VERTICAL OBJECTS IN A HORIZONTAL ARCHITECTURE: DESIGN ISSUES IN A COMPONENT BASED ARCHITECTURE FOR DOCULIVE

Kasper Østerbye
Norwegian Computing Centre,
OSLO, Norway.
Kasper.Osterbye@acm.org

Abstract

Doculive is a system for handling electronic healthcare records developed by Siemens Healthcare Systems (SHS) in Norway. The system provides a generic object oriented document model, which enables definition and management of documents and collection of documents. The system allows both definition of the structure and behaviour of documents. The behavioural aspects are programmed either in Transact-SQL (server side behaviour), or in C++ (client side behaviour). The main problem in this is that behavioural extensions require knowledge of a very rich data model, and its realisation in TSQL or C++. This is a major obstacle for the development of extensions outside the original development team. Components are an attractive way to enable developers outside the SHS team to specify behavioural extensions. This position paper focuses on the difficulties in achieving this in Doculive.

1 Introduction

A frequently cited motivation for component technologies is binary reuse, that is, the development of binary components that can be reused in different settings. A different attraction behind components is that one can create a kernel application, which can be extended using customised components. Such extensions can be developed in a language independent way, and they do not require that the extension developer have access to the source code of the kernel system. A main attraction to the kernel/extension architecture is the possibility for programmers outside the kernel group to develop the program for a specific purpose, freeing the resources of the kernel group to improve the quality of the kernel application.

Doculive is currently developed solely for the windows platform. Therefore, the component technology under consideration is COM/DCOM. One nice feature of COM is that communication between components is very efficient (under the right circumstances). This allows for a very fine-grained object model behind the components.

This paper will present the current architecture of Doculive, and show where we are heading. The contribution to the workshop is a real life example of a component architecture, and the problems we are facing in its design. Today, we have no "best" solution to the main problem, and hope to find one if not in advance then during the workshop. The main problem encountered is how a logical object can be divided into several physical layers, which are running on different machines, and how one can program such a logical object with minimal knowledge of the physical layers, as these logical objects are the very objects to be written by extension programmers.

We believe the problems encountered are general, and highlighting a problem inherent in any physically layered object-oriented system of some complexity.

2 The current architecture

The current architecture of Doculive is presented in Figure 1. The client program is a general presentation and editing tool, which is able to present the user with a consistent interface for arbitrary document types stored in the Doculive system. The client is highly optimized, as is the connection and interface to the server. The user interface provided by the client is one of the main selling points of Doculive, and hence that user interface must be maintained in future revisions of Doculive as well.

The server consists of three primary storage components. Firstly, Doculive provides a generic document modeling system. Therefore the schema definitions for documents are stored in the server, and new document types can be defined without recompiling the server or the client. The documents handled by Doculive are all sorts of structured documents relating to patient information at a hospital. The fact that different hospitals and departments within hospitals have their unique set of documents is one of the fundamental motivations behind having a generic document model.

Secondly, the server provides storage for the documents themselves. Thirdly, there are a number of auxiliary tables which represent administrative information about users, installed software, versions etc, but also tables which contain redundant information to allow faster access to often used information.

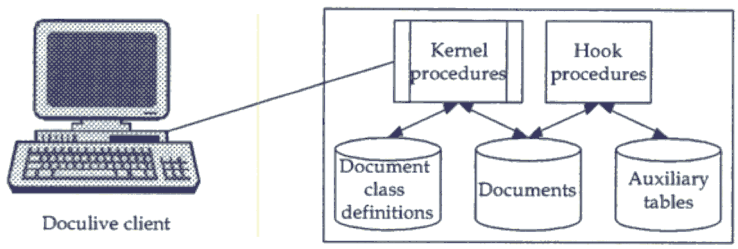

Figure1. Current Doculive architecture

The kernel procedures are stored procedures written in Transact SQL, and provide a set of access routines so that the client need not access the tables representing the documents directly. As an example, there is a kernel procedure which can create an instance of a given document class.

Documents do not only have structural aspects but also behavioural aspects. For example, if one changes a value for cholesterol, this might notify that a doctor should be notified under certain circumstances. As another example, if a document of a give type is created, it should be initialised with values from other documents. To separate such document specific behaviour, Doculive uses hook-procedures [Nørmark, 1995]. The Doculive kernel uses naming conventions for such document specific procedures. If a field named X in a document class Y in module Z is changed, the Doculive kernel will check to see if there is TSQL procedure defined that matches, X, Y or Z and call it if it exists. This way, it is possible to provide document specific behaviour without changes to the kernel procedures. The changes done by a hook procedure might initiate new calls of kernel procedures, which will recursively activate the application of hook procedures.

3 A new architecture

3.1 Target situation

We are currently faced with two architectural requirements. First, it should be easier to extend the system with new functionality that is not closely related to a specific document. One example is extraction of information form several patients; another is extracting selected pieces of information from a specific patient. Doculive encapsulates its data model behind stored procedures. Rather than making Doculive expose its relational representation, it is more attractive to develop a component based object model that reflects the document structure more faithfully. Providing a COM based interface to the kernel functionality of Doculive is also a request that has been brought up by our third party developers.

Second, it should be easier to extend the system with new types of documents with specialised behaviour. The specialised behaviours include 1) client-side extensions that affect how a specific aspect of a document is to be presented. 2) How changes in a document should course changes in other systems. 3) How fields in other document might be changed as a side effect of a change in the presented document. As already mentioned, we are interested in becoming in a situation

where such extensions can be developed using COM technology, preferably through visual basic components, as VB is easy to learn.

In addition, we would like to address these two requirements by providing an extension development environment, which should make it easier to develop extensions that address both the first and the second requirement.

Regarding the second requirement, the functionality needed for an extension to fit into the architecture differs from one physical level to the other. The client only need two methods, one which will produce a presentation of the underlying information, and one which allows editing of that information. The server has a large and document type dependent set of methods that it might call (corresponding to a COM based version of the HOOK procedures). Mostly, only a few of these will be used for any specific document class. Finally, the business layer has a set of document-type specific methods that enable access to individual fields on the document, and allows navigation in document folders. These methods can be automatically generated, and are useful at all three levels. The client part can use it to extract information, and store information received from the editor method. The hook procedures of the server can use the navigational and access methods to ensure consistency as is done in hook procedures today.

3.2 Architectural approaches

The first requirement can be addressed by moving to a three-tier architecture, in which the new middle layer provides an object-oriented component model, which can be used to manipulate the documents in the database. The present architecture can be extended to automatically generate components that can provide access to documents as COM objects. The extension development environment around Doculive is expected to include a component generator in the future, which will produce COM component that are specific to the document classes defined in the database. The COM based extension API of Rational Rose [Rational, 1996] is a good example of the kind of object model we want to address this issue.

This second requirement does not fit well with the three-tier model. The layers the architecture is horizontal, separating the system into three physical layers, a database, business objects, and a client. However, a document object has logical aspects that belong in all physical levels of the architecture, and thus provides a vertical organisation of the system. There are two different approaches to the problem.

- First, one object type is created for each document class, and objects are instantiated in one place in the system. The other layers can then access the objects through DCOM. The advantage of this approach is that there is only one object to consider for the developer. The interfaces necessary for all three layers can be generated automatically. In addition, the access methods for navigation in the business layer can have their behaviour generated automatically. The main problem with this approach is that the methods on the object become context independent. The methods that relates to presentation and interaction should not be invoked from the database or business objects layer, as there might not be any client available to present the object.
- The second approach is to generate three objects, one for each layer. This adds complexity for the developer, but avoids the overhead of DCOM, as the objects can be located in the layer where they belong. The problem with this approach is to keep the three object representations consistent.

The problem is that the first approach looses the context of the layer, and thus provides access to methods without taking into account the layer of the caller. E.g. the database layer should only access hook procedures, which in turn have restricted access to the object. In the first approach, some physical layer must be chosen as the "home" of the object, and no layer seems more appropriate than any other. The main problem with the second approach is how to keep the three physical objects consistent. We are seeking an automated way to do this, because we want the extension development environment to be as easy to use as possible. It is not very attractive for an extension developer to have to maintain consistency between three objects manually.

An extension of object-oriented models to support roles is discussed in [Kristensen & Østerbye, 1996] and [Østerbye & Kristensen, 1995]. It is possible to combine the two approaches, as illustrated in Figure 2 using this model.

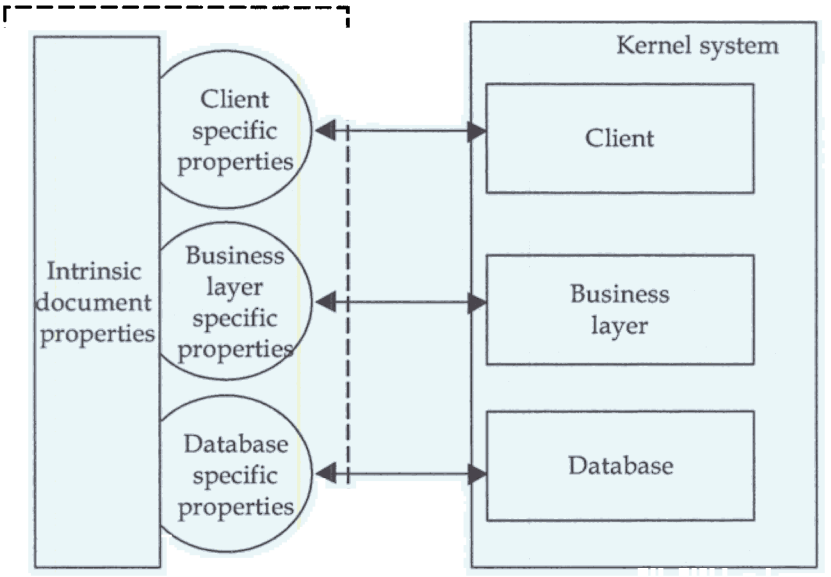

Figure2. Role-based approach to verticalbjects in a horisontal architecture.

The left part of the figure is the extension object, which has an intrinsic part, and three roles. The properties in the intrinsic part are shared between the roles. The properties of one role cannot be accessed from other roles (roles do not know about each other). Because the client in the kernel system has a reference to the client role, the client can access properties of the client role and the properties of the intrinsic object.

This approach avoids the problem of consistency by placing shared behaviour as intrinsic properties, and context dependence is achieved by providing a role corresponding to each layer.

3.3 Open issues

The role-based approach was not designed with a distribution of intrinsic object and roles in mind. Nor was not designed with components in mind. Mapping the role model into a component architecture will be the next step, and one of the issues we want to discuss at the workshop.

4 References

[Kristensen & Østerbye, 1996] Bent Bruun Kristensen and Kasper Østerbye. Roles: Conceptual Foundation and Practical Usage in Analysis, Design and Programming. Theory and Practice of Object Systems (TAPOS), Vol 2, No 3 1996 Pages 143-160.

[Nørmark, 1995] Kurt Nørmark. Hooks and Open Points. In: *Quality Software - Concepts and Tools*, edited by Jan Stage, Kurt Nørmark, and Kim Guldstrand Larsen, The Software Engineering Programme, Institute for Electronic Systems, Aalborg University, Denmark.

[Rational, 1996] *Rational Rose - Extensibility Reference Manual*. Revision 4.0, November 1996.

[Østerbye & Kristensen, 1995] Kasper Østerbye and Bent Bruun Kristensen. Roles. Technical Report R 95-2006, Aalborg University, Department for Mathematics and Computer Science, 1995.