Proceeding of the International Conference on Computing, Communications and Control Technologies,
August 14-17, 2004, Austin, Texas, USA

Rule-based Approach to the Validation of Subway Engineering Work Allocation Plans

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

Dennis YEUNG, Wai Ming
Synergicorp Limited
Unit G004B, G/F, Tech Centre
72 Tat Chee Avenue, Kowloon Tong
Hong Kong SAR
dennis.yeung@synergicorp.com

ABSTRACT

This paper describes our current progress in designing and developing an AI module that automatically validates individual subway engineering work requests as well as the manually produced weekly allocation plans and schedules. The work documented in this paper is part of a larger AI project that will eventually generate the entire weekly allocation plans automatically. Hundreds of various types of construction, engineering and maintenance tasks need to be performed each week within Hong Kong's busy subway system; on average over two million people travel through the four non-airport lines daily. Each work request will be evaluated for priority, urgency, and resource availability before they get approved. The AI Validation Module (AVM) ensures firstly that data submitted in a work request is consistent with operational constraints and safety guidelines. For example, engineering train formation must be correct and adequate resources are requested. Secondly, the AVM checks whether there are any resource conflicts with other jobs that have already been requested and whether there are adequate resources to handle the job based on job priority. AVM also validates the final allocation plan for conformance to all the operational rules, constraints and guidelines as well as resource availability. Finally, AVM monitors and validates any change in the allocation plan. This paper describes the domain problem and our AI approach and architecture.

Keywords: Expert System, Scheduling, Planning, Validation.

1. INTRODUCTION

Hong Kong's subway system is one of the busiest in the world. The traffic-flow though each station is roughly 46,000 passengers daily or 500,000 passengers per line. This can be several times that of other large cities around the world. For example, New York MTA's statistics [2] show roughly 14,000 passenger per station and 250,000 passengers per line daily. Ensuring that the entire subway system runs smoothly each day without any delays falls on the shoulder of a team of planners that decides what engineering or maintenance work need to get done, who to assign tasks to, and what equipment to use. This planning is done once a week in an Engineering Works Meeting (EWM). An allocation plan is produced three weeks prior to execution. During those three weeks, amendments, changes and additional requests can be made. The plan will need to be updated regularly to reflect these changes.

Similar to the Paris Métro [3], Hong Kong's subway 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.

2. CURRENT WORKFLOW

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.

For any given week, there will never be enough time slots to accommodate all the Possession Requests that would have been made. Therefore, as part of the EWM work, the planners must try to "combine" two or more requests from different parties together to maximize resource utilization. There is an entire set of rules that govern when and how "combines" can be made. For example, requests being combined must be in the same vicinity of each other and the engineering trains 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.

Once all the allocations have been made, the final approved allocation plan is then updated back into the ETMS. Once a request has been approved, additional details are 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.

Because of the large amount of knowledge involved in the EWM and the potential dangerous consequences if any safety rules were overlooked, it makes sense to use Artificial Intelligence (AI) to streamline the process, validate allocation results as well as to optimize the allocation plan and maximize resource utilization.

The subway system is incorporating AI into their process in three stages – Stage A uses AI for validation, Stage B uses AI for automatic allocation plan generation, and Stage C uses AI for interactive rescheduling as well as what-if analysis. This paper describes the design and implementation of the current Stage A of this AI project.

3. VALIDATION

In Stage A of the project, AI techniques are used to perform three different tasks: (1) validate individual Possession Requests prior to allocation, (2) perform preliminary resource loading estimations, (3) validate the finalized allocation plan produced by the EWM, and (4) validate any subsequent changes to the allocation plan. To perform these tasks, all the relevant operational and safety rules and constraints will need to be encoded into the system, as well as rules and constraints relating to how "combines" can be made. A rule-based approach was selected to encode this knowledge. In addition, the AI module will also need to keep track of the availability of different types of resources.

The AI module not only ensures that all allocation plans are absolutely safe to execute, it also helps streamline the overall allocation process. For example, at the time possession requests are made, the AI module ensures the requests are totally valid to eliminate wasting time in making unnecessary amendments later. At the same time, the AI module also performs an initial resource loading estimation to determine the likelihood that the request might be allocated. With this information, the requester can immediately adjust his/her request to maximize the chances of allocation and thus reduce the time wasted due to a simple mismatch between resource availability and requested work schedule.

4. RELATED RESEARCH

The design of the AI validation module is related to research in the areas of user interface validation as well as the use of XML as an AI knowledge representation. Although part of the role of our AI validation module is to validate client input from a

Web browser form, it is a bit more complicated than standard user interface validation as validation does not rely only on the current set of input data but also on other previously made inputs. In addition, the AI validation module must also be applied to objects in the middle-tier without user input.

There are many tools and approaches to validating user inputs. The common approach for Web-based forms might be to use validator controls such as those provided in MS ASP.NET [8] or, for J2EE, the Struts Validator [9]. Validator controls for ASP.NET are designed mainly for client-side use and does not allow validation rules to be centralized in separate XML files. The Struts Validator, on the other hand, allows validation rules to be centralized and stored in XML files. However, since we need our validation rules to be useable for both user input and in the middle tier, user interface validator would not be suitable for this project.

The Struts Validator relies on another Apache validator project called the Jakarta Commons Validator [7] that provides validation for JavaBeans based on XML-formatted rules. The Commons Validator addresses the issue of verifying the integrity of received data and the potential need of applying different rule sets to the same data, for example based on locale. Error messages may also need to be varied by locale. By using loosely coupled validation rules in XML files, these differences can easily be maintained.

Nolton [10] presents another validator framework for the middle tier, but for MS .NET. The framework supports the validation of method parameters, class properties and fields using .NET attributes and reflection. However, the validation rules must be hard-coded into the source code.

Tabet, Bhogaraju, and Ash [4] pointed out the advantages of using XML as a language interface for AI applications. The first being the ability to quickly create a new AI language customized for a particular domain or problem by simply designing a new XML Schema or DTD file. The XML format also makes interfacing an application to the AI knowledge-base a lot easier simply using standard XML parsers and tools. In [4], a mortgage underwriting prototype was developed using MindBox's ART*Enterprise* [5] expert system tool and XML for rule input and result output.

Venugopalan [6] defined a similar design pattern called the "User Interface Validator" pattern to verify user input on the server-side while centralizing validation rules as data (possibly XML) making it easily configurable and maintainable. It not only reduces development time but also makes maintenance and bug fixing easy since a change in validation rules only requires a change in the rule data.

Our design borrows upon these ideas, but is somewhat different. All our validation rules are centrally stored in XML files. However, they can be applied equally well to validate either user input as well as objects in the middle tier. This means the validation rules cannot be tightly linked to the client code. In some ways, our approach is similar to Jakarta's Commons Validator. However, the rules are not just associated with an individual property of a class but possibly a set of properties from one or more classes.

5. VALIDATION RULES

To represent the validation rules in XML, we must first select or design an XML-based markup language. The 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. Besides XML-only format, there are also XML/RDF and RDF-only formats of RuleML as well as Object-Oriented RuleML (OO RuleML). Many rule-engines, especially open source ones, such as jDrew [13] and Mandarax [14] also supports RuleML.

There is also a considerable amount of interest in applying RuleML work to Semantic Web. For example, the Semantic Web Rule Language (SWRL) [15], combines OWL with RuleML to perform Semantic Web inferencing. SWRL is part of the DARPA Agent Markup Language (DAML) project [16].

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.

6. IMPLEMENTATION

Our AI module is part of the ETMS system, which is implemented entirely on the MS .NET platform. All the user interface screens were coded in ASP.NET. The AI module itself is a server process that is coded in C# with rules in XML. The following describes some of the domain knowledge that are encoded into the system.

DOMAIN KNOWLEDGE

The AI validation module encodes several types of knowledge regarding how engineering works should be assigned. The main types of knowledge encoded into our system include:

- Track Possessions
- Train Protections
- Train Consists
- Train Availability
- Personnel Availability
- Combining Requests
- Operational Heuristics

Track Possessions
Engineering work requests are also know as "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 Protections
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 consists," 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.

KNOWLEDGE REPRESENTATION

The AI 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, the 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:salience="high priority"
    ai:direction="forward">
  <dc:title xml:lang="en-US">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="r1" ai:type="Request"/>
      <ai:Property ai:var="priority" ai:key="r1.JobPriority"/>
      <ai:Condition ai:check="Request.IsEngergized(priority)"/>
    </ai:And>
  </ai:Antecedent>
  <ai:Consequent>
    <ai:And>
      <ai:Validate ai:check="r1.CombineLevel==DoNotCombine"/>
    </ai:And>
  </ai:Consequent>
  <ai:Message
    ai:format="Cannot combine energized possession: {0} - {1}"
    ai:arg0="r1"
    ai:arg1="priority"/>
 </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, our Rule Engine automatically generates a .NET DLL that can be called from the application server without any additional rule-related coding. All necessary information needed to compile the rules are extracted from domain business objects via Object Reflection; the DLL was then generated dynamic in memory. The GUI rule-editor uses .NET Attributes, i.e. metadata annotated on the business objects, to create the editing environment for the domain.

7. SUMMARY AND CONCLUSION

In this paper we presented the current progress in designing and developing a rule-based AI module to perform validation of possession requests and engineering work allocation plans for the Hong Kong subway system. The work documented here represents the first stage in a three-stage AI project. The current stage will be fully deployed by end of June with the final stage completing around December of this year.

8. ACKNOWLEDGEMENT

The authors would like to thank the MTRC Limited for providing us with an opportunity to participate in this project. We would also like to thank MTRCL for allowing us to share our project experiences in this paper. In particular, we would like to thank MTRCL's Jerome Lam and Helena Chan for their support!

9. RESEARCH FUNDING

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 and 6980002).

10. REFERENCES

[1] *MTR Corporation* (n.d.), "Patronage – Monthly Total," Retrieved 1 May 2004 from http://www.mtr.com.hk/eng/investors/pad.htm

[2] *Metropolitan Transportation Authority* (n.d.), "The MTA Network," Retrieved 1 May 2004 from http://www.mta.nyc.ny.us/mta/network.htm

[3] *RATP* (n.d.) Retrieved 1 May 2004 from 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.), Retrived 1 May 2004 from http://www.mindbox.com/

[6] Venugopalan, Vivek, "User Interface Validator Pattern," *TheServerSide.com*, February 17, 2002, Retrieved 1 May 2004 from http://www.theserverside.com/patterns/thread.tss?thread_id=11947#40970

[7] *Commons Validator* (n.d.), The Jakarta Project, Retrieved 1 May 2004 from http://jakarta.apache.org/commons/validator/project-info.html

[8] *Introduction to Validating User Input in Web Forms* (n.d.), MSDN, Retrieved 1 May 2004 from 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, Retrieved 1 May 2004 from http://jakarta.apache.org/struts/userGuide/dev_validator.html

[10] Nolton, Mathew (23 March 2004), "Validators," In theserverside.com, Retrieved 1 May 2004 from http://www.theserverside.net/articles/showarticle.tss?id=Validators

[11] *Simple Rule Markup Language (SRML)* (17 May 2001), In Cover Pages, Retrieved 1 May 2004 from http://xml.coverpages.org/srml.html

[12] The RuleML Homepage (n.d.), Retrieved 1 May 2004 from http://www.ruleml.org/

[13] *A Java Deductive Reasoning Engine for the Web* (jDrew) (n.d.), Retrieved 1 May 2004 from http://www.jdrew.org/

[14] *The Mandarax Project* (n.d.), Retrieved 1 May 2004 from http://mandarax.sourceforge.net/

[15] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML* (30 April 2004), Retrieved 1 May 2004 from http://www.daml.org/rules/proposal/

[16] *The DARPA Agent Markup Language Homepage* (n.d.), Retrieved 1 May 2004 from http://www.daml.org/

[17] Drools: Object-Oriented Rule Engine for Java (23 January 2004), codehaus, Retrieved 1 May 2004 from http://drools.org/

[18] Resource Description Framework (RDF) (11 May 2004), Retrieved 1 June 2004 from http://www.w3.org/RDF/

[19] The Dublin Core Metadata Initiative (n.d.), Retrieved 1 June 2004 from http://dublincore.org/