Experiences of generating COTS components when automating medicinal product evaluations.

Radmila Juric
Stephen Williams
Peter Milligan*

School of Informatics
* Enterprise Architecture, DCSIT, GlaxoSmithKline, Greenford, UK.

This is an electronic version of a paper presented at the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05), 14-16 Jun 2005, Taipei, Taiwan, Republic of China.

Experiences of Generating COTS Components when Automating Medicinal Product Evaluations

Radmila Juric, Stephen Williams, Peter Milligan*
Cavendish School of Computer Science, University of Westminster,
115 New Cavendish Street, London W1W 6UW, UK
*Enterprise Architecture, DCSIT, GlaxoSmithKline, Greenford UB6 0NN, UK
juricr@wmin.ac.uk, williast@wmin.ac.uk , pete.s.milligan@gsk.com

ABSTRACT
This paper reports on experiences of generating COTS components when designing and deploying component based software architecture for automation and interoperation of medicinal product evaluations across different countries in the world. Our generic architectural model renders two sets of software components that are candidates for COTS components. We identify which role such COTS components may play and outline our approach of generating them. We advocate that such COTS components are developed with a specific component platform in mind and must adhere to constraints of our software architecture.

KEY WORDS
COTS components, healthcare, design patterns, EJB

1. Introduction

COTS Based Software Development configures software systems from reused components, which are plugged-in into software applications and bought/sold at their marketplaces [18]. The research into the CBSD is numerous and range from COTS component acquisition [21], problems and risks of selecting them [20][4], developing and deploying COTS components to deliver tailored software systems [2], to their the role in requirements engineering [7] and in software architecture [22]. The most intriguing is the problem of making them compatible with other heterogeneous components that make complex software systems [8], [6], [11].

Our work adds more to the above research and shows how COTS components can be generated, if they cannot be found at their marketplace. They can also be integrated to fit within our software solution and be reused by a family of related applications. We use a layered and component based software architecture that automates procedures for medicinal product evaluations [13]. When deploying our software architecture we identify two families of software components that are candidates for COTS components. We recognize which role such COTS components may play and outline our approach of generating them. Our COTS components are specific: they are developed with a certain component platform in mind, they comprise design patterns and they adhere to constraints of our software architecture. We believe that with such an approach we have not only alleviated the deployment of software components from our architecture by discovering COTS components, but have also addressed the problem of COTS components dependability on component technologies [6] and interoperability in run-time environments [8].

Section 2 details the related background and outlines our previous work. In section 3 we raise the issue of using COTS components within our problem domain. We give the aim of the paper and comment on related works. Section 4 describes how we generate COTS components: the constraints of our software architecture, the creation of design patterns and design decisions dictated by a component technology have created a set of COTS components. Section 5 lists their characteristics. We conclude and list future works in section 6.

2. Related Background

Medicinal product evaluation is one of the most important tasks undertaken by government health departments and their regulatory authorities in every country in the world. Each country has its own systems and procedures for evaluating medicinal products, which represents a serious drawback for their efficient local and worldwide registration. The automation of such evaluation procedures and adequate software support is a critical task that can improve the efficiency of regulatory authorities and interoperation of regulatory systems across the world.

In our previous works [13] we derived the layered and component based generic architectural model given in Fig.1 that allows automation of medicinal product evaluations, according to any regulatory authority evaluation procedure.

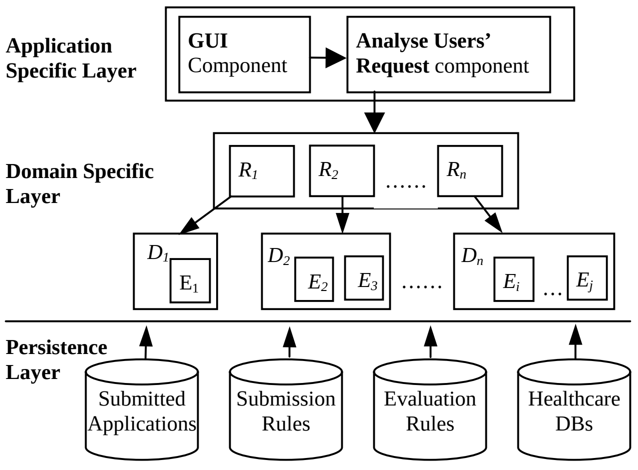

Figure 1. The Generic Software Architecture

The *application la*yer provides a basic GUI functionality and controls interaction between users and any other layers within the system. The *domain layer* consists of two families of components. The $\boldsymbol{R_i}$ family of components contains a set of *rules* that are to be followed if we want to have an automated application submission within a particular regulatory authority. The $\boldsymbol{D_i(E_i)}$ family of components contains all available *evaluation procedures* $\boldsymbol{E_i}$ that originate in different regulatory authorities and which can be applied to any submitted application (after the application has conformed to a set of the submission rules in $\boldsymbol{R_i}$). With $\boldsymbol{D_i}$ we denote that we chose any combination of evaluation procedures $\boldsymbol{E_i}$ which are relevant for a particular $\boldsymbol{R_i}$ and of interest for a particular country/regulatory authority. Components from the *domain layer* use various data repositories stored within components of the *persistence layer*, where data on submissions and evaluations are kept. We have implemented a prototype, where example components, placed within Fig.1, are modelled as an EJB application, using Studio Enterprise 7 [17].

3. Problem Formulation and Related Work

Our experiences from the previous works and from the prototype implementation have raised two issues:

(a) The deployment of components from our architecture requires a component technology whose communication infrastructure is embedded within our example components hence compromises their independence and our solution.
(b) The complexity of the problem domain should lead us towards acquiring COTS components, which may alleviate the implementation of the software architectural model from Fig. 1 and address the issue (a) above. Could we claim that some of or components are COTS component and if yes, which characteristics should they have?

The aim of this paper is to primarily address the question in (b) and see which role such COTS components may have when populating our generic software architecture.

We are not aware of any other work involving both COTS components and the problem of automation of medicinal products evaluations. There are related works within the EMEA http://www.emea.eu.int/htms/human/presub/q24.htm (European Medicine Agency) and within the US Food and Drug Agency - guidelines for Computer Assisted New Drug Application are at http://va.evolvingtech.com/etc/resources/FDAGuide94TOC.html). However, none of these solutions gives a generic architectural model that may serve any regulatory authority across the world and make medicinal evaluation practices interoperable. Bringing in new applications such as Updating Electronic Medical Records takes our architecture closer to the Distributed Healthcare Environment (DHE) which has been formalised in European Health Information Systems Architecture (HISA) [5]. Our components from the domain layer (and some from the application layer) can find a place within the CEN middleware [9] of common services.

4. Generating COTS Components

The potential COTS components have emerged from the final model of our application that automates medicinal product evaluations, i.e. after the decision on component technology was made and after a few design patterns that suited our application were generated. Therefore in this section we describe the process of modelling the application, i.e. designing the application components, using and generating design patterns and extracting potential COTS components.

4.1 Designing the Application Components

When designing our application we adopted the four principles in the following order:

(i) Choosing an adequate component technology,
(ii) Adhering to the layering principle and constraints from the architectural model and
(iii) Applying the Model-View-Controller (MVC) pattern [1] and generating design patterns.

We briefly comment on each of them.

<u>(i) Choosing an adequate component technology:</u>
Our analysis of available component platforms has short-listed the EJB and the J2EE platform, the CORBAmed standards and the Artemis architecture from [12]. The complexity of the CORBAmed framework and experimental status of the Artemis prototype lead us towards EJB. We have been geared towards J2EE because of our previous positive experiences of implementing software architecture for interoperable database as an EJB application [14]. Our decision to use EJBs has also been based on the fact that (a) EJBs are portable between various vendor implementations of J2EE, (b) EJB standard has been adopted by a number of vendors in order to provide EJB-compliant servers EJB and (c) EJB containers could shield us from component implementation complexities [16].

<u>(ii) Adhering to the layering principle and constraints from the architectural model</u>
(a) We separate components into layers, which conform to [3]. The components from the domain layer push away application specific requirements from generic functionality of computing platforms, making systems more adaptable to changes.

(b) The content of a particular component may be decided upon which layer it is appropriate to reside, i.e. knowing the layer in which the component resides, we know which functionality it implements.

(c)We can extend families of domain specific components $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ without affecting existing components in the same and adjacent layers. Furthermore, we may generate in advance domain specific $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components to suit new requirements/applications.

The architecture from Fig.1 shows the *Strategy* pattern [10]. We generate a family of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components that implement the functionality of checking the adherence to rules for submitting an applications and the functionality of evaluating submitted applications. These families of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components provide different implementations of the same behaviour, where the user's request (and user's understanding of the problem) decides the most suitable combination of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components.

<u>(iii) Applying the MVC pattern</u>
The components from the application specific layer are represented by JSP and Servlets in order to display and obtain information from the user. Servlets also implement workflow and session management. Components from the application layer accept a user input, analyse it, make invocations to the EJB components, and issue a response to a user. We use Servlets as the common entry point into the application. It is supported by a controller role given to Servlets in JSP/Servlet/EJB scenarios of the MVC pattern. It enforces a separation of Model (Entity Beans, JavaBeans), View (HTML, JSP) and Controller (Servlets, Session

Beans) aspects. The MVC pattern in Fig. 2 shows accessing DB records and performing the functionality of evaluating submitted medicinal product.

Servlets control *submission* of new applications through $\boldsymbol{R_i}$ and *evaluation* of submitted applications through $\boldsymbol{D_i(E_i)}$. We show in Fig. 2 the *evaluation* part, which is executed by component $\boldsymbol{D_i(E_i)}$. Components $\boldsymbol{R_i}$, their servlets/JSPs that assist in *submissions* are available in [17]. In Fig.2 the evaluation of a submitted application is controlled by `ApplyEvalServlet`, which delegates `SessionBeanEvaluating` to carry out the evaluation. However, before the evaluation starts `EvaluationServlet` uses a look-up session bean called `SessionBeanLook-upEvaluations` for retrieving all evaluation procedures available locally or globally, which are stored within the persistence layer. After displaying them through `DisplyEvals.jsp` we chose one suitable evaluation procedure. We use an entity bean `EntityBeanEvaluation` to retrieve a chosen procedure from an adequate persistence. In other words `EntityBeanEvaluation` PLUGS into $\boldsymbol{D_i(E_i)}$ `SessionBeanEvaluating` that perform evaluations. The results of the evaluation are stored using `Report` entity bean. Fig. 2 shows our design pattern named *evaluation pattern* [24], which can be used for applications of different evaluation procedures.

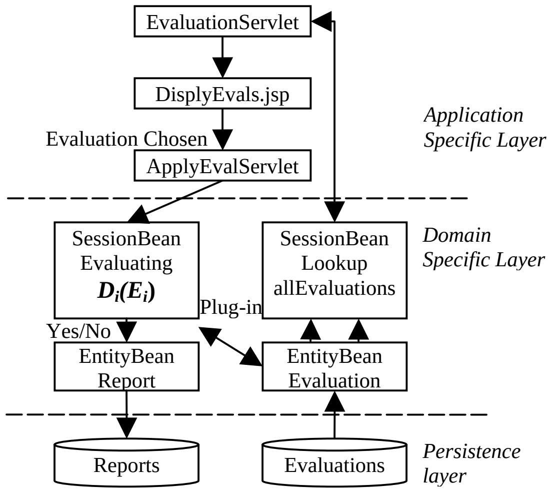

Figure 2: *Evaluation Pattern*

4.2 Candidates for COTS Components

The constraints from our generic architectural model listed in (ii) from section 4.1 allow the set of $\boldsymbol{R_i}$ and ***Di(Ei)*** components to be extendible, standardized and dynamically generated or posted from any persistent data stores. They are ideal candidates for COTS components because they represent an implementation of certain functionality, which can serve a family of related applications. In our example we could use $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components when evaluating medicinal products in any country and according to any evaluation procedure. Thus our $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components could be applied to a problem domain where we automate submissions of various kinds of applications and perform their evaluations according to a prescribed procedure (e.g. visa applications and their evaluation for the Home Office).

What makes the $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components very suitable for COTS components is that they operate on the principle of 'plugging-in' submission rules or evaluation procedures - stored in a persistent data store - into Session Beans that implement these functionalities, as in Fig.2. Thus, the programming code stored within `SessionBeanEvaluating` could remain the same for a variety of evaluations and submissions. What changes, are the submission rules and evaluation procedures stored within our Evaluation and Rule DB, which are plugged-in, using EntityBean into a SessionBeans. Thus `SessionBeanEvaluating` from Fig.2 is an example of $\boldsymbol{D_i(E_i)}$.

5. Characteristics of COTS Components

The COTS components, which can populate our generic software architecture in Fig.1, are sets of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components that populate the core layer of the software architecture. However, they are also EJB components, which means that they may participate in their own composition and exercise bindings with some other components according to communication principles dictated by the J2EE technology.

Furthermore, our COTS components comprise a functional core of the software that automates evaluation of medicinal products. However, they are generated according to constraints of the given software architecture, various design patterns used and design decisions dictated by the J2EE component technology. In the application specific layer we have JSP and Servlet components, which need services from the EJB (or COTS) components from the domain specific layer in order to carry out the required functionality. Thus the procedure of extracting the COTS components renders their main characteristics:

(a) Our COTS components are EJBs, i.e. we develop them having a dedicated component platform in mind;

(b) Our COTS components conform to our software architecture: a family of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components belong to the domain specific layer and participate in the Strategy pattern [10]. We also create and use our own design pattern called *evaluation pattern,* given in Fig.2, which emphasises that a given set of components are carrying out an evaluation procedure. Hence, the patterns and constraints from our software architecture put the Servlets and EJB components into a certain 'context' [24] [15] in which our application layer and COTS ($\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$) components may co-operate. The user defines the 'context' by choosing the most suitable implementation or combination of $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ components and their associated Servlets.

(c) Our COTS components are not necessarily middleware components: they are part of the MVC pattern where $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ are 'controllers', delivering business functionality.

6. Conclusions

In this paper we report on our experiences of implementing component based architecture for the automation of medicinal product evaluations, which generates COTS components. However, our COTS components have specific characteristics as outlined in (a)-(c) above: they are generated with an exact component platform in mind and they must adhere to constraints of the generic architecture.

Our views on the issue of COTS components integration into an application are rather strong. In today's world of ubiquitous computing, where the problem of interoperability is rapidly growing, we need to start trading-off and paying the price for heterogeneities we encourage in software systems. COTS components can alleviate the problem, but they raise the issue of their interoperability and dependability on component platforms [6], [8]. Hence our COTS characteristics from (a)-(c) in section 5 are the answers to such concerns. Furthermore, component's deployments are often given in the perspective of deploying EJBs or .NET-based components. From this point of view our proposal is not different.

We plan test our solution it in a real life example where more complex submission rules and evaluation procedures take place. Consequently we will be able to examine the way of marketing our $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ as COTS components. We also plan to test them for their interoperability, by placing them within frameworks such as [19] that manage components' dependencies when assembling them into an application. We want to see if Health Level Seven (HL7) (http://www.hl7.org) can act as a deployment mechanism for our component model. Such a work might give an insight into a role and different characteristics of COTS components, generated from the same generic software architecture in Fig. 1 and might also address (a) from section 5.

7. References:

[1] Alur D., Crupi J., Malks D. *Core J2EE Patterns*, 2nd edition, Prentice Hall, 2003

[2] Bandini S., De Paoli F., Manzoni S., Mereghetti P. *A Support System to COTS-based Software Development for Business Services*, Proc. of the SEKE '02, Ischia, Italy, 2002, pp. 307-314.

[3] Bass L., P. Clements, R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998, ISBN 0-201-199300.

[4] Bhuta, J. and B. Boehm, *A Method for Compatible Components Selection*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 132-143.

[5] Blobel B. *Application of the component paradigm for analysis and design of advanced health systems architectures*, International Journal of Medical Informatics, 60(2000), pp. 281-301.

[6] Chiang C.C. *Development of Reusable Components through the Use of Adaptors*, Proceedings of the 36th Hawaii Int. Conf. on System Sciences (HICSS), IEEE, 2002.

[7] Chung L and Cooper K *A Knowledge-Based COTS-Aware Requirements Engineering Approach*, Proceedings of the SEKE '02, Ischia, Italy, 2002, pp. 175-182.

[8] Davis L., Gamble R.F., Payton J. *The Impact of Component Architectures on Interoperability*, The Journal of Systems and Software, 61(2002), pp. 31-45.

[9] Ferrara F.M. *The standard 'Healthcare Information Systems Architecture and DHE middleware*, International Journal of Medical Informatics, Volume 52(1998) , pp. 39-51

[10] Gamma E, Helm R, Johnson R and Vissides J *Design Patterns*, Addison-Wesley Professional; 1st edition (January 15, 1995)

[11] Goebel S., Nestler M. *Composite Component Support for EJB***,** Proc. of the Winter Int. Symp. on Information and Communication Technologies, Cancun, Mexico, 2004.

[12] Jagannathan W., *The Careflow Architecture, A Case Study in Medical Transcription*, in IEEE Internet Computing, May/June 2001, Volume pp. 59-63.

[13] Juric R., and J. Juric *Applying Component Based Modelling in the Process of Evaluation of Medicinal Products***,** Proceedings of the IDPT-2002 conference, June 2000, Pasadena, CA, US, Society for Design and Process Science Press, ISSN 1090 – 9389

[14] R. Juric, J. Kuljis, R. Paul, *A Software Architecture to Support Interoperability in Multiple Database Systems,* Proc. 22nd IASTED Int. Conf. on Software Engineering, Insbruck, Austria, Feb. 2004.

[15] R. Juric, J. Kuljis and R. Paul, *Contextualising Components when Addressing the DB Interoperability*, in Proc. of the IASTED – Int. Conf. on Software Engineering Applications, Boston, MA, Nov. 2004.

[16] R. Juric R. and LJ. Beus-Dukic, *COTS components and DB Interoperability*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 77-89.

[17] R Juric, S. Williams, L. Slevin, R. Shojanori S. Courtenage, P. Milligan., *Component Based Automation of Regulatory Affairs in the Pharmaceutical Industry*, submitted to the Journal of Healthcare Informatics, 2005

[18] Morisio M., Seaman C.B., Basili V.R., Parra A.T., Kraft S.E., Condon S.E. *COTS-based software Development: Processes and Open Issues*, The Journal of Systems and Software, 61(2002), pp. 189-199.

[19] M. Northcott and M. Vigder *Managing Dependencies between Software Products*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 201-211.

[20] Torchiano M, Jaccheri L, Sorensen C.F. Wang A I *COTS Product Characterization*, Proceedings of the SEKE '02 Conference, Ischia, Italy, pp 335-338

[21] Ulkuniemi P and Seppanen V. *COTS Component Acquisition in an Emerging Market,* In IEEE Software, Nov/Dec 2004, pp 76-82

[22] Vigder M. and Dean J. *An Architectural Approach to Building Systems from COTS Software Components*, 22nd SE Workshop, NASA/Goddard Space Flight Center SEL, Greenbelt, MD, December 1997, NRC Report Number 40221, pp.99-131.

[23] Szypersky C., *Component Software-Beyond Object Oriented Programming*, Addison Wesley, 2002

[24] Williams, S., R. Juric and P. Milligan *Design Patterns for Marketing Authorisation in Pharamceutical Industry*, to appear in the 26th Int. Conf. On Information technology Interfaces (ITI '05), University of Zagreb, June 2005, Croatia