Towards Flexible Business Software

Ahmed Elfatatry
Information Technology Department
Alexandria University
Alexandria Egypt
elfatatry@alexu.edu.eg

Abstract—**Software flexibility is a multidimensional problem. Solving one side of the problem might not enhance the situation significantly. This work is motivated by both the problem of software flexibility and the need for a solution for highly volatile business software. The work presented here is based upon ongoing research into software flexibility. The main contribution of this work is the proposal of a new framework to facilitate frequent changes in both the business layer and the presentation layer. Among systems that benefit from such design are workflow systems and document oriented.**

Keywords-Software Flexibility; Document Oriented Systems; presentation layer

I. INTRODUCTION

Software flexibility is the ease with which a software system can be modified in response to changes in system requirements. Software flexibility is a multidimensional problem. Solving one side the problem may not improve the situation significantly. When software is built out of layers, often, applying changes to one layer affects other layers.

Changing one part of a system may require changing a number of related parts; this is known as the "propagation effect" of change. Each of the related parts may need to be dealt with differently. For instance, a change request may affect business rules, user interface, and data. Each of these facets needs to be designed in a way that facilitates change.

The focus of this work is flexibility in business software systems. While all software systems could be subject to change, business software systems are more likely to change as result of their changing environments. Flexibility problems in business systems vary according to the type of the system. Business software systems include business information systems, workflow systems, and document oriented systems [1]. In workflow systems, for instance, modelling techniques produce tightly coupled systems [2]. Minimal change in business requirements may require the change of many parts of a given model. A case in point is the model adopted by the Workflow Management Coalition (WFMC) which embeds transition information within activities [3]. As a result, changing the sequence of activities may require rewriting such activities. Other models integrate business rules within the specification of the activities. This results in activities that are complex and hard to maintain.

A Document-Oriented Application (DOA) is a type of business applications that is built around business documents. User interface in DOAs is both stage-based and role-based where it displays and manipulates business documents in several stages for different roles. Such characteristics bring about a common requirement for applying consistent stage-based and role-based presentation behaviour throughout the entire application.

Adapting DOA after it has been deployed in production usually involves allowing business-experts to change business rules including specifications about stages and/or roles for business documents. Combining this requirement with the stage-based and role-based characteristics brings about a design challenge: the application should be designed to support flexibility both in the business layer and the presentation layer. In other words, the changes made to the business layer should also affect the presentation layer in a consistent manner.

This paper is structured as follows. In Section 2, the problem of building flexible business systems is analysed. Section 3 introduces a framework for dealing with flexibility issues. The evaluation of the proposed work in presented in Section 4. Section 5 discusses the contribution of the work and outlines the future extensions.

II. PROBLEM AND MOTIVATION

Large changes in business requirements naturally lead to large changes in the supporting software systems. When small changes in business requirements lead to large changes in the supporting software system, this indicates the presence of a design problem. In this work, flexibility related problems are classified into two main classes. Each class exposes a different perspective of the system.

A. User Interface problems

An important class of business software is Document Oriented Applications (DOA). A Document Oriented Application is a type of business applications that is built around business documents. In such systems, work procedures are done by exchanging documents according to some rules related to both the persons using the documents and the state of the given document. A case in point is the exchange of legal documents in a court. Current approaches used in building DOAs fail to solve the issue of reflecting changes in business logic to user interface in a way that retains flexibility [4]. Such approaches have a number of problems discussed below.

• Violating the separation of concerns concept by injecting large crosscutting concerns into user interface [5]. Crosscutting concerns are software features whose implementation is spread across many modules in the form of tangled and scattered code [6]. For example, reflecting presentation behaviour for the active role using current approaches of security architectures results in software that has application code tangled with security code. Such tangling makes it difficult to change security architecture once the software has been deployed [7].

• Concealing the high abstract view of business logic behind presentation changes and blending it within the presentation code. This hardens any attempts to understand or extract business logic that leads to a specific behaviour.

• Producing inflexible solutions that cannot cope with changes in business rules. This leads to DOAs that lose its ability to adapt change once it has been deployed in production. The typical solution to modify or to include new business rules requires a new cycle of development and testing for each modified rule.

• Preventing business-experts who have the required knowledge in a business domain from participating in adapting DOAs. Usually, business experts do not understand programming languages and therefore they cannot directly change the application [8]. Instead, they have to wait for IT-professionals to implement new business rules and to change the behaviour of the user interface.

B. Modelling Problems

Decisions at the conceptual level strongly affect flexibility. The chosen model of decomposition has a direct effect on the cost of change. This sub-section outlines a number of problems that may result from the modelling phase.

• *Inability to respond to frequent changes of business processes*. Most workflow modelling techniques produce tightly coupled systems. A minimal change in a business attribute may require the change of many parts of a given model. For instance, the model adopted by the Workflow Management Coalition [WFMC] embeds transition information within activities [3]. As a result, changing the sequence of activities may lead to rewriting of the activity body itself. Other models integrate business rules within the specification of the activities [9]. Such activities are complex and hard to be maintained**.**

• *Model inconsistency*. The addition or deletion of tasks, relationships, or rules at runtime may cause system inconsistencies especially when changes are done in an ad-hoc manner [10]. Consider a simple order processing where the billing step and the shipping step take place at same time. Assume that a change at run time is made so that the shipping step is performed after the billing step. If at the time of the change, a job had started with shipping, it will never perform the billing step according to the instructions of the new procedure. Thus, a customer will not be billed for the goods that he receives. If there are a large number of jobs being in the same situation at the time of change, then a large number of customers will not be billed. This is a very simple example of a "dynamic bug". Many of these bugs are much more difficult to detect and can have unexpected effects. In the following section, the proposed framework addresses these problems.

C. Research questions

The previous discussion of flexibility problems leads to a number of research questions. First: how can we build user interfaces that can accommodate changes in other layers of the software system? Second: how can workflow systems be more adaptive to change?

III. THE PROPOSED FRAMEWORK

To address the issues described above, we propose a framework for flexibility. The following sub sections describe the proposed framework.

D. Conceptual view

The proposed framework defines a workflow as a set of activities as shown in Figure 1. The upper part of the figure shows a design time view of a workflow. The lower part of the figure shows the runtime view of the figure. A workflow consists of one or more activities ordered according to some transition flow rules. Transition flows are not embedded within activities. They are modelled as first class entities. Each activity is assigned to a specific role according to binding conditions. Role binding rules postpone the assignment of an activity to an available user until runtime [11].

At runtime, activities are bounded to the appropriate services through service requests. Business rules can be bound to workflow at any time during its life cycle, providing the ability to customize the workflow while it is executed.

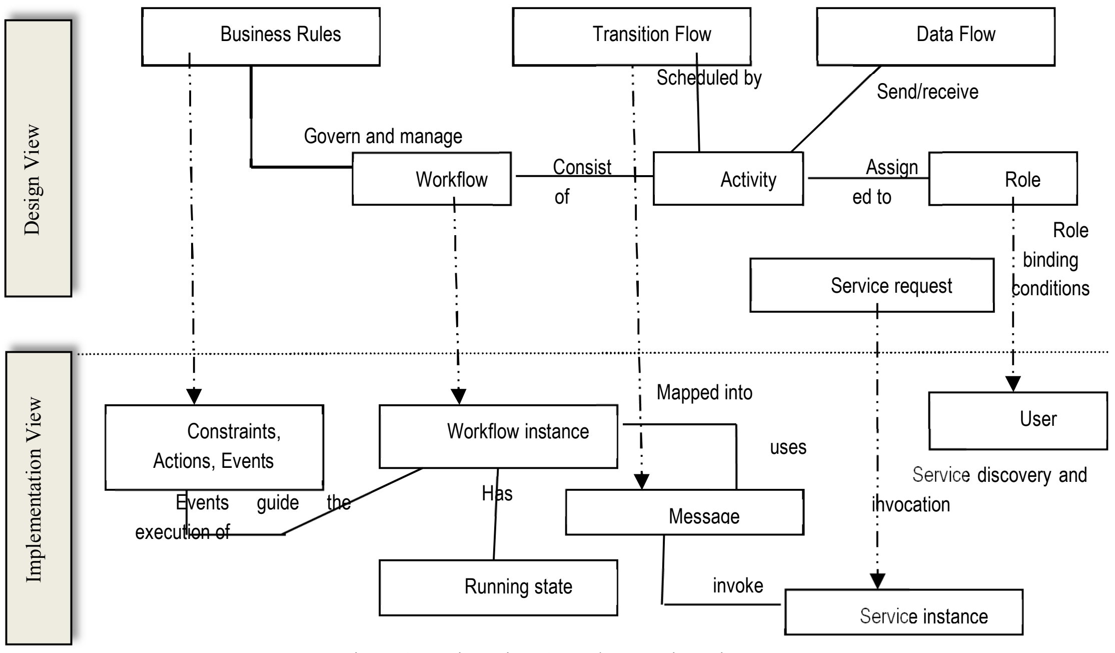

Figure 1. Design View & Implementation View

E. 3.2 Presentation Behaviour Layer (PBL)

In typical DOAs, a system is divided into three layers: Data-Access layer, Business layer and Presentation layer. In the proposed approach, we introduce a fourth layer: Presentation-Behaviour Layer (PBL) as shown in Figure 2. The main goal of this layer is to provide a mechanism for applying presentation changes in a consistent manner.

The PBL externalizes the logic of applying presentation-behaviours instead of hard coding it within the presentation layer. This externalization provides support for building flexible DOAs. The PBL consists of (PBM) and Presentation-Behaviours Run-time (PBR). The PBM is responsible for defining and storing presentation behaviours, while the PBR is the responsible for applying such behaviours during the runtime. The arrows show that PL uses services from BL and BL uses services from DAL. Arrows on the left, show the interaction between PBL and PL in response to a given change.

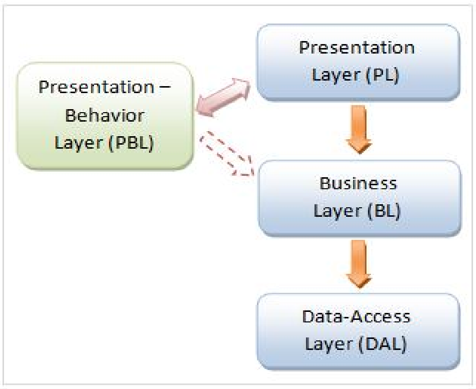

Figure 2. Presentation Behavior Layer

Presentation-Behaviour Model (PBM). The PBM consists of state machines and sequence flows. Each state diagram describes the behaviours that the system should apply at each stage of the process. One of the main objectives of PBM is to externalize and store full specifications about presentation changes outside the presentation code. The specifications are stored in XML documents which contain all the information required to describe how and when to apply presentation behaviours. When a change happens, it is analysed to its atomic element and then reflected to the presentation behaviour layer.

State machines. State machines are the ideal placeholders to store specifications about presentation behaviour for each process stage. They are suitable for representing the stages of business documents. In contrast to other approaches that blend presentation behaviour within the source code, the state diagrams keep the original definition of these behaviours inside the BPM model. Obviously, this simplifies the understanding of business rules that lead to a specific presentation behaviour. In addition, storing presentation behaviours in state diagrams representations rather than source code allows business-experts to participate in the development process by defining presentation behaviours for each business requirement.

In the proposed approach, we employ state diagrams to store specifications about business processes and their related presentation behaviours. Therefore, we need to store extra specifications about presentation behaviours for each combination of a stage and a role.

State: a state corresponds to a document stage in a business process. Usually the state identifies a significant point in the lifecycle of a business process.

Actions: an action represents a business logic that should run to perform a business task. In our approach, actions are modelled as sequence diagrams which provide simplicity and flexibility. Operations and Transitions are concrete forms of actions. From the user interface perspective, actions (Operations and Transitions) are reflected to user interfaces as tasks that can be triggered by end-users.

Transitions: a transition represents a change in the document stage. The transition connects a source to target state. At any given time only one transition can be executed for each document.

Guard conditions: a guard condition is an optional specification that describes business rules. It has to be evaluated before a transition can be executed.

Operations: an operation represents a business logic that should run to perform a business task. Operations can range from simple and common actions such as CRUD (Create, Retrieve, Update, and Delete) operations, to complex and custom tasks such as "Calculating Taxes".

Attributes: an attribute represents a document element that can be entered, modified and displayed. The concept of attributes is introduced to the proposed state diagrams to allow presentation behaviours to be defined at the granularity of attributes.

Roles: the role-based nature of business documents requires proper communication with access control model. In the proposed approach, we enriched state diagrams to define access controls for each element in each stage.

Specifying Presentation Behaviour. The proposed state machines have additional attributes that describe presentation behaviour. The objective of these attributes is to provide specifications that allow PBR to apply presentation changes automatically to user interfaces. The additional attributes deal with the following issues.

- **Controlling tasks.** User interfaces in document oriented applications provide end-users with a set of tasks that are appropriate for both active stage and role. Storing specifications about such tasks allows PBR to display proper tasks upon each stage change. Definitions of tasks include both visual and functional aspects. These specifications transform the tasks from being code-oriented to a higher and more abstract form. Such form is more business-expert oriented. It treats tasks as standalone elements that can be granted to or denied to certain roles.
- **Controlling default presentation modes and exceptions.** A document stage usually defines whether the user interfaces allow end-users to modify document information or not. The default mode allows readers to easily figure-out the expected behaviour especially in user interfaces that represent documents with large set of attributes.
- **Controlling common handlers.** The architecture of business documents results in common and redundant operations that could be applied to any document instance. For instance, all business documents provide common business operations such as CRUD operations, validation handlers, state transitions and etc. Although these operations are usually written centrally in the data access layer (DAL) and the business layer (BL) respectively, however, the code that calls them and displays their results to end-users is usually written in each user interface. Externalizing the decisions to activate or deactivate such common operations into the definitions of state machines provides more flexibility to adapt user interfaces according to the characteristics of each document stage.
- **Controlling default authorization mode and its exceptions.** Similar to the presentation mode, the default authorization mode simplifies defining authorizations to document information.
- **Controlling role access**. Although the default authorization mode discussed above facilitates the definitions of implicit authorizations, however, there is a need in some situations to define access roles in the granularity of attributes, transitions, and operations. We believe that this part is the most complex and is responsible for most of the crosscutting code.

IV. EVALUATION

At the architectural level, software quality attributes such as flexibility are hard to measure using direct quantitative measures. Other indirect methods are more suitable for the nature of this work. Two methods have been adopted to evaluate this work. The first method examines the effects of different types of changes on the proposed system and compares the results to those of traditional workflow systems. The second method evaluates this work by cross-referencing the features of this solution and a number of flexibility requirements.

A. Comparing the proposed framework with related work

One way to measure the success of the proposed solution to achieve flexibility is to test it on different scenarios of change and compare the ease of change with the results of traditional workflow management systems.

A common area of change in businesses is policy change. Policy changes usually have a substantial effect on workflows. Existing workflow models deal with business policies and rules in different ways. Usually, workflow systems introduce only a limited type of constraint that could be defined within an activity as a transition condition. Modeling business policies with such a model will be very hard. It may only be modeled as a new activity with different behavior, and different pre and post conditions which leads to a complex design.

Another way to model policies is to use a rule based workflow model. The entire workflow composition logic is specified in the form of if/then rules. Such a model determines the boundaries of a workflow, and leaves the freedom to the designer to specify the transitions between the activities. The workflow components such as activities, flows, roles, business policies are expressed in terms of activities built in process specification. This results in processes that are not modular, complex, and hard to maintain. In such a case, business rules are hard to change without affecting the core composition of the model. This way of modeling decreases the flexibility of the workflow.

The Proposed model introduces rules as a first class abstraction that governs and guides workflow execution. Each rule has enforcement conditions which state when and how such a rule is enforced inside the flow. Rules are not embedded within processes. Change in policies is enforced by changing related rules. This principle makes the workflow more simple and easy to maintain. Workflow enactment engine enforces policies by checking rules related to each step before performing it. Rules do not only govern activities but also govern role binding, services specifications, and exception handling.

The Model-View-Controller (MVC) is a software pattern for implementing the separation of concerns concept in the implementation of software systems. The work presented here focuses on providing a mechanism for reflecting changes on the presentation layer specifically.

SNATA defines service oriented architecture for N-tier application [11], however, it does not provide a mechanism for change propagation between layers.

B. Matching the features of the solution to the specified flexibility requirements

The proposed solution has been evaluated against a set of flexibility requirements. This set of requirements is derived from a number of well-established software engineering principles such as abstraction, separation of concerns, and loose coupling. The requirements are discussed below.

R1: Support model evolution. Evolution of workflows occurs over time as a result of changing tasks, priorities, responsibilities, and people. Modifications should be allowed at design time as well as at runtime. The proposed solution allows structural changes as well as behavioural changes. Structural changes allow model evolution. The Rule manager provides an interface to accomplish this requirement.

R2: Allow function/provider decoupling. The provider of a specific functionality may not be specified until runtime. Hard coding such information at design time leads to systems that are not flexible. In the proposed solution, activities are implemented as services. Services are selected according to some criteria that may not be known until runtime. Service selection constraints are sent through service requests to each running instance to select a suitable service and source of provision. A new activity or behaviour could be added at runtime to allow composition of a complex task.

R3: A workflow has to provide an integrated multiple view of a business system.
A workflow model has to provide high level of abstraction, and support visualization of its parts. The Proposed framework combines an activity based model, role model and a rule based model. A business system may be viewed from one or more perspectives: roles, processes, or rules. The proposed framework provides a multi-view modeling of a business system.

R4: Support the management of evolving workflow schema. Changes in business environment have to propagate to running workflow instances. A robust management system has to support propagation of change to running instances in a consistent way. The presented work didn't address this requirement.

V. CONCLUSION

The main contribution of this work is the introduction of a framework for dealing with change in business software. The focus is on workflow systems and user interface in document oriented systems.

A major drawback of current approaches for building document oriented applications is neglecting the impact of change in business rules on user interfaces. The result is having systems that are hard to change when business requirements change. While it may be easy to change the code related to business rules, the impact of such changes on the user interface may cause undesirable knock-on effect. For instance, many researches focus on how to provide flexibility in the business layer by providing workflow based solutions. However, the impact of such changes on user interface is usually ignored.

It is necessary that flexibility should be addressed in each logical layer and also between different communicating layers. That is why it is common that many business applications that provide flexibility in the business layer and also provide flexibility in presentation layer fail to sustain flexibility across the boundary between the two layers.

To address such problems, we introduced the Presentation Behaviour Layer (PBL) as a solution of providing flexibility between business layer and presentation layer. We believe that, the PBL can eliminate most of the crosscutting concerns usually found in document oriented applications to apply presentation changes while keeping flexibility. In addition, the visual representation of PBMs allows business-experts to modify their applications based on business rules without the need to touch the source code.

Building flexible workflow systems comes at a cost. The main cost is the implementation efficiency. While separating roles, business rules, and invocation conditions, leads to a flexible design, it certainly adds processing overhead.

Although a complete analysis of flexibility problems and limitations has been discussed, the proposed solution has mainly focused on modelling problems. Runtime limitations still need more research. Currently, we are working on enhancing the performance of workflow engines. The ongoing work focuses on the development of more propagation strategies and building workflow engines able to efficiently weave rules with activities.

Three medium sized companies with average of seven developers each have been chosen to implement the proposed framework. The framework will be applied to existing systems that are subject to frequent change requests. A comparison between the performance before and after using the framework will be published later.

REFERENCES

[1] C. Wiehr, N. Aquino, K. Breiner, M. Seissler and G. Meixner, "Improving the flexibility of model transformations in the model-based development of interactive systems," in *Proceedings of the 13th IFIP TC 13 international conference on Human-computer*

interaction - Volume Part IV, Lisbon, Portugal, 2011,pp. 540-543.

[2] S. Bhiri, G. Khaled , O. Perrin and C. Godart, Overview of Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services, Springer Berlin / Heidelberg, 2005, pp. 440-445.

[3] WfMC, "Interface 1: Process Definition Interchange," [Online]. [Accessed May 2015].

[4] O. Chapuis, D. Phillips and N. Roussel, "User interface façades: towards fully adaptable user interfaces," in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, Montreux, Switzerland, 2006, pp 309-318.

[5] A. Marot, "reserving the separation of concerns while composing aspects on shared joinpoints," in *4th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009*, Orlando, Florida, USA., 2009, pp. 837-839.

[6] A. Sabas, S. Shankar, V. Wiels and M. Boyer, "Undesirable Aspect Interactions: A Prevention Policy," in *Theoretical Aspects of Software Engineering, Joint IEEE/IFIP Symposium*, Montreal, Montreal, QC, Canada, 2011, , pp. 225-228.

[7] G. Chao, "Human-Machine Interface: Design Principles of Visual Information in Human-Machine Interface Design," in *IHMSC '09 Proceedings of the 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics*, IEEE Computer Society Washington, 2009, pp. 262-265.

[8] M. Mike and D. Dwight , "End user developer: friend or foe?," *J. Comput. Small Coll.,* vol. 24, no. 4, pp. 40-45, April 2009, pp. 42-49, 2009.

[9] T. Sterling and D. Stark, "A High-Performance Computing Forecast: Partly Cloudy," *Computing in Science and Eng.,* vol. 11, no. 4, pp. 42-49, 2009.

[10] M. Blake, A. Bansal and S. Kona, "Workflow composition of service level agreements for web services," *Decision Support Systems,* vol. 53, no. 1, April 2012, pp. p. 234–244,.

[11] A. Elfatatry, Z. Mohamed and M. Eleskandarany, "Enhancing Flexibility of Workflow Systems," *80 Int.J. of Software Engineering, IJSE,* vol. 3, no. 1, pp. 79-92, 2010.

[12] T. . C. Shan and W. H. Winnie , "Solution Architecture for N-Tier Applications," in *Proceedings of the IEEE International Conference on Services Computing*, September 2006, pp. 234-244.

[13] C. Ackermann, M. Lindvall and G. Dennis, "Redesign for Flexibility and Maintainability: A Case Study," in *Software Maintenance and Reengineering*, Kaiserslautern, Germany, 2009, 2009, pp. 259-262.

[14] A. Bruno, F. Patern and C. Santoro, "Supporting interactive workflow systems through graphical web interfaces and interactive simulators," in *TAMODIA '05 Proceedings of the 4th international workshop on Task models and diagrams*, Gdansk, Poland, 2005, pp 63-70.

[15] D. Gaurav, "A survey on guiding logic for automatic user interface generation," in *Proceedings of the 6th international conference on Universal access in human-computer interaction: design for all and eInclusion - Volume Part I*, Orlando, FL, 2011, pp. 365-372.

[16] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990,* pp. 1-84, Dec 1990.