Refactoring Legacy Software Systems into SOA Compatible Style to Support e-Business Development in Enterprise Organizations: The Case of Umm Al-Qura University's Systems

Basem Y. Alkazemi, Abdullah Baz, and Grami M. Grami

***Abstract*—The paper analyzes the current state of software system in a leading Saudi University described as 'legacy system' in terms of functionality and ability to be modified in the face of ever increasing e-government requirements. The natural course of action taken by CEOs in similar settings is purchasing and installing an entirely new system, a decision which inevitably entails the risks of costing huge sums of money and compromising older business models. Subsequently, lengthy and complicated setting up may be required and the new system therefore is not always financially viable. An alternative model to re-architect the legacy systems instead of installing a new one suddenly makes sense. We attempt to establish the viability of our approach and account for the commercial and practical advantages as opposed to the other solution. The proposed solution is the first step towards facilitating the adoption of e-business models to automate the entire processes within that institute.**

***Index Terms*—Legacy systems, SOA, e-government, business process, Baz tool.**

I. Introduction

Responding to rapidly changing IT environments – including but not limited to expanding e-government applications in various institutions and corporations- requires adopting a reliable, versatile yet exceptionally flexible computing systems capable of accommodating the old, current and future applications. The modifications must be carried out efficiently and smoothly while keeping old business needs intact [1]–[2]. In this sense, such a system is as a mandatory requirement for any company/institute with ambitions, not an option.

Adaptable systems nevertheless are not readily available even for large private companies, public organizations, government agencies, hospitals, municipalities, and universities in Saudi Arabia. Studies show that these institutes run and maintain their respective legacy systems as far as long as they provide the basic necessary functionality. Despite that, these organizations are aware of the rapidly changing IT market and are duly planning to replace their old systems at some point should finical resources become available, given the high costs associated with buying and installing a new replacement. They may also consider the cost-effective yet unproven option of modernizing their current [3].

Many challenges stem from the nature of legacy systems which, as the name suggests, are not always modifiable. Systems are usually treated as black boxes not because they lack documentations or availability of the source code. Rather, the systems are poorly architected in the sense they can no longer cope with new business needs. This scenario in fact is one of the key barriers to adopt any potential e-government business models we identified. Poorly architected legacy systems entail in many instances the decision by CEOs to buy and install entirely new systems are reported earlier. However, such a decision should be informed and well researched. It entails significant consequences in terms of costs associated with purchasing and efforts in installing and setting up. Moreover, legacy systems usually provide highly customized functionalities none of the available solutions in the market can usually provide if purchased as is. For example, setting up new systems may require significant modifications for up to several years to comply with old business needs while accommodating newer ones.

This research aims at investigating an architectural model to analyze the feasibility of refactoring legacy systems. The outcome should help CEOs make informed decisions based on findings of thorough analytical studies on whether to initiate modernizing legacy systems projects or simply approve purchasing new solutions from the market.

II. Background Work

Umm Al - Qura University (UQU) [4] is a typical Saudi organization running a legacy system and subsequently is in need of an urgent update. UQU therefore was selected as the case study of the proposed model with the purpose of creating a fully integrated environment that supports e-government business. While the institution needs a fundamental solution to cope with the changing environment, interrupting the routine working activities should not be affected. Funding is also a major consideration that influences any decision regarding major development plans.

From a historical perspective, Umm Al-Qura University launched its information systems in early 2001 to serve around 3600 employees and just over 40,000 students at the time. The system runs an outdated code systems based on Oracle 6i for forms and reports, which are built entirely on client-server pattern [5]. The major subsidiaries include an

Manuscript received November 10, 2012; revised December 21, 2012.

Basem Y. Alkazemi is with Umm Al Qura University (bykazemi@uqu.edu.sa).

in-house Enterprise Resource Planning (ERP), Student Information System (SIS), Library Information System (LIS), and Healthcare Information System (HIS).

Twelve years later, the system started to struggle with the new environment because of the increase of the number of users and the pressure to support e-government models. The 12-year old systems are still in operation today serving around 75,000 students and more than 7000 employees, an almost two-fold increase compared to 2001, be it with minor changes to the original core functionality. Moreover, software systems lack many capabilities which are considered core-requirement these days including compatibility with different environments (e.g. Mobile devices) and the services provided to students and faculty members in the University.

Additionally, due to the emerging e-government movements in Saudi Arabia, organizations need to apply major changes to their core systems in order to accommodate new requirements; one of which is process automation that solely requires splitting functional aspects of an application from the business aspects.

Unfortunately, the modifications attempts carried out by IT departments to add features to the systems are largely random. Specifically, business processes are implemented directly into the forms confusing the functional aspects of an application with the non-functional parts. As a result, the complexity of UQU systems is rapidly complicating in a pattern that will inevitably become hard to manage in continues.

III. Business Requirements at UQU

As expected from a major educational and research institution, UQU's main objectives are centered on providing high quality teaching and research services to the community in different knowledge areas including medical sciences, engineering, and computer sciences. In the backend, many organizational activities take place in the form of processes that provide services to various users within the university. All services provided (i.e. faculties, employees, students) must meet the standards of certain Service Level Agreement (SLA) in order to guarantee the quality of service provided to users. Based on this criterion, we identified a number of driving forces that describe the potential business requirement for adopting a sophisticated software system to serve the needs of the university members; these include:

- **Considerable increase of users**: the number of students in the university has increased sharply which in turn required a considerable increase in the number of faculties and employees. This extra pressure calls for a smooth platform of interaction between the different parties and also a reliable system architecture that can be expanded to accommodate different services (e.g. payroll, allowances, admission and registration, scheduling).
- **The establishment of new campuses**: the university has established a number of branches since the original systems were first installed all of which expect a similar level of IT support. Currently the university uses paper based correspondences an application forms to provide services. This traditional system of management causes considerable delays in processing applications especially when dealing with dependent campuses. As a result, there is mounting pressure to establish a fully integrated e-services system that can be accessed remotely from the web, significantly reducing delays of sending documents for processing.
- **Automated processes within/between departments**: even though some departments in the university run software applications which in theory should help the application processing, these applications are in fact data entry forms fed into databases. However, the flow of data within/between departments is very much carried out manually; this practice much like the one discussed before it requires handling the applications between different offices.
- **Facilitate data exchange with other establishments**: UQU occasionally receives requests from other establishments (e.g. Ministry of Higher Education) to provide statistics and other facts about activities and projects. We observed that the requested data are usually posted as reports to the requestors. Reports generation is a time consuming task. As a result, we propose establishing secured web services that allow different governmental entities to retrieve their required data automatically without disturbing UQU business.
- **Provide services on different platforms**: with the emergence of smartphones and tablet computers it becomes necessary that the university provides its services not only electronically but also in a form compatible with these different platforms. The current application architecture is built on client-server pattern. While this pattern might have served UQU well when the system was first envisaged, the system will not support the adoption of modern business model. So, the establishment of web services in a SOA [6] composite is a necessary requirement. We have generated a comparison matrix of three commonly known architectural styles in the market namely, client-server, n-tier, SOA. Table I below presents this matrix.
- **Support higher management and decision makers**: the significant growth in operations makes it necessary to monitor the progress of its different businesses by higher managements (i.e. Rector). Currently, few reporting applications are provided to the higher managements that give very limited KPIs. This shortcoming requires constantly visiting different sites of the universities to get live updates of progression. We believe this is another time consuming practice that may affect the core duties of higher mangers. We also believe a fully integrated system architecture can facilitate the establishment of Business Inelegance (BI) model on top of it in order to serve the higher management business needs.
- **Facilitate smooth cross-applications interactions**: Different applications interact with each other randomly where glue-code is used frequently to establish the linkage. Fig. 1 below exemplifies the current interaction between different applications within the ERP system of university.

TABLE I: ARCHITECTURAL STYLES MATRIX

Characteristics	Client-Server	N-Tier	SOA
Language/Technology Dependency	Dependent	Dependent	Independent
Extendibility	Hard to Extend	Hard to Extend	Fully Extendable
Modifiability	Major changes to source code	Major changes to source code	Support Plug-&-Play effectively
Re-use Legacy system	Requires an Adapter	Requires an Adapter	Full support of Re-using legacy systems
Data Exchange	Internally	Internally	Internally/Externally
System Integration	Homogeneous	Homogeneous	Heterogeneous
Maintenance	Affect all parts of the system	Affect most of the system parts	Only a single service is affected
Separation of Concerns	Low	Medium	High

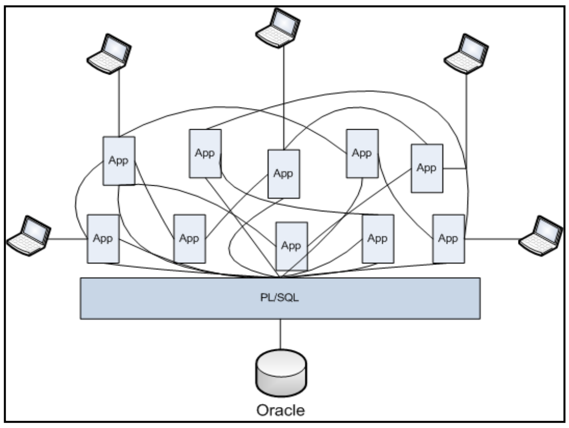

Fig. 1. Current view of system design at UQU

As the above diagram shows, the ERP system is highly complex and the situation is most likely going to worsen should the university expanded the system to accommodate new emerging business needs in the same manner. Our work mainly addresses this point as we are trying to establish a well-organized SOA based system architecture that can facilitate extensibility in a controlled manner.

Based on these business requirements, we can conclude that current systems architecture at UQU is not capable of expanding the university business. The goal of this project is to establish new system architecture capable of accommodating emerging business needs and subsequently fulfills UQU IT potential.

IV. REFACTORING LEGACY SYSTEMS

Legacy software systems [7] represent the backbone for many organizations nowadays due its significant value to fulfill their business needs. However, it is always the case that organization might be at a great risk if their legacy systems suffer from any sort of failures as maintaining such systems can be very hard if not entirely impossible in some cases. As a result, legacy systems can be either removed with brand new systems or they can be refactored.

Generally speaking, refactoring is a process of improving the underlying design and structure of source-code components that subsequently can improve their performance and maintainability [8]. Software programmers and developers exploit this aspect by implementing different codes to achieve a desired behavior. While different codes can lead to having the same software behavior, the software code's readability and comprehensibility are significantly affected. However, a more streamlined approach to software refactoring involves reducing the size of software source codes. This is achieved mainly by taking certain code paths which will then be transformed into structures or data at runtime. Refactoring software is probably a new approach to software design because, in conventional terms, software undergoes design before it is developed or built, especially in the Object Oriented Programming technology that uses object models. However, developers have learned during object code implementation that a better design befitted the object. Refactoring ensures that such a need for improvements can be accommodated on demand, although it affects the design review process. However, it is important to assert that refactoring affects the design and not the functionality of the software system.

We have I identified a number of different usages of the term refactoring in the literature. One usage has described refactoring as the process of removing duplication from source code in order to enhance its readability and structure [9]. Another one described refactoring as a mechanism to generalize source code in order to make it reusable into deferent contexts [10]. A third usage of the term refactoring in the literature describes refactoring as a way to identify design patterns in the internal structure of a system [11].

Our view of refactoring is slightly different than the ones described above. We are more concerned with modifying the architecture [12] of software systems than the internal design and behavior. A comparable model to ours can be found in [13] which introduced the concept of *architecture refactoring* and establish the idea of breaking the code into more logical units. However the latter's refactoring approach deals with extracting methods from huge chunk of code to enhance the modularity of the code and not like our view that concerns the overall architecture of the enterprise system.

The term *re-architecting* will be used to refer to refactoring. The precise definition of the term is *the process of modifying the architecture of a legacy software system without affecting the functional and workflow aspects of it in order to comply with new architectural style*.

V. Conceptual System Architecture

Currently, many systems hardcode their business processes into the source code. In other words, whenever new business processes are required the original code has to be modified. Moreover, applications are integrated in a one-to-one pattern by writing a glue code to achieve the integration. This glue code is usually written as a mediator between two applications. Although this approach sounds like a simple - yet effective - solution to developers, it in fact results process design being incredibly complicated. In some cases the glue code is injected into one of the applications themselves. This is the worst setup as it generates a tangled code that gets only more complicated over the years especially when developers are dealing with an enterprise solution.

The framework uses SOA as an integration facilitator mechanism not as service delivery. The framework is composed of different layers that, based on our previous work for analyzing a number of systems [14], any enterprise solution in the market must satisfy in order to ensure flexibility and extensibility of their systems. Figure 1 presents the proposed architecture for an enterprise solution.

Each layer is independent from the surrounding layers in terms of main function. The description of these layers is as follows:

- *Data Access Layer*: this layer is responsible for managing the interaction between application and database, and hiding the databases used in the organization. Subsequently, if different database technologies are used (e.g. MYSQL, Oracle), this layer will manage the connectivity with the corresponding source.

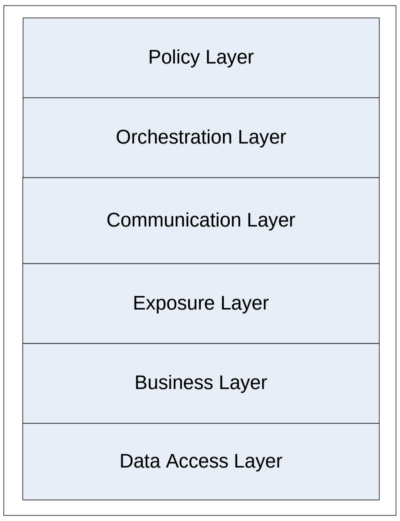

Fig. 2. Architectural layers of enterprise solutions

- *Business Layer*: this layer is responsible for executing the basic functionality representing organizational business needs. In the context of an ERP solution, this layer represents the fundamental modules offered by the solution such as HR, Finance, Projects, and Sales. Every one of these modules must be a standalone application that is not aware of any other modules.
- *Exposure Layer*: this layer is responsible for exposing the available applications from the application layers into services (e.g. web services, com components). All applications are therefore decoupled from their underlying environment and made available through request-response interaction mode.
- *Communication Layer*: the integration layer is responsible for establishing the communication pattern and routing protocols that enable service discovery and interaction. It defines the policies that comply with the standards adopted by vendors. For example, web services interact by exchanging SOAP messages over HTTP protocol. Any interaction between services must be accomplished through this layer. This is usually referred to as the Enterprise Service Bus (ESB) layer.
- *Orchestration Layer*: this layer defines the business processes that are employed by an organization. It is responsible for establishing the sequence by which services are going to be invoked to satisfy business requirements. For example, an attendance service might need to issue a request to a finance service to deduct a certain amount from an employee salary.
- *Policy Layer*: this layer is responsible for defining the privileges for accessing services. A different level of access rights can therefore be granted at this layer according to the defined policy.

The identified layers are not interchangeable as they must build up from the bottom-up. For example, a database can be established and tables created for an ERP system. Then, a number of standalone applications are developed on top of these tables to utilize the data in the tables. These applications must then be exposed in a standard manner in order to facilitate their integration with other applications to achieve new business needs. The new exposed interfaces are pooled and made ready for requests. Workflows can then be defined on top of the available pool of services in order to integrate different applications seamlessly. In fact, a workflow defines the design of a system where different components can be executed in a pre-defined sequence. Once all the business requirements are established (i.e. all functionality is implemented), there should be privileges assigned to personnel who are authorized to execute certain processes in the system.

VI. Proposed Architecture

UQU is moving steadily and progressively toward providing its clients with e-government services which goes in line with the university's IT ambitions. A main objective from the university's website reads "fully automating all the internal processes and establish rigorous infrastructure capable of supporting internal and external exchange of data." In fact the organization dedicates huge resources and funds in order to achieve this objective.

This objective requires establishing a comprehensive architecture in order to facilitate seamless integration of different systems. UQU currently operates in three different environments, SharePoint, PHP, and Oracle. Our proposed architecture is designed to integrate all systems regardless of technology in a rather neutral manner. The proposed architecture model is described below.

The figure illustrates the proposed architecture for facilitating the adoption of the emerging business need of UQU with available resources in mind. The main objective of the model is to promote a fully integrated environment that facilitates internal and external data exchanges, in addition to supporting scalability for future development. UQU currently owns the full package of SharePoint 2010, an in-house built Oracle ERP solution, a website (and other PHP services), and an Internet Information Server (IIS).

In the proposed solution, SharePoint is utilized to fulfill two main roles; the web presence and the service orchestration layer where business processes are defined through windows workflow foundations (WWF) provided by the SharePoint workflow engine. Services are exposed to SharePoint through the Microsoft-IIS layer where web services are defined. As a result, every application must be wrapped and exposed as a standalone web service that can be consumed by SharePoint.

This capability simulates the basic functionality of the well-known Enterprise Service Buss (ESB) pattern for service integration and management which represents the communication layer for integrating the various applications in an organization. SharePoint 2010 must work only on SQL server, hence, in this solution; we propose to use the SQL server for document flow management purposes without interfering with the university database by any means.

This architecture proposes a flexible solution to the ERP system within the UQU and also establishes rigorous platform for any potential ECM functionality required by the university. SharePoint 2010 and Microsoft-IIS jointly provide the necessary pool and management of services. They facilitate services orchestration in order to enable the interaction between the different services of the system. Any new service can be exposed into this layer and then composed with other services by defining a workflow that corresponds to a predefined business process model.

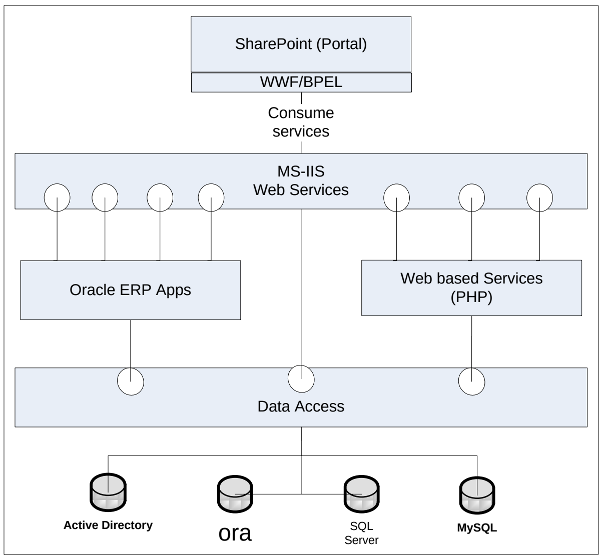

Fig. 3. UQU proposed architecture model

Ideally, the resulted architecture should promote high degree of extensibility and flexibility where different business processes within or between departments become fully automated. The first step toward that ultimate goal, as far as system structure is concerned, is re-architecting of the legacy system in order to increase the flexibility of IT within UQU. Re-architecting UQU legacy systems with SOA concepts in mind allows for a quick response to changing market needs, can implement IT systems that can quickly adapt to changing markets, shifting customer requirements, and new business opportunities.

VII. Anticipated Benefits of the re- Architecting Approach

Legacy systems' re-architecting is a cost-effective modernizing alternative to the creation of software systems in an organization when a new market or business need arises. Given that purchasing new software entails huge cost implications on an organization, it is commercially viable to consider improving the proven legacy systems instead.

The costs associated with buying a new system are rapidly adding up to the actual price of the software, rollout, and training. In fact, new systems might cost even more to maintain in early stages because a system might require vendor intervention periodically [15]. The alternative approach in comparison requires low operational costs, with the software built on existing hardware and other systems [16].

The re-architecting approach is also a low-risk modernization option in the sense that existing software is already tested in dealing with existing business needs. This is a huge advantage over software systems yet to be piloted and deemed fit for an organization's needs. It allows an organization to transform its existing legacy applications to meet the current market demands without overhauling the entire system. This, in turn, minimizes the loss of existing IT systems' investments in which the legacy systems hold crucial information and data that is required in the daily operations of the organization. Another advantage of this approach is that re-architecting encourages the development of a custom software system architecture that is based on the organization's demands and capabilities. This is because this process allows the re-architecting team to survey and understand not only the requirements of the new system, but also the overall capabilities of the organization in managing the new system.

Consequently, this enables the development of a system that the organization can use and manage with relative ease. During the re-architecting operation, highlights that re-architecting legacy systems gives the development team an opportunity to transform the current system user interface to the popular web-based user interface if it is not in place already. This helps the users interact with the new system in a friendly manner and, thus, enable the usual operations of the organizations to run with minimum delays. Software system re-architecting approach allows customization in the training of the system users and maintenance teams. This is realized through re-architecting experts who train the users at each stage of the development process, thus, enabling them to understand the new system. This is achievable since re-architecting targets certain system enhancements concerning the central part of the solution which aims at handling given business needs.

Improving legacy systems helps sustain an organization's reputation because it principally helps minimize any interruption to routine business operations. This means that customers are faced with a minimal impact, if at all. This is critical in businesses where reputation is very important, particularly due to competition. A situation where rolling out a new system takes many days to complete is not acceptable. Business operations would have to halt until the system is up and running while clients may lose patience with an organization unappreciative to their specific requirement. Finally, this approach grants an organization the opportunity to employ modern technical architecture such as Service Oriented Architecture and Cloud Computing Architecture. These have tested and happen to require certain levels of flexibility. For instance, when SOA is implemented to support business intelligence (BI), it allows a flawless technology integration to form a consistent BI environment that makes the delivery of data straightforward while simultaneously simplifying low latency diagnostics.

VIII. Proposed Migration Roadmap

SMART [17] process, briefly described earlier, helps examine the feasibility of migrating the legacy systems into the new SOA-based environment. The following steps are required to successfully conduct the process:

- Refactor applications in order to eliminate potential decoupling applications in a way that each provides its standard set of functionality without reference to other applications in the system.
- Extract stored PL/SQL procedures in the Oracle DB and wraps them with containers to be exposed as web services.
- Business logic must be separate from the Oracle forms using the Model-View-Controller (MVC) architectural pattern. In other words, business logics can be accessed from different views and not restricted to a single usage. This might be achieved through the migration to the ADF [18] and extracting the source code from oracle forms and encapsulate them in a well-defined business component (BC) models that can be invoked directly by forms. Thus, functionality embedded in forms can be de-coupled in self-contained components.
- Establish the linkage between forms, BC web-service, and PL/SQL web-service. Ideally, forms should be hardcoded to invoke BC services. However, BC services must interact with the PL/SQL services via a defined work flow in order to support dynamic binding. Thus, no code will be used to establish the linkage between services.
- The resulted web services must be exposed through Microsoft-IIS that establishes messages routing protocol between web services. The Microsoft-IIS is considered as the service layer.
- Active Directory must be integrated to the service layer in order to provide credential check and assign basic privileges to users according to their pre-defined profiles.
- Utilize the workflow (WF) engine provided by SharePoint 2010 in order to implement business processes. The implemented WF represents the main thread of control that establishes the design for consuming the exposed services. Subsequently, services can be placed and executed in a sequence to fulfill business requirements.
- The functional interface must be separated from the architectural interface. Therefore, UQU team must identify the business logic including data link, connectors, and modules life-cycle control code and separate them from the core functional business logic. This helps identify the potential functional services directly consumed by clients and separate them from any supporting services that may be related to the architecture of the legacy system.

IX. Re-Architecting Experiment

Smart Developer [19] provides the Baz tool that converts legacy applications built using Oracle Forms to Oracle ADF 11g in a very high speed and punctuality. It saves the conversion effort by more than 80% from the total project duration. Baz is a tool to consider for re-architecting legacy systems in order to comply with the MVC architectural pattern. One of its distinguished characteristics can be expressed in the ability to convert directly from any Oracle Forms version to ADF without converting to Oracle Forms 10G. The tool supports all natural languages including Right-to-Left direction languages like Arabic. Possibility to build tailored conversion that suits each customer special needs. The operation of the Baz tool is conceptually described in Fig. 4 below.

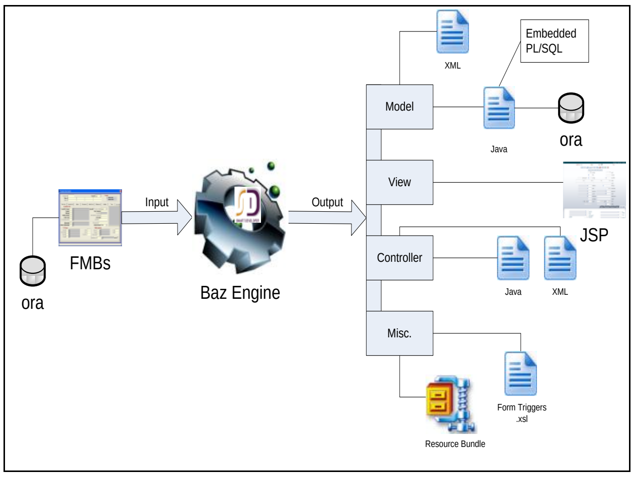

Fig. 4. The operation of Baz tool

UQU has conducted a pilot project in collaboration with Smart Developer to evaluate the applicability of the Baz tool to migrate one of its admission modules from Oracle 6i into ADF, hence provide the information on proceeding in the proposed re-architecting approach.

The project included a training package to UQU team to help understand ADF and Baz tool. The overall duration of this project was 60 days. However, the migration tasks only lasted for two weeks.

A total of 21 forms from the admission system were selected for this pilot project; they include the triggers listed in Table II.

The selection of forms reflects the needs of the University's Department of Admission and Registration as its operations require accessing the forms off-campus frequently. The Department used Citrix [20] and VPN to establish the connection with the university's network. However, the response time was extremely slow, in some cases committing a transaction took more than 25 minutes.

As a result, our selection was in favor of performing the migration to ADF which will be accessible via the web as web-services. Thus, eliminates most of the network related issues that caused the delay in the previous connection scenario.

TABLE II: SIS ORACLE FORMS

FORM_NAME	TITLE
SIS_FORM089	Application Request
SIS_FORM091	Admission Policies
SIS_FORM095	Pre-Approval
SIS_REP270	Smartcard Details
SIS_FORM101	Blacklist
SIS_FORM104	Applicant Information
SIS_FORM023	Applicant Contacts
SIS_FORM147	Admission Procedure
SIS_FORM148	Final-Approve
SIS_FORM153	Enrolment
SIS_FORM151	Qualifying Exams Results
SIS_FORM221	Application withdrawal
SIS_FORM224	Admission Amendment
SIS_FORM270	Admin Exams Date
SIS_FORM300	Qyas Data
ATS_FORM01	Logon
SIS_FORM307	Edit Student Information
SIS_FORM277	nomination
SIS_FORM313	Change Admission type
SIS_FORM324	Filtering Application Requests
SIS_FORM311	Notifications

Baz tool has been applied to every single form listed in the table above. All the user interfaces have been successfully converted to JSP.

The tool has been applied to convert 120 triggers within the FMBs, 89 of which were successfully converted automatically using the tool and 31 were not. To validate the output of the Baz tool, all the generated forms and classes were examined manually to check their functionality. It was found that 89 JSPs provide similar business logics as those in the input FMBs. The remaining 31 were considered either deprecated or required manual conversion. The provisional listing of omitted triggers is given below:

WHEN-BUTTON-PRESSED

RUN_MNU

WHEN-NEW-FORM-INSTANCE

SET_MENU_ITEMS

PRE-INSERT

POST-QUERY

POST-INSERT

WHEN-NEW-BLOCK-INSTANCE

WHEN-VALIDATE-ITEM

POST-FORMS-COMMIT

WHEN-NEW-FORM-INSTANCE

KEY-COMMIT

WHEN-BUTTON-PRESSED

The overall results of the experiment revealed that the Baz tool has converted 74.16% of the triggers automatically into the new architecture in comparison to 25.84% not converted. The unconverted triggers were the business logics that require human intervention. The average time and effort required to complete the conversion of the remaining triggers were estimated at 45 minutes per trigger totaling 8 working days.

X. DISCUSSION

Technically speaking, the outcome of this experiment provides ample support to opt for re-architecting legacy systems. The SIS module has been converted successfully into MVC that fits into the SOA style, and the experiment proved that Baz tool is capable of performing wide range of conversion as the selected module is fairly complex compared to the other modules in the system.

Considering other aspects that may affect a decision to proceed; we identified a matrix to compare buying and re-architecting. The results displayed in the table are obtained upon consulting vendors in the market for purchasing a new system and also upon Smart Developer's proposal for the migration project. The summary is in Table III.

TABLE III: BUY VS. RE-ARCHITECT

	Buy	Re-Architect
Cost	5M$-8M$	~= 1.2M$
Customization	substantial	none
Training	substantial (users + Developers)	very minimal
Ownership	UQU + (X-Company)	UQU only
License	yearly	support only

The table explains the different factors CEOs may wish to consider prior to making a decision on whether to proceed in one route or the other. The cost is a major factor and, as far as re-architecting is concerned, is a great advantage over purchasing. Other important factors include the amount of customization and training needed in both situations which again supports the model proposed. Finally, CEOs should also consider the long term effect of their decisions including legal matters of ownership and licensing both of which pay dividends to the proposed model. The return investment is a plus point that should always be a factor in the decision.

As a result, we strongly believe that re-architecting legacy system is much more beneficial to cost-conscious organizations with a capable IT team who can ensure conducting the migration smoothly. However, for organizations which usually outsource their development projects, we recommend purchasing over re-architecting as that will be much more applicable especially if funding is not a major issue.

XI. Conclusion and Future Work

This paper accounts for one major obstacle that affects the decision to adopt e-business solutions in any given organization which is the lack of standard architecture of the legacy systems. The paper indicates that re-architecting legacy system can be beneficial to many organizations in improving the architecture of their systems without affecting the underlying business logics. We summarize the main advantages of re-architecting in the following points:

- Re-architecting connects legacy business logic with modern technologies and concepts.
- Re-architecting can evolve legacy applications into SOA-based deployments.
- The new system will require less time spent coding when modifying or developing logic.
- Being based on SOA concepts and built on an advanced framework, the new system will be flexible, transparent, and reliable.
- The new system will be expandable without developing the risk of a 'spaghetti architecture'.

The Baz tool has been applied to one module of student information system as a pilot scheme to evaluate its applicability. The experiment revealed that the Baz tool was successful in converting Oracle 6i forms into Oracle ADF, and this is goes in line with the planned re-architecting outcome where business components have been successfully extracted and converted into web services.

The next stage is to utilize the Baz tool in performing the conversion of 6i forms into ADF compatible components for the remaining applications of the ERP system. Upon the completion of the conversion process, we will develop different layers of the proposed design in order to migrate the entire UQU systems into this new architectural pattern.

References

[1] C. Holland and B. Light, “A Critical Success Factors Model for ERP Implementation,” *Software IEEE*, vol. 16 no. 3, pp. 30 – 36, 1999.

[2] K. Bennett, M. Ramage, M. Munro, and M. Decision “Model for Legacy Systems,” *Software, IEE Proceedings*, vol. 146, no. 3, pp. 153 – 159, 1999.

[3] R. C, Seacord, D. Plakosh, and G. A. Lewis, *Modernnising Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*, Boston: Pearson Education, 2003.

[4] Umm Al-Qura University. [Online]. Available: http://www.uqu.edu.sa

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, “Pattern-Oriented Software Architecture,” vol. 1, 1996.

[6] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, 2005.

[7] J. McGee, “Legacy Systems: Why History Matters,” *Enterprise Systems Journal*, 2005.

[8] T. Mens and Tourwe, “A survey of software refactoring,” IEEE Transactions on Software Engineering, vol. 30, no. 2, pp. 126-139, 2004.

[9] M. Fowler and K. Beck, Refactoring: improving the design of existing code. Addison-Wesley Professional, 1999.

[10] H. Washizaki and Y. Fukazawa, “Automated Extract Component Refactoring.” in *Proc. the 4th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2003)*, pp. 328-330, 2003.

[11] D. Heuzeroth, T. Holl, G. Högström, and W. Löwe, “Automatic Design Pattern Detection,” in *Proc. the 11th International Workshop on Program Comprehension co-located with 25th International Conference on Software Engineering*, 2003.

[12] L. Bass, P. Clements, and R. Kazma, *Software Architecture in Practice*, 3rd Edition, SEI Series in Software Engineering, 2012.

[13] S. Michael, Software Architecture Refactoring, OOPSLA, Tutorial T12, Montréal, Québec, Canada 2007.

[14] B. Y. Alkazemi, “A Conceptual Framework to Analyze Enterprise Business Solutions from a Software Architecture Perspective,” *the IJCSI*, vol. 9, no. 3, 2012.

[15] D. Reeves, “Legacy systems re-engineering: leveraging your existing assets,” Revenue Solutions, Inc, 2009.

[16] A. Umar and A. Zordan, “Reengineering for service oriented architectures: a strategic decision model for integration versus migration,” *Journal of Systems and Software*, vol. 82, no. 3, pp. 56 – 64, 2008.

[17] L. Grace, M. Edwin, S. Dennis, and S. Soumya. “SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment,” CMU/SEI-2008-TN-008. Software Engineering Institute, Carnegie Mellon University, 2008.

[18] M. Driver. (2012). Oracle Application Development Framework: Past, Present and Future, Gartner. [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-1BU762S&ct=120827&st=sb

[19] Smart Developer Co. [Online]. Available: http://www.sd4it.com/Baz.html

[20] Critix Website. [Online]. Available: http://www.citrix.com/