Prototyping Freight Trains Traffic Management
Using Multi-Agent Systems

Alessio Cuppari, Pier Luigi Guida
Sistemi Informativi, Divisione Infrastrutture - FS S.p.A
P.zza Croce Rossa 1, 00161 Roma, Italy
cuppari@disi.unige.it, pierluigi.guida@guidafs.inet.it

Maurizio Martelli, Viviana Mascardi, Floriano Zini
D.I.S.I. - Università di Genova
Via Dodecaneso 35, 16146 Genova, Italy
{martelli,mascardi,zini}@disi.unige.it

Abstract

Applications dealing with railway traffic management have been usually modeled adopting classical technologies such as Operations Research and Constraint Programming. These technologies are suitable to model static situations where the information is complete, but they lack to cope with the dynamics and uncertainty of freight trains traffic management. The paper presents a new approach to the problem based on the Multi-Agent System technology. CaseLP, a logic programming based environment for MAS prototyping, has been adopted to face a real case-study: the management of freight trains traffic along the railway line between the Italian stations of Milano and La Spezia. The research, conducted within the framework of the EuROPE-TRIS Project [7], has successfully demonstrated the advantages of the MAS approach to this field of application.

1. Introduction

Due to the constantly increasing demand of short-term train schedules, automatic tools for decision support that can be used by Transport Operators in the management of freight trains traffic are more and more necessary. This is particularly true when the number of trains running on a railway line and the availability of tracks are only known on a day-by-day basis. Decision support systems can prove very useful to maximize tracks demand granting and optimize the traffic flow.

The railway line connecting Milano and La Spezia yards is strongly characterized by short-term train scheduling. In order that a freight train may reach the port of La Spezia, a loading authorization for each wagon has to be granted and train dispatching along the line has to be authorized. Train traffic flow in La Spezia depends on a great number of variables, such as the average number of trains that La Spezia yard will contain, the average train flow from the yard to the port terminals and the number of loading authorizations already granted for a certain day. The possibility of delivering a new train on the line depends both on local congestion indexes of railway sections and on global information about the train traffic situation. Due to the inherent dynamism characterizing the values of these variables, a long-term scheduling of loading authorization and train delivering is very hard to achieve. Instead, dynamic real time coordination and cooperation between the personnel in charge on the railway line is necessary to obtain a good scheduling agreement.

This complex scenario highlights the usefulness of the adoption of some decision support mechanism. Classical techniques have widely been adopted for the realization of prototypes for decision support of this kind [13, 4, 5, 2]. The paper presents an approach based on more recent techniques, i.e., multi-agent systems (MAS) [16] and ontologies [6], to this aim. These instruments allow to organize the prototype realization along two dimensions. MAS permit a task-based organization, by which different tasks are identified and allocated to various agents, corresponding to the human or artificial entities that are involved in the application. Ontologies are instead used to give the knowledge-based organization of the prototype. The information used in the application to execute the various tasks is represented as concepts and relationships among them.

The prototype has been developed using CaseLP [11, 12], a Complex Application Specification Environment
There are three levels of information that have to be considered in the realization of a CaseLP prototype.

- Information related to the application domain of the prototype. Domain knowledge is incorporated into a domain ontology, whose definition is independent of the fact that it can be used in a MAS. This structuring of domain knowledge can be profitably shared among different agents in a MAS and/or reused for building multi-agent systems that operate on similar domains.

- Information related to characteristics and properties of agents. Agents need to know features of other agents with which they interact. In CaseLP agent knowledge is expressed in an agent ontology which describes features of CaseLP agents and does not depend on the domain to which a multi-agent system is applied. In particular, part of the agent ontology includes vocabulary to identify basic agent services, such as creation and deletion of an agent or search for an agent that provides some services.

- Domain dependent information related to agents. Every time a prototype is realized, the agent ontology is extended to include domain level knowledge that is manipulated by agents.

Ontologies are exploited to perform various kinds of analysis on the correctness of the different steps of prototype realization. Moreover, agents use domain level ontology during computation as knowledge repositories.

2.1. Prototyping method

The prototyping method proposed in CaseLP includes informal and more formal phases and tries to conjugate rigor and simplicity of use. It is composed by the 7 steps outlined below.

1. Informal description of prototype features. In the first stage features of the prototype and its domain of application are described in natural language. Both task-based and knowledge organizations are taken into consideration and the informal description concerns tasks that have to be executed, as well as information used to perform these tasks.

2. Ontological analysis. This step concerns the definition of the first and third level of prototype information previously listed.

   - According to the domain description obtained from step 1, the prototype developer decides if there is a suitable already existing domain ontology that can be used or if a new one has to be realized ad hoc for the application.
3. **Static description of prototype architecture.** The set of tasks identified in the previous step drives the definition of the agents that form the prototype and their interconnections. A class of agents is designed to execute a certain task, providing and requesting a set of services. In this step the prototype developer defines:

- the classes of agents that the application needs; their *kind, architecture, interpreters* (for interface agents) and required and provided *services*;
- the *instances* of each class that will form the MAS;
- the *interconnections* between agents, that is, how a service provided by an agent is linked to a service required by another agent.

4. **Description of communication between agents.** Each provided or requested service needs a specific *conversation* between the agent that provides the service and the one that requires it. This step allows to specify the sequence in which messages are exchanged. Some conversation can start some other (sub)conversation. This is properly captured defining a relation *sc (sub-conversation)* between conversations. The *conversation model* of the MAS includes the set of all conversations and the relation *sc*.

5. **Description of initial state and behavior.** This step sets the initial beliefs and behavior of the agents in the MAS. Specification of behavior takes into consideration the conversation model defined in step 4. For reactive agents, behavioral rules consider the received messages and the current beliefs. For proactive agents, only the current beliefs determine future actions.

6. **Prototype implementation.** In this step the MAS prototype is built: each agent is implemented as an EC-LiPSe module. This module is obtained from the specification given in the previous steps. This step allows integration of external software modules.

7. **Execution and testing.** The obtained prototype is executed. This allows the developer to test its behavior, in order to eventually iterate the previous step and refine the application against the initial requirements.

The agent ontology does not change from application to application and for this reason it does not explicitly appear in the prototyping method.

2.2. **Prototyping tools**

A set of tools is provided within CaseLP in order to support the prototype development. In the following a brief description of these tools is given. See [12] and [17] for further details.

2.2.1 Ontologies

An ontology is modeled as a 5-tuple

\[
< \text{name}, \text{terms}, \text{predicates}, \text{kb}, \text{constraints} >
\]

where *name* identifies the ontology, *terms* is a set of constants and function symbols, *predicates* is a set of predicate symbols to be applied to terms, the knowledge base *kb* is a set of definite Horn clauses, and finally *constraints* is a set of constraints that *kb* has to satisfy.

The agent ontology can be exploited to perform a semantic check on an architectural description of a CaseLP class of agents after step 2 in the prototyping method. We can use the domain dependent part of the agent ontology to check agent behavioral rules and verify if every reactive rule that is defined in the behavior is related to a service provided by the agent. This check can be executed when the MAS prototype is implemented in step 6 of the method. As far as domain ontology is concerned, predicates that are defined in it can be exploited in agents’ behavior as *conditions* to be satisfied to proceed computation.

2.2.2 MAS-adl

In the third step of the method MAS-adl, a simple, customized, *architectural description language* for MAS, is adopted. It allows to define the classes of agents used in the MAS under construction, the instances of such classes (i.e., the agents that constitute the MAS) and the directed links between instances of agents, from a service provided by an agent to a service required by another agent.

2.2.3 Tools for conversation model

The fourth step of the method describes the conversation model of the MAS. The agent communication language is a subset of KQML [14]. The conversation model is defined by choosing, for each service, the sequence of messages (conversation), as well as their *performative*. Message contents correspond to the information terms identified in the agent ontology. A relation *sc* defines which conversations eventually start during other conversations.
2.2.4 ACLPL

The language ACLPL is used in the fifth step of the method to program the reactive and proactive agents of the MAS. The ACLPL code for an agent is structured in three parts. They express respectively the initial set of agent’s beliefs, the set of reactive or proactive rules defining its behavior and the set of auxiliary procedures. The initial state of an agent is a (possibly empty) set of beliefs, which are atomic ground formulas, while the behavior of an agent is described by a sequence of reactive and proactive rules. A reactive rule has the form

\[
\text{on message Message check Condition do Actions.}
\]

It is fired by the reception of a message matching with Message and if Condition is satisfied the corresponding Actions are executed. Single actions can be either state updates or messages sending. A proactive rule is similar to a reactive one, but it has not the on message Message part.

2.2.5 Architectural task control

When the prototype is being implemented, in step 6 of the method, each agent is given a particular task control. According to its architecture, reactive or proactive, a different meta-interpreter is used as execution engine of the code of the agent. The task control can include several execution policies, for example selection of which fired rules are actually executed.

2.2.6 CaseLP Simulator and Visualizer

Execution and visualization of the MAS are performed in step 7 of the method by the CaseLP Simulator and Visualizer. The simulator engine is implemented in ECLiPSe, and is based on a round-robin scheduler that activates in turn all the agents in the MAS. When activated, an agent behaves according to its architectural task control and to the rules defining its behavior. It is possible to follow what happens at the agent level during MAS execution by means of a graphical interface, the Visualizer. According to the developer needs, the code of the agents is automatically instrumented. Instrumentation adds probes to agents code; events related to state changes and/or exchanged messages can be recorded and collected for on-line and/or off-line visualization.

3. Freight trains traffic management

The so-called train dispatching problem concerns the control of trains movement over a railway line, in order to guarantee traffic regularity and service performance. It is subject to several operational constraints, particularly when deviations from scheduled timetable and other running disturbances (e.g. stochastic delays, unforeseen track maintenance, etc...) occur. The task is to conceive support systems in order to help the rail dispatchers or controllers in taking the best possible decisions and regulating actions (e.g. overtakings and temporary train stops at appropriate stations). Computer technology is increasingly being used to assist this critical functions. One dispatcher usually governs a line section (e.g. in European experience it is about 100-300 kilometers long) before passing train control to a bordering one. For very long and densely operated lines, several regional control centers cover the whole “corridor” and a more strategic, higher co-ordination center is considered effective to increase the overall quality of service.

The real case-study we have chosen to demonstrate the suitability of the MAS technology in this context is the railway line connecting the Italian stations of Milano and La Spezia. The demonstrator we have developed aims to be the base for a decision support system for Train Dispatching to be supplied to the Traffic Co-ordinator in Milano (COIM) and the Production Deputy in La Spezia (yard operative manager). The case-study has been developed within the framework of the EuROPE-TRIS Project [7], as a result of the co-operation between the Information Systems Division of Italian Railways and the Computer Science Department at the University of Genova, Italy.

3.1. Problem description

3.1.1 Line model

The railway line connecting Milano and La Spezia is composed by three main sections, named $S_1$, $S_2$, $S_3$. Sections are delimited by nodes, respectively Voghera, Arquata and Genova, that are accessed by minor lines serving several container terminals (see Figure 1).

3.1.2 System actors

The following real entities operate in this scenario: $T_{[vog,ar]}x$ is a container terminal operating inside the railway network. It is connected to the node $[Voghera, Arquata, Genova]$...
via a minor line. COIM is the line traffic manager, responsible for train dispatching on the overall line. Production Deputy of La Spezia (PD) manages trains arrival, recovery and departure operations in La Spezia yards. Section Traffic Co-ordinator (STCi) is responsible for train dispatching on line section Si.

3.1.3 Management of wagons loading authorization (LA) requests

The loading operation of train wagons directed to La Spezia is performed in a terminal, and has to be authorized. The requesting terminal sends to COIM the LA request; the COIM updates the congestion index (number of trains per Km) of the concerned section in the indicated date, assuming the granting of the LA. If the new index is greater than the maximum limit, the COIM transmits the request to the PD. PD grants or refuses a LA request based on: average number of trains that La Spezia yard will contain at wagons arrival date; average train flow from the yard to the port terminals; number of LA requests already granted for the requested date.

3.1.4 Train dispatching authorization

A terminal asks its STCi for dispatching a train along the main railway. The generic STCi knows the congestion index of his section. The decision concerning the running of the train is based on the comparison between such index and a maximum limit value: if this value is not overcome, the train is sent and the congestion index of section Si is updated. Otherwise, the decision is delegated to COIM, that is asked by STCi. COIM knows the congestion indexes of all the line sections; the decision rule is based on the comparison between the average of these indexes and a reference parameter: only if this value is not overcome, the train is sent and the congestion index of section Si is updated.

3.1.5 Model assumptions

In order to realize the prototype, some assumptions have been made. Furthermore, some constraints related to actual structure of the railway has been taken into consideration. The system time-frame (day) is divided into four time bands, each elapsing six hours. Train progress between two different nodes is managed at the end of each band. Infrastructures in La Spezia are limited and the prototype considers only one yard, having the capacity (i.e., number of tracks) of all the real yards of La Spezia. Average wagon flow from La Spezia yards to the port terminals is considered, as well as train arrival forecasts in La Spezia. It is assumed that each node has infinite capacity i.e., it can contain an infinite number of trains.

Figure 2. Case-study: part of agent ontology.

3.2. Prototype realization

An agent based approach certainly suits the realization of a prototype for the problem described above. The description given in Section 3.1 can be seen as the informal description of prototype features required by step 1 of the CaseLP development method. In the following, we show how the other steps have been followed for the realization of a working prototype.

3.2.1 Ontological analysis

The domain ontology for this application has been realized ad hoc. It contains representation for all the relevant objects of the domain: stations, sections, terminals, train number and date of delivering. The ontology knowledge base contains predicates to classify these objects and relationships that describe the topology of the railway line, as depicted in Figure 1.

As far as the domain dependent part of agent ontology is concerned, in this particular application each task to be performed by the prototype corresponds to a service. Each service has an associated set of information terms. Figure 2 presents the corresponding part of the knowledge base of the agent ontology. Predicate domain.service holds for terms that represent services. Predicate domain_msg holds for information terms that are possible contents of messages that are exchanged for providing services. Predicate domain.associated_to relates a message content to the corresponding service.

3.2.2 Static description of the application architecture

Figure 3 depicts the MAS-adl architectural description of the MAS. Five classes and eight instances of agents are defined. Four classes correspond to the four system actors previously described and define logical reactive agents. The fifth class defines logical agents that proactively check the system time and inform section agents when a time band
3.2.3 Description of communication between agents

A sample of the conversation model of the MAS is shown in Figure 4. For each previously defined link, a conversation is given (only two of them are presented here). Furthermore, the conversation model defines relation se on conversations.

3.2.4 Agent initial state and behavior

Part of the ACLPL code for COIM behavior is presented in Figure 5. Some abbreviations are used for messages, the initial state is not shown. The code for the other agents in the system has a similar form.

3.2.5 Execution and testing

The test of the prototype has been carried out running the system with various initial configurations and inputs. The resulting behavior was satisfactory. Figure 6 gives an idea of how the CaseLP Visualizer works. It presents offline visualization of the execution of the MAS defined in the previous steps and shows how concurrent LA requests have been managed. For each instrumented agent, both exchanged messages and state updates are depicted.

4. Discussion

Railway dispatching or scheduling problem has been widely investigated by Operations Research (OR) combinatorial methods, also motivated by the closely linked issue of timetable planning. In the OR traditional approach the short (one rail line section) is generally addressed and the scheduling or re-pathing problem is the main target, e.g. minimizing some cost function of weighted train delays. In [4, 5] real-time scheduling is modeled as an off-line timetable planning, due to its abstract problem similarities. Freight train scheduling is specifically addressed in [10]. In EuROPE-TRIS project [7] a state-of-the-art Lagrangian and network flow heuristic optimization algorithm for scheduling over long railway lines is also proposed. A scheduling optimization algorithmic approach is also implemented in the new Italian railways traffic control system (“SCC - Sistema di Comando e Controllo”, [15]), which will manage long “corridors” dispatching.

Another classical approach which has been adopted in railway dispatching is Constraint Programming (CP). As an example, [13] describes a system developed using the ILOG Solver and the ILOG Scheduler libraries. The input of the system consists of both static and dynamic information.
Static information concerns the network itself and incorporates railroad stations, topology of the railroad network and operational constraints (yard capacity, travel times, loading and unloading times). Dynamic information deals with request for empty cars to load/unload trains, demands for loaded cars, current standing car locations and some predefined (operating or planned to operate) trains. The program task is to create a constraint-consistent set of trains that will satisfy all requests and demands, and also optimize some parameters.

Finally, some applications of Expert System technology have been reported [2, 9], but their widespread application seems likely hampered by difficult tuning for real life operations.

The adoption of the MAS technology for railway dispatching problems is quite recent. Nevertheless, interesting results have already been obtained. In [8] the MARS multi-agent system is presented. MARS models a society of cooperating transportation companies. The functionality of the system as a whole, namely the solution of the global scheduling problem, emerges from local decision-making and problem-solving strategies. More recently, the TELETRUCK approach [3] has been developed in close collaboration with a forwarding company to develop a MAS with holonic agents\(^1\) that will support dispatch officers in transport scheduling. The system can handle real-world requirements like dynamics and uncertainty. Route planning and allocation of resources are controlled by cooperation protocols.

Various reasons motivate the adoption of MAS technology to the applications like the one described in this paper:

- Many of the problems concerning with planning and scheduling are known to be NP-hard. This means that obtaining a global view of the modeled system is impossible due to its complexity. The only reasonable approach is decomposing the problem into simpler subproblems, finding subsolutions which are locally optimal and combining these solutions to find a good (even if not optimal) global solution. The task decomposition is quite easy to perform in a multi-agent setting, while it should not be feasible following more standard approaches.

- The planning and scheduling of freight transport is performed in a highly dynamic environment, under a high degree of information incompleteness and uncertainty; the extension of standard approaches in order to cope with the dynamics and incompleteness of this domain is far from being consolidated. The adoption of the MAS technology helps in developing entities which perceive the environment and react to the changes occurring in it, also dealing with incomplete information.

- The application domain is inherently distributed and is composed by communicating entities. A model in which a set of agents exchange information using message passing is closer to the reality than a set of constraints to be solved. Thus, a greater scalability and flexibility can be obtained. The agents in the system show a behavior which is designed according to the possible messages coming from other agents and the environment, and to the local status. This behavior can be independent from the number of agents instances and the way the environment is modeled. Modifying the way an agent manages an incoming message can be done by changing a few lines in the agent’s code. Provided that the conversation model remains the same, the change of an agent’s behavior has no impact on the description of the other agents. The behavioral method which can

---

\(^{1}\)Holonic agents are agents which are composed by subagents acting in a corporate way.
be adopted in a MAS, besides working at higher level than classical scheduling techniques, could be easily fitted with the railways process-oriented automation.

- One of the main features of MAS is to provide interfaces towards legacy software, allowing its integration and reuse. This allows to take advantage of the best features of standard OR or CP approaches, maintaining the overall description of the system at a high and very realistic level of abstraction.

These considerations apply for the MAS technology in general. The choice of CaseLP as a specification and simulation tool can give another advantage, concerning with the execution of the specification. CaseLP allows to model the application in a quite easy way. Once the application has been modeled, the simulation of its behavior is obtained with a very small effort. Through the use of CaseLP increasingly wider scenarios and business models can be therefore simulated and tested before full scale investments.

References


