A HOSPITAL PHARMACY PATIENT INFORMATION SYSTEM

ROBERT FRANKEN

Part 1. Abstract

Hospital pharmacy operations primarily consist of the logistics of supply, budgeting and the provision of clinical services to wards, staff and patients. The focus of this project will be on part of these operations, namely ward pharmacy supply and clinical services, otherwise referred to as medicines management. In most hospitals, all records of processes carried out on wards by pharmacy staff are handwritten. The procedures followed are fairly robust, but slow and susceptible to errors of various kinds, with the information gathered not easily available to other staff members.

Part 2. Introduction

Part 3. Design

This section describes the requirements of the system and how the requirements were used to design the system. By following an object oriented design approach with each real world entity being represented as a class, the design of the system may closely reflect the structure of the real world objects and the information expert gang of four pattern.

1. Rails and MVC

Except for small, simple web application projects, the MVC (Model View Controller) design approach should be used when developing web applications.[2] The MVC model is used in place of a 2-tier, client/server model. In the 2-tier model a majority or all of the business logic processing occurs on the client with the server providing the database. This structure ties views and logic together on remote clients. Proposed in 1979 by Trygve Reenskaug when designing GUI systems at Xerox and widely used in OOP frameworks including that of the iPhone[1]. The MVC approach describes the 'separation of concerns' within a system. Related to the three-tier, or n-tier model, it breaks a system into several independent encapsulated modules[7].

The MVC approach makes a system more modular and thereby more robust and amenable to maintenance and change.[4]

The Model represents the problem domain of the system - the data and elements of the business logic (data validation for example)[5]. It is independent from the other parts of the system, using generic data formats for communication. Models may

be considered observers of controller. In the Rails approach, the model consists of the class representing an object and the database record holding its attributes.
The Controller contains elements of the business logic of the system. The Controller is responsible for processing requests, forming requests to make to the Model and choosing which View mechanism to call. The Controller should be able to function with failure of the Model and View components[5].
In Rails, the controller methods for a given model define the variables and direct which view is to be rendered. The controller is where the view will be set to either iPhone or standard web interface is set. There may be multiple controllers for a single model.

The View contains all presentation information frequently being implemented with a template language. In most systems there are several Views available making it possible for the Control to choose the appropriate View for a given situation. Views may be considered observers of controllers and models.
The view consists of the layout to be applied, either that for iPhone or standard, and the HTML passed to the layout.
The encapsulation of elements of a system in this way provides several benefits over the 2-tier alternative. By executing much of the processing of the data on a server before transmission, network activity is reduced. Executing business logic on the server also allows for caching and reuse of results by multiple clients leading to greater overall efficiency. Offloading processing from clients is of particular importance where mobile or lower power clients will be accessing the system. The primary advantage of the MVC architecture is the benefit to design, development and maintenance that encapsulation provides. By allowing for views to be managed separately from logic, different views for different clients may use results from the same logic, and by keeping related code together the points of interaction are kept to a minimum and as a result changes in each component are possible without affecting the others.[6]
Within Rails applications there are further levels of encapsulation. The classes representing each object in the model layer interact with the database via a compatibility layer provided by Rails called ActiveRecord. ActiveRecord provides methods for database data manipulation and is database independent. This allows for business logic code to function if the underlaying database system is changed, for example from MySQL to PostgreSQL. Another example of further encapsulation in Rails applications is on the view side, with html views referencing calling partial views and then being passed to a layout, which calls CSS (cascading style sheets) and JavaScript libraries.

2. Rails and REST

Analogous to OOP and first proposed by one of the principle authors of the HTTP specification, Roy Fielding[3], REST(representational state transfer) is the use of URLs to represent unique conceptual resources that are available to users in hypermedia systems. An example of a REST resource may be `/patients/1` or `/discharges` - the former will always point to the patient with an id of 1, and that patient will never change, and the second will always point to patients who are being discharged. The base routes follow the `:controller/:action/:id` format. The path `/patients/edit/1` passes the `:id =>1` to the edit action of the patients

controller. The representation of a resource may change but its underlying concept doesn't.
For each view required for a particular resource, for example for Patient, there is an associated model and action in the model's controller. The system will define a views to index, new, show and edit patients and the patients controller will have an action for each of those, and there will be a separate ruby html file for each view.
The design of the system will follow the REST approach, focusing on conceptual resources.

3. Context and resources

The focus of this system is on the requirements of ward pharmacy staff. Ward pharmacy staff work with information concerning wards, patients, doctors, prescriptions, products, orders, lab results, stores, and local policies. These represent the REST resources that will guide the design. The details of the these entities and how the information concerning them is detailed in this section.

3.1. **Ward pharmacy staff.** Each member of the ward pharmacy staff is either a pharmacist or ward technician. Pharmacists and technicians have different roles, but the line between them continue to blur in the duties they execute. The extent of this blurring depends on the site. On traditional sites, technicians are only concerned with supply, whereas at increasingly more numerous sites pharmacy technicians carry out preliminary clinical checks and take drug histories from patients in addition to their supply functions. While their roles are essentially identical in this system, a mechanism for changing access to functionality in the site will be made available, partially by making pharmacists and technicians different types of user.

3.2. **Ward.** Wards are where patients stay for the majority period of each admission. Each ward has a team of nurses and one or more regular medical teams associated with it. All wards are assigned a patient group based on their current care requirements. The following properties are relevant ward pharmacy staff:

- Name - used to identify the ward
- Type - most wards care for patients admitted for specific problems, such as respiratory disease.
- Pharmacist - a ward will have a single pharmacist who is responsible for pharmacy services to that ward
- Ward notes - there is a variety of information about a ward and its workings a user may want to store to help other staff members who are charged with responsibility for the ward. An example may be the times that are best to visit the ward, which beds hold the most critical patients, or who to speak to about a specific recurring issue.
- Beds - a ward has several beds which may or may not be occupied.
- Orders - wards have their own small stocks of medications that are supplied by the pharmacy department. Ward staff assist in managing stock levels.

Ward pharmacy staff need to be able to list, edit and update ward orders, ward patients and ward notes.

3.3. **Patient.** : The patients recorded in this system will be the focus of the system. Most ward pharmacy functions centre on individual patients. When working from a ward, pharmacy staff need to record and screen current medications, medication history, allergies, primary indications or diagnoses, supply levels, lab results and a brief medical history for each patient they see.

3.3.1. *Properties.*

- statutory details, including hospital identification
- current and previous prescriptions - and by extension, current pharmaceutical agents
- current and previous admissions
- allergies
- diagnoses and selected lab results
- current and previous results of blood tests and values of other medical variables used to make clinical judgements.
- notes other ward pharmacy staff have made about the patient
- probable discharge date

Ward pharmacy staff need to view, list, update and add to these properties.

3.4. **Product.** A product may be an agent(drug) or combinations of agents, or other medical products such as dressings, delivery aids such as pill boxes or cutters. A product has a form, a strength and a variety of doses and frequencies that may be prescribed for it.

3.4.1. *Product properties.*

- brand - the brand name of the product.
- agents - combination products have multiple agents in them. Pharmacists are usually more concerned with prescribed agents than what a specific brand is. Only occasionally different brands have differences significant enough for the brand to be important. Each agent belongs to a pharmaceutical class of drugs. Much like classes in other areas of science, agents in the same pharmaceutical class share common characteristics. For example, a patient may be allergic to beta-lactam antibiotics, of which one is penicillin. Classes also share common side effects and clinical indications, and the prescribing of multiple agents of the same class may be a prescribing oversight. As such, being able to query the classes of product agents would be useful.
- form - a product may be have one of many forms, including several types of solid oral, liquid oral and topical preparations. The dose of each form depends on this type - a liquid is dosed in volume, and a solid oral form is dosed in discrete units such as capsules. To this end, a form has both a form type and a name to allow for calculations to be applied to them.
- doses - how much of a product a patient is treated with is described by several variables. These are the strength of each agent, e.g. 20 milligrams per unit, the number of product units, e.g. two tablets, and the frequency, e.g. twice a day or every eight hours. For a given strength, each product has a collection of valid quantity/frequency pairs associated with it. A period of treatment must also be recorded.

- policies - a significant aspect of the potential benefit of recording information digitally is the prospect for some level of automation. Part of the challenge of updating policies in a large organisation is disseminating them and encouraging compliance. To this end, each product may have several policies. Each policy is essentially text, intended to describe special cases regarding the use of the product. These usually include conditions, such as 'do not use when taking aspirin'. To assist in automation of some of the checks that are carried out by pharmacy staff, each policy has a number of checks associated with it. Checks are entered by the user as a type of conditional statement. Executing a policy's checks should result in some output indicating their results, either that all is fine or that something requires attention.
- stores - each product is associated with one or more stores. There are ward stores and a main pharmacy store. Having access to current levels of a product in the pharmacy store while on a ward assists in early warning of potential supply problems. A common problem is occurs when ordering prescribed items from the pharmacy store for a patient who is soon to leave. If a product is in stock then all goes smoothly. However, if levels are low or supply is entirely exhausted, this will not become apparent until it is too late to arrange additional supplies from external suppliers. If when making an order the current level may be checked the orderer may immediately take action to prevent a break in supply continuity.

Ward pharmacy staff need search products, view the details of products and prescribe products. A product must be able to be compared with the patient to whom it is prescribed in order to identify potential problems. Other details include stock level on the ward and in the pharmacy store.

3.5. **Prescription.** The recording of a prescription is this system will not be made by its prescriber but by ward pharmacy staff. A prescription has the following properties or associations:

- product - each prescription is for a single product object.
- dose - may be chosen from the available doses for a specific product.
- patient - a one-to-many relationship.
- prescriber - a one-to-many relationship, one doctor may have many prescriptions.
- treatment period - a start and end date is recorded, as is necessary with all prescriptions in hospitals, even if it is to be repeated indefinitely.
- ancillary instructions - text that gives additional dosing instructions beyond dosing instructions.
- orders - for a given prescription there are usually multiple supplies in a hospital setting, often as the period of treatment may continue for some time.
- amount supplied - by recording the amount supplied following the completion of an order on the prescription, it will be possible to check where supplies are running low. The quantities are recorded with two fields, packs and units. The total may be inferred from this information, as the pack size will be recorded. It is often useful to distinguish between a supply of one pack of 28 tablets and a supply of 16 tablets and 14 tablets.

Users will be able to create, update and view prescriptions for a specific patient, and create new orders for supply of the prescription.

3.6. **Prescription order.** Each prescription has multiple supplies made for it. In existing systems, orders consist of orders for multiple prescriptions, or prescription order lines, each of which has the product and prescription details, but also instructions to the dispensary, such as priority for supply and whether or not instructions are required.

- Packs - indicates how many full packs of the product are to be supplied.
- Units - indicates how many individual units are requested.
- IP - a boolean indicating if the patient is to have 'inpatient' labelling. Inpatient labelling does not include instructions on the label, as the dose may be changed multiple times during admission which would require a change to the label. As dosing instructions are specified on the patient's drug chart, potentially conflicting instructions are not included on the label.
- Priority - this is a boolean indicating to the dispensary staff if the supply needs to be completed within 24 hours or the standard 3 days is sufficient.
- Process delivered orders - a function to add delivered orders to the recorded supply level for the associated prescription. In this way patient supply levels will be updated. The function will also reflect this in the order status.

3.7. **Team.** A team, or more specifically a medical team, is a group of doctors. Each team typically has two interns, a registrar and a specialist.

- Ward - the activity of a team typically centres on a single ward, though commonly cares for patients in more than one ward.
- Patients - teams naturally have a collection of patients for whom they are responsible.

3.8. **Doctor.**

- Name
- position - a doctor may have one of three positions, intern, registrar or specialist. Within those positions are grades, but for this system, those three roles are all that is relevant to pharmacy staff.
- page number - the need to communicate with prescribers means that page numbers are required.
- interns, registrar and specialist - functions to return doctors with specific roles.
- specialty - a team is defined by its leader, the doctor with the 'specialist' role. So, a team's speciality will be found by querying its specialist's specialty.

3.9. **Store.** A store is associated with many products and with a single ward or with the pharmacy department. For the purposes of this system, pharmacy stores are identical to ward stores, except the value in the ward foreign key will be nil. It would be possible to use a polymorphic relationship, similar to that used for policies, but this would introduce complexity without utility. Each store has a current store level for each product and an order for each product. This means that connecting models are required that are more complex than those for the other many-to-many relationships in this system.

- shelf - conceptually, each shelf is in a specific store, holds a specific product and has a current level. In this system, conceptual shelves are represented by records in product_stores. Stores and products have a many-to-many relationship via product_stores, each product_stores record contains its foreign keys for both store and product, and a stock level for that product.
- store orders - each store has multiple store orders.

3.10. **Store order.** A store order is an order to the pharmacy for a top up of the ward's stock.

- store - each store order is associated with a single store.
- store order line - product_stores share a many-to-many relationship with store_orders through store_order_lines.
- has many store order lines, each of which belongs to a specific product in a specific store, a product-store line.

3.11. **Level.** Most patients when admitted to hospital will have multiple tests carried out during their admission. Most of these will be from blood samples, and a majority of those will be repeated for every sample received. This results in something similar to a set of collections of the tests for each patient, each collection having a time of collection or measurement. A 'level' in the system represents a collection of results of tests for a given time. Each patient has a collection of levels.

3.12. **Policy.** As described earlier, a policy reflects some of the checks carried out by pharmacy staff. Most of the checks pharmacy staff carry out are standard throughout the profession, but there are local policies in each hospital that are very specific and not obvious. An example of a local policy may be a product that may only be prescribed by members of a specific medical team, or a brand of drug that is peculiar to a specific indication.

- Content - this is plain text that is intended to be used to describe the policy.
- Checks - a policy may have many checks associated with it. These checks are conditional statements that may be executed for a patient.

3.13. **Check.** A check is a collection of conditional statement elements which are added to a policy by a user used to generate executable checks against a given patient. What is entered by the user is not code, but something that may be used to create to necessary code for the checks.

3.14. **Admission.** An admission is the period between an admission and discharge of a patient, as well as the reason (indication) the patient has been admitted. Usually a single medical team is responsible for the treatment of a patient at any one time. During an admission, a patient may be transferred between teams, but the final team for an admission may be considered the primary team. Having access to some basic information regarding previous admissions easily assists in finding pertinent information concerning the patient quickly for future admissions. Properties of an admission:

- date of admission
- ward admitted to

- discharge (departure) date - this is useful even for current admissions. An expected departure date assists in discharge planning and ensuring sufficient supplies of medications are supplied in a timely fashion.
- primary indication (reason for admission)
- summary of clinical information regarding that admission
- a function to find all patients being discharged in range of days.

4. User interface

The following functions need to be part of the system:

- view all wards -a list of all wards registered on the system with ability to navigate to a more detailed view of the system
- view details of a ward
- a list of the patients on a specific ward
- the details of each patient on the ward
- the admissions associated with each patient
- currently and previously prescribed products for each patient
- orders for supply of a prescription
- orders for a ward supply
- details of a product
- ward medical team
- search for patients or products

Other operations: The main function of a ward visit by a pharmacist is checking each patient's medical condition and treatments. Checks are made in reference to professional knowledge, and to local policies, such as availability of a given brand of a medication.

- check prescribed treatments against medical condition
- check prescribed treatments against one another
- check current stock levels in ward stores
- check patient medication supplies
- order medications when supplies reach a given level

4.1. **Dispensary.** Dispensary users need to interact with the ward and patients concerning supply issues. As such, dispensary pharmacy staff need to be able to view following:

- patient discharge date
- patient current prescriptions
- patient supply orders
- store orders
- ward details

4.2. **Administrator.** An administrator will need to administer users, wards and products. To this end, user with administrator privileges will need to perform CRUD operations of the following:

- products - adding new products and updating or removing existing products.
- wards - adding new wards.
- users - creating new users and updating existing ones.
- policies and checks.

Part 4. **Implementation**

The structure of rails applications closely mirror that of the underlying database which allows for straight forward translation of classes and instances to and from the database by ActiveRecord, and mapping of classes to their CRUD operations. This structure The Rails framework convention provides CRUD methods for objects and some associations when objects and their associations are created. For example, when creating a new model with the Rails model generation script, a standard Rails class is represented in the database by a single table witch is named the plural of the class name. In the case of the Patient class, the associated table is called 'patients'. The existing class objects' attributes are represented by a row in their class' table. With this standard structure, methods for finding all patients, editing patients, creating new patients and deleting patients are provided, along with views for these actions. The views provide basic forms with which to update. These provide the basis for extension to show and manipulate the data of instances of multiple associated models, and providing tailored views.

5. Ruby on Rails features

5.1. **ActiveRecord.**

5.2. **External libraries.**

5.3. **Generator scripts.**

5.4. **Routing and REST.** A central tenet of Rails applications is the that they should be 'RESTful'. REST organisation is enforced by the scripted scaffolding and default routing. One of the ways in which Rails leverages a REST architecture is in routes. The routes file enables the nomination of controllers as REST resources with associated paths. An example of this is the route `/discharges`. The route is defined in the routes.rb file with `map.resources :discharges` which references the `discharges_controller.rb` controller. In the controller, the only action defined is `index` where the collection of patients to be indexed is defined, in this case with a function `leaving(1)`. Routes also assist in defining nested associations, such as `map.resources :products, :has_many => :agents`. Assuming the correct associations are defined in the relevant models for Product and Agent, and there is a connecting model in the case of a many-to-many relationship, this route maps the `product/:product_id/agents` providing a meaningful path.

6. Polymorphism and inheritance

Most of the polymorphism employed is part of the rails scaffolding with the rails base classes. However, the store object could be a pharmacy store or a ward store and they needed to be differentiated. Rather than have a separate attribute for each kind of association, the convention in rails at the database layer is to use two attributes to identify what kind of store it is, a single foreign key attribute that stores the primary key of all off the associations and a type attribute to indicate the class of the association. Both are later used in the model layer to define associations for ActiveRecord.

7. Interface

Multiple layouts are required to cater for system administration from a desktop or standard screen and for mobile users. The system chosen for defining the view uses the user agent reported by the client in the application controller and sets the format for the respond_to methods in the various controllers. If it is a mobile safari client the iPhone views are rendered with the iPhone layout.

7.1. **Standard web interface.**

7.2. **iPhone interface.**

8. Classes and methods

8.1. **Users and session.** There are three types of sessions supported. The primary user type is a standard user, usually a pharmacist or technician, who can create prescriptions and notes, add patients to wards and edit their own notes. There needs to be an system administrator role, who can edit users and add policies, and an audit role enabling only viewing of the information recorded on the system, and some additional views for information such as all notes made by a particular user. Rails and the rails plug-in AuthLogic create a session cookie (user_credentials) upon user login. By this mechanism the current session and user is persisted. Data associated with a session may be stored in the session hash but will be lost if the user logs out, closes the browser or clears their cookies. As a result, storing data objects in a session is not advisable. Doing so would also lead to stale data in the object or the database.
As the system will have a limited number of users and roles, a simply access restriction pattern is appropriate. In this case a boolean property, admin, is employed. The model controllers check the property on instantiation so that both the controller and associated views have access to the admin attribute of the user. In combination with a check of the current_user and user associated with records, editing of notes and other records are restricted.

8.2. **Check.** In order to interpret the checks entered by a user a three hashes are used to map the user choices, which are made available by a menu, to actual method calls. The structure of implementation means that in order to add additional potential conditional tests all that is required is to enter the appropriate values in the hash. The matching values are method calls that are called on the patient object, for example:

```
@@type_hash = { "categories" => ["current_agents_categories_names"],
"agents" => ["current_agents_names"], "level" => ["levels", "last"],
"specialist" => ["team", "specialist", "first"],
"specialty" => ["team", "specialty"] }

@@check_hash = { "bp" => "dbp", "k" => "potassium", "surname" =>
"surname"}

@@operator_hash = { ">" => ">", "<" => "<" , "=" => "==", "has" =>
"include?" }
```

```
@patient.send (@@type_hash["level"].send(@@check_hash["bp"]
.send(@@operator_hash[">"], value)
```

would result in, where value is 90:

```
@patient.levels.last.dbp.>(90) #=> boolean
```

The actual code is slightly more complex, but this example illustrates the mechanics. The methods called in the result are all generated when the models are created. This arrangement allows for checks that have not been defined in the code itself. Allowing the user to execute code directly is potentially dangerous, but using a hash to define the conditions prevents access to methods other than those deemed appropriate.

8.2.1. *Attributes.*

- check_type - essentially the first function to call on the patient instance.
- check - second function, may be nil as some potential checks may only require a single method call, for example `@patient.surname == 'Smith'` only needs `check_type`
- operator - indicates the final method, which is passed the value with the ruby method `send(String method, value)`.
- value - a string, integer or float that is passed to the 'operator'.

8.2.2. *Methods.*

- perform_check(patient) - passes check attributes to private function (arrayit) which returns an array of function calls, which is then passed to a second private function (process) that recursively calls each element of the array, calling the last with the value to be passed. The result, which is returned, is returned as a hash containing the check itself and the boolean result of the check.
- description - returns a string representing the check.

8.3. **Patient.**

8.3.1. *Patient methods.*

- current_products - returns the products the patient is currently prescribed.
- current_agents - each product may consist of more than one agent, and the agents are what clinical staff are most concerned with.
- current_agents_names - returns a collection of strings representing current agents. Allows for simpler listing of current agents.
- current_agents_categories_names - returns a collection of strings representing the categories, or pharmaceutical classes, of the agents that the patient is currently prescribed. This information is useful in performing checks on the current treatment of the patient, as some pharmaceutical classes are contraindicated (should not be prescribed) in some situations.
- current_level - returns the last set of results for a patient. Most are lab results, but may be others, such as diastolic blood pressure, weight or water input/output balance.

- has_currentnotes? - returns boolean indicating whether the current admission for this patient has notes associated with it.
- inpatient? - returns boolean indicating whether the patient is currently admitted or not.
- current_ward - if currently admitted returns an instance of ward representing ward the patient is currently admitted to, otherwise returns nil.
- fullname - returns a string representing the full name of the patient. Used in views.
- check_allergies - returns a set of strings resulting from the intersection of a set of agents representing the patient's allergies and a set of currently prescribed agents.
- team - returns instance of team that represents the medical team currently responsible for the patient's care.
- policy_results - returns an array of hashes, each hash containing a product and an array of check results.
- failed_policies - returns an array of hashes, each hash containing a product and an array of the checks that failed to pass.
- print_failed - prints a description of failed checks
- CRUD methods

8.4. **Level.** Each individual level type, such as potassium or creatinine (kidney function), is represented as a column in the levels table.

8.4.1. *Table description.* `Level(id: integer, potassium: float, creatinine: float, inr: float, ppt: float, albumin: float, created_at: datetime, updated_at: datetime, patient_id: integer, dbp: integer, collected: datetime)`

Part 5. Improvements

9. Testing

Given the intended realm of use for the system, issues of security speed and adherence to the multitudinous local and national policies, guidelines, regulations and laws specifying the appropriate handling of personal medical information would need careful consideration. As it stands, the security of the system would be relatively assured if the network used for access to the patient data were provided over encrypted wireless networks and local policies prescribed sensible behaviours for its users.

10. Authentication

11. Integration

Data duplication is a already existing problem in most hospital data, so a new system such as this would be of little use unless it was integrated into existing systems. As most are almost entirely proprietary integration would likely require cooperation from the suppliers of existing systems and present some technical challenge.

12. Interface

It is likely that a variety of mobile devices capable of using the system would be available. The design allows for multiple views and layouts. Its possible that simply rendering existing views with a different layout may suffice in these cases, in which case all that would be needed would be an additional layout file.

13. Refactoring

13.1. **Third party libraries.** A third party library for rendering elements of the iPhone interface was employed. While capable, it has not been maintained and suffers performance problems. It is sufficient for prototype development, but if the system were to be adopted the iPhone interface styling and associated javascript would probably need to be improved.

14. Constraints and validation

The system is to be used by its owners, and for the sake of flexibility there are few constraints employed as to the nature of the data that may be entered.

14.1. **Checking product policies during prescribing process.** One of the more significant potential uses of a system such as this would be the ability for local policies to be implemented with some level of automation. Given the MVC architecture and model associations employed the system may be customized to allow for checking against policies on prescription entry. An example may be if a product is prescribed that has an alternative preferred brand a check against local policies would generate a message to prompt the pharmacist reviewing the patient to query the prescription. Policies may also be associated with specialties or medical teams, as is often the case in hospitals. A product may not be available, except for a specific indication under a specific team/specialty/ward.

15. Native application

There are drawbacks in implementing the system as a web application. While it increases the speed of development and improves accessibility to a variety of platforms, the browser is not as powerful or as fast as a local application. It is likely that the web application implementation would be used as a prototype of the system and interface, and once tested and fine tuned, would act as a template for the implementation of a native application for the relevant platforms

16. Separate system administration interface

As it stands the system's administration is part of the standard interface, which is unnecessary. A number of creation and update functions are not relevant in the day to day workings of a pharmacist on a ward. For example, adding a new product or user administration would be best served by a different view, as would auditing functions, such as viewing all notes by all pharmacists.

17. Performance

Some of the information held in separate relations, such as in the status relation, could have been held as hashes instead. However, doing so would have required changing the code to add new status types, whereas with a relation an administrator for the system can maintain status types.

18. Logic

The system is a structure into logic may be inserted, thanks largely to the MVC nature of rails applications. Though some logic has been implemented, the specifics rely largely on the workings of a specific site. Some sites would probably like the restrict dosing for a specific product to approved doses, while others may find that approach overly limiting. The benefit of the restriction is that simply by not finding an odd dose would prompt the user to query the necessity for a non-standard regimen. Due to the structure of the application, adding new logic is relatively straight forward - new methods can be defined in the model or controller and ActiveRecord.

References

[1] Christopher Allen and Shannon Appelcline. *iPhone in Action: Introduction to Web and SDK Development*. Manning Publications, December 2008.
[2] Martin Bond and Debbie Law. *Tomcat Kick Start*. Sams, November 2002.
[3] Trotter Cashion. *Rails Refactoring to Resources: Using CRUD and REST in Your Rails Application*. Addison-Wesley Professional, April 2007.
[4] Mark Ramm, Kevin Dangoor, and Gigi Sayfan. *Rapid Web Applications with TurboGears: Using Python to Create Ajax-Powered Sites*. Prentice Hall, November 2006.
[5] Juan R. Rodriguez, Jerome Curlier, Werner Frei, Denise Hatzidakis, Juergen Moetzel, Nancy Ting, and Shawn VanRaay. *BM WebSphere Portal V4 Developer's Handbook*. IBM Redbooks, March 2003.
[6] Jesse Skinner. *Unobtrusive Ajax*. O'Reilly Media, Inc., July 2007.
[7] Kevin Skoglund. Ruby on rails essential training. Instructional video, January 2007. Available from: `Lynda.com`.