Winner of the American Association for Artificial Intelligence's
"2005 Innovative Applications of AI Award" in the category of "Deployed Applications"

Scheduling Engineering Works for the MTR Corporation in Hong Kong

Andy Hon Wai Chun

City University of Hong Kong
Department of Computer Science
Tat Chee Avenue, Kowloon Tong
Hong Kong SAR
andy.chun@cityu.edu.hk

Dennis Wai Ming Yeung, Garbbie Pui Shan Lam

Synergicorp Limited
c/o City University of Hong Kong
Department of Computer Science
Tat Chee Avenue, Kowloon Tong
Hong Kong SAR
{dennis.yeung, garbbie.lam}
@synergicorp.com

Daniel Lai, Richard Keefe, Jerome Lam, Helena Chan

MTR Corporation Limited
MTR Tower, Telford Plaza
Kowloon Bay, Hong Kong SAR
{daniell, jerome, keefe, helenac}
@mtr.com.hk

Abstract

This paper describes a Hong Kong MTR Corporation subway project to enhance and extend the current Web-based Engineering Works and Traffic Information Management System (ETMS) with an intelligent "AI Engine." The challenge is to be able to fully and accurately encapsulate all the necessary domain and operation knowledge on subway engineering works and to be able to apply this knowledge in an efficient manner for both validation as well as scheduling. Since engineering works can only be performed a few hours each night, it is crucially important that the "AI Engine" maximizes the number of jobs done while ensuring operational safety and resource availability. Previously, all constraint/resource checking and scheduling decisions were made manually. The new AI approach streamlines the entire planning, scheduling and rescheduling process and extends the ETMS with intelligent abilities to (1) automatically detect potential conflicts as work requests are entered, (2) check all approved work schedules for any conflicts before execution, (3) generate weekly operational schedules, (4) repair schedules after changes and (5) generate quarterly schedules for planning. The AI Engine uses a rule representation combined with heuristic search and a genetic algorithm for scheduling. An iterative repair algorithm was used for dynamic rescheduling.

Task Description

The privatized MTR Corporation Limited owns and operates the MTR metro system of Hong Kong. The first subway line opened for service in 1979. Since then the network has expanded to 6 lines and 50 stations, with a new Disneyland Resort Line starting operations in the second half of 2005. The MTR currently carries 2.4 million passengers each weekday [1], making it one of the busiest in the world. For example, New York MTA's statistics [2] show roughly 14,000 passenger per station and 250,000 passengers per line daily.

Despite the large traffic-flow, the MTR has set for itself a very high service quality standard – train punctuality and delivery must be 99% and 99.5% on time respectively. In 2004, it achieved 99.7% and 99.9% respectively. Ensuring the required maintenance works are undertaken according to program assists the smooth running of the railway, and this task falls on the shoulders of a team of planners who decide what engineering or maintenance works need to get done, who to assign tasks to, what equipments to use and when to perform the work. This planning is done once a week in an Engineering Works Meeting (EWM). From this planning session, an allocation plan is produced three weeks prior to execution. During those three weeks, further amendments, changes and additional requests can be made, requiring the plan to be updated regularly.

Similar to the Paris Métro [3], Hong Kong's MTR service ends after midnight and resumes early morning next day. All the engineering works are performed during this "non-traffic hour" (NTH) when no passenger trains are running. Each night, there is only a precious 4 to 5-hour window to perform all the necessary engineering works and repairs. Obviously, it is crucial that the NTH window is used wisely and efficiently.

Previous Manual Approach

Any type of work that needs to be performed during the NTH will need to be approved and scheduled during the EWM. Different parties will make requests by submitting a "Possession Request" form via a Web-based application called the Engineering Works and Traffic Information Management System (ETMS). The ETMS collects all the requests and prints a hardcopy report that is used during the EWM for the week being scheduled.

During the EWM, the planners determine which tasks are more urgent or have higher priority and allocate them one at a time while considering all the appropriate

operational and safety rules, constraints and guidelines. An example of a safety rule is “pedestrian works are not allowed within 600m from possession boundary of energized possessions.” There are roughly 60 to 70 similar rules and constraints that must be considered each time. Understandably, the EWM is a very knowledge-intensive meeting, as it is utmost critical that no safety rule or regulation is overlooked.

At the same time as the allocation plan is being formulated, resource assignments are also tentatively made to ensure there are adequate personnel, train operators, engineering locomotives and wagons to meet the assigned workload as well as available at the requested depots.

For any given week, there will not be enough time slots or resources to accommodate all the possession requests that would have been made. Therefore, as part of the EWM work, the planners must decide which jobs are more time critical and need to be performed first. For those that cannot be assigned, usually because of either safety rule or resource conflicts, the EWM team then tries to “squeeze them in” by “combining” them with already allocated jobs.

Combining two or more requests, potentially from different parties, may save both personnel and equipment resources. This allows jobs that cannot be allocated before to be allocatable now. There is an entire set of rules that govern when and how “combines” can be made. For example, requests being combined must be nearby each other and their engineering trains, if any, should ideally come from and return to the same depot. The “combine” operation not only allows more possession requests to be satisfied, it also opens the possibility of reducing resource needs, as locomotives and train operators can potentially be shared and only 1 EPIC (Engineer Person-In-Charge) is needed per combine. Most importantly, by combining two requests that are close to each other, some safety protection conflicts can be resolved.

Once all the allocations have been made during the EWM, the final decisions and approved allocation plan are then manually entered back into the ETMS. Once a request has been approved by the EWM, additional details are then filled in, such as train paths and movements. The system also keeps track of any changes or last-minute requests that need to be slotted in. Any such changes or adjustments made after the EWM must be resolved and rescheduled separately.

Because of the complexity of knowledge involved in the EWM and the potential consequences if any safety rules were overlooked, it makes sense to use AI to streamline the entire process.

New AI Approach

The previous manual approach is quite tedious and time-consuming, involving careful validation and negotiation between staff of different responsibilities and departments. Requests may need to be changed and refined several times before they can finally be scheduled.

In early 2004, MTR decided to work with the City University of Hong Kong (CityU) to review their current scheduling processes and investigate how AI technologies might be used to help streamline their workflows, maximize resource utilization and minimize errors.

Application Description

To meet the project objectives, we designed and built an AI Engine that can be used to streamline MTR’s business processes for scheduling. Since the original ETMS application was already in production, we decided to use a service-oriented architecture (SOA) and expose the new AI functionality as Web services for existing applications to use. This approach allowed us to effortlessly enhance the existing application with AI capabilities without much, if any, impact to existing daily operations.

Besides the new AI services, additional screens had to be built to display Gantt Charts and subway engineering maps. These screens had to be interactive to take advantage of the interactive problem-solving capabilities of the AI Engine. We decided to use SVG technology as it was standards based and more flexible. In addition, we wanted to use XML as the common language for all communications and data exchanges within the system.

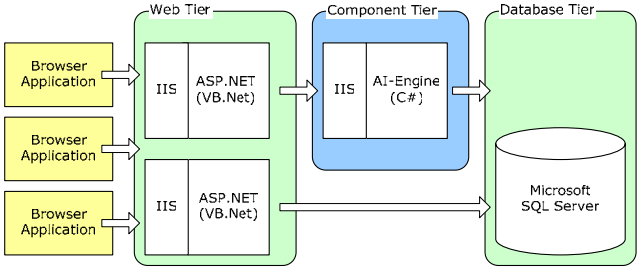

Figure 1. Overall system architecture.

Figure 1 shows our architecture. The AI Engine resides in a dedicated application server for better performance and reliability. It interfaces with the existing ETMS Web servers using Web-Service/.NET Remoting. The existing ETMS application was extended to n-tiered SOA for screens that needed AI.

Application Use Cases

The AI-enhanced ETMS is used in many different situations. Firstly, when possession requests are first entered, the AI Engine ensures each request is valid and that there will be enough resources for the request. This

eliminates time wasted in making amendments later as well as allows the requester to decide whether or not to continue with his/her request if there are not enough resources. With this information, the requester can immediately adjust his/her request to maximize the chances of allocation and thus reduce time wasted due to a simple mismatch between resource availability and requested work schedule.

Secondly, after all the requests have been entered and just before the EWM, the AI Engine automatically generates a schedule or allocation plan. This plan provides an opportunity to resolve minor issues off-line prior to the EWM. The final plan will then be reviewed during EWM for approval by the planning team, see Figure 2.

Thirdly, as changes come in, the AI Engine is used to validate the changes. If a change causes conflicts, the AI Engine will automatically resolve these conflicts and propose schedule changes for the human planner to approve.

Lastly, the AI Engine is used for long-term quarterly planning. Technically, it works similar to weekly planning except that the time span is a quarter instead of a week. The actual number of requests need not be proportionally larger as quarterly planning deals only with larger projects and higher priority jobs.

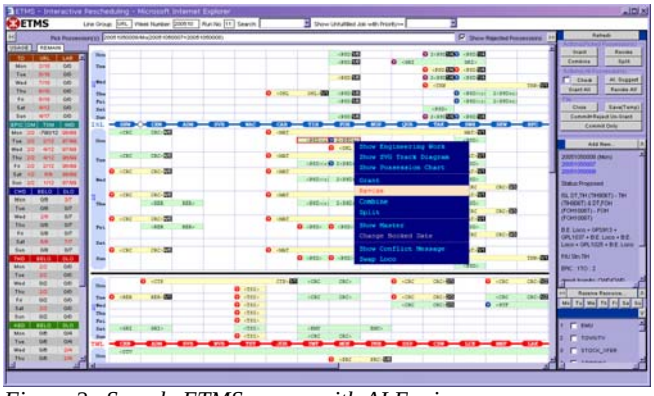

Figure 2. Sample ETMS screen with AI Engine.

Domain Knowledge

The AI Engine encodes several types of knowledge regarding how engineering works should be assigned:

- **Track Possessions**
  Engineering work requests using engineering trains are called "possession" requests as engineering works require the possession of a segment of tracks for dedicated use by those engineering tasks. There is a set of rules to ensure that adequate length of possession is requested for different types of jobs. For example, possession booking must be from station to station if the engineering task involves an electric train that must be run. Or for the case of rail grinding, possession must be taken of all tracks between stations/landmarks on both sides as well as adjacent to worksite. Obviously, track possessions requested by different jobs cannot overlap.

- **Train Protection**
  There are also rules regarding the length of tracks that must be reserved between two neighboring protections to be used as a safety buffer. This "protection zone" is defined for both ends of a possession request. For example, energized possessions require 200m for protection, while non-energized possessions require 100m for protection. The rules may further differ for different types of neighboring work. For example, a pedestrian access (PA) work next to an energized work will require at least 600m protection. Related to train protections are rules governing the placement of track circuit operating clips and red flashing lights.

- **Train Consists**
  Along with each possession request is the request for an engineering train that might be needed to support that engineering work. There are rules governing how the engineering train can be formed from different types of locomotives and wagons. The details of how a train is to be formed is called the "train consist," which is basically a patterns of how locomotives and wagons can be used to form a train plus additional constraints. For example, having a battery electric locomotives at both ends of a train consists can at most support 3 wagons. Another example is that some wagons need to be coupled together. The rules may be different for different types of locomotives.

- **Train Availability**
  Besides ensuring there is no conflict in track possessions and that the engineering train is of proper formation, the system has to also ensure that the requested locomotives and wagons are indeed available and that they will be located at the requested depot at the night of the job. Engineering train movements during the day must therefore be recorded as well.

- **Personnel Availability**
  Our system also keeps track of personnel resources. This includes making sure there are adequate train operators (TO) and supervisors or "engineer's person-in-charge" (EPIC) for each job. There are rules governing how TOs and EPICs are assigned, making sure they have adequate skills for the job as well as adequate number of available staff.

- **Combining Requests**
  When resources are not adequate to support all engineering work requests, some jobs may be "combined" together to allow multiple non-conflicting jobs to be performed at the same vicinity with some potential for resource savings. There are rules on how and whether track possessions may be combined as well as train consists. In some cases, the same locomotive and train operator may be used for more than one job.

- **Operational Heuristics**
  Besides these general rules and guidelines, there is also a set of very specific rules that relate to particular scenarios of operational needs. These rules are called "operational heuristics." For example, there is a heuristic that deals with how particular set of sidings must be reserved for overnight use to park trains.

Uses of AI Technology

Several different AI techniques were used in our AI Engine implementation. To represent the operational rules and regulations used in validation, we used a rule based approach. To perform weekly/quarterly scheduling, we created a heuristic search algorithm that is combined with a GA for optimization. To ensure non-stop processing, we coded a "self-healing" mechanism within the AI Engine that provides fixes to data problems on-the-fly. We also have an iterative repair algorithm for rescheduling.

The Rule Engine

One of our design objectives is easy of maintainability – to ensure that rules can be maintained easily by any developer and that the rule engine implementation should be separated from the domain objects. To achieve this, we took a "non-intrusive" approach to AI where the rule base resides as XML data and is totally separated from the domain objects that are coded in C#. Automatic code-generation techniques were then used to dynamically generate the rule engine library (as DLL) from XML. This is made possible with .NET object reflection and attribute-based programming.

To represent the rules in XML, we must first select or design an XML-based markup language. User interface validation rules, such as [7, 8, 9] can only represent simple types of value validation, such as checking if an input is within a particular range of values. On the other hand, there is also a body of XML markup languages to represent AI rules. For example, there is a rule-engine neutral Simple Rule Markup Language (SRML) that can represent common language constructs to support forward-chaining rule engines. SRML is a relatively higher abstract-level markup that can easily be read by a programmer. Drools [17], an open source Java rule engine, also has a high-level markup language called DRL.

The most widely used XML markup for rules is probably RuleML [12] from the Rule Markup Initiative. It permits both forward and backward-chaining rules for deduction, rewriting, and further inferential-transformational tasks. Several rule-engines, especially open source ones, such as jDrew [13] and Mandarax [14] supports RuleML.

Unfortunately, RuleML was designed to be processed by a computer and not really to be used directly by a programmer encoding knowledge. For this project, we have designed a markup that is midway between RuleML and higher-level representations like SRML.

Our rules are represented in W3C's RDF/XML [18] format. Since our representation is designed to be editable by both a programmer or via a GUI rule-editor, our XML Schema was designed so that the rules can easily be comprehended by a programmer. The RDF/XML syntax was selected so that programmers have the flexibility of using either RDF or XML editors to process the rules.

```
<ai:Rule>
  <ai:RuleStructure
      ai:ruleset="Combines"
      ai:salience="high priority"
      ai:direction="forward">
    <dc:title>No Combine for Energized Possession</dc:title>
    <dc:identifier>K1_R1</dc:identifier>
    <dc:subject>ETMS</dc:subject>
    <dc:description>
      Energized possessions may not be combined.
    </dc:description>
    <dc:date>2004-05-10</dc:date>
    <ai:Antecedent>
      <ai:And>
        <ai:Object ai:var="r" ai:type="Request"/>
        <ai:Condition ai:check="r.IsEnergized"/>
      </ai:And>
    </ai:Antecedent>
    <ai:Consequent>
      <ai:And>
        <ai:Variable ai:var="requests" ai:key="r.CombinedRequestList"/>
        <ai:OrValidate>
          <ai:OrValidateStructure>
            <ai:Validate ai:check="requests==null"/>
            <ai:Validate ai:check="r.IsAllEnergizedCombine"/>
          </ai:OrValidateStructure>
        </ai:OrValidate>
      </ai:And>
    </ai:Consequent>
    <ai:Message
      ai:format="Cannot combine energized possession {0}"
      ai:arg0="requests"/>
    <ai:Remark
      ai:format="Cannot combine energized possession with others,
unless others are also energized. Conflicts: {0}"
      ai:arg0="requests"/>
  </ai:RuleStructure>
</ai:Rule>
```

The above example illustrates the syntax of our AI rules. (The dc: namespace is the Dublin Core [19].) Once the rules are encoded into our RDF/XML format, we use code generation techniques to automatically create a .NET DLL from the RDF/XML rule file that can be called directly from the application server without any additional rule-related coding. All necessary information needed to compile the rules are extracted from domain objects via object reflection; the DLL was then generated dynamic. The GUI rule-editor uses .NET attributes, i.e. metadata annotated on the business objects, to create the editing environment for the domain.

The Scheduling Algorithm

The rule engine is used to valid individual requests as well as to guide the scheduling algorithm. The scheduling algorithm was originally designed as a genetic algorithm (GA) [10]. This was later extended with heuristic search. The reason was that although GA produced an optimized solution, it was hard for humans to comprehend the rationale behind the evolved solution. GA is driven mainly by its fitness function, a complex formula that embodies all

the necessary knowledge to evaluate the quality of a plan. The human planners found it hard and time consuming to interpret the meaning behind differences in fitness values.

The final approach we took was to use heuristic search for higher priority jobs and then use GA to optimize the lower priority ones. This combined the best of both worlds. The heuristic search was coded with the same heuristics used by the human planners so results were easy to comprehend. The heuristics included scheduling jobs chronologically and according to their priority, while "combines" are done separately and afterwards. GA was then used to optimize on lower priority jobs which are harder to schedule with heuristics alone as many different options must be tried.

This scheduling algorithm is used for both weekly and quarterly scheduling. It determines which requests to allocate, which days to allocate these requests to, and how to combine requests if there is not enough resources. Finally, the scheduling algorithm determines locations to place track circuit operating clips and red flashing lights for protection.

We compared results from our scheduling algorithm with historical human-generated schedules prior to AI Engine deployment. The results were quite interesting. Firstly, out of the 21 weeks of data that we compared, there were on average 5 to 10 hard rule violations daily for the human-generated schedule; rules were divided into hard, medium and soft rules. The AI Engine-generated schedules, of course, did not contain any hard rule violations. After analyzing the results, it turns out that some of the hard rule violations were due to human data entry errors, while others were real errors that were fixed in later processes after scheduling and prior to execution. With the deployment of the AI Engine, we ensure that these types of human errors were caught upstream in the workflow.

We also found that our approach produced schedules that had similar, if not better than, "fitness" scores compared to human schedules. The fitness was measured using a following formula that maximizes on the percentage of high priority jobs that are allocated, while minimizing rule violations. The formulae also had a "combine penalty" component to define different degrees of preference for different types of "combines." For example, combining requests from the same line or that result in resource savings has higher preferences.

The AI Engine not only mimics what human planners do today, it also opens up new functionality that were not considered before. For example, users can now specify a range of days to be allocated, such as "2 days out of 5 days." Users can also specify preferences for certain scheduling actions, such as which other requests it would like to be combined with if possible.

To further assist the human planners in understanding the AI Engine-generated schedule, explanations are automatically generated by the AI Engine for requests that cannot be allocated. These explanations justify why those requests were not allocated, for example listing rules that would have been violated or resources that were unavailable.

The Self-Healing Mechanism

The ability to perform "self-healing" is important for any mission critical application [15]. The main objective for self-healing code is to allow the system to dynamically heal itself when minor inconsistencies are found and continue processing with possibly some graceful degradation in results. Without self-healing, the system might just throw an exception and exit. In our case, the self-healing mechanism handles different types of potential data inconsistencies. These data inconsistencies are then logged so that similar problems might be avoided in the future.

It is particularly important for ETMS as data input for the AI Engine comes from many different sources – the user, the user interface, database tables, historical data, and other software components. It is unavoidable that some data inconsistencies may occur from time to time, while software or database tables are being upgraded or improved. Self-healing is used in several types of situations in the AI Engine such as illogical parameter values, incorrect data items, or missing information. We found self-healing mechanisms to be extremely useful, especially in transient situations where software is being updated and different components are being integrated.

The Iterative Repair Algorithm

The iterative repair algorithm [11, 24, 25] is used during interactive rescheduling to handle changes and modifications. The main objective of the algorithm is to determine if a change can be accommodated without any impact to the existing schedule. If not, it then tries to find a way to satisfy the request through various different types of "combine" operations. This algorithm is an "iterative repair" algorithm as it iteratively finds problems and repairs the schedule. It uses heuristics similar to those used by the weekly and quarterly scheduling. The same set of validation rules are used as well. The performance of the iterative repair algorithm is fast enough (within seconds to tens of seconds) to be used during the EWM for interactive problem solving and rescheduling.

Application Use and Payoff

The AI Engine has been in daily use since July 2004. Practically all reports for engineering work requests now contains rule violation warning messages, if rules were violated, and summaries of resource usages as computed

by the AI Engine. In particular, hard rule violations are highlighted. Experiments and end user feedback tells us that schedules generated by AI Engine are very similar to those generated by humans.

There are numerous benefits to extending the ETMS with an AI Engine. Some of the key payoffs include:

- **Improved productivity –** with automated AI scheduling, human planners can focus on resolving difficult operational problems and resource contentions that requires human negotiation
- **Ensured operational safety** – the AI Engine eliminates any potentials for human errors
- **Maximized resource utilization** – different combinations are explored to maximize utilization
- **Streamlined workflows** – the human aspect of scheduling is streamlined for efficiency
- **Streamlined decision making & problem solving** – changes/modifications are automatically resolved
- **Improved long-term quarterly planning** – planning is now more flexible and dynamic
- **Improved Quality of Service –** by providing better schedules for engineering works, the quality of service provided during the day for passenger trains will be improved.

Application Development and Deployment

This project began in early 2004 and involved several different IT enhancements to the existing ETMS application. CityU was responsible for identifying the software requirements and performing the architecture and software design. Implementation was then performed by different development teams in parallel – an MTR team handles all database enhancements, a third party handles the client-side ASP.NET and SVG development, while CityU coded the AI application server and Web services. Total development team size was roughly a dozen people.

With several development parties involved, all located in different places, it was important to have a loosely-coupled architecture that simplifies development and minimizes any potential integration problem. A Service-oriented Architecture (SOA) was selected with data exchanges between modules through standard XML messages.

To further ensure that all the development teams were in-sync at all times and that requirements were in-line with user expectations, we adopted an agile development methodology with fine-grained iterations and releases, each with incremental functionality.

Business Process Re-engineering

The project began with an extensive user requirements study and business process re-engineering (BPR). Since the final application will be used by different types of end users and at different phases of the scheduling process, we had to interview key staff members from different departments within MTR to thoroughly understand their specific needs and the potential impact any changes to workflows might have on their work. By April 2004, we have designed new workflows that leveraged upon new features to be provided by the AI Engine. Initial screen designs and API interfaces were also made.

Knowledge Engineering

At the same time as BPR, knowledge engineering was performed to define the exact set of rules to be coded into the AI Engine. The knowledge engineering process was simplified by the fact that detailed operational rules and regulations were already readily available. However, there were still tacit undocumented knowledge as well as "grey areas" that needed to be clarified. Once the final set of rules were mutually agreed upon, they were easily translated and coded into our RDF/XML rule syntax. Since we follow agile test-driven development, test cases and test data were created for each rule. With hundreds of test cases in hand, we were able to precisely determine project progress at any time by the number of NUnit [16] "green lights" (passed test cases).

The initial development involved business object modeling, design and C# coding as well as mapping business objects to MS SQL Server tables in the MTR database. A highly decoupled "non-instructive AI" approach was taken where the rule engine interacts with the business objects without rule engine-specific changes to object coding. This is done through automatic code generation from RDF/XML using object reflection that produces the rule engine as a separate DLL.

Just like any other modern software project, we made extensive use of open source tools to accelerate our development effort. For example, we used #develop for IDE, NUnit [16] for unit testing, NDoc [20] for documentation, Log4Net [21] for event logging, NAnt [22] for software building, and Subversion [23] for source code management.

First Release

Using this approach, we were able to quickly produce our first release in June 2004, which included important application features that immediately helped streamline and simplify some of the MTR workflows. This release included the rule engine that was used firstly to validate all newly entered possession requests to ensure they followed all the necessary rules and regulations. Previously, there was no way of knowing if there was a problem until the request was reviewed by managers during the weekly EWM scheduling process. By then, a lot of time would have been wasted. Now, problems can be rectified immediately during initial data entry.

In addition, the rule engine also verifies whether or not the newly entered request conflicts with any prior approved or yet-to-be approved requests and what priority levels those requests are in. With this information, the party making the request will have a fairly good idea of the chances that his/her request will be granted or not. This information will also guide the user in modifying the request so that it will have a higher chance of approval, for example, changing the work request to another day that does not have any conflict. Alternatively, the user may decide to negotiate with the conflicting party to mutually resolve the problem. In any case, this allows the human aspect of problem solving to start immediately instead of waiting for the weekly EWM. Although AI cannot replace the human aspects of scheduling, in our case, it helps guide and streamline this time-consuming process.

Second Release

A second release of the AI Engine was made in October 2004 with the addition of automatic weekly scheduling capabilities. The scheduling algorithm combined heuristic search with genetic algorithm to automatically produce an allocation plan, within a few minutes. This plan is generated and circulated to all the managers a day prior to the EWM so that minor issues and simple conflicts can be resolved prior to the weekly meeting. By resolving trivial problems before the meeting, the planning team can then be more productive and focus on discussing and resolving more complex problems, thus reducing the amount of time needed for the EWM.

This second release also contained an iterative repair algorithm to reschedule requests after changes were made. Since plans are produced 3 weeks ahead of time, there may inevitably be changes to the requests as well as last-minute additional urgent work requests. The iterative repair algorithm detects conflicts and resolves problems due to these changes in tens of seconds.

Third Release

A third release of the AI Engine was made in November 2004 to support quarterly planning. Since many different parties, some external to MTR, will be performing work in the subway, they all need to have an overview of the coming month's committed schedule to help them formulate their own schedule. The task of quarterly planning is to produce a 3-month plan that includes long-term projects or high-priority jobs that must be performed, such as construction work. The quarterly plan allowed more difficult issues to be resolved first before the weekly scheduling. Given the quarterly plan, others can then try to work their own work request schedule around it.

The challenge in creating the quarterly plan was to produce an algorithm that was able to handle such as large search space within reasonable time. We solved this by partitioning the search space into regions that can be solved independently. This partitioning makes use of the fact that some possession requests are recurrent within a quarter.

Fourth Release

A fourth release of the AI Engine was made in April 2005 to support additional rules and operational requirements needed to support the new Disneyland Resort Line that connects the current subway system to the soon to open Hong Kong Disneyland.

Future Plans

The ETMS is an ongoing MTR strategic project. New enhancements and software features will be added in the future to support the evolving business requirement. Potential future AI enhancements include adding train path planning to the engineering work scheduling process. Train path planning is currently performed manually after the request allocation plan has been finalized and just before execution. The task of train path planning is to find an optimized train path for all the engineering trains. The path must take the train to their designated work area before the nightly power shutdown and without any conflicts with other engineering trains. However, this is also a very tedious task requiring a lot of domain knowledge on MTR-specific train operations and railroad network structure. This is also a good candidate for AI automation. In addition, crew rostering is also being considered as candidates for future enhancements.

Maintenance

Just like any other mission critical software, there will inevitable be changes and upgrades to the AI Engine after deployment to reflect business and operational changes in MTR. The architecture design and technology selected for the AI Engine makes it easy to maintain. Firstly, the client-side is Web-based and any required plug-in is downloaded as needed, such as the SVG viewer. The data used by the AI Engine is stored in MTR's own MS SQL Server database and maintained by the MTR IT team. The operation of the AI Engine is parameter-driven to reduce the need for maintenance. Exposing the AI Engine as Web services also helps reduce maintenance and integration needs.

The only potential need for maintenance is the rule base, which will probably not change much, unless when there is a new line or station. In any case, the rule base was designed to be maintainable by anyone with basic RDF/XML knowledge and an understanding of our simple rule syntax. Changes to the rule base to reflect changes in operational needs can be easily done without any source code modifications to the AI Engine itself. This is done

through our "non-destructive" AI approach as explained earlier.

For the ETMS, MTR's IT team provides front-line technical and end-user support while we provide additional assistance on the AI Engine whenever requested. In addition to maintenance, CityU and its partners provide continued enhancement and consulting services to MTR on the AI Engine.

Conclusion

This paper is an overview of our AI project to enhance Hong Kong MTR's Web-based subway engineering work processing software with AI capabilities and automated scheduling. Through the use of AI techniques, we were able to help the subway streamline their scheduling/rescheduling processes and maximize their resource utilization, while providing early identification of potential violations of safety and operational regulations and guidelines for all scheduled engineering works. In addition, valuable domain knowledge and expertise related to these regulations and guidelines are now quantified, coded and preserved within the organization, for use by this and other systems. Experiments showed that the AI-generated schedules were comparable in quality with human-generated schedules while eliminating all errors and conflicts. Our AI Engine also makes use of several innovative techniques, such as a non-intrusive XML rule-engine, intelligent self-healing coding, and combining heuristic search with genetic algorithm. This is probably the first AI system to be deployed in Asia Pacific that uses these innovative techniques as well as a modern service-oriented architecture.

Acknowledgements

The authors would like to thank the MTR Corporation Limited for providing us with an opportunity to participate in this exciting project. In particular, we would like to thank Richard Keefe, our key user and knowledge expert and Jerome Lam, who heads the IT project team.

The research described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 9040517, CityU 1109/00E) and a grant from the City University of Hong Kong (Project No. 7001286).

References

[1] *MTR Corporation* (n.d.), "Patronage – Monthly Total," http://www.mtr.com.hk/eng/investors/pad.htm

[2] *Metropolitan Transportation Authority* (n.d.), "The MTA Network," http://www.mta.nyc.ny.us/mta/network.htm

[3] *RATP* (n.d.) http://www.ratp.fr/

[4] Said Tabet, Prabhakar Bhogaraju, David Ash, "Using XML as a Language Interface for AI Applications," PRICAI Workshops 2000, Springer-Verlag Heidelberg, 2000, pp.103-110.

[5] *MindBox* (n.d.), http://www.mindbox.com/

[6] Venugopalan, Vivek, "User Interface Validator Pattern," *TheServerSide.com*, February 17, 2002, http://www.theserverside.com/patterns/thread.tss?thread_id=11947#40970

[7] *Commons Validator* (n.d.), The Jakarta Project, http://jakarta.apache.org/commons/validator/project-info.html

[8] *Introduction to Validating User Input in Web Forms* (n.d.), MSDN, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontovalidatinguserinputinwebforms.asp

[9] *Struts Validator Guide* (n.d.), The Apache Software Foundation, http://jakarta.apache.org/struts/userGuide/dev_validator.html

[10] Holland, J. H., *Adaptation in natural and artificial systems*, Ann Arbor: The University of Michigan Press, 1975.

[11] Zweben, M., Daun, B., Davis, E., and Deale, M., "Scheduling and Rescheduling with Iterative Repair," *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, 1994, pp. 241-256.

[12] The RuleML Homepage (n.d.), http://www.ruleml.org/

[13] *A Java Deductive Reasoning Engine for the Web* (jDrew) (n.d.), http://www.jdrew.org/

[14] *The Mandarax Project* (n.d.), http://mandarax.sourceforge.net/

[15] Jim Gray, *Why do computers stop and what can be done about it?*, Technical Report 85.7, Tandem Corp., 1985.

[16] The NUnit Homepage (n.d.), Retireved 1 May 2004 from http://www.nunit.org/

[17] Drools: Object-Oriented Rule Engine for Java (23 January 2004), codehaus, http://drools.org/

[18] Resource Description Framework (RDF) (11 May 2004), http://www.w3.org/RDF/

[19] The Dublin Core Metadata Initiative (n.d.), http://dublincore.org/

[20] NDoc Homepage (n.d.), http://ndoc.sourceforge.net/

[21] log4net Homepage (n.d.), http://logging.apache.org/log4net/

[22] NAnt Homepage (n.d.), http://nant.sourceforge.net/

[23] Subversion Homepage (n.d.), http://subversion.tigris.org

[24] David E. Wilkins and Marie desJardins, "A Call for Knowledge-Based Planning," *AI Magazine*, Vol 22, No.1, pp.99-115, 2001.

[25] Gregg Rabideau, Russell Knight, Steve Chien, Alex Fukunaga, and Anita Govindjee, "Iterative repair planning for spacecraft operations using the aspen system," In Proceedings of the *International Symposium on Artificial Intelligence Robotics and Automation in Space* (ISAIRAS), Noordwijk, The Netherlands, June 1999.