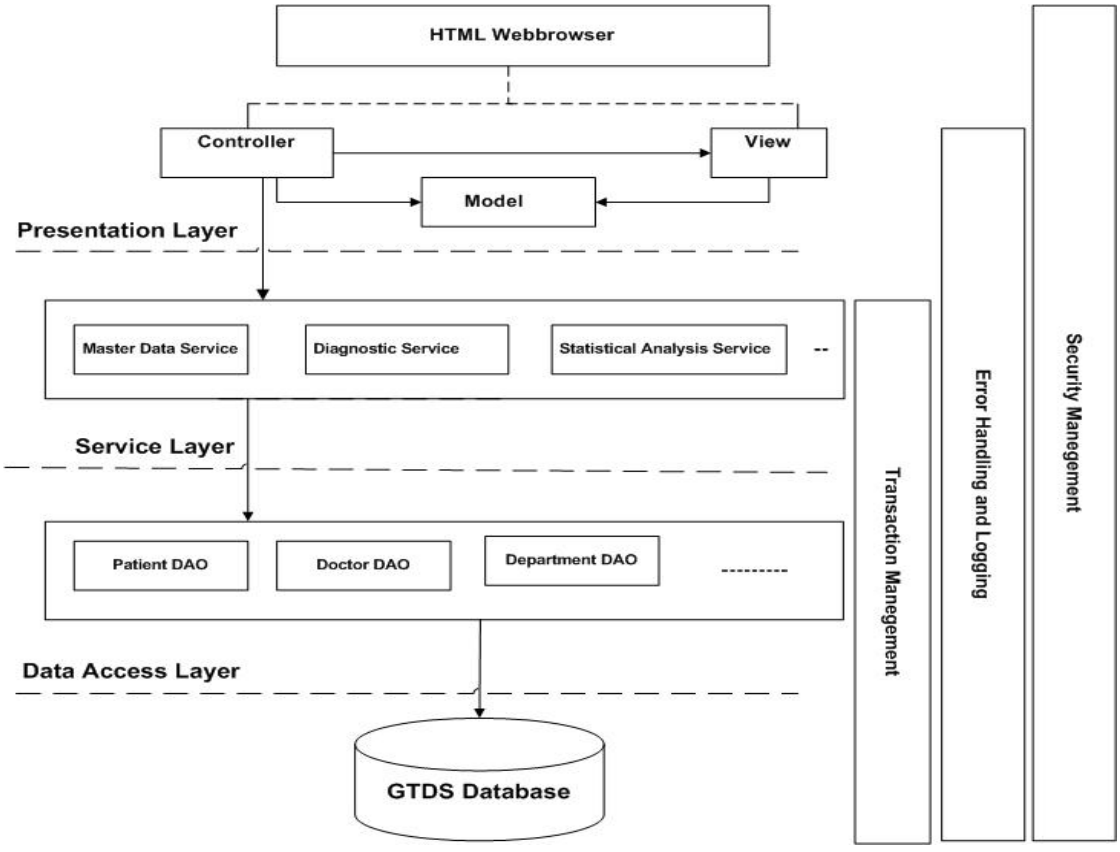
Error and Exception Handling: Standard processing of errors and their logging.

Multilgualism: Support of different languages by the user.

Rules and Validation: Centralization of the validation logic and use on all levels of the architecture.

Workflow: Integration of the central work list of a user and, consequently, related data.

Security: Management of identity, verification of access rights and encryption of data.



instances stands for a record from the respectively represented database object.
The entities therefore serve as data holders and normally do not contain any logic.
In the DAO objects (see Fig. 3), requests to the database are formulated and the results delivered to the calling service. The delivered results in the most cases take the form of entity instances.

III. The New Proposed Architecture

In the new architecture the system is divided into different layers, which subsequently carry out clearly defined responsibilities. This architecture is designated a multi-tier application and allows for the further development of individual aspects of the application, or its adaption to new environments -- without having to worry about unfortunate side effects affecting other applications (see Fig. 2). This is achieved by the encapsulation principle. This principle refers to hiding the actual implementation behind a layer, which only provides an interface to the layer on top of it. Therefore, every layer virtually consists of an interface and its implementation. The interface defines what functionalities the layer makes available to the one on top of it, while its implementation defines how the functionalities of the layer are realized. This design facilitates the exchange of individual implementation details without the rest of the application being affected. For example, the entire application does not have to be reprogrammed just to enable use of another database, such as Oracle database. Another design principle in this architecture defines the communication between the layers: a lower layer does not know the layers on top of it and therefore cannot use their functionalities.

a. Data Access Layer

The Data Access Layer is responsible for the permanent storage of data, and, for example, accesses tables and stored procedures of a relational database [6]. It provides data to the layers above it in a form independent of the data structure of the storage type, for example in the form of Java objects. In this way, it cannot be recognized whether the momentary storage is implemented over a relational database or based on a file. The database objects on this layer, such as tables or views, are represented on objects (entities) of the chosen programming language (see Fig. 3). An entity simply reflects the schema of the represented database object. Each one of its

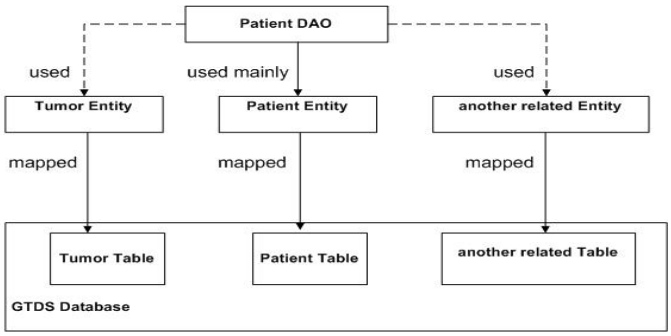


Fig. 3: Data Access Layer (Data Access Objects)

b. Service Layer

The service layer on top of the data access layer contains the business logic of the system. Each service forms a self-contained unit intended to represent related professional applications [6]. A service can access several data access objects (DAOs), to provide its services to the controllers from the representation layer (see Fig. 4). The transactional behavior of the system and the rollback mechanism are defined on this layer. In other words, every service is carried out as a transaction.

c. Presentation Layer

The user works exclusively with an optimal presentation layer for his environment, which prepares the data of the system graphically. The separation of representation and application makes it possible to offer different surfaces for different environments. The presentation layer of the new GTDS architecture follows the MVC pattern (Model- View- Controller- Pattern) [6]. Shown in simplified form, the presentation layer is divided into (see Fig.2)

Model: Data container without further logic functions (e.g., HashMap)

View: Shows the data from the model on a GUI surface (e.g., per JSP for the web browser)

Controller: Controls the sequence, generates and processes the data of the model and starts the view. By making use of multiple services, a controller receives the necessary data for his model.

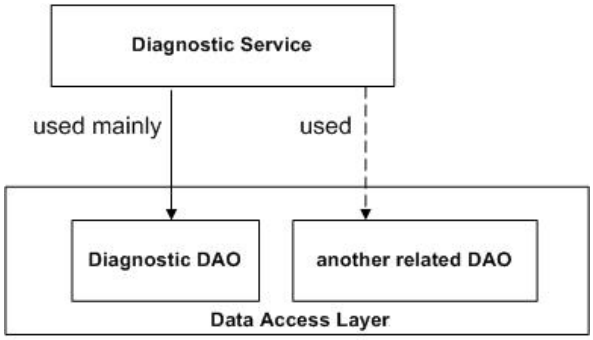


Fig. 4: Service Layer

III. Approach and Challenges

a. Extracting Knowledge from Oracle Forms

In the GTDS, a part of the specialist medical knowledge is implemented in the database in the form of "stored procedures" and triggers. This can simply be used further by the data access layer. The other part is manifested in Oracle Forms in many attributes, triggers and program blocks. From these, the relevant medical knowledge to extract and outsource in the database, or to implement in the chosen programming language is provided[4].

Another challenge, in addition to the extraction of specialized knowledge from Oracle Forms, is the extraction of the validation logic and the identification of the workflow of the masks for each use case. The validation logic and the workflow should then be outsourced in the presentation layer or implemented [5].

b. Development Team

In addition to mastering a modern programming language, e.g., Java for the implementation of the new design, the development team must also be competent in the area of PL/SQL programming and Oracle Forms development. It is not easy to find developers who are able to master both programming paradigms (the object-oriented paradigm and Oracle Forms programming) [4]. What makes this more likely is the fact that the Oracle Form technology is relatively old, and therefore the difference between the Oracle Form developer generation and the generation who is knowledge about a modern object-oriented programming is fairly large. A solution to this problem can be training the team in Oracle Forms development. From this solution, the following questions arise: how is it possible to interest team members in an old technology or to motivate them? When and how is the training effective to the point that the extracting of the professional logic from the Forms can be carried out correctly and efficiently in view of the effort expended?

c. Effort Estimation:

A multiple layer procedure can be very helpful for estimating effort. First, it is possible to select several use cases and to implement them in the new architecture. The created prototype is then provided to users of the system to ascertain their level of acceptance and satisfaction. The prototype helps, for example, to check whether the team selected has sufficient knowledge in PL/SQL and Oracle database. It could be necessary to add experienced Oracle Forms developers to the team in cases where a simple training in Oracle Forms programming is not sufficient. Despite prototyping and the formation of a team, the effort estimation remains problematic for a number of reasons. One of these reasons is that the demands on the new system are themselves defined by the existing solution, meaning by the existence of GTDS. It is very difficult in a limited time period to examine hundreds of binary modules (Oracle Forms). This investigation proves all the more difficult when the team lacks an experienced Oracle Forms developer. Another reason for the difficulty of the estimation is that Oracle Forms applications have many automatically delivered detail functions (e.g., in the Search). To offer a suitable substitute for all the functions makes a realization very complex [4].

d. Selection of the Framework

A modern object-oriented programming language, such as Java, is suitable for the implementation of the new architecture The Java framework, „Spring" [7] offers many components (Spring-Core, Spring-MVC, Spring-Security), which are very well suited to fulfill the requirements posed. The ORM Java

framework „Hibernate“[8] is particularly suited for the representation of database objects on Java objects.

e. Test

An especially important aspect in the implementation is testing, which should accompany all phases of development. An automatic test demands that appropriate medical data is prepared and made available for every use case. Another possibility for testing is implementing specific use cases in defined time intervals (cycles), and making this so-called documentation available to experienced GTDS users. These users will then manually test those use cases implemented in the respective cycle and report on errors and areas for improvements. The errors and suggestions for improvement are then eliminated, or taken into account in the next development cycle. Therefore working together and constant cooperation is required by the responsible parties.

III. Summary

The Gießener Tumor Documentation System is an Oracle Forms application. This technology is based on the client/server architecture paradigm and is subject to vendor lock-in. This makes the system user-unfriendly, and not well adaptable or extensible. Among other things, these technical properties justify the need to redesign the architecture of the system. In this article, we have conceptually presented a new architecture for the GTDS and discussed its properties. In addition, we have broadly discussed an approach for the implementation of the new design and the challenges connected with it. A special component of this architecture in regard to the SOA-GTDS is the service layer. This can help in the identification and realization of web services in SOA-GTDS.

REFERENCES

[1] http://www.med.uni-giessen.de/akkk/gtds/

[2] Y. Li, Ch. Meinel: The Framework of SOA-GTDS. In Proceedings of XIX Winter Course of the CATAI (CATAI 2011) Canary Islands, Spain, 3, 2011

[3] Oracle, Orcale Forms – Oracle Reports – Oracle Designer, Oracle Statement of Direction, 2005. www.oracle.com/technology/products/forms/pdf /10g/ToolsSOD.pdf

[4] M. Bertelmeier: Altsystem im neuen Kleid: Migration von Oracle-Forms nach Java. Objektspektrum (05/2007)

[5] S. Price, G. waite: Oracle Forms to SOA: A Case Study in Modernization. An Oracle Forms Community Whilte Paper. (06/2008). http://www.oracle.com/technetwork/developer-tools/jdev/griffithswaite-129182.pdf

[6] M. Fowler: Patterns of Enterprise Application Architecture. Addison- Wesley (2003)

[7] http://www.springsource.org/

[8] http://www.hibernate.org/

Nuhad Shaabani is a researcher in the Internet Technologies and Systems group at Hasso-Plattner-Institute for IT-Systems Engineering (HPI) at the University of Potsdam, Germany.
His main research interests are applying semantic and intelligent methods in telemedicine and software architecture.

Prof. Dr. sc. nat. Christoph Meinel is President and CEO of the Hasso-Plattner-Institut for IT-Systems Engineering (HPI) and full professor (C4) for computer science at the University of Potsdam.
His research field is Internet and Web Technologies and Systems (www.hpi.uni-potsdam.de/meinel). Beside he is a teacher at the HPI School of Design Thinking, a visiting professor at the Computer Science School of the Technical University of Beijing (China) and a research fellow of the interdisciplinary center SnT at the University of Luxembourg. Since 2008 he is program director of the HPI–Stanford Design Thinking Research Program. Since 2010 he chairs the Steering Committee of the HPI Future SOC lab.
Christoph Meinel is author or co-author of 12 text books and monographs and of various conference proceedings. He has published more than 380 peer-reviewed scientific papers in highly recognised international scientific journals and conferences. His high-security solution Lock-Keeper is international patented and licensed by Siemens AG. His tele-TASK system provides an innovative mobile system for recording and Internet broadcasting lectures and presentations used in many universities all-over the world. The virtual tele-lab for Internet Security provides the possibility to get hands-on experiences in practical issues of

internet and information security. The recently developed Tele-Board supports remote work of creative teams.

Furthermore, Christoph Meinel is chairman of the German IPv6 council, and chairs the advisory board of UTD Meraka in South Africa. In 2006, he hosted together with Hasso Plattner the first German “National IT-Summit” of the German Federal Chancellor Dr. Angela Merkel. From 1998 to 2002 he was the founder and CEO of the Research Lab “Institute for Telematics” in Trier. Christoph Meinel is chief editor of the scientific electronic journals “ECCC – Electronic Colloquium on Computational Complexity" and “ECDTR – Electronic Colloquium on Design Thinking Research”, the “IT-Gipfelblog”, and the “tele-TASK”-archive.