A Framework for Migrating Traditional Web Applications into Multi-Tenant SaaS

Eyad Saleh, Nuhad Shaabani, and Christoph Meinel
Hasso-Plattner-Institut
University of Potsdam
Potsdam, Germany
{*eyad.saleh, nuhad.shaabani, christoph.meinel*}@*hpi.uni-potsdam.de*

***Abstract*—Software-as-a-Service (SaaS) is emerging as a new model of delivering a software, where users utilize software over the internet as a hosted service rather than an installable product. Multi-tenancy is a core concept in SaaS. It is the principle of running a single instance of the software on a server to serve multiple companies (tenants). Re-engineering traditional web applications from scratch into multi-tenancy requires tremendous efforts in terms of cost, manpower, and time. Thus, we provide a framework to migrate traditional web applications into multi-tenant SaaS. The framework provides a detailed overview of the proposed multi-tenant architecture that helps software architects and developers to migrate their applications into multi-tenancy.**

***Keywords*-Multi-tenancy; Migration; Software-as-a-Service; SaaS.**

I. Introduction: Multi-tenancy Evolution

History has shown that advances in technology and computing changes the way software are designed, developed, and delivered to the end users. These advances yield to the invention of personal computers (PCs) and graphical user interfaces (GUIs), which in turn adopted the client/server architecture over the old big, super, and expensive mainframes. Currently, Fibers and fast internet connections, Service-Oriented Architectures (SOAs), and the high-cost of managing and maintaining on-premises dedicated applications raised the flag for a new movement in the software industry, and the result was the introducing of a new delivery model called Software-as-a-Service (SaaS) [1].

Cloud computing has several definitions, one of those is 'delivering computation to end-users over the Internet'. This computation could be software, hardware, or even information. Users use this computation in a pay-as-you-go model, this means that they will pay for their usage of this computation. For instance, renting a server for two hours or one day, or using a certain financial application for two weeks, without the need to provision a complete data center or buy a full license of the software.

SaaS provides major advantages to both service providers as well as consumers. Service providers can provision a single set of hardware to host their applications and manage hundreds of clients (tenants). They can easily install and maintain their software. As for consumers, they can use the application anywhere and any time, they are relieved from maintaining and upgrading the software (on-premises scenario), and benefit from cost reduction by following the pay-as-you-go model [2].

Multi-tenancy is a requirement for a SaaS vendor to be successful (Marc Benioff, CEO, Salesforce) [3]. Multi-tenancy is the core of SaaS; it is the ability to use a single-instance of the application hosted by a provider to serve multiple clients (tenants). Multi-tenancy is different from multi-instance architecture (Figure 1) where separated instances of the same software are hosted on different servers to serve different tenants.

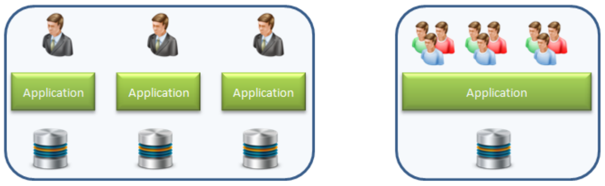


Figure 1. Multi-instance vs. Multi-tenant Architecture.

Software applications had been built for decades into non-SaaS mode, re-engineering or re-designing such applications from scratch requires tremendous efforts in terms of cost, manpower, and time. Therefore, researchers have been proposing several approaches to migrate such applications into SaaS mode.

While migrating the non-SaaS applications into SaaS mode, certain issues need to be considered, such as database architecture, data partitioning, UI customizations, data-model extension by tenants, scalability issues, and workflow management. This paper introduces a framework that supports the migration of traditional web applications into SaaS mode, and discusses certain solutions to the concerns above, as well as introducing new features that could utilize the multi-tenancy mode.

The main contribution of the paper is the framework itself and its components, especially, the business logic configuration and workflow customization, as well as the techniques described to implement the framework, which can be applied in different use cases.

The rest of this paper is organized as the following: Section II discusses the related work. Section III outlines the framework and its main components. In Section IV, we detailed the configuration and customization layer. Section V outlines the use case. Section VI discusses the different database architectural designs. Finally, Section VII concludes the paper and outlines the future work.

II. Related Work

There is much research that has been carried on SaaS and multi-tenancy; however, to the best of our knowledge, there are only a few who proposed a complete framework for migrating traditional web applications into SaaS mode. Moreover, none of the related work touches critical components such as business logic configuration or workflow customization.

A. *SaaS-ization*

Cai et al. [6] propose an end-to-end methodology for SaaS-ization based on identifying isolation points between tenants, such as UI, constant fields, and configuration files. Once these points are identified, the development team can modify these points to be configurable per tenant using a toolkit which was built for the purpose of their work. This approach addresses only the look-and-feel and basic configuration options between tenants, however, business logic and data-model are not covered.

A new approach based on migrating traditional web applications into SaaS automatically and without changing the source code has been proposed by Song et al. [7]. They adopt several technologies to accomplish this goal, mainly (1) page template to fulfill configurability, (2) memory in thread to maintain tenant-info, and (3) JDBC proxy to adopt the modified database. Additionally, they propose a SaaSify Flow Language (SFL) which models and implements the flow of the migration process. Practically, migrating an application from non-SaaS mode into SaaS without having or changing the source code is very hard to achieve. For instance, how the developers are going to handle the session data, how the database changes would be reflected on the Data Access Layer (DAL), even the simple UI components, how it could be managed.

B. *Migration into SaaS*

Bezemer et al. [8] report on a use case of converting an industrial, single-tenant application (CRM) for a company called Exact [9] into a multi-tenant one. They propose a pattern to migrate the application taking into account hardware sharing, high degree of configurability, and shared database instance. They propose three components that need to be added to accomplish the migration process: (1) Authentication module to map end-users credentials to tenants, (2) Configuration module to handle tenant-settings, and (3) database module to adapt insert, modify, and query tenant-oriented data. It is not possible in this approach to have a different business logic or workflow for each tenant.

C. *Customization and Configuration*

Since multi-tenancy is based on sharing the same application by more than one tenant, and the business requirements can vary among those tenants, there is a need to customize the application to appear as tenant-specific, in terms of the look-and-feel, workflow, business logic, etc.

Nitu [10] focuses on the configuration of SaaS applications, basically on user interfaces and access control. They store the configurations in XML files that are injected into applications at runtime. Based on a simple case study they introduce about a university grading system in two Indian universities, they suggest that a tenant should not expect all aspects of the software to be customized. If the tenant needs a complete customized software, then SaaS is not the right choice. This approach is very simplified, and does not cover other critical configuration items, such as workflow and data-model.

Müller et al. [11] identify different categories of customization, such as desktop integration and UI customization. Additionally, they identify two cornerstones for a customizable, multi-tenant aware infrastructure, namely, dynamic instance composition and abstraction from the persistency layer. However, they focus mainly on back-end customization by injecting custom business logic at runtime.

D. *Database Architecture*

Database design is considered as one of the most critical issues in multi-tenant SaaS, because multiple tenants are mapped into one physical database. Therefore, non-traditional concerns are involved, such as data-model extension, workloads, and database scalability.

Several approaches for designing a database for multi-tenant applications from different point of views are discussed by Chong et al. [12], such as completely isolated databases for each tenant versus shared databases with different schemas, or shared databases and shared schemas. Choosing specific approach depends on several factors, such as cost, security level, tenant requirements, scalability options and SLA.

W. Tsai et al. [13] propose a new database partitioning schema to maximize SaaS customization called two-layer schema. Their approach combines read-optimized column store and update-oriented writeable operations.

A new technique for mapping logical schema into physical schema is introduced by Aulbach et al. [14]. They categorize the tables into two types; conventional and chunk tables. The most and heavily utilized parts of the database are placed in conventional tables while the remaining parts are vertically partitioned into chunks. These chunks are folded into different multi-tenant physical tables and joined as needed.

All the aforementioned research proposals strongly contribute to our research; however, in this paper, we tried to introduce a complete framework to migrate traditional web applications into multi-tenancy and introduces new components such as, business logic configuration and workflow customization.

III. The Framework

In this paper, we propose a framework for migrating traditional web applications into SaaS mode as shown in Figure 2.

The process flow of the migrated application will be as follows:

- A user belonging to a certain tenant logs into the system by entering his username and password.
- A dedicated authentication module is used to map this user to the tenant he belongs to, to create a token for the tenant including the Tenant-ID as well as other relevant information (such as locale settings), and finally to pass it to the customization layer.
- In the customization and configuration layer, the UI components, such as logos and colors, the business logic, and workflow configuration data for this tenant are restored, and passed to the application server.
- The DB configuration data will be passed to the DB server for query transformation.
- The application server receives the above specified data from the upper layer and pass it to the run-time customization engine, which integrates all components and lunches the application instance.
- A log service is used to record the application actions and store them in text files.
- A dedicated monitoring service is used to monitor the performance and status of the application, and detects any faults or bad resource usage.

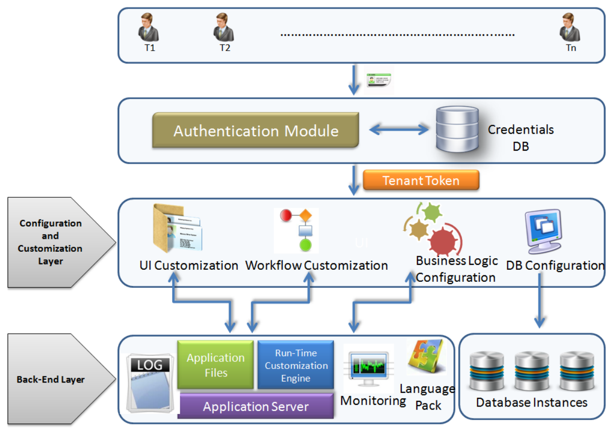


Figure 2. The Architecture of the Proposed Framework

A. The Back-End Layer

The main components of this layer are as follows:

1) The Performance Monitoring Service: Monitoring means getting feedback from the usage of the current software, which leads to enhance and improve the current version of the software.

This service monitors the performance of the software, for example, which queries respond slowly, what are the most heavily-used components of the application, which tenant is overusing the resources, etc.

This collective data will enable the vendor to enhance (upgrade) the software and better isolate tenants to improve the performance.

2) The Log Files: Log files are important to several applications, and more importantly to the multi-tenant ones. They can be used for many reasons, such as, monitor the performance of the application, figure out processing bottlenecks, discover software bugs in the early stages of the release and fix them immediately.

3) The Language Pack: A multi-tenant application is used by several tenants, and they might be from different cultures or having specific language requirements. Therefore, a language pack is additional component the tenant may use to personalize the language settings he needs.

This component is responsible for managing language files, and provide the settings that correspond to the tenant preferences to the run-time customization engine. Several languages could be defined in the language pack, such as English, Arabic, German, Chinese, etc.

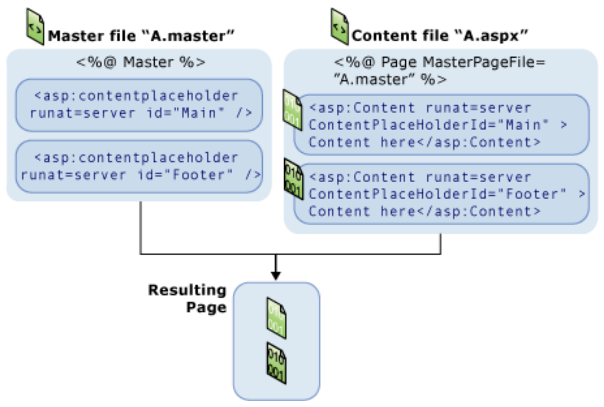
IV. The Configuration and Customization Layer

A. User Interface Customization

UI customization means changing the look-and-feel of the application to be tenant-specific. This includes general layout, logos, buttons, colors, and locale settings, such as date and time. To utilize this customization, we propose the usage of Microsoft's ASP.NET master page concept [17].

ASP.NET master page allows the developer to create a consistent look for all pages (group of pages) in the application; one or more master pages could be created for each tenant and used in the application. The master page provides a shared layout and functionality for the pages of the application, when users request any page, ASP.NET engine merges the master page that contains the layout with the requested content-page, and send the merged page to the user as shown in Figure 3.

The application developers would be able to define a master page for each tenant by applying the master page technique, which contains the required layout, color, buttons, and other design components. Moreover, several master pages could be defined for each tenant. Therefore, tenants will have the chance to get benefit of using dynamic look-and-feel. It is worth to mention that applying the ASP.NET concept does not require Microsoft's technologies or platform, our approach aims to apply a similar concept regardless of the technology or the platform.



```
<%@ Master %>
<asp:contentplaceholder
runat=server id="Main" />
<asp:contentplaceholder
runat=server id="Footer" />
```

```
<%@ Page MasterPageFile=
"A.master" %>
<asp:Content runat=server
ContentPlaceHolderId="Main" >
Content here</asp:Content>
<asp:Content runat=server
ContentPlaceHolderId="Footer" >
Content here</asp:Content>
```

Figure 3. ASP.NET Master Page [17]

B. Workflow Customization

The workflow of the application might vary from one tenant to another, for instance, a recruitment agency (Tenant A) might wait until they receive a request for a specific vacancy (from a company looking for employees), then start looking for applicants, while another agency (Tenant B) would collect applications, meet applicants, and then short-list them according to their potential, and have them ready for any vacancies from companies looking for employees (Figure 4). Therefore, we assume that customizing the workflow of the multi-tenant software is important. In order to achieve this, two steps are required. First, identify the components of the software that need to be customized, second, change the design of these components to be loosely coupled, thus they can be easily replaced by other versions and integrated with other components, and therefore, each tenant can have his own version of the same component.

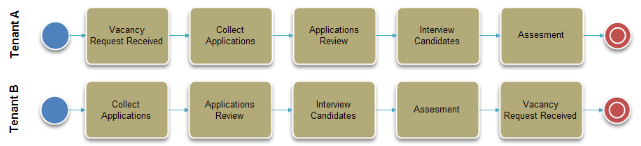


Figure 4. Different Workflow for two recruitment agencies

Changing the design of the entire application into loosely coupled components is a difficult task, on the other hand, customizing the complete workflow of the application may not be necessary since the majority of the application components are normally common among all tenants.

The second step is crucial to make workflow customization successful. The benefit of changing the components selected in step one into loosely coupled is twofold; first, it will maximize the utilization of workflow customization, by allowing the tenant to architect the workflow according to their needs; second, these services/components will become interoperable, thus they can integrate with each other as well as with other services or components from other applications, and even with third party components.

C. Business Logic Configuration

In software engineering, a multi-tier architecture enables developers to divide the application into different tiers to maximize the application re-usability and flexibility. One of the most common implementations of multi-tier is the three-tier architecture, which divides the application into three-tiers, namely, the presentation layer (PL), the business logic layer (BLL), and the data access layer (DAL) (Figure 5).

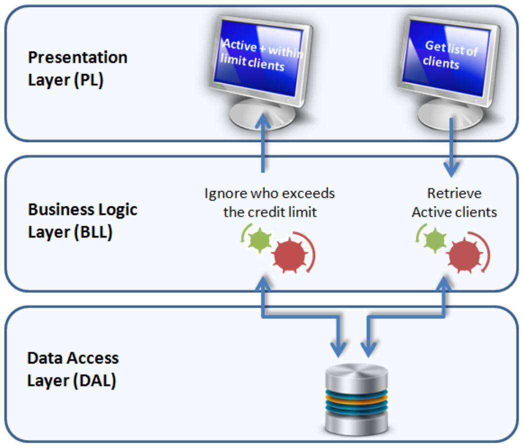

Presentation Layer (PL)
Active + within limit clients
Get list of clients
Business Logic Layer (BLL)
Ignore who exceeds the credit limit
Retrieve Active clients
Data Access Layer (DAL)

Figure 5. Multi-tier Architecture

Business rules are part of the business logic layer (BLL), and these rules varies from one organization to another. For instance, in a travel agency, if a reseller or a client exceeds his credit limit, all his upcoming purchases will be rejected, while another agency, may apply a different rule which state that if the reseller exceeds his credit limit for three weeks without any payment, he will be blacklisted.

In order to achieve and maximize multi-tenancy, we propose that these business rules need to be tenant-specific, this means that the tenant should have the ability to design, apply, and re-configure his own rules at the run-time. Therefore, a tool that offers this feature is needed as a part of the proposed framework.

D. Database Configuration

Database design is considered as one of the most critical issues in multi-tenant SaaS because multiple tenants are mapped into one physical database. Therefore, a robust database architecture must be modelled and implemented.

Consolidating multiple tenants into one database requires changes to the design of the tables and the queries as well, thus, a query transformation is required. For instance, in a traditional hospital management system, a simple query to fetch a patient record would be "select * from patient where

SSN=1234", while in a multi-tenant system, this will not work, since the "patient" table would have information for many tenants (i.e., hospitals). Therefore, the query should be changed to something similar to "select * from patient where tenant_id=12 and SSN=1234"; in this case, the patient record that belongs to the tenant 12 (i.e., hospital 12) will be retrieved. Based on that, the rules for query transformation should be stored in the database configuration files and restored by the transformation engine when required. Further details are discussed in Section VI.

V. The Use Case: GTDS

Gießen Tumor Documentation System (GTDS) [18] is a software developed mainly for hospitals treating cancer. The project was funded by the federal ministry of health in Germany. GTDS is a powerful and comprehensive system for the documentation of cancer data. GTDS is widely accepted in all over Germany, and it is used in more than 60 hospitals and clinics. From technical point of view, GTDS was initially implemented with Oracle Forms, and its relational data model contains about 400 tables [19]. The base schema for GTDS that we used in the experiments is shown in Figure 6. There are some efforts to redesign the software with modern technologies to produce a standard version that could be used in all hospitals around Germany. Therefore, we are designing and implementing a new modern multi-layer architecture for GTDS. This new architecture will enable us to speed-up the process of migrating GTDS into multi-tenant one.

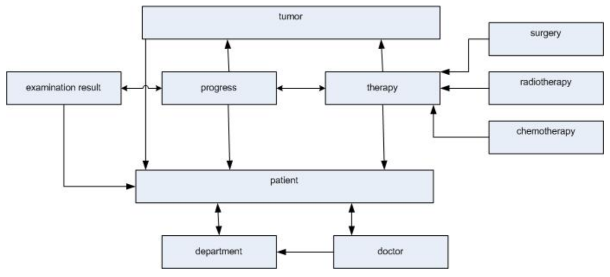


Figure 6. GTDS Basic Schema Architecture

VI. Database Architecture

There are several database architectures that could be used for multi-tenancy, such as completely isolated database for each tenant, shared database with different schemas, or shared database and shared schemas.

The isolated database design makes it easy to extend the data model to meet the requirements of individual tenants. Additionally, it works well in terms of backup and restore, tenant migration between VMs or hosts, and requires only minor changes to the software or database layer. However, it does not reflect the real benefits of multi-tenancy, at least on the database level, such as consolidating several tenants into one physical database. Moreover, it is costly since we need a separate database instance for each tenant, and we will be limited with the number of databases the server can support.

Another approach is the shared database with separate schemas, several tenants will share the same database instance, while everyone is having his own schema. This approach is relatively easy to implement, tenants can extend the data model, and a moderate degree of logical isolation is gained. However, the main limitation of this approach is maintainability, recovering the data for a single tenant in case of failure if a complex task. The database administrator cannot restore the entire database (e.g., from a backup) since this will overwrite the data for other tenants. Therefore, additional efforts need to be taken.

The third approach is shared database with shared schema, where all tenants share the same database with common schema. Obviously, this approach overcomes most of the shortcomings of the aforementioned approaches. However, few questions arise, such as, how to accomplish data isolation between tenants, how to extend the data model since the extension is different from one tenant to another, etc.

The most simple technique to accomplish data isolation is adding a tenant-id column to all tables, then change the queries, functions, and triggers to add the tenant-id filter. However, other advanced techniques can also be used, such as extension table layout, pivot tables, and chunk folding. For more details about these techniques, please refer to [14].

VII. Conclusion and Future Work

Migrating traditional web applications into multi-tenant SaaS poses several research challenges in terms of software customization, database architecture, and isolation between tenants. In this paper, we presented a complete framework to facilitate the migration process. We explored the configuration and customization of the application from several layers, such as UI, business logic, workflow, and database design. Our future work will focus on implementing the framework on GTDS. A set of tools will be developed to facilitate the business logic configuration and workflow customization. Further, security in terms of isolation between tenants will be investigated. Furthermore, how to protect the privacy of tenants will be studied. Finally, several validation experiments of the framework will be conducted on different use cases.

Acknowledgment

The authors would like to thank Mohammed AbuJarour from SAP for his valuable feedback, insights and comments.

References

[1] T. McKinnon: "The Force.com multitenant architecture: understanding the design of Salesforce.com's internet application development platform", White Paper, USA, 2008. [Online]. Available:

http://www.developerforce.com/media/ForcedotcomBookLibrary/Force.com_Multitenancy_WP_101508.pdf [retrieved: 08, 2012]

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and M. Zaharia: "Above the clouds: A Berkeley view of cloud computing". Technical report, University of California, Berkeley, USA, 2009.

[3] D. Woods: Salesforce.com Secret Sauce, Forbes, January 2009. [Online]. Available: http://www.forbes.com/2009/01/12/cio-salesforce-multitenancy-tech-cio-cx_dw_0113salesforce.html [retrieved: 08, 2012]

[4] X. H. Li, T. C. Liu, Y. Li, and Y. Chen: "SPIN: Service Performance Isolation Infrastructure in multi-tenancy environment". Proc. 6th Int. Conf. on Service-Oriented Computing, Sydney, Australia, 2008, pp. 649-663.

[5] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao: "A framework for native multi-tenancy application development and management". Proc. 9th IEEE Int. Conf. on E-Commerce Technology and The 4th IEEE Int. Conf. on Enterprise Computing, E-Commerce and E-Services, Tokyo, Japan, 2007, pp. 551-558.

[6] H. Cai, K. Zhang, M. J. Zhou, W. G., J. J. Cai, and X. Mao: "An end-to-end methodology and toolkit for fine granularity SaaS-ization". Proc. IEEE Int. Conf. on cloud computing, Bangalore, India, 2009, pp. 101-108.

[7] J. Song, F. Han, Z. Yan, G. Liu, and Z. Zhu: "A SaaSify tool for converting traditional web-based applications to SaaS application". Proc. IEEE 4th Int. Conf. on Cloud Computing, Washington, DC, USA, 2011, pp. 396-403.

[8] C. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. Hart: "Enabling multi-tenancy: An industrial experience report". Proc. 26th IEEE Int. Conf. on Software Maintenance, Timi oara, Romania, 2010, pp. 1-8.

[9] Exact Website. [Online]. Available: http://www.exact.com [retrieved: 08, 2012]

[10] Nitu: "Configurability in SaaS (software as a service) applications". Proc. 2nd India Software Engineering Conference, Pune, India, 2009, pp. 19-26.

[11] J. Müller, J. Krüger, S. Enderlein, M. Helmich, and A. Zeier: "Customizing enterprise software as a service applications: Back-end extension in a multi-tenancy environment". Proc. 11th Int. Conf. on Enterprise Information Systems, Milan, Italy, 2009, pp. 66-77.

[12] F. Chong, C. Gianpaolo, and R. Wolter: "Multi-tenant data architecture", Microsoft Corporation, http://www.msdn2.microsoft.com, 2006.

[13] W. Tsai, Q. Shao, Y. Huang, and X. Bai: "Towards a scalable and robust multi-tenancy SaaS". Proc. Second Asia-Pacific Symp. on Internetware, Suzhou, China, 2010.

[14] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger: "Multi-tenant databases for sSoftware as a service: schema-mapping techniques". Proc. 2008 ACM SIGMOD Int. Conf. on Management of data, Vancouver, Canada, 2008, pp. 1195-1206.

[15] D. Lin and A. Squicciarini: "Data protection models for service provisioning in the cloud". Proc. 15th ACM symp. on access control models and technologies, Pittsburgh, Pennsylvania, USA, 2010, pp. 183-192.

[16] Y. Shen, W. Cui, Q. Li, and Y. Shi: "Hybrid fragmentation to preserve data privacy for SaaS". Proc. 2011 Eighth Web Information Systems and Applications Conf., Chongqing, China, 2011, pp. 3-6.

[17] ASP.NET Master Page on Microsoft.com. [Online]. Available: http://msdn.microsoft.com/en-us/library/wtxbf3hh.aspx [retrieved: 08, 2012]

[18] GTDS Website. [Online]. Available: http://www.med.uni-giessen.de/akkk/gtds/ [retrieved: 08, 2012]

[19] U. Altmann, FR. Katz, and J. Dudeck: "A reference model for clinical tumour documentation". Institute of Medical Informatics, University of Gießen, Germany, 2006.