Development of a Decision Support System for Facility Maintenance and Management Technical Report

Article

4 authors, including:

Yingzhao Xue
Azienda Sanitaria Locale Torino 1
148 PUBLICATIONS **4,411** CITATIONS
SEE PROFILE

Jie Zhu
15 PUBLICATIONS **202** CITATIONS
SEE PROFILE

NRC·CNRC Institute for Research in Construction
CNRC·NRC Institut de recherche en construction

National Research Council of Canada

Report RR#286

Development of a Decision Support System for Facility Maintenance and Management: Technical Report

Qi Hao, Yunjiao Xue, Brian Jones, Jie Zhu

August 2009

National Research Council Canada | Conseil national de recherches Canada
Canada

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

One of the major activities of IRC-CCCT's strategic project "Decision Support Tools for Critical Facilities Maintenance Management" is to develop a demonstration decision support system for facility maintenance management (FMM Demo for short) which aims to achieve the following objectives:

- Develop basic FMM operation modules (such as Preventive Maintenance) and as a result, establish a functional R&D foundation/test bed for future FMM-related research, development and demonstration.
- Implement some advanced features that can distinguish it from other existing commercial or R&D-based FMM solutions.
- Build and learn from a case study from the industry; reproduce the case study using the FMM Demo system.
- Improve the efficiency to collect requirements/needs from the industry which in turn can drive the evolution of this research and the demonstration system.
- Accelerate recognition of CCCT's capabilities in construction process integration to the industry.

The first prototype of the FMM Demo system was finished within the past four months, from May to August of 2009, based on a software system we developed for the DND [1]. This report will document the developed system in its requirements, system architecture, functional design, technical details, and a special algorithm created for advanced FMM workload optimization. Regarding the industrial case study, NRC-ASPM facility maintenance practices and processes were studied and the case was re-built in the developed FMM Demo system. Details about the ASPM case study is reported in a separate IRC research report [2].

2. Requirements

In the operations and maintenance domain, assets are physical objects that need to be maintained. Assets can be organized as a hierarchy. Some assets are regular physical maintainable assets; some are virtual assets – functions, locations or areas that help build the asset tree. These virtual assets are generally referred to in other CMMS systems as “Technical Systems” or “Functional Locations” to represent grouping information.

Facility maintenance activities can be classified into three major categories: Corrective Maintenance (CM), Preventive Maintenance (PM), and Condition-Based Maintenance (CBM). Corresponding to these different sources, three types of maintenance orders are created, scheduled and issued:

1) Corrective maintenance orders that are manually entered directly by FM personnel;

2) Preventive maintenance orders that are automatically generated by the PM planning process;

3) Condition-based maintenance orders that are raised from condition monitoring and assessment.

The first prototype of the FMM system will include functionalities of CM and PM. CBM functions, though not implemented, have already been considered in the system design stage for future extensibility.

Preventive maintenance work can be pre-defined for assets at different levels. However, only the assets that are “maintainable” can have PM settings. Multiple PM settings can be applied to a given asset. The system should be able to generate “preventive maintenance orders (PMOs)” for a given time period according to:

- Plan information of past maintenance work (“finished” or “pending” work);

  AND

- A pre-defined preventive maintenance interval (frequency);

  OR

- A default time (or date) for establishing the first maintenance cycle.

Except from regularly routine maintenance orders, major maintenance work is required to be managed separately on a project-by-project basis; for example, major repairs, renovations, and building expansions. This requires the coexistence of major facility maintenance projects along with routine maintenance orders. In order to re-utilize the project management and dynamic scheduling modules we developed for

the DND project, all types of maintenance orders were ultimately converted and linked to individual projects at the back end.

When projects for routine maintenance orders and maintenance projects are created, the system automatically generates the first feasible schedule for each of the projects. But the schedule of these projects may have conflicting demands on the shared resources. Therefore, the system needs to check conflicts, coordinate schedules, perform workload analysis and perform optimization based on priority, cost, condition and locations. Graphical interfaces for showing project/task schedules and resource workload are required to present the calculated results before and after the optimization to assist in the decision-making process.

3. System Architecture

The functions that were developed in this research (shown in Figure 1) are based on the requirements summarized in Chapter 2. Figure 1 shows an overview of system functions. Among them, Actual Maintenance Record function is developed as a group of Web-based interfaces for iPhone's Safari® browser. This shows the system's potential to support mobile devices.

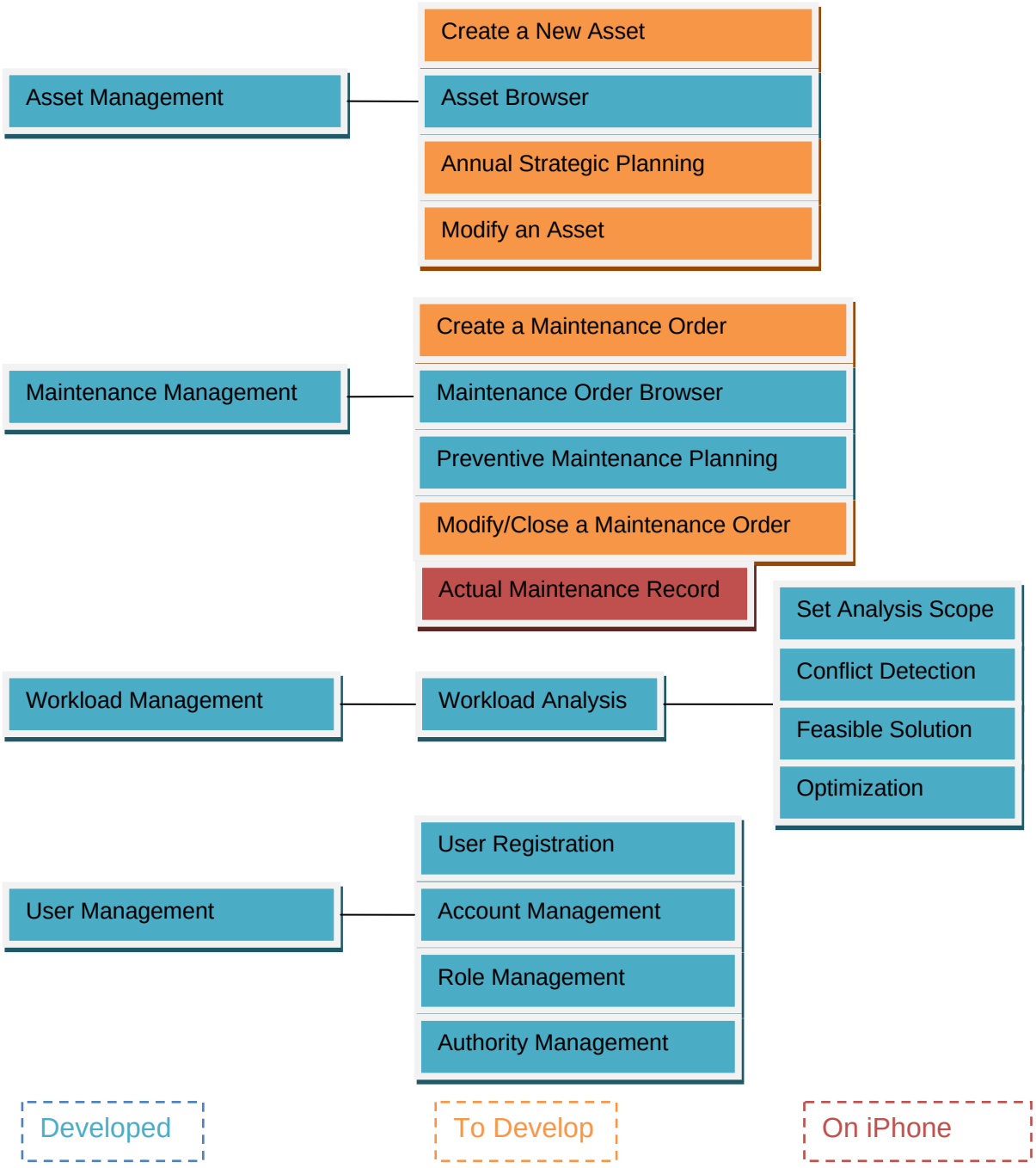


Figure 1: System functions

The FMM Demo system was built as a Web-based application using the popular MVC architectural pattern. The following sections will identify why the MVC design pattern was chosen with regards to basic MVC concepts, principles and benefits.

3.1 Traditional Web application

If information is an important aspect of the application, a traditional Web application is composed of a set of dynamic Web pages and a database server. Each dynamic Web page may contain both the HTML code for page presentation and the script code for implementing business logic. In this kind of architecture, web pages require hardcoded SQL statements for handling database connections and operations. Figure 2 shows the architecture of a traditional Web application using Java technology.

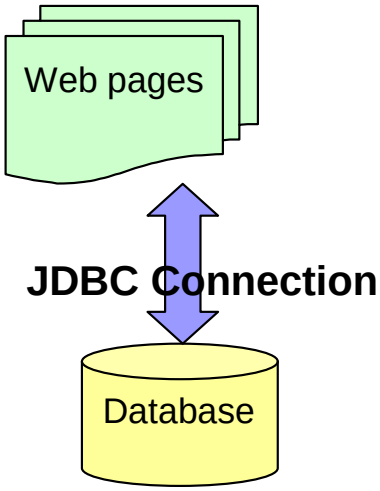


Figure 2: Traditional Web application architecture

There are several drawbacks with the traditional Web architecture, as listed below:

1) The business logic is mixed with HTML code, which is not a good programming practice.

2) The transition of pages is programmed in the pages.

3) Manually handling JDBC connections and database operations will reduce efficiency and increase the possibility of introducing errors.

4) The business logic is developed based on a business model, which is usually an Object model. Directly dealing with SQL for database operations defies the concept of 'Object-Oriented Programming' adopted by most modern programming languages, since the widely-used relational databases are using a different entity-relationship model.

3.2 MVC architecture

Model–view–controller (MVC) architecture is used to isolate business logic (or program logic) from the user interface completely, permitting one to be freely modified without affecting the other. It is a promising paradigm to overcome the weaknesses of the traditional Web application design pattern.

With the MVC architecture, the "*view*" is the user interface, the screens that the end user of the application actually sees and interacts with. In the JavaEE technology, views are typically JSP files. For collecting user input, there will be a JSP that generates an HTML page that contains one or more HTML forms. For displaying output, a JSP generates an HTML page containing the information to display. Each of these JSPs is a view: a way for the end user to interact with the system, putting data in, and getting data out. The "*model*" is a set of classes that contain the actual business logic. It also includes the database which stores business data. The model is responsible for persisting data, manipulating data, and providing computation results according to specific business rules. The "*controller*" is a mechanism to define how the model and the view are related to each other; for example, when an action is performed on a Web page, which part of the model should be invoked.

Figure 3 shows the MVC architecture, where the model is composed of a set of Java classes, an Object Persistence Layer, and a database. The object persistence layer is a third-party Java library. The Java classes interact with the object persistence layer via messages. The object persistence layer manages the JDBC connection to the database to perform database operations. The controller is a configuration file that defines the transition between Web pages.

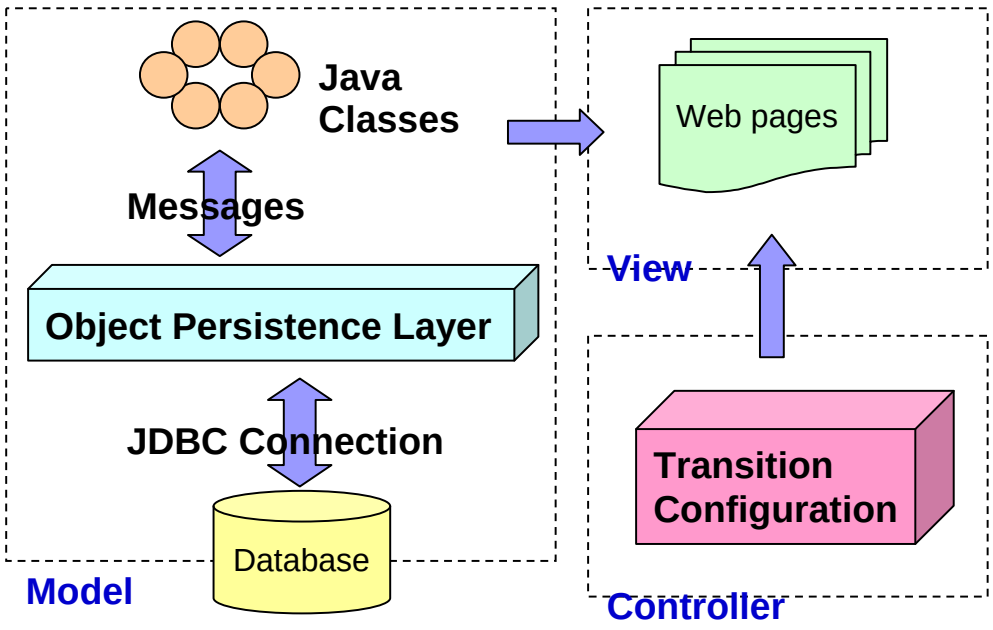


Figure 3: MVC architecture

The beauty of Model-View-Controller separation is that new views and controllers can be created independently of the model. The model defines a set of business functions that only called by controllers, and the controllers act as proxies between the end user (interacting with the view) and the business logic (encapsulated in the model). This means that a developer, without making any changes to the model, can add a new view and its associated controller for human beings to interact differently with the application.

3.3 Implementing MVC in the system

Struts 2.0 framework was adopted to implement the MVC architecture in the FMM Demo system. The detailed partition of objects and layers is shown in Figure 4. It clearly shows that the system is implemented based on an optimal maintainable MVC architecture with objects at different layers to maximize the modifiability, extensibility, and reusability of the code.

Figure 4: Implementation of the MVC Architecture

4. Database Design and Process Flows

4.1 Asset Management

4.1.1 Asset information and views

Assets are the objects that need to be maintained. Assets have multiple types of information at different layers and all data is logically linked to each other. Figure 5 gives an overview of the general concept of an asset that contains information at three layers:

Figure 5: Asset information wheel and tree structure

1) Basic information

   Basic information describes the properties of the asset itself. The most important data is the identification. Apart from the unique ID# that is referred to by the system, an asset has other ways to distinguish itself in various environments; for example, RFID, bar code, and ID in BIM models. The tree structure helps describe assets and the asset tree hierarchy. Currently, only

the “parent-children (1:n)” relationship is considered and the hierarchy allows multiple roots. There is a flag to mark whether an asset is a maintained item. So in an asset tree, we distinguish two types of assets: “maintainable” and “functional”.

2) Maintenance information

Maintenance information is a set of data that defines the rules for maintenance as well as the planned, pending, and past maintenance orders. As for maintenance settings, each maintainable asset can have multiple pre-defined rules/options for carrying out its preventive maintenance activities. The PM setting contains the following information:

- Temporal type: periodic or scheduled
- Work type: inspection, minor repair, major repair or replacement
- Time interval
- Counter
- Counter interval
- Shifting flexibility
- Default start time
- Default contractor
- Duration and cost
- Task list template
- Work content (trade or skill, i.e. plumbing)
- Requirements of resources (equipment, trades, materials, tools, contractors)

For the purpose of maintenance planning tracking, all maintenance orders generated by the system or entered by the user (including CMOs, PMOs, and CbMOs) are permenently stored in the database along with their status information. Using the status, one can easily distinguish planned orders from historical orders. In reality, more states are defined for MOs other than “planned” and “finished” in the database.

3) Life cycle condition-based information and models

The information and processes at this level are focused on “condition” as to record and assess the current and future condition of the asset. With deterioration models and condition assessment models in place, the system needs to calculate remaining life, condition and impacts for assets, based on maintenance work done (or to be done) and real-time data from sensor system monitoring models. This level of information and models can be designed and developed in the future, depending upon the evolution of the FMM Demo System.

4.1.2 Database tables

The main data objects related to asset management are shown in Figure 6. Some tables and important data are listed in the following paragraphs.

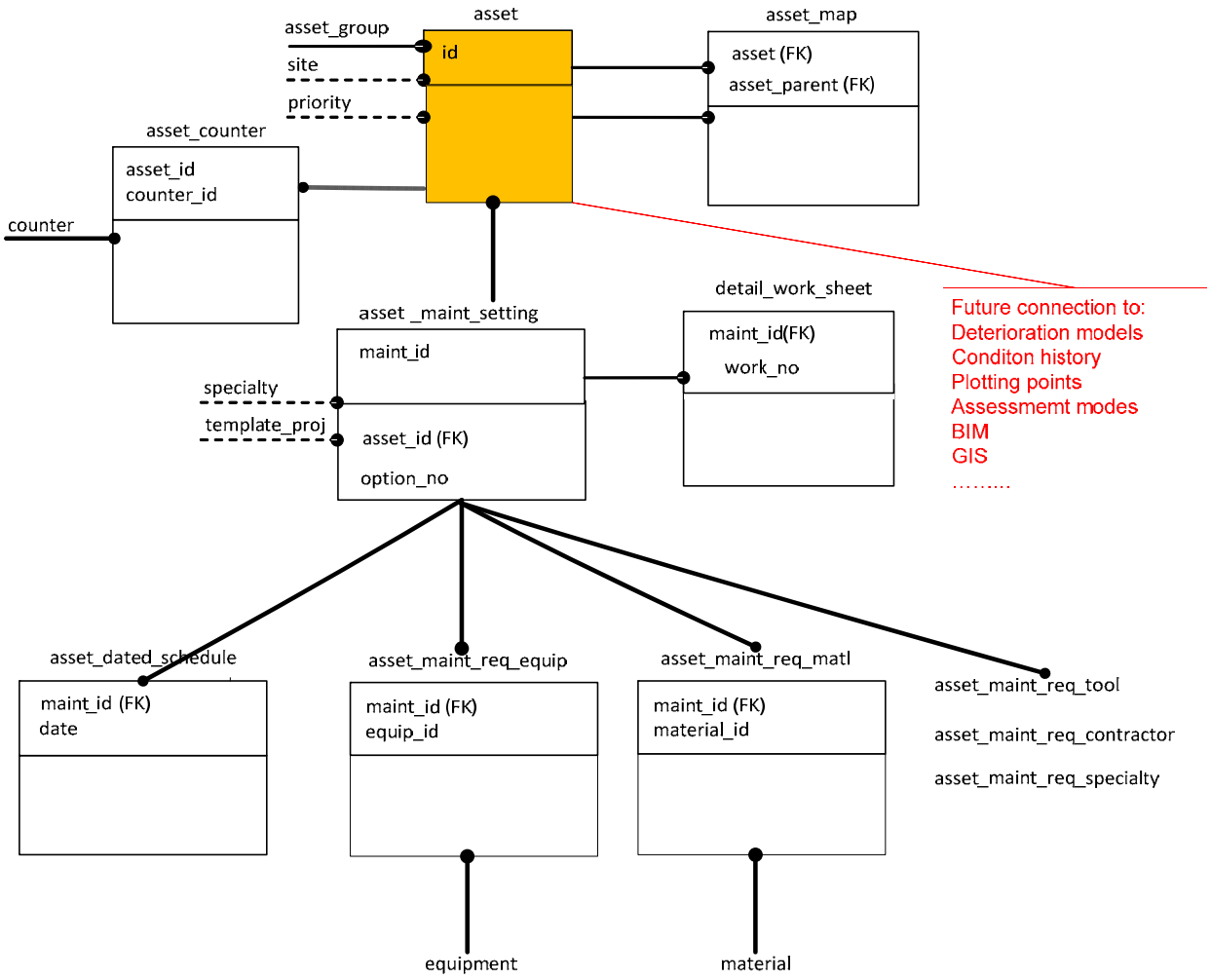


Figure 6: E-R diagram - Asset Management

[Asset]

The `*Asset*` table contains information and maintenance settings for assets for the purpose of facilities maintenance. Asset management software used by financial departments would have a different view of assets.

- `*asset_group*`: group assets into categories/views other than the main tree structure; for example, financial perspectives. This field is designed for future use.
- `*status*`: marks whether the asset is currently in use, temporarily or permanently out of use. In this demo, one always uses the default value of "in-use".
- `*serial_no*`, `*rf_code*`: keep and link with other identification numbers.
- `*site*`: the location of an asset. This field is designed for future use.
- `*capacity*`: the area (in square feet) that this asset occupies or requires.
- `*date_in_service*`, `*purchase_value*`: the date and the value that an asset is first purchased or a building is first delivered for occupancy.
- `*expected_life*`: the total life span of an asset (in years) from the date in service.
- `*current_cond_rank*`: an integer less than or equal to 100; it is used to keep the most recent estimation of an asset's condition. This number is required to be entered manually when an inspection or a maintenance work order has been finished. The number, representing the condition, generally elevates after a repair/replacement depending on the work being done and the re-assessment after completion. It is used to calculate a new condition assessment value at a future time according to the deterioration curve.
- `*init_condition_rank*`: the initial condition when an asset is created in the system.
- `*in_condition_assessment*`, `*weight*`: indicates whether an asset is involved in the complete condition assessment model and its weight. For future use only.
- `*estimated_value*`, `*depreciation_rate*`: to be used in calculating the current value of the asset in financial terms. This field is designed for future use.
- `*annual_budget*`: is a reference limit of annual budget for this asset if it applies. This budget includes only the sum of annual M&Os but not the capitals.
- `*priority*`: an integer indicating "priority" in optimization.
- `*is_leaf*`, `*level*`: whether this asset is a leaf node and its level in the asset hierarchy. These two fields combined with the asset_map table define an asset tree.
- `*manager_responsible*`: information only
- `*is_regulated*`: if an asset is regulated, its MOs scheduled have to be completed irrespective of budget and condition reasons. PMOs are always selected in the PM planning once they are generated.
- `*is_maint_item*`: indicate whether an asset is a maintenance item ("maintainable"). Only a leaf asset can be a maintenance item. Only a

maintenance item can have maintenance settings, detail work sheets or dated maintenance schedules.

- `*replacement_cost*`: the value in case an asset is to be replaced with another one.

[Asset_maint_setting]

The `*Asset_maint_setting*` table contains information and pre-defined rules for preventive maintenance work.

- `*option_no*`: a maintenance item can have multiple options for its inspection/ maintenance optimization. Decisions among these options are part of this decision support system at the strategy planning level and will be implemented on an “as needed” basis.
- `*temporal_type*`: a regular maintenance item, depending on the nature of work, may need purely periodical maintenance work or some discrete work on pre-determined dates:
    - Periodical maintenance/inspection schedule defined by `*time_interval*` and/or `*counter_interval*` depending on the nature of an asset. The default value for time interval is always “Month”, for simplicity’s sake.
    - Scheduled maintenance/inspection defined by a dated schedule in the `*maint_dated_schedule*` table.
- `*work_type*`: inspection, minor repair, major repair or full replacement.
- `*time_interval*`, `*interval_unit*`: define a periodic cycle based on time (month as default).
- `*counter*`, `*counter_interval*`: define a periodic cycle based on counters. Counter cyclic maintenance is not implemented at the current stage.
- `*deterioration_curve*`: always use a default curve for this demo.
- `*estimated_cond_improve*`: the estimated escalation of condition after the maintenance work is done. It will be useful in future optimization for selecting maintenance options.
- `*estimated_cost*`, `*labour_required*`, `*estimated_duration*`: estimations of the maintenance work.
- `*flexibility*`: a flexibility of time schedule in “Days”. The maintenance job can be done in a range of the scheduled start date plus or minus the "flexibility".
- `*work content*`: reference to work specialties defined in the *specialty* table. It is used to match the specialty of contractors or trades for selection. Examples of work contents are electric, plumbing, etc.
- `*default_start_time*`: sets a start time when history records of this asset cannot be retrieved, the cycle the periodic maintenance is broken, or when the user chooses to reset this time as a MO’s start time.
- `*template_id*`: if the content of the detailed maintenance work contains a predefined task list along with inherent precedence requirement, one can fill in its template in this field. A template is not needed for maintenance/

inspection that can be considered as having only one task.

*The detailed tasks in a work order can be defined using two options:

1) Simply use the `*detail_work_sheet*` and write whatever details you want;

2) Define a complete task network as a project template and specify this template in `*asset_maint_setting*`→`*template_id*`.

[Equipment]

The `*Equipment*` table is a basic table that describes the type of equipment resources. Other resource types (materials, tools, specialities, contractors) all have a similar definition.

- `*cost_base*`: lump sum / by hours
- `*standard_hour_rate*`, `*overtime_hour_rate*`
- `*minimum_hours*`: minimum hours that charge to a maintenance work. This is useful when scheduling maintenance work at the operation level to group small maintenance orders at the same site to reduce costs.
- `*startup_base_cost*`: the (virtual) cost of a maintenance work that is to be done on a site. This field is useful when considering the impact of site/locations on maintenance orders.
- `*idle_rate*`: the cost when the resource is idle which is useful in balancing the workloads on resources.

*Notes about resources:

1) `*Material*`, `*specialty*`, `*tool*` tables have “quantity” fields to declare a group of resources that are available at a certain amount.

2) Cost information, such as start up cost, hourly rate, and idle rates, works well with contractors, equipment and trade people but not with usage-based materials or tools.

4.1.3 Functional Requirements

The *Asset management function* requires an asset tree of hierarchy on the left (mark assets which are “*is_maint_item*” with a specific icon) and a tabbed window to see the following aspects of detailed information of a specific asset. The views marked with * are only available for “*is_maint_item*”.

- Asset Basic Information View: asset_name, model, serial_no, maker_name, date_in_service, purchase_cost, expected_life, capacity, warranty, specification, attachment, note.

- Asset Maintenance Setting View*: a table showing multiple options of preventive maintenance setting for the asset including the option_number,

work-content, time_interval, priority, is_regulated, estimated_cost, labor_required, estimated_duration, flexibility, specialty, default_contractor, default_start_time, manager_responsible and one (or many) of the following links:

    - View dated_schedule (only provided for temporal_type = “scheduled”)
    - View detail work sheet (only applies to valid records)
    - View template

- Asset Condition View*: asset_name, date_in_service, expected_life, designed_condition (to date, calculated from the deterioration curve), last_estimated_condition (equals cond_rank in asset), current_condition (to date, get from the most recent “condition_modified” from the issues related to this asset).
    - View Curve. Two plotted curves show: 1) the designed curve; and 2) the real history of condition records and the projection into future years. This function for showing the deterioration curves is not implemented in the current FMM Demo system.
- Asset Budget View (issue table; last 5 years): year, asset_name (a specific asset / list of assets), maintenance_cost_planned (issues of a specific asset / planned issues of lower level maintenance assets), maintenance_cost_spent (issues of a specific asset / list of finished issues of lower level assets), annual_budget.
    - Sequenced by asset_name at each level.
    - Highlight the assets over budget: cost planned > annual_budget
    - Highlight the assets over spending: cost spent > cost planned
- Asset Maintenance Plan View* (issue table; status = “planned”): maint_type, estimated start date, planned start date, duration, contractor, manager, estimated_cost, labour_required.
    - Sequenced by date.
    - Differentiate maintenance and inspections.
    - Highlight delayed issues.
    - Show summary information.
- Asset Maintenance History View (issue table; status = “finished”, “pending”, “expired”): maint_type, planned_start_date, actual_start_date,

estimated_duration, actual_duration, contractor, estimated_cost, actual_cost, status.

- Sequenced by date. Grouped by status.
- Differentiate maintenance and inspections.
- Differentiate status.
- Show summary information.

4.2 Preventive Maintenance Planning

4.2.1 PM planning process

The complete preventive maintenance planning process is shown in Figure 7. Currently, the PM planning cycle is fixed to the calendar year. The flexibility of the planning period can be implemented in the future depending on needs. The PM planning process can be divided into six major steps.

1) The system finds a set of maintainable assets in the asset tree by looping through the complete sub-tree for each root node.
2) For each maintainable asset, the system generates a list of its PMOs according to the following rules:
    a. If there is a dated schedule for this asset, the system generates PMOs directly using the pre-set dates that fall into the current planning period.
    b. If there is a periodical schedule setting for this asset, the system generates PMOs according to the following steps:
        i. Create the initial cycle when there are no historical MOs. The “estimated start time” of the first MO in the current planning cycle will be set to the “default start time” by default, as specified in the *asset_maint_ setting*.
        ii. If there are historical MOs, find the last PMO (finished or pending) with the most recent planned time (NB: not the actual time). Add a time interval to that time and create the first PMO. The system needs to check whether the estimated time of the first PMO falls in the current PM planning cycle. If not, re-set the estimated time for the first PMO to its initial cycle as in 2)-a.
        iii. Add another interval to the first PMO. If the estimated time is still in the current planning period, create another new PMO. Repeat until the planned time exceeds the current planning period.

Figure 7: Preventive maintenance planning process

3) For the list of generated PMOs, the system finds a sub-list of PMOs as executable MOs and marked the remaining PMOs as “pending”.

4) For each executable MO, be it a PMO, CMO or CbMo, the system will generate a project and link the project with the MO.

5) The system calculates a schedule for the generated project. The schedule is “feasible” because all constraints are guaranteed by using a time-constrained network based on the RCPSP algorithm developed for the DND project [3].

6) After all executable MOs are planned, the system can do a cross-project schedule optimization to solve the conflicts in regards to resources and optimize their utilization. This is another major decision point during the PM planning process to perform optimization at the operational level.

*Notes about the PM planning process:

1) The “executable MO” and “Schedule Opt.” are two major modules where decision support is very much required.

    a. The objective of the executable MO selection is to find a list of PMO work for the next PM planning period based on current condition, remaining life, deterioration model, priority, cost, and prediction of future impacts leveling against a limited M&O budget base. This module is

currently implemented as a manual selection process but it can be upgraded to an automated process featuring a strategic optimization algorithm to direct the allocation of the M&O budget within an organization.

b. The schedule optimization at the operational level has been implemented in the current FMM Demo system. Details of the algorithm will be introduced in Chapter 6.

2) Treating all MOs as projects at the back end provides a good merge of maintenance orders with maintenance projects, which is one of the major distinguishing features of this system. This feature allows the representation of maintenance operations as a complex task network. Other CMMS systems on the market, even the big players such as MAXIMO and SAP-PM, can only describe a sequential task list for a MO. Complex relationships and constraints between tasks are not allowed. Refer to publication [1] and [3] for the complexity of the RCPSP problem and the numerous constraints that were handled in the project.

3) Corrective MOs, entered manually by the user directly, become executable in the system because the decision of whether it needs to be done is already decided. Also in the future, when the condition-based maintenance process is in place, the CbMOs will directly become executable for the same reason.

4) Major maintenance projects enter the system directly as a project without linkage to any MO.

4.2.2 Database tables

The main data objects related to PM planning management and MO management are shown in Figure 8. Some tables and important data are listed in the following paragraphs.

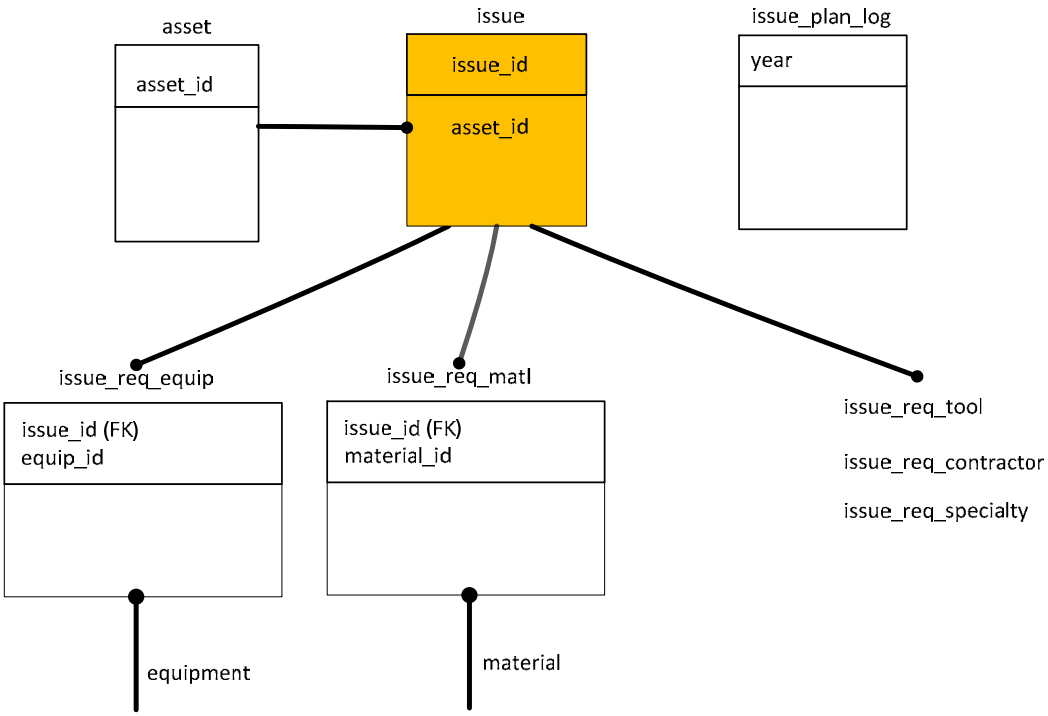


Figure 8: E-R diagram - Issue Management

[Issue]

This table contains all maintenance/inspection orders and a repository of maintenance history for all assets. The term “issue” is synonymous with “maintenance order” in the rest of the report.

- `*maint_type*`: PM, CM or CBM
- `*temporal_type*`, `*work_content*`: same meaning as in `*asset_maint_setting*`
- `*flexibility*`, `*priority*`, `*is_regulated*`, `*template_id*`: carried from the `*asset*` table
- `*is_system_generated*`: 0-yes/1-no. This flag is designed to identify whether the issue item is generated by the system. CMOs are entered by the user and thus this flag is always marked “1” for CMOs.
- `*estimated_start_time*`, `*estimated_end_time*`, `*estimated_duration*`, `*estimated_cost*`, `*estimated_labor*`: the initial maintenance schedule and cost information that the system calculates or carries over from PM setting. Please refer to Section 4.2 for the details of this logic.
- `*planned_start_time*`, `*planned_end_time*`: the planned time period calculated by the planning process in its corresponding project.
- `*actual_start_time*`, `*actual_end_time*`, `*actual_duration*`, `*actual_worker_id*`, `*actual_cost*`, `*actual_labor*`, `*actual_overtime*`: entered after an issue has been finished.
- `*status*`: the status of an issue. The value of status could be: “created”,

“planned”, “pending”, “finished”, etc. Figure 9 shows a state transition graph of a typical issue.

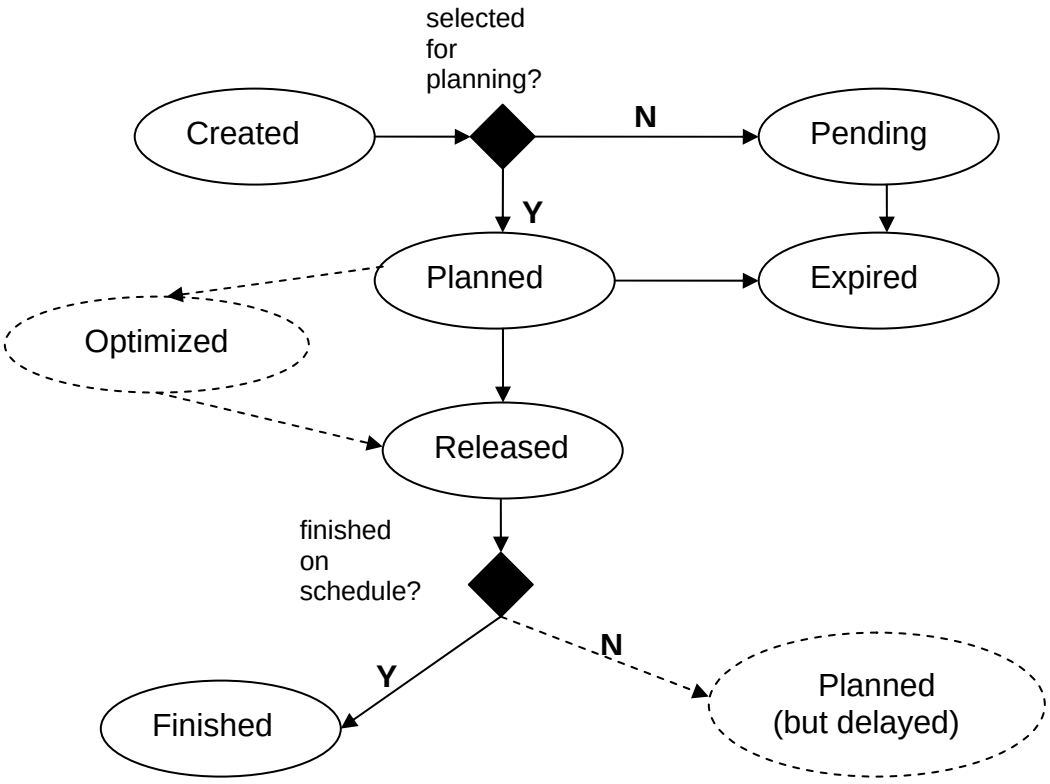


Figure 9: State transition for maintenance orders

- `*condition_rank_initial*`: the initial cond_rank read from the `*asset*` table.
- `*condition_rank_modified*`: re-assessment of the condition value entered by the user or evaluated by the system after the issue is finished.
- `*source_issue_id*`: is for the follow-up issues to record their resource of request issue.
- `*need_contractor*`, `*contractor*`: default contractor from the `*asset*` table; or otherwise, a contractor that the user selected.
- `*responsible_person*`: default from the `*asset*` table; and could be changed to another person.
- `*no_of_defaults*`, `*default_desc*`: entered after the issue has been finished. It could be a source for rating contractors in the future.
- `*project_id*`: refer to the linked project. All MOs have a linked project but not valid vice versa.

[Issue_plan_log]

This table contains a plan log for issue planning. It indicates whether the maintenance issues have been planned for a certain year.

4.2.3 Functional requirements

The *Preventive maintenance planning function* is done on a yearly basis following the process described in Figure 7. The user first selects a year. The system will check whether the PMOs for the selected year are already in the database.

- If no issues are observed, the system automatically continues the process by generating issues for the year.
- If issues are already planned, the system lists the issues of the year.
    - The user can remove all the issues, and ask the system to regenerate issues.

For the generated issues, each issue is provided with a selector for the user to decide whether it is to be “executed”.

- “select all” and “un-select all” buttons are provided for this step.

After issue selection, when the user elects to continue, the system generates projects for all executable issues and calculates an initial plan for each project. Now, all issues have a status “planned”, marking the end of the preventive maintenance planning.

The *Issue management function* shows a list of issues in a tabular view with maint_type, work_content, status, estimated start date, planned start date, actual start date, duration, manager, estimated_cost, actual_cost, worker_id, etc.

- Sequenced by date
- Grouped by status
- Differentiate maintenance and inspections.
- Highlight delayed issues.
- Show summary information.
- *buttons to transfer from delayed “planned” or “pending” issues to “expired” issues. Make changes in projects as well.
- Buttons to input actual information: actual_start_time, actual_duration, actual_cost, actual_labor, actual_overtime, condition_rank_modified, no_of_defaults, default_desc, etc.
- Filters: status, time period, asset

The *Issues entry function* is for the user to manually enter a maintenance order, most likely the corrective orders. These issues are always “selected” and thus “planned” after the project is generated. This function is not implemented because it is very simple.

The *Actual information record function* is to enter actual information to a maintenance order and close the order. Many groups of interfaces of the same function are required for a regular desktop configuration and mobile handheld devices. This function is only developed on iPhone because the technologies and interfaces are very simple on a desktop PC.

4.3 Workload Analysis and Optimization

4.3.1 Overview

Workload analysis and optimization is a group of functions/interfaces to find conflicts, resolve conflicts and optimize schedules based on resources. As mentioned before, five groups of resources are considered; they are: equipment, material, tool, specialty trade, and contractor. As for the scope of workload analysis, the user is allowed to set his/her scope freely through:

- *A time window* in weeks, months, or even longer, to analyze the workload. However, the system does not suggest setting a very long period to avoid lengthy calculations inherent with very large scope selection for optimization.
- *The issues and projects* that he/she is interested in analyzing by selecting from the full lists of issues and projects the system provides that fall within the time window.
- *The types of resources* to optimize by selecting from the full lists of resource types provided by the system.
- *The list of individual resources* by selecting from the resource list.

After setting the scope, first, the system needs to detect schedule conflicts and display the conflicting situations, if there are any. The system then tries to solve the conflicts with a choice of three types of optimization:

- Calculate the first feasible solution based on Local Search;
- Calculate the first feasible solution based on Schedule Repair;
- Calculate a complete optimized solution.

Even if there are no conflicts, which means the initial solution is already feasible, the system can still perform a complete optimization (option 3) to optimize schedules on projects and resources.

Because each issue has a corresponding project, the inputs to the optimization algorithm can be unified to purely projects or tasks at the back end. Therefore, the user interface needs to do the translation, or pre-processing of data before execution of the algorithm.

The implemented algorithm uses a meta-heuristic approach called “complete local search” to optimize the maintenance work schedules on each individual resource, based on utilization. The current algorithm considers the shifting flexibility of projects and resource requirements of tasks. The objective function is makespan. However, one is able to add economical (setup cost, idle cost, hourly rate, minimum hours, etc.) and location aspects into the objective function.

4.3.2 Functional requirements

1) 1st page

a. The user specifies a time window by entering a start date and an end date
b. The user clicks the UPDATE button and the system updates the issue list (#, asset code, issue name, planned start time) and project list (#, asset code, project name, planned start time) that fall into or overlap the time window.
c. The user can narrow the list of issues and projects by selecting the check boxes beside each item.
d. The user then continues to select the types of resources that will be included in the optimization.
e. The user can view a detailed list of resources based on their selection by clicking the ‘SHOW RESOURCES’ button.
f. The user clicks the CONTINUE button to go to the next page.
g. The system gets a list of selected projects by converting the selected issues to projects. (The project list is formed by both the issue list and project list selected in step b.)
h. The system creates an instance of Optimization Class by passing in a project list and a resource list. Set a start time (LONG type), an end time, and methodType=”ORIGIN” before running the instance.
i. The results of the optimization are accessed through the following objects:
    - *Timetable*
      HASHMAP (KEY=TASK ID, VALUE=START TIME)
    - *Detail*
      HASHMAP (KEY=RESOURCE ID, VALUE=HASHMAP (KEY=INTERVAL(string, string), VALUE =HASHSET (task list)))
    - *Usability*
      HASHMAP (KEY=RESOURCE ID, VALUE=HASHMAP (KEY=INTERVAL(string, string), VALUE =PERCENTAGE)

2) 2nd page with three tabs -“General”, “Task”, and “Resource”

a. The “General” tab shows:

- Time window information
- Number of projects in scope;
- Number of tasks in scope;
- Number of resources in scope;
- Number of conflicts detected;
- Number of resources that have conflicts; list the resources
- Number of time intervals that have conflicts; list the intervals
- Number of tasks that have conflicts; list the tasks
- Solution status: original plan, feasible plan, or optimized plan
- Number of moved tasks

b. Three options will show at the bottom of the “general” tab – “SCHEDULE REPAIR”, “RESOLVE CONFLICT”, and “OPTIMIZE”. *“SCHEDULE REPAIR”* and *“RESOLVE CONFLICT” only show when t conflicts exist in the initial solution.*

i. in case of “RESOLVE CONFLICT”
The system creates an instance of Optimization Class by passing in a project list and a resource list. Set a start time (LONG type), an end time, and methodType=”FIRST” before running the instance. And immediately a feasible solution can be calculated and the results are stored in the same objects and data structure as in 1.i.

ii. in case of “OPTIMIZATION”
The system creates an instance of Optimization Class by passing in a project list and a resource list. Set a start time (LONG type), an end time, and methodType=”OPT” before run the instance. The results are stored in the same objects and data structure as in 1.i.

iii. in case of “SCHEDULE REPAIR”
Same as above except setting the methodType = “REPAIR” before run the optimization instance.

The optimization process might take a very long time to terminate, depending on the size of the problem. So, an optimization progress page is required as described in 3.

** The system gets the results of FIRST, REPAIR or OPT execution, and shows the solution using the same group of interfaces as described above in this step.*

c. The “Task” tab will show the task plan list in scope in a table (task name, project name, start time, end time, duration, estimated cost)

    i. Order the items by planned start time.
    ii. Highlight the conflict tasks in the table.
    iii. GANTT button would direct to graphical display.
    iv. SHOW CONFLICT ONLY button would show only the tasks in conflict.

d. The “Resource” tab will show the resource plan in scope in a table (resource name, time period, utilization percentage, and link to task list)

    i. Sort the items by resource name; then order by time period.
    ii. Highlight the conflict time periods on resources (i.e. for utilization rate greater than 0.8).
    iii. BAR CHART button would direct to graphical display of bar chart.
    iv. SHOW CONFLICT ONLY button would show only the resources and time periods that have conflicts.

3) 3rd page – optimization progress

a. Example data on this page would be: number of iterations, best solution, value of objectives, number of solutions without improvement, estimated iterations remaining, etc. and a status bar with percentage showing.

b. The system needs to check the status every two seconds.

c. There is a TERMINATE button to allow the user to terminate the calculation anytime when the user deems it appropriate.

5. Technical Solutions

5.1 Java

Java technology is an object-oriented, platform-independent, multi-threaded programming environment. It is the foundation of the Web and other networked services, applications, platform-independent desktops, robotics, and other embedded devices.

In the family of Java technology, *Java Platform Enterprise Edition (JEE)* builds on the solid foundation of Java Platform, Standard Edition (Java SE) and is the industry standard for implementing enterprise-class, service-oriented architecture (SOA) and next-generation Web applications.

JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic Web content. JSP technology enables rapid development of Web-based applications that are server and platform-independent.

The Java technology family provides essential capability, performance, flexibility, and expandability, and therefore, JEE was adopted as the fundamental technology to build the entire application and use JSP to create the dynamic Web pages.

5.2 Object Persistence

The “impedance mismatch” between relational databases' tabular orientation and Java's object-oriented hierarchical one has led to the development of several different object persistence technologies in an attempt to bridge the gap between the relational world and the object world. Among the solutions for object persistence, *Hibernate* is a powerful, high performance object/relational persistence and query service. Hibernate enables developers to focus on the business model and adheres to the notion of object-orientation without concerning database operations since they are managed by the technology itself.

Hibernate is a Java framework that provides object/relational mapping mechanisms to define how Java objects are stored, updated, deleted, and retrieved. In addition, Hibernate also allows the developers to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with object-oriented Criteria and Example API. Ultimately, Hibernate reduces the effort needed to convert relational database result-sets and graphs of Java objects.

One of Hibernates unique features is that it does not require developers to implement proprietary interfaces or extend proprietary base classes in order for classes to be made persistent. Instead, Hibernate relies on Java reflection and runtime augmentation of classes using a powerful, high-performance, code-generation library for Java called CGLIB. CGLIB is used to extend Java classes and implement Java interfaces at runtime.

Based on these features, Hibernate was chosen as the solution for object persistence in the system. More information regarding Hibernate can be accessed from https://www.hibernate.org/.

5.3. Struts

Web applications, based on Java Server Pages, sometimes commingle database code, page design code, and control flow code together. Unless these concerns are separated, larger applications become very difficult to maintain.

The Model-View-Controller (MVC) architecture is a practical way to separate these concerns in a software application. In the MVC architecture, the *Model* represents the business or database code, the *View* represents the page design code, and the *Controller* represents the navigational code. Struts is a free, open source framework from Apache Jakarta for building large-scale Java Web applications using the MVC architecture.

The framework provides three key components:

- A “request” handler provided by the application developer that is mapped to a standard URI.
- A “response” handler that transfers control to another resource which completes the response.
- A tag library that enables developers to create interactive, form-based applications with server pages.

In our work, we adopted Struts 2.0 to implement the MVC architecture. Detailed information about the Struts framework can be accessed from http://struts.apache.org/.

5.4 Yahoo! UI Components

In the past, the most widely cited advantage stand alone computer applications held over web applications was the availability of rich user interface widgets, which were not initially available for web development. Recently, however, there have been major advances in this area. Currently, there are several JavaScript libraries that enable a programmer to utilize complex and more visually appealing widgets when creating a web application. One such library is the *Yahoo! User Interface Library*, more commonly referred to as YUI.

YUI is a coherent collection of JavaScript and CSS resources that makes it easier to build richly interactive environments within Web browsers. A few of the most important features provided by YUI include (but are not limited to): animations, auto completion, calendars, carousels, colour pickers, drag and drop, non-standard

events, image loading and cropping, menus, “pagers”, rich text editors, selectors, sliders, tabs and “tree views”. Use of these widgets promotes the level of familiarity and comfort to the client using advanced controls more commonly used in standalone applications.

We selected YUI for this project for several reasons:

1) The YUI library is free for both commercial and non-profit use, licensed under a ‘liberal’ BSD license, which is always a bonus for a development project.

2) YUI is developed, maintained and supported by developers at Yahoo!, and is used in many of the Yahoo! pages. This provides ample examples, test-beds, and continuous use, testing and development.

3) The YUI library is light and flexible, minimizing the amount of resources sent across a network while still providing full functionality for a wide variety of applications.

4) YUI’s modular design allows a programmer to include only the pieces he or she needs, which again contributes to the minimization of data transfer. For example, a programmer who uses only one widget from the YUI Library is therefore only required to include the resources necessary for that specific widget, not the entire YUI library.

5) Browser support is also a major benefit of the YUI library. Currently, the library supports all ‘A-grade’ browsers, including Firefox 3.x, Internet Explorer 6.x, 7.x, and 8.x, Opera 9.6 as well as Safari 3.2 and 4.0 (on Mac OS 10.4 and 10.5, respectively). This is a critical consideration, as limiting users to a Web application with only one type of browser could hinder the feasibility and attractiveness of a project to potential clients.

6) The final factor which makes the YUI library a great choice for integration into our project was the vast and informative API documentation provided by the Yahoo! developers. Not only is there documentation for every YUI component, there is a plethora of real-world examples, user guides, and “cheat sheets” available.

All of these factors combined made the Yahoo! User Interface JavaScript library an obvious choice for integration into the development of the FMM Demo system. The YUI library can be downloaded from http://developer.yahoo.com/yui/.

5.5. Ajax and jQuery

Ajax (shorthand for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side for interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the Web server asynchronously in the background without interfering with the display and behaviour of the existing page. In the background, it trades data with the server; and

in the front, it updates data to the current page without reloading the page. User experience can be greatly improved since only a small portion of the page is updated while other parts stay the same. It also increases the performance of an application because much less data is transferred over the network and the response speed is significant increased.

In our development, we adopted jQuery as the primary JavaScript framework to support Ajax calls. *jQuery* is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid Web development. It emphasizes the interaction between JavaScript and HTML.

Just as CSS separates "display" characteristics from the HTML structure, jQuery separates the "behaviour" characteristics from the HTML structure. For example, instead of directly specifying the on-click event handler in the specification of a button element, a jQuery driven page would first identify the button element, and then modify its onclick event handler.

jQuery contains the following features:

- DOM element selections using the cross-browser open source selector engine Sizzle, a spin-off jQuery project.
- DOM traversal and modification (including support for CSS 1-3 selectors and basic Xpath).
- Events
- CSS manipulation
- Effects and animations
- Ajax
- Extensibility
- Utilities - such as browser detection and each utility function
- JavaScript Plug-ins

Detailed information about jQuery can be found at http://www.jquery.com/.

5.6. iPhone-based Web Application

The on-site facility maintenance work can be better accomplished if a worker can carry a mobile computing device to enter real time information instead of carrying a computer. This requires us to explore the feasibility of providing support to mobile computing devices such as smart phones.

The iPhone is one of the latest devices for mobile computing platforms. Its unique features, including a multi-touch LCD screen, a fully-functioning browser (Safari), embedded GPS, portable media player, WiFi, email, and 3G connectivity, have made it today's most popular smart phone. Therefore, iPhone was chosen as the first platform to provide mobile support.

Apple provides a set of APIs and an IDE to develop native applications for iPhone. With the support of the Safari Web browser, it is also possible to develop Web applications for the iPhone. Safari functions just like other industry standard browsers such as Firefox and IE. The major difference is that Safari in iPhone must accommodate to a small screen (480x320 pixels). Therefore, the Web applications designed for the iPhone should carefully consider the size of the screen. Furthermore, the applications can benefit from the multi-touch screen, which requires the designer to think more about how users can interact with such a device.

Thus, the Web pages developed for the iPhone are comprised of a set of Web pages that are separated from the regular pages. Also, in the index page of the FMM Demo system, an auto-detection mechanism is needed to recognize the source of the HTTP request (either from a regular computer or from an iPhone in this case); and then the system needs to automatically load the correct set of Web pages.

This can be achieved by the following JavaScript code. This code segment should be inserted into the HEAD segment of the index page of the Website.

```
<script language="javascript">
  <!--
    if((navigator.userAgent.match(/iPhone/i)) ||
         (navigator.userAgent.match(/iPod/i)))
    {
      location.replace("iphone/index.jsp");
    }
  -->
</script>
```

In this work, iUI was adopted to create the iPhone version of the Web pages. *iUI* is a framework consisting of a JavaScript library, CSS, and images for developing iPhone Web applications. iUI has the following features:

- Ability to create navigational menus and iPhone interfaces from standard HTML.
- Experience or knowledge of JavaScript is not required to create basic iPhone pages.
- Ability to handle phone orientation changes.
- Provide a more "iPhone-like" experience to Web apps (on or off the iPhone).

iUI can be accessed from http://code.google.com/p/iui/.

5.7. Flot

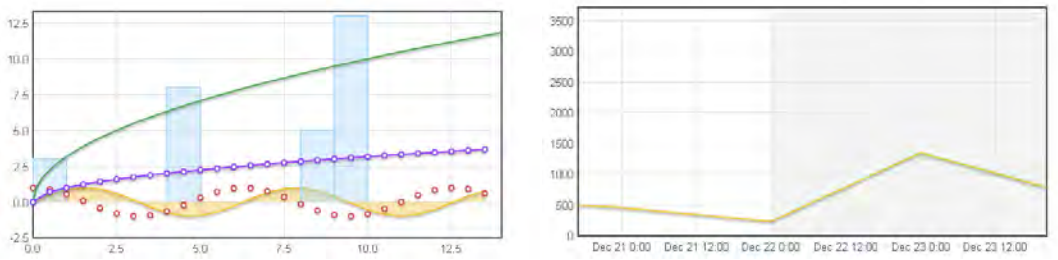
Flot is a pure JavaScript plotting library which relies on jQuery. It is capable of producing graphical plots representing arbitrary data ‘on-the-fly’, client side. The ability to generate a plot ‘on-the-fly’ enables the programmer to dynamically create illustrations, where the data being represented may change constantly depending on the selection or modification by the user. This is very important for most web applications due to the nature of an application versus that of a static web page.

A web page’s content is always displayed in a single manner, and changing its content requires modification of the HTML code. However, a web application is dynamic in nature because of the fact that an application provides interfaces for the user to create, remove, modify or delete some sort of data. In this way, data being presented to the use, via a web application, could be different every time the user visits the site, and therefore, representing d ata graphically requires a library that can handle dynamic data and not only “pre-defined” or ‘hard-coded’ data.

This feature also renders the related client-server processing more efficient. Because Flot generates the plots on the “client side”, it reduces the amount of time and processing required by the server. The data is created on the server, sent to the client’s machine, and their browser is then responsible for running the JavaScript program to instruct Flot how, where and what to plot. This in turn improves the application’s response times, and, in effect, the application becomes more efficient.

Currently, Flot is supported by almost all web browsers. Flot has been tested thoroughly with main-stream browsers such as Firefox 2.x+, Safari 3.0+, Opera 9.5+, Konqueror 4.x+ and Internet Explorer 6 and 7. The only exception is Internet Explorer 8 where the “excanvas” JavaScript emulation helper is used. Flot only supports IE8 in “development mode”.

Flot provides a wide range of capabilities for representing data. Graph types that are supported include pies, bars, connected lines, broken lines, filled areas, points, and connected points. Also, with the exception of pie graphs, one plot is not limited to being represented in one style. For instance, one plot may have bars, points and lines all representing different data sets (Figure 10-a). Flot also supports the concept of customizability, so as the programmer can control the axes (for example, denoting timestamps as an axis as shown in Figure 10-b), legend, graph type, the look of the grid, colours, etc.



(a) (b)

Figure 10: Flot graph samples – customized graph styles

More advanced interactive features of Flot include functionalities such as turning a data series on or off, selection support, zooming, overviews of data, data interaction and mouse tracking, which will be introduced in the following paragraphs.

Turning a data series on or off is fairly self explanatory, and can be accomplished relatively easily by providing the user with checkboxes, each representing one of the data sets illustrated in the graph. When the user selects or deselects that data set, the plot responds by hiding or showing the corresponding data on the graph. In doing so, Flot also automatically scales the axes to best represent the data in the space allowed.

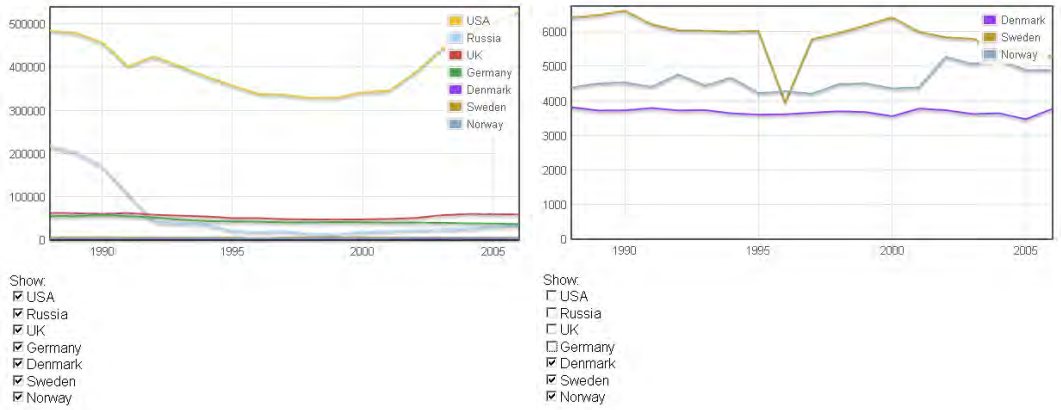


Figure 11: Flot graph features – turning data series on and off

Selection support, data interaction and mouse tracking are all very closely tied together and most often would be used in conjunction with one another. However, it is not necessary to use any or all three can be used. Selection support allows the users to select points or areas on the graph, regardless of how data is presented. Data interaction refers to the ability of the plot to relay information about the users' selection to the browser. For instance, selecting a point may update an area on a webpage where information regarding the selected point is displayed. Mouse interaction is inherent here, and simply allows Flot to track where the mouse is on the plot, and ultimately what is being selected by the user. (Illustrated in Figure 12).

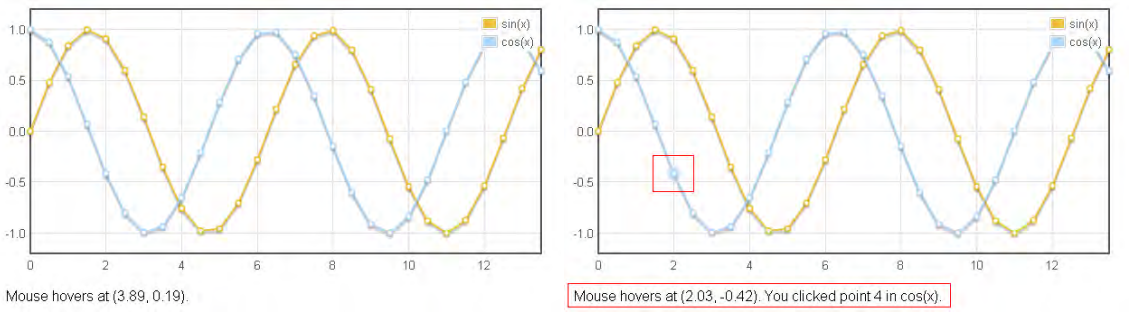


Figure 12: Flot graph features – mouse tracking and data interaction

One of the most unique and advanced features of Flot is the ability to zoom and provide an overview graph. Using the Zoom feature, the programmer is able to provide either pre-defined zooming buttons, or provide what is called an “overview” plot representing the entire data series. Upon clicking one of the buttons, the axes on the plot will automatically be adjusted to correspond to the predefined scale settings designated by the button and adjust the data where necessary. On the other hand, an overview plot provides the user with the ability to explicitly select an area of the graph one is interested in, rather than some pre-defined zooming scales. In this way, the user is in complete control of the data illustrated in the plot (as shown in Figure 13).

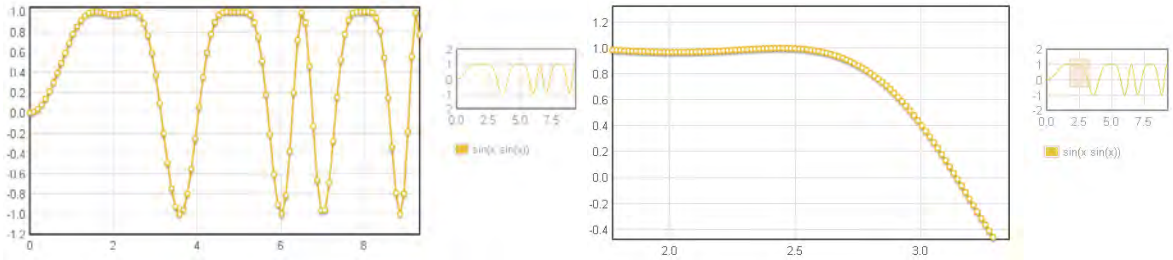


Figure 13: Flot graph features – zoom through “overview”

The only requirements for Flot are a JavaScript enabled browser and jQuery. Since all modern browsers have JavaScript capabilities built in and jQuery is already being utilized in the project for Ajax, it is very convenient to use this library. Information about Flot can be retrieved from http://code.google.com/p/flot/.

6. Scheduling and Optimization Algorithms – Design and Implementation

In this work, a complete local search algorithm to do optimization for project schedules based on resources was developed. The objective function is currently makespan minimization; in the future it may be decided to add economical and location concerns to fulfill more practical needs from the industry. Note that the FMM Demo project and the algorithm were wrapped up in separate papers [4] [5]. The following sections will briefly introduce the problem, formulation, heuristics, processing logics and results of the optimization algorithm.

6.1 Problem description

The scheduling problem in the system can be modeled as a very classical scheduling problem, i.e. the *Resource Constrained Project Scheduling Problem (RCPSP)*. The problem is NP-hard in a strong sense. The problem can be described as follows:

A project consists of $n+2$ activities where each activity has to be processed in order to complete the project. Let $J=\{0,1,\cdots,n,n+1\}$ denote the set of activities to be scheduled and $K=\{1,\cdots,k\}$ be the set of resources. Activities 0 and $n+1$ have a zero duration and represent two dummy activities – dummy source and dummy sink, respectively. The activities are interrelated by two types of constraints:

1) Precedence constraints force each activity j to be scheduled after all predecessor activities P_j are completed;

2) Activities require resources with limited capacities.

   While being processed, activity j requires $r_{j,k}$ units of resource type $k \in K$ during the whole period of its duration d_j. The assumption we made is that no preemption is allowed. Resource type k has a limited capacity of R_k available at any point in time. The parameters d_j, $r_{j,k}$ and R_k are supposed to be non-negative, deterministic integers. For the dummy source and dummy sink activities, we have $d_0 = d_{n+1} = 0$ and $r_{0,k} = r_{n+1,k} = 0$ for all $k \in K$.

Solving the RCPSP problem means finding a schedule for the activities, taking into account the resources and the precedence constraints, that minimizes the makespan $C_{\max}$, i.e. the completion time of the dummy sink activity.

Figure 14 shows an example of a project network, where the activities on nodes and precedence on directed arcs are presented. The resource requirements are given on top of each node and the total available numbers of resources are provided at the bottom of the network.

Figure 14: A project network example

In the system, solutions are represented in the form of a precedence-feasible activity sequence $\pi(\pi_{[1]},\cdots,\pi_{[n]})$ where $\pi_{[i]} \in J$ and i is the index of an activity in the sequence. The *precedence-feasible activity sequence* is a sequence of activities satisfying the precedence constraints, i.e. any activity must have all its predecessors sequenced ahead. In the context, a precedence-feasible activity sequence is also called a valid sequence for short. For example, $(1,4,2,3,5,6,7,8,9,10)$ and $(2,5,6,1,4,7,8,3,9,10)$ are two valid sequences for the example project network in Figure 14.

Let $t_{[i]}$ represent the start time of activity $\pi_{[i]}$. A schedule for a valid sequence can be represented by a vector of start times $(t_{[1]},\cdots,t_{[n]})$. For the same example, Figure 15 shows a feasible schedule with the makespan of 17.

Figure 15: The Gantt chart of a feasible schedule with the makespan of 17

The RCPSP can be decomposed into two sub-problems: (1) the schedule generation problem, in which a feasible schedule for a given valid sequence is calculated and (2) the sequencing problem, in which a valid sequence is found to have the optimal schedule. The schedule generation problem is embedded in the sequencing problem. A forward and reverse schedule generation method is developed for the schedule generation problem and a complete local search is proposed for the sequencing problem.

6.2 The schedule generation method

Given a valid sequence, the schedule generation method generates a feasible schedule for the sequence and evaluates the obtained schedule. Two kinds of schedule generation schemes are adopted in the system: forward schedule and reverse schedule.

A forward schedule S_F is a mapping where $\pi_{[1]},\cdots,\pi_{[n]}$ are mapped to a set of start times which are set to be as early as possible while still satisfying the resource constraints.

The reverse schedule S_R is developed based on the forward schedule scheme by reversing all the precedence in the project network and the given valid sequence. A reversed project network is the network with all the precedence linkages reversed, i.e. an activity's predecessors in the original network become successors and its successors are changed to predecessors (as shown Figure 16). Similarly, a reverse sequence reverses the order of activities in the sequence; for example, the reverse of the sequence $(1,2,3,4)$ is $(4,3,2,1)$. The reverse sequence of a valid sequence is precedence-feasible for the reversed project network as well.

Figure 16: The reverse project network and reverse sequence

The procedure to generate a reverse schedule is shown as follows:

1) Reverse all the precedence.

2) Reverse the given valid sequence.

3) Calculate the forward schedule of the reversed sequence based on the reversed precedence.

4) Transform the schedule obtained by step (3) to a feasible schedule satisfying the original precedence constraints.

Suppose the schedule obtained by step (3) is $(t'_{[1]},\cdots,t'_{[n]})$ with a makespan of $C_{\max}$, the transformed schedule is generated by step (4) using $(C_{\max}-t'_{[1]}-d_{[1]},\cdots,C_{\max}-t'_{[n]}-d_{[n]})$. Notice that the transformation in step (4) does not change the makespan.

The procedure of the forward and reverse schedule generation method consists of three steps:

1) Calculate the forward schedule S_F of the given valid sequence.

2) Calculate the reverse schedule S_R of the given valid sequence.

3) The schedule with the smaller makespan value is selected as the final schedule of the given sequence.

6.3 The complete local search method

There are three containers in the algorithm: (1) NEWGEN, which accommodates the newly found valid sequences which will be explored in the current iteration; (2) NEIGHBOR, which accommodates the neighbors of the sequences in NEWGEN; and (3) DEAD, which accommodates all the explored valid sequences to avoid recalculating.

CLLM starts with a random valid sequence as the initial solution and puts it into NEWGEN. At the beginning, DEAD and NEIGHBOR are both empty.. The following process is iterated:

1) Generate valid sequence for NEWGEN.

    a. If NEWGEN is empty, the algorithm stops;

    b. The algorithm generates neighbors for all solutions in NEWGEN and checks for their validity so that only the precedence-feasible neighbors are put into NEIGHBOR; if NEIGHBOR is not full.

Two methods are used to generate potential neighbors: 1-insertion and the partial pair-wise exchange. The former method simply picks up an activity and moves it to every possible slot one at a time; the latter generates neighbors by simply swapping two activities in the sequence.

2) Empty NEWGEN.

3) Look through sequences contained in NEIGHBOR and check whether they are "good" enough.

   a. Check each valid sequence in NEIGHBOR; if it is not in DEAD, assume it is "good".

   b. Calculate a schedule for a "good" sequence using the forward and reverse schedule generation method.

   c. The schedules with makespan lower than a certain threshold value are transported to NEWGEN for the next iteration and the others are moved into DEAD.

   It was observed that a small threshold value (for example, the best-so-far makespan) turns the algorithm into a greedy search and stops very soon, whereas a large-enough threshold value could lead to an enumerative search. Therefore, $C_{best} \times (\tau + 1)$ was used as the threshold value where C_{best} is the best-so-far makespan and the speed of search and optimality of the solution can be adjusted by factor τ.

4) Empty NEIGHBOR.

5) If no improvement has been made for a certain number of successive iterations, or a predefined maximum number of sequences have been explored, the algorithm stops; otherwise go to the first step and repeat the process.

6.4 Computational results

The proposed algorithm has been tested on J30 and J60 of the Project Scheduling Problem Library – PSPLIB [6]. J30 has 480 instances containing 30 activities. J60 has 480 instances containing 60 activities. The effectiveness of CLS is also compared with two published evolutionary algorithms EA1 and EA2 in the following table. %NBS stands for the percentage of the optimal solutions found by the corresponding algorithm and %ARPD is the average relative percentage deviation from the optimal solution.

ALGORITHM	%NBS		%ARPD	
	J30	J60	J30	J60
CLS	85.6%	70.2%	0.34%	3.8%
EA1	84.8%	72.5%	0.37%	1.23%
EA2	82.9%	71%	0.47%	1.23%

6.5 Applications in the FMM Demo system and future extensions

The proposed algorithm is initially designed from pure mathematical aspects to solve the classical RCPSP. In order to apply to the FMM Demo system, the algorithm was modified to take into account some more practical constraints. In the following items, “task” equals to “activity” in a project network.

1) The earliest start time of each task. Each task has a predefined earliest start time, which indicates it cannot start earlier than that time.

2) The latest end time of each task. Each task has a predefined latest end time, which indicates it must be finished before that time.

3) The deadline of the project. Each project has a predefined deadline, which indicates all the tasks in the project must be finished before the deadline.

4) The business hours. All tasks must be processed within the business hours excluding night hours, weekends and holidays.

The algorithm is implemented as a runnable and sychronized Java thread that can execute at four modes:

- Checking the original schedule. Find the quality of the initial schedule, especially the number and situation of conflicts in the schedule.

- Schedule Repairing. Given an unfeasible schedule, repairing can adjust the schedule by shifting as few tasks as possible. Only the tasks in conflicting time periods will be shifted. The resulted schedule repairing is a feasible schedule.

- Finding the first feasible schedule. For some time-critical situations, it is necessary to calculate a schedule as quickly as possible. The complete local search algorithm stops when the first feasible solution is found.

- Optimization. The complete local search tries to find the best schedule depending on predefined heuristics; for example, running with a certain number of iterations, population, time period, or the quality of the best solution.

At this current stage, the makespan was used as the only objective to evaluate a schedule. In the future, economic factors (such as hourly rate, setup cost, base cost) can be considered as the project proceeds.

7. System Configuration and Deployment

7.1 Server Configuration

Server Name: lonpc0558r.lon.ca

IP Address: 10.10.18.132

System: Microsoft Windows XP Professional Version 2002 Service Pack 3

Hardware: Intel Core 2 CPU 6600@2.40GHz, 3.00 GB of RAM

7.2 Software Configuration

Technology	Version	Installation Path	Comment
Java	JDK 1.6_0_03	C:\Program Files\Java\jdk1.6.0_03	Environment variable defined as: JAVA_HOME=C:\Program Files\Java\jdk1.6.0_03
MyEclipse Enterprise Workbench	7.5	C:\Program Files\Genuitec\MyEclipse 7.0	Workspace location: C:\Facility_maintenance_demo_project\Development\Code
Tomcat	6.0.14	C:\Facility_maintenance_demo_project\Deployment\tomcat6	Web app deployment path: C:\Facility_maintenance_demo_project\Deployment\tomcat6\webapps\Facility_Maintenance_Demo
MySQL	5.0.5	C:\Program Files\MySQL\MySQL Server 5.0	Data dictionary: C:\Program Files\MySQL\MySQL Server 5.0\Data\
MySQL Connector	5.0.4	WEB-APP-PATH/WEB-INF/lib/mysql-connector-java-5.0.4-bin.jar	
Hibernate	3	WEB-APP-PATH/WEB-INF/lib/hibernate3.jar, WEB-APP-PATH/WEB-INF/lib/hibernate-annotations.jar, WEB-APP-PATH/WEB-INF/lib/hibernate-commons-annotations.jar	WEB-APP-PATH is the root path of the Web app. In the system it is C:\Facility_maintenance_demo_project\Deployment\tomcat6\webapps\Facility_Maintenance_Demo
Struts	2.1.6	WEB-APP-	The library used by the system is

		PATH/WEB-INF/lib/struts2-core-2.1.6-nrc.jar	modified from the original version. Our version is named struts2-core-2.1.6-nrc.jar.
jQuery	1.3.2	C:\Facility_maintenance_demo_project\Deployment\tomcat6\webapps\Facility_Maintenance_Demo\js\jquery-1.3.2.min.js	
Ruby	1.8.6-26	C:\NRC_DND_Project\Deployment\ruby	Installed from One-Click Ruby Installer for Windows version 1.8.6-26. Ruby is used to support Gantt chart with redmine.
redmine	0.5.1	C:\Facility_maintenance_demo_project\Deployment\rails\redmine-0.5.1	This package is used to support Gantt chart.
iUI	0.20	WEB-APP-PATH/iphone/iui	Used to develop iPhone compatible interfaces.
Chrome	2.0	WEB-APP-PATH/js/chrome.js	Used to create animated menus.
Flot	0.5	WEB-APP-PATH/js/flot	Used to create graphical plots
Yahoo!UI	2.7.0	WEB-APP-PATH/yui	Used to enrich the interfaces.

8. User Interfaces

User interfaces are developed as Web pages using JSP technology. Multi-lingual support is provided since bilingualism is a mandatory requirement within the NRC. A distinguishing feature of this system is that multi-lingual support is mapped to a separate XML file so that adding languages or modifying terms does not require any change in code.

After the user selects the language and has logged in using his/her account credentials, a main page is shown (Figure 17).

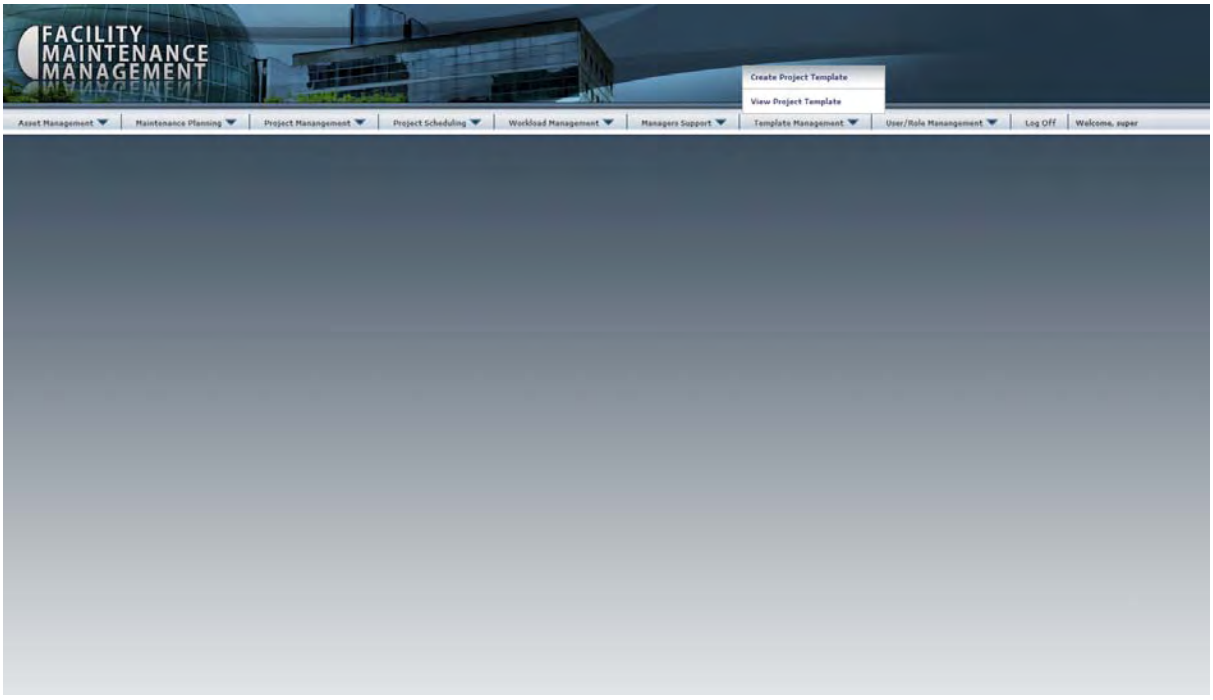
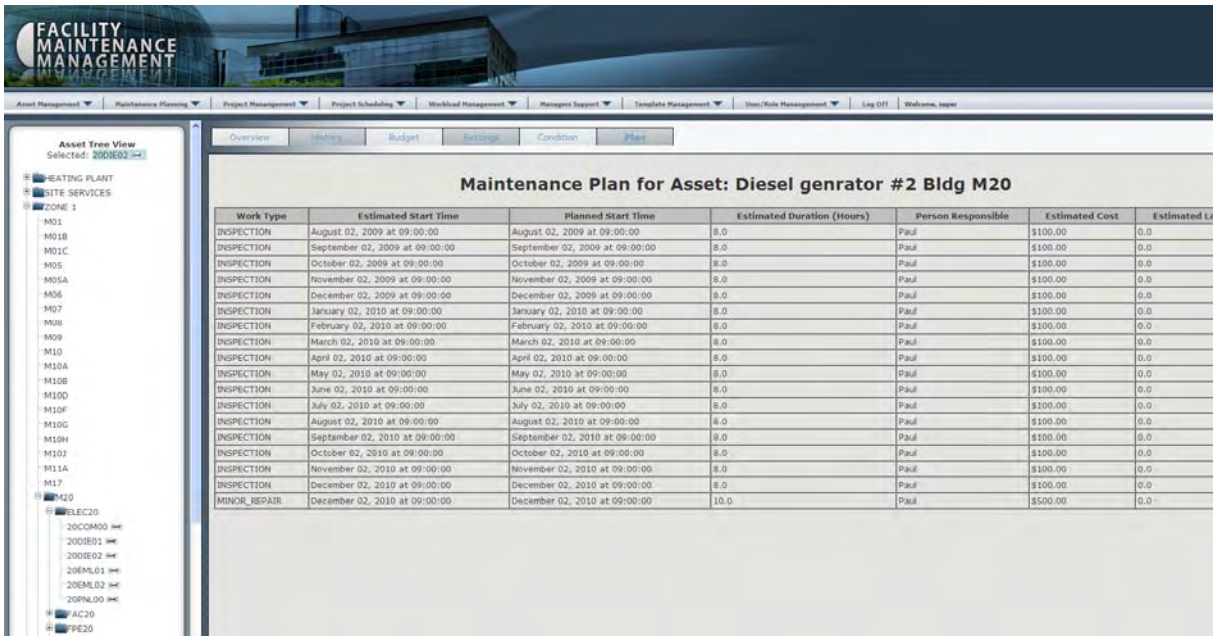


Figure 17: User interface – main page

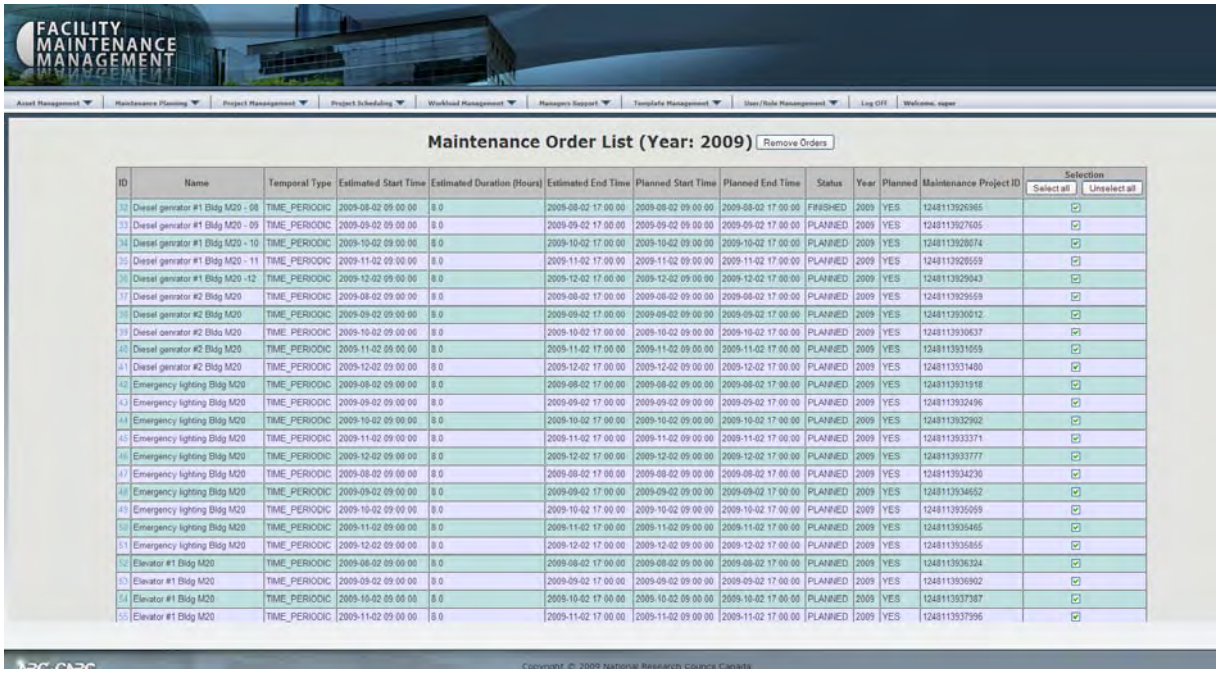
Asset management constitutes an asset browser showing an asset tree on the left and a tabbed view on the right. Each tab is implemented as a separate JSP page for populating the corresponding aspect of asset information on the screen. The asset management user interface with asset “ASPM-> ZONE1->M20->ELEC20->20DIE02” and the “Plan” tab opened is shown in Figure 18.



Work Type	Estimated Start Time	Planned Start Time	Estimated Duration (Hours)	Person Responsible	Estimated Cost	Estimated La
INSPECTION	August 02, 2009 at 09:00:00	August 02, 2009 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	September 02, 2009 at 09:00:00	September 02, 2009 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	October 02, 2009 at 09:00:00	October 02, 2009 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	November 02, 2009 at 09:00:00	November 02, 2009 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	December 02, 2009 at 09:00:00	December 02, 2009 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	January 02, 2010 at 09:00:00	January 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	February 02, 2010 at 09:00:00	February 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	March 02, 2010 at 09:00:00	March 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	April 02, 2010 at 09:00:00	April 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	May 02, 2010 at 09:00:00	May 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	June 02, 2010 at 09:00:00	June 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	July 02, 2010 at 09:00:00	July 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	August 02, 2010 at 09:00:00	August 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	September 02, 2010 at 09:00:00	September 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	October 02, 2010 at 09:00:00	October 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	November 02, 2010 at 09:00:00	November 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
INSPECTION	December 02, 2010 at 09:00:00	December 02, 2010 at 09:00:00	8.0	Paul	$100.00	0.0
MINOR_REPAIR	December 02, 2010 at 09:00:00	December 02, 2010 at 09:00:00	10.0	Paul	$500.00	0.0

Figure 18: User interface – asset browser

Maintenance order management includes two functions: “preventive maintenance planning” and “maintenance order browser”. Detailed processes and functional design is already described in Section 4.2. Figure 19 gives an idea of what these functions look like.



ID	Name	Temporal Type	Estimated Start Time	Estimated Duration (Hours)	Estimated End Time	Planned Start Time	Planned End Time	Status	Year	Planned	Maintenance Project ID
32	Diesel genrator #1 Bldg M20 - 08	TIME_PERIODIC	2009-08-02 09:00:00	8.0	2009-08-02 17:00:00	2009-08-02 09:00:00	2009-08-02 17:00:00	FINISHED	2009	YES	1248113926965
33	Diesel genrator #1 Bldg M20 - 09	TIME_PERIODIC	2009-09-02 09:00:00	8.0	2009-09-02 17:00:00	2009-09-02 09:00:00	2009-09-02 17:00:00	PLANNED	2009	YES	1248113927605
34	Diesel genrator #1 Bldg M20 - 10	TIME_PERIODIC	2009-10-02 09:00:00	8.0	2009-10-02 17:00:00	2009-10-02 09:00:00	2009-10-02 17:00:00	PLANNED	2009	YES	1248113928074
35	Diesel genrator #1 Bldg M20 - 11	TIME_PERIODIC	2009-11-02 09:00:00	8.0	2009-11-02 17:00:00	2009-11-02 09:00:00	2009-11-02 17:00:00	PLANNED	2009	YES	1248113928659
36	Diesel genrator #1 Bldg M20 -12	TIME_PERIODIC	2009-12-02 09:00:00	8.0	2009-12-02 17:00:00	2009-12-02 09:00:00	2009-12-02 17:00:00	PLANNED	2009	YES	1248113929043
37	Diesel genrator #2 Bldg M20	TIME_PERIODIC	2009-08-02 09:00:00	8.0	2009-08-02 17:00:00	2009-08-02 09:00:00	2009-08-02 17:00:00	PLANNED	2009	YES	1248113929659
38	Diesel genrator #2 Bldg M20	TIME_PERIODIC	2009-09-02 09:00:00	8.0	2009-09-02 17:00:00	2009-09-02 09:00:00	2009-09-02 17:00:00	PLANNED	2009	YES	1248113930012
39	Diesel genrator #2 Bldg M20	TIME_PERIODIC	2009-10-02 09:00:00	8.0	2009-10-02 17:00:00	2009-10-02 09:00:00	2009-10-02 17:00:00	PLANNED	2009	YES	1248113930637
40	Diesel genrator #2 Bldg M20	TIME_PERIODIC	2009-11-02 09:00:00	8.0	2009-11-02 17:00:00	2009-11-02 09:00:00	2009-11-02 17:00:00	PLANNED	2009	YES	1248113931059
41	Diesel genrator #2 Bldg M20	TIME_PERIODIC	2009-12-02 09:00:00	8.0	2009-12-02 17:00:00	2009-12-02 09:00:00	2009-12-02 17:00:00	PLANNED	2009	YES	1248113931480
42	Emergency lighting Bldg M20	TIME_PERIODIC	2009-08-02 09:00:00	8.0	2009-08-02 17:00:00	2009-08-02 09:00:00	2009-08-02 17:00:00	PLANNED	2009	YES	1248113931918
43	Emergency lighting Bldg M20	TIME_PERIODIC	2009-09-02 09:00:00	8.0	2009-09-02 17:00:00	2009-09-02 09:00:00	2009-09-02 17:00:00	PLANNED	2009	YES	1248113932496
44	Emergency lighting Bldg M20	TIME_PERIODIC	2009-10-02 09:00:00	8.0	2009-10-02 17:00:00	2009-10-02 09:00:00	2009-10-02 17:00:00	PLANNED	2009	YES	1248113932902
45	Emergency lighting Bldg M20	TIME_PERIODIC	2009-11-02 09:00:00	8.0	2009-11-02 17:00:00	2009-11-02 09:00:00	2009-11-02 17:00:00	PLANNED	2009	YES	1248113933371
46	Emergency lighting Bldg M20	TIME_PERIODIC	2009-12-02 09:00:00	8.0	2009-12-02 17:00:00	2009-12-02 09:00:00	2009-12-02 17:00:00	PLANNED	2009	YES	1248113933777
47	Emergency lighting Bldg M20	TIME_PERIODIC	2009-08-02 09:00:00	8.0	2009-08-02 17:00:00	2009-08-02 09:00:00	2009-08-02 17:00:00	PLANNED	2009	YES	1248113934230
48	Emergency lighting Bldg M20	TIME_PERIODIC	2009-09-02 09:00:00	8.0	2009-09-02 17:00:00	2009-09-02 09:00:00	2009-09-02 17:00:00	PLANNED	2009	YES	1248113934652
49	Emergency lighting Bldg M20	TIME_PERIODIC	2009-10-02 09:00:00	8.0	2009-10-02 17:00:00	2009-10-02 09:00:00	2009-10-02 17:00:00	PLANNED	2009	YES	1248113935059
50	Emergency lighting Bldg M20	TIME_PERIODIC	2009-11-02 09:00:00	8.0	2009-11-02 17:00:00	2009-11-02 09:00:00	2009-11-02 17:00:00	PLANNED	2009	YES	1248113935465
51	Emergency lighting Bldg M20	TIME_PERIODIC	2009-12-02 09:00:00	8.0	2009-12-02 17:00:00	2009-12-02 09:00:00	2009-12-02 17:00:00	PLANNED	2009	YES	1248113935855
52	Elevator #1 Bldg M20	TIME_PERIODIC	2009-08-02 09:00:00	8.0	2009-08-02 17:00:00	2009-08-02 09:00:00	2009-08-02 17:00:00	PLANNED	2009	YES	1248113936324
53	Elevator #1 Bldg M20	TIME_PERIODIC	2009-09-02 09:00:00	8.0	2009-09-02 17:00:00	2009-09-02 09:00:00	2009-09-02 17:00:00	PLANNED	2009	YES	1248113936902
54	Elevator #1 Bldg M20	TIME_PERIODIC	2009-10-02 09:00:00	8.0	2009-10-02 17:00:00	2009-10-02 09:00:00	2009-10-02 17:00:00	PLANNED	2009	YES	1248113937387
55	Elevator #1 Bldg M20	TIME_PERIODIC	2009-11-02 09:00:00	8.0	2009-11-02 17:00:00	2009-11-02 09:00:00	2009-11-02 17:00:00	PLANNED	2009	YES	1248113937996

Figure 19: User interface – preventive maintenance planning

Workload analysis is the real substance of this demo system. As the user goes through this function, he/she needs to do the following:

1) On the first page shown in Figure 20, the user selects a time period, selects maintenance orders and projects from the generated lists, selects the resource types, and presses the 'Continue' button at the bottom. On this page, all the lists are populated dynamically by the system based on the scope set up by the user. Javascript and Ajax are heavily used for this reason. For example, when the user clicks the "List Resources" button, the system only updates the pre-defined area on the page and populates only a list of resources for the selected resource types (Figure 21).

2) On the second page, the user can view and compare both the original schedule and the computed schedules calculated by the proposed algorithm, as described in Chapter 6. The user can check whether there are conflicts, judge the quality of the schedule, and select an option (mode) to run the optimization through the "general" tab (as shown in Figure 22); view the tasks' schedule and a graphical Gantt chart on the "task" tab (as shown in Figure 23); and see all individual resources, their utilization and graphical bar charts on the "resource" tab (as shown in Figure 24).

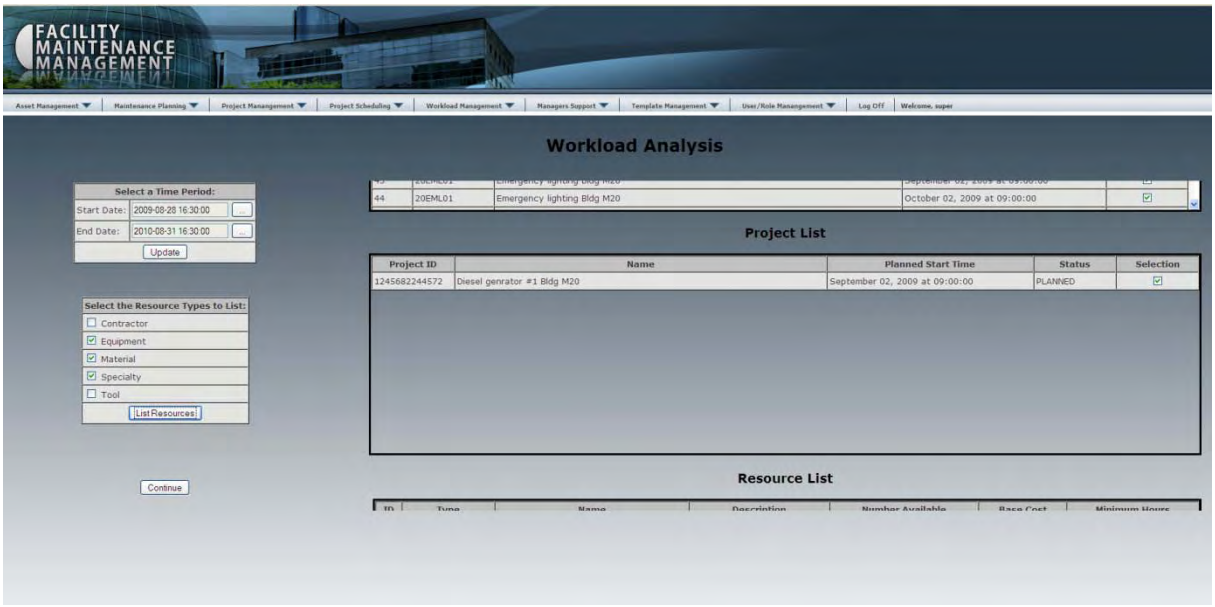


Figure 20: User interface – Workload analysis – select analysis scope

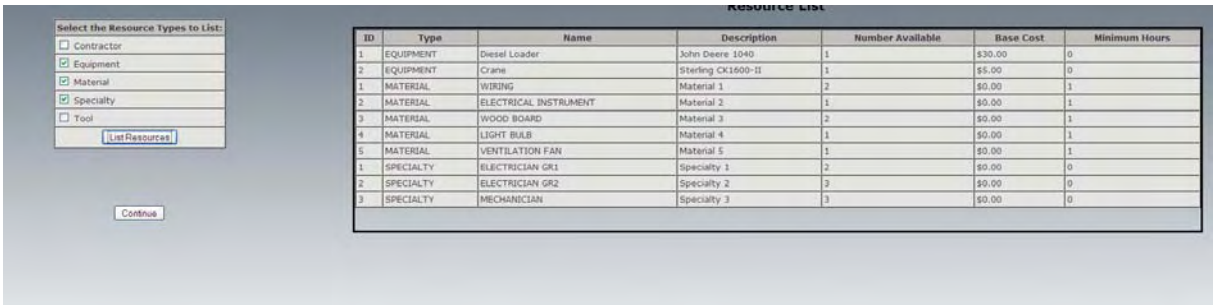


Figure 21: User interface – using Ajax to update only the resource list

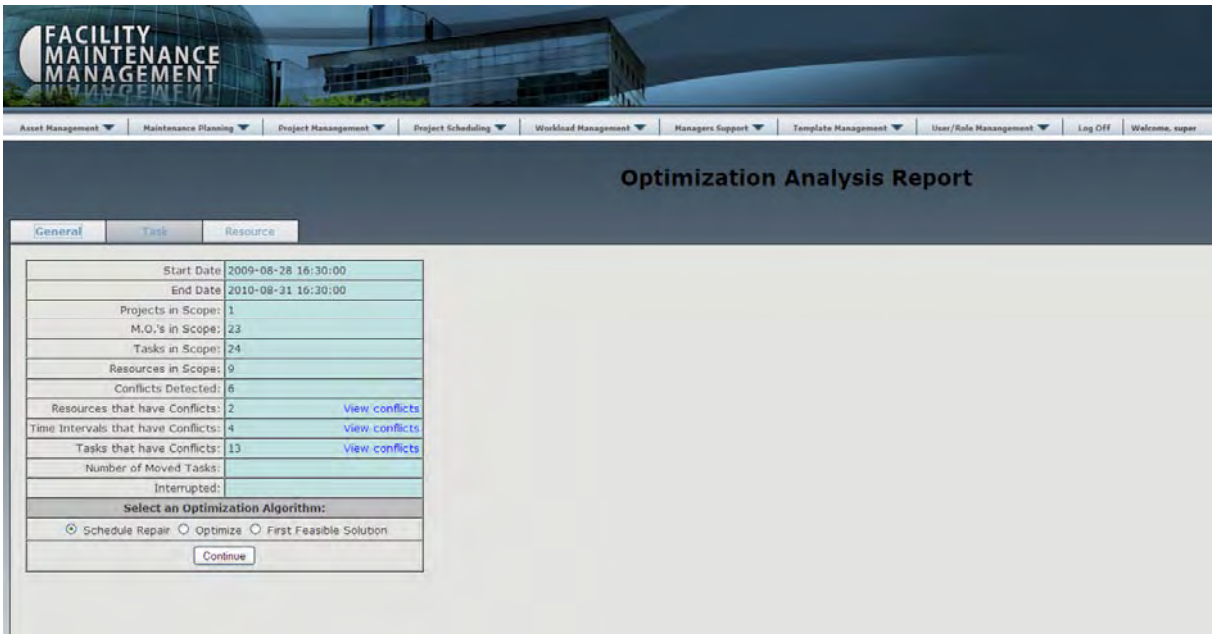
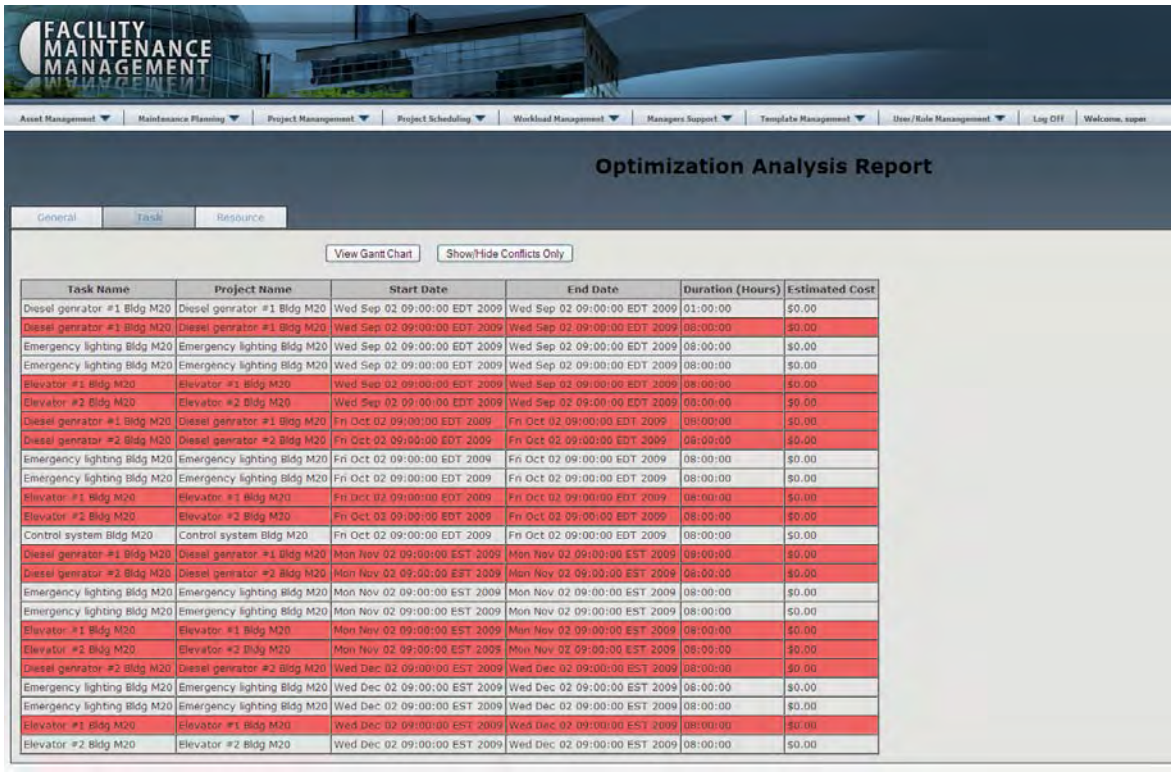
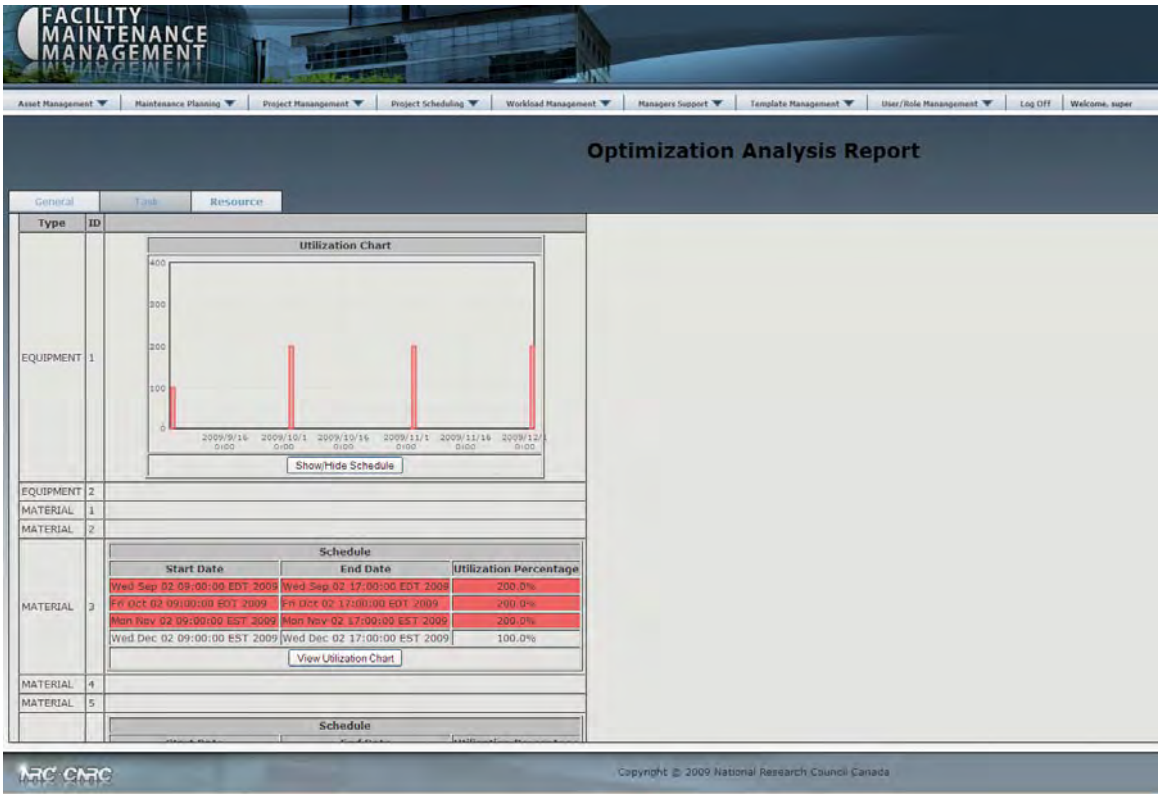


Figure 22: User interface – workload analysis – general tab



Task Name	Project Name	Start Date	End Date	Duration (Hours)	Estimated Cost
Diesel genrator #1 Bldg M20	Diesel genrator #1 Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	01:00:00	$0.00
Diesel genrator #1 Bldg M20	Diesel genrator #1 Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	08:00:00	$0.00
Elevator #1 Bldg M20	Elevator #1 Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	08:00:00	$0.00
Elevator #2 Bldg M20	Elevator #2 Bldg M20	Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 09:00:00 EDT 2009	08:00:00	$0.00
Diesel genrator #1 Bldg M20	Diesel genrator #1 Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Diesel genrator #2 Bldg M20	Diesel genrator #2 Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Elevator #1 Bldg M20	Elevator #1 Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Elevator #2 Bldg M20	Elevator #2 Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Control system Bldg M20	Control system Bldg M20	Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 09:00:00 EDT 2009	08:00:00	$0.00
Diesel genrator #1 Bldg M20	Diesel genrator #1 Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Diesel genrator #2 Bldg M20	Diesel genrator #2 Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Elevator #1 Bldg M20	Elevator #1 Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Elevator #2 Bldg M20	Elevator #2 Bldg M20	Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 09:00:00 EST 2009	08:00:00	$0.00
Diesel genrator #2 Bldg M20	Diesel genrator #2 Bldg M20	Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 09:00:00 EST 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 09:00:00 EST 2009	08:00:00	$0.00
Emergency lighting Bldg M20	Emergency lighting Bldg M20	Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 09:00:00 EST 2009	08:00:00	$0.00
Elevator #1 Bldg M20	Elevator #1 Bldg M20	Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 09:00:00 EST 2009	08:00:00	$0.00
Elevator #2 Bldg M20	Elevator #2 Bldg M20	Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 09:00:00 EST 2009	08:00:00	$0.00

Figure 23: User interface – workload analysis – task tab

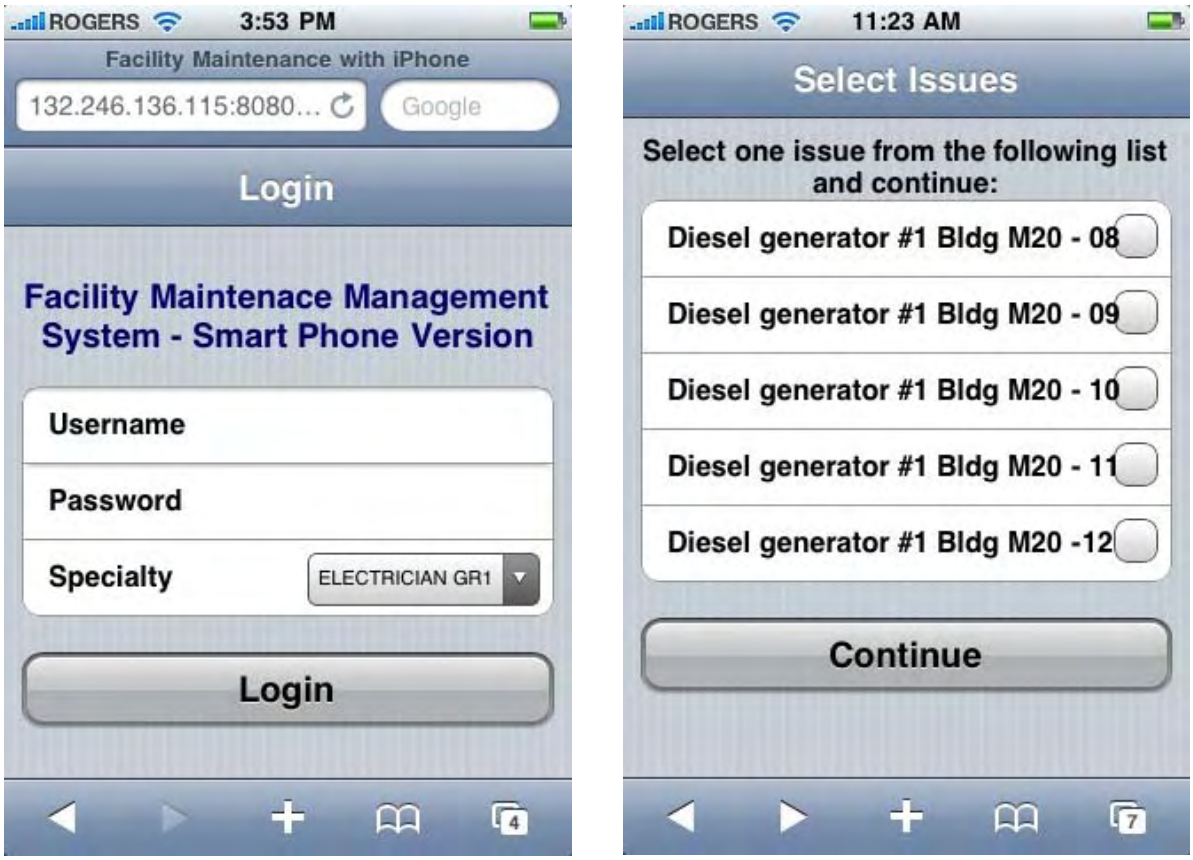


Type	ID
EQUIPMENT	1
EQUIPMENT	2
MATERIAL	1
MATERIAL	2
MATERIAL	3
MATERIAL	4
MATERIAL	5

Start Date	End Date	Utilization Percentage
Wed Sep 02 09:00:00 EDT 2009	Wed Sep 02 17:00:00 EDT 2009	200.0%
Fri Oct 02 09:00:00 EDT 2009	Fri Oct 02 17:00:00 EDT 2009	200.0%
Mon Nov 02 09:00:00 EST 2009	Mon Nov 02 17:00:00 EST 2009	200.0%
Wed Dec 02 09:00:00 EST 2009	Wed Dec 02 17:00:00 EST 2009	100.0%

Figure 24: User interface – workload analysis – resource tab

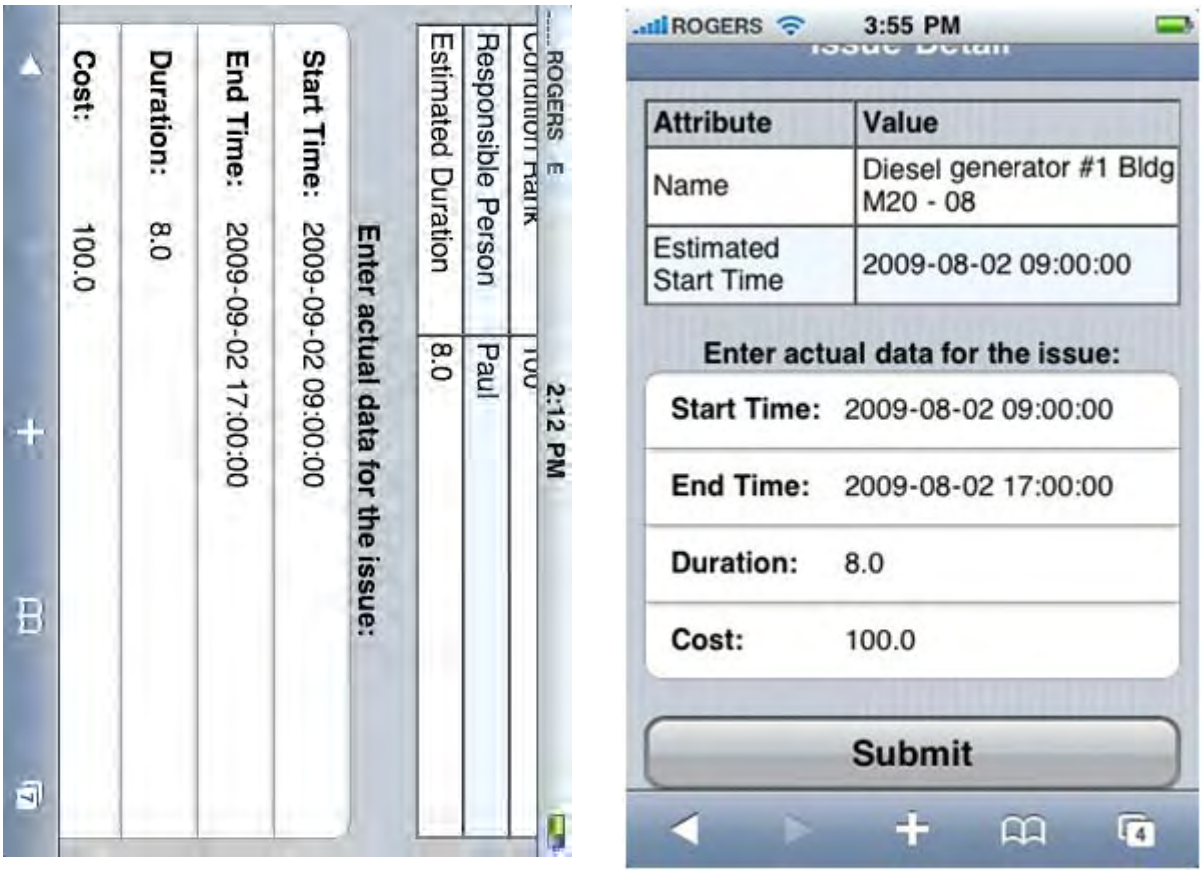
A set of user interfaces is provided, developed for the iPhone's Safari Web browser to enter actual information for maintenance orders at the job site. When the user accesses the system's website through a mobile device, the system can automatically recognize the type of device and display a specific set of pages developed for that device from the correct server directory. The first login page is shown in Figure 25-a and then the system will pop up the to-do maintenance list for the logged-in person according to his trade specialty (Figure 25-b). So workers from different trades will see different maintenance work lists.



(a) (b)

Figure 25: User interface – iPhone – login and work list

After the user selects a maintenance item from the list, he/she can then view the detailed description of the work (Figure 26-a) and after the work is done, enter the actual information of the work (Figure 26-b). Actual information is directly saved to the database and the status of the maintenance order is updated immediately.



(a) (b)

Figure 26: User interface – iPhone – work details and actual data recording

9. Future Consideration

The first prototype of the FMM Demo system was completed during the past four months, from May to August 2009, following a software design and planning stage from the beginning of the year. This report has documented the developed system in its requirements, system architecture, functional design, technical details, and a special algorithm proposed by the authors for advanced FMM workload optimization. As for an industrial case study, the NRC-ASPM facility maintenance practices [2] and processes were studied and the case was re-built in the developed FMM Demo system.

The distinguishing features of the FMM Demo system include:

- The optimization of maintenance schedules in three different modes: find a feasible solution through schedule repair; find a feasible solution through local search; and find an optimal solution through complete local search (CLS). This design originated from the viewpoint of practical system development.

- The scheduling and optimization of maintenance orders and maintenance projects is unified in the system based on projects and tasks. This feature offers greater potential for the system's application in general project management, as well as in construction.

- Multiple language support in XML is developed at a separate layer so that adding a language or modifying terms can be done without modifying any code.

- An optimal maintainable MVC system architecture is implemented with objects at different layers to maximize the modifiability, extensibility, and reusability of code.

As mentioned at the beginning of this report, the demo system could be used for both one's own R&D activities and for collaboration with partners. For future development, we can consider the following directions:

1. **Develop additional functions for strategic maintenance decisions, based on condition-based operations and maintenance management**

    - Deterioration models and condition assessment models for critical assets

    - Condition-based operations and maintenance processes

    - Annual asset budget allocation and optimization based on current conditions and prediction of future conditions

    - Performance measures for organizational assessment

2. **Systems integration with other data models or systems**

- Integration with RFID technologies, sensing devices/systems, and other local applications.

- Integration with geometric models and asset information contained in Building Information Models

- Integration of FMM with location maps in Geographical Information Systems

3. **Develop and demonstrate more case studies in facility maintenance management**

    - General purpose buildings and structures

    - Maintenance of the NRC building at 800 Collip Circle

    - Asset management of NRC London IT facilities

    - Maintenance of critical facilities such as hospitals and utilities

    - Maintenance of Urban Infrastructures

4. **Develop collaboration opportunities with universities and industrial partners**

References

[1] Qi Hao, Weiming Shen, Yunjiao Xue, Shuying Wang, Joseph Neelamkavil, Steve Kruithof, Brian Jones. Development of a Decision Support System for Aircraft Periodic Inspection and Maintenance: Technical Report. IRC Client Report. July 2008.

[2] Qi Hao, Weiming Shen, Brian Jones. Facility Maintenance Management Case Study: NRC-ASPM. IRC Research Report. To submit.

[3] Qi Hao, Yunjiao Xue, Shuying Wang and Weiming Shen, A dynamic scheduling algorithm for time- and resource-constrained task networks, IEEE 2009 International Conference on Systems, Man and Cybernetics, San Antonio, Texas, October 11-14.

[4] Jie Zhu, Xiaoping Li, Qi Hao, Weiming Shen. A New Approach for Resource-Constrained Multi-project Scheduling. Construction Research Congress 2010. May 8-11, 2010. Banff, Alberta.

[5] Qi Hao, Yunjiao Xue, Weiming Shen, Brian Jones, Jie Zhu. A Decision Support System for Integrating Corrective Maintenance, Preventive Maintenance, and Condition-based Maintenance. Research Congress 2010. May 8-11, 2010. Banff, Alberta.

[6] Project Scheduling Problem Library – PSPLIB. http://129.187.106.231/psplib/