A CTSC Request Management System: Design, Development and Deployment

Jacek Szymanski\textsuperscript{1}, Carolyn Apperson-Hansen\textsuperscript{2}, and Guo-Qiang Zhang\textsuperscript{3}

\textsuperscript{1}Case Center for Proteomics and Bioinformatics
\textsuperscript{2}Center for Clinical Investigation
\textsuperscript{3}Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio, USA

Abstract

This paper presents the design, development, and deployment of a novel request management system (RMS) for the Clinical & Translational Science Collaborative (CTSC). The RMS’s main goal is to provide a variety of domain expertise in areas such as statistical analysis, data management, and regulatory policy and procedures to clinical investigators who are unlikely to have them available. Consultation activities provided by domain experts are managed through the RMS in a role-based and structured way so that consultation areas and names of experts are visible and consultation services are easily reportable. Clinical translational science is a demanding area due to its inherent interdisciplinary nature. Our RMS is the first of its kind that streamlines the process of creating a catalog of expertise within a CTSA site and managing the requests and services provided domain experts in a scalable system with an intuitive user interface.

1 Introduction

The ultimate goal of the CTSC [6] in Cleveland is to provide complete service and integrated clinical translational research capability within the community that will improve the health of patients in Northeast Ohio through patient-based research.

The Research Concierge is the CTSC’s entry point and facilitates access to all CTSC services. Responsibilities of the Research Concierge include streamlining the process of obtaining regulatory advice and technological consultation, identifying institutional resources for research project support, assisting inexperienced research investigators in navigating complex processes, and identifying the potential use of CTSC cores to appropriately triage research projects. The Research Concierge also tracks requests and provides reports to CTSC leadership summarizing core and resource usage as well as utilization trends and requests for support not currently included in the CTSC.

The Request Management System (RMS) is designed to support the daily operations of the Research Concierge in such a way that maximizes operational efficiency through cooperative, decentralized management of services among clinical investigators and CTSC cores, with the Research Concierge in the loop in an administrative and mediating capacity.

2 Requirements Overview

The RMS should provide a single entry point for collaboration within the CTSC. An investigator will use the RMS to request assistance and receive consultation resources. The Research Concierge will use the system to triage requests to cores and their resource experts for assisting a clinical investigator’s research project planning and execution. The core administrators can further assign such requests to experts in respective domains who will then consult with investigators to identify project needs and develop a project plan. All such activities need to be tracked by the RMS. Even requests for support not currently available in the CTSC are to be tracked for future resource enhancements. Over time, the RMS will also serve as a CTSC-wide resource bank.

The RMS should also support program evaluation and reporting in a measurable way to assess whether our activities achieves explicit CTSA goals. The CTSA goals include: 1) promoting multidisciplinary clinical investigation; 2) demonstrating new clinical research activities in tangible ways in type, number, and scope that are distinguishable from existing multi-disciplinary clinical investigations presently underway.
within our CTSC.

The evaluation is designed to meet the highest professional standards of utility, feasibility, propriety, and accuracy as well as to adhere closely to the five principles of systematic inquiry, competence, integrity/honesty, respect for people, and responsibility for general and public welfare. Activities tracked by the RMS will help collect critical raw data to support the program evaluation.

3 Architecture

Figure 1 showcases the high-level architecture of the RMS. The system consists of a web-application server and a separate dedicated database server. The web-application server is virtually partitioned into the model, view, and controller components of the Model-View-Controller (MVC) paradigm [1], which promotes architectural organization. The web-application server also features a role-based access control security policy, which we discuss next.

3.1 Role-based Access Control

Role-based access control (RBAC) is the security policy of choice for the request management system. RBAC involves assigning one or more roles to a user, where a role may be viewed as an implicit classification to perform certain actions. We chose RBAC for its ease-of-use and the ability to decentralize access control management [3]. In a large-scale system such as the RMS, it is simply unfeasible to expect a single person to effectively and efficiently manage access control. RBAC partitions the daunting task of comprehensive access control management across multiple personnel, which decreases the burden on the Research Concierge. Since the conventions of RBAC are “human-friendly” with the use of checkboxes, assigning roles to users can happen with ease. The RMS uses four roles to delegate access control privileges.

System administrator – System administrators have complete, unrestricted access to the RMS’s functionalities. The Research Concierge has this role.

Core administrator – A core administrator manages all core-specific aspects such as users, consultation types, and requests.

Consultation expert – Consultation experts are assigned to carry out particular tasks.

Default investigator/user – Default investigators/users represent the base role in the RMS.

3.2 The RMS Data Model

The data model for the RMS is displayed in Figure 2. The name of each table in the database which corresponds to a model is represented by a text entry. Such an entry relates to other entries (i.e. models) through :has_many or :belongs_to attributes. The use of a :has_many attribute implies that a model can be associated with zero or more instances of some different model. Similarly, the presence of a :belongs_to attribute defines that a model is a child of another model.

In Figure 2, we use a solid arrow adjacent to three dots to denote a :has_many:belongs_to relationship between models, where the arrow’s direction points to the “child model.”

We chose the data model shown by Figure 2 for use in the RMS due to its ability to effectively act as a unified entry point into the CTSC for the investigator. The model’s Help Requests/Sub-requests/Consultations hierarchy allows users to request help via consultations from many cores simultaneously. This is possible since the consultations that users request are individually associated with corresponding core-specific Sub-request models, which in turn become the “children”
of one Help Request model. Figure 3 shows the interface for creating help requests. Consultation types are selected via checkboxes, and a description is entered in a free-form text box. Furthermore, the attachments checkbox denotes whether attachments will be uploaded upon request submission.

3.2.1 The Expertise Tree

The implementation of the RMS’s interface for issuing help requests features the presence of an “expertise tree” for each individual core. We define an “expertise tree” as an entity consisting of a combination of levels and selectable consultation types, where levels are analogous to tree branches, and consultation types represent core-specific services that can be instantiated into active consultations.

We refer to each core’s “expertise tree” as dynamic to illustrate its flexible modifiability. We assigned core administrators with the capacity to make changes to the tree through a core administration interface, which is shown in Figure 4. A core administrator may modify a core’s name by changing the contents of a text box and clicking the “Update” button. It is also possible to edit existing consultation types, create new consultation types or levels, and view relevant core personnel. The decision to implement a core administration interface was made in order to conform to a self-sufficient operational model, which allows a core to operate independently from other cores when dealing with core-specific data.

4 Development

The development process of the RMS led to the incorporation of relevant design features. We describe some of these design features in the following sections. In Section 4.1, we show how the RMS displays existing help requests using a role-based filtering mechanism and explain why this mechanism has been used. In Section 4.2, we elaborate on an aspect of the RMS’s user interface (asynchronous data requests) and provide support for why the RMS is a logical fit for its use. Then, in Section 4.3, we introduce the RMS’s primary operational environment (Ruby on Rails) and state the rationale behind selecting this particular environment for both the development and deployment stages.

4.1 Displaying Existing Help Requests

The RMS displays any relevant consultations via a list which is generated through a hybrid role-based access control mechanism. For example, we define the corresponding help requests for the system’s core administrator role as:

\[
\{ x \mid \exists y \in cs(x) \land (c.u).admin \in (c.(c.t(y))) \}
\]

In the previous definition, \( cs(x) \) refers to the set of consultations of help request \( x \), \( c.u \) is the current user, and \( (c.u).admin \in (c.(c.t(y))) \) is a boolean value which corresponds to whether the current user is a core administrator in the parent core of the consultation type affiliated with consultation \( y \). If the current user is a core administrator for multiple cores, we compute the union of the resulting sets.

The hybrid role-based access control mechanism has been put in place to centralize the display of help requests across multiple cores. Its cascading-union-based design also provides a scalable foundation that can accommodate the introduction of additional cores at a future date.

4.2 User Interface Design

The RMS’s user interface integrates the latest web-application technologies, which include asynchronous requests. Asynchronous requests can be defined as client-originating requests to a web-application server which execute independently and do not block the client from interacting with the server. Asynchronous requests are generally used to make changes to an existing client-side web page without the overhead of
entirely reloading it anew. The use of asynchronous requests is commonly referred to as simply “AJAX,” which is an acronym for Asynchronous Javascript and Extensible Markup Language [2]. We decided to use AJAX as a means for many client-side interactions because the RMS data model is rich in :has_many relationships (please reference Section 3.2) which can be extended. For example, having the Research Concierge triage a help request is a prime candidate for asynchronous request use. Figure 5 shows the user interface for accomplishing this task, where the “Send to core” button is used to issue an AJAX request that selects a consultation, and the “Update” button commits the consultation to the database.

The Research Concierge may update a help request’s description and view, edit, or delete core-specific sub-requests.

### 4.3 Ruby on Rails

We identified the Ruby on Rails [5] framework as the development environment for the RMS. Ruby on Rails satisfied two important criteria which were instrumental in its adoption: Promoting high developer productivity and having first-hand experience with the development environment. High productivity via Ruby on Rails results from a combination of factors. Coding is human-friendly, elegant, and concise due to the use of the Ruby Programming Language. Ruby on Rails also features built-in support for the MVC paradigm and efficient, database independent read/write wrappers through the inclusion of Active Record Classes [4]. Our extensive hands-on experience with Ruby on Rails in developing previous web-applications serves as evidence which further solidifies its selection [7].

### 5 Results/Deployment

The RMS has been deployed since February 2009 for internal core use. Each core’s administrative personnel may work with the RMS, however, external users such as investigators will obtain access to the system at a later date. We decided to adhere to this “tiered deployment model” to allow the RMS to be as refined as possible before being fully released to an unrestricted user base.

**Platforms** – The deployment environment consists of a Dell rack-mount web-application server which runs Microsoft Window 2003 and Ruby on Rails 2.1.1. The deployment server interfaces with clients by using Microsoft’s Internet Information Server, which supports full SSL data encryption via a trusted certificate. The deployment environment is replicated by two instances on a virtual web-application server for training and development purposes. We include a training environment since it acts as a “playground” for users to become acclimated with the RMS without affecting any data on the deployment server. The development instance facilitates adding new features to the RMS because updates can be safely evaluated before being transferred to the deployment and training environments. All three environments integrate the same database server, which runs Microsoft SQL Server and is accessed securely by the web-application servers. Each environment is distinguished uniquely by being associated with a separate database instance. Maintenance involves automatically backing up the contents...
of the web-application and database servers to external media on a nightly basis.

6 Conclusion and Future Work

The RMS has been designed to serve as the primary management portal for all help requests through the CTSC. The system’s advanced architecture promotes a self-sufficient operational model across cores and is highly scalable due to decentralized data management and a role-based access control security policy. Its conformance to strict data encryption standards provides the foundation for an environment featuring patient data exchange. We also feel that the RMS’s user interface has matured to the point where it enriches the user’s experience and that any future development will be accomplished efficiently and elegantly via the Ruby on Rails environment.

To further improve the RMS, we plan on obtaining feedback from two sources.

Investigators – Once the initial deployment for internal core use has been completed, we will release the RMS to the KL2 scholars (a subset of the investigator population) and consider their feedback before performing a full-scale deployment to the entire investigator population.

Domain experts – After the system has been fully deployed for a certain period of time, we will analyze its help request pool through domain expert feedback to ascertain any patterns that would help in refining the structure of the RMS’s “expertise trees.”

Acknowledgment. The project described was supported by NIH grants UL1RR024989. Its contents are solely the responsibility of the authors.

References


