An Agent Model for a Railway Consist Communication System

M.J. Fernández-Iglesias and J.C. Burguillo-Rial

Departamento de Ingeniería Telemática
{manolo,jrial}@det.uvigo.es,
WWW: http://www-gist.det.uvigo.es
ETS de Ingenieros de Telecomunicación
Campus Universitario S/N
36200 Vigo (SPAIN)

Abstract. In this paper we describe a case study to illustrate the use of a multi-agent system for designing and modeling an on-board communication system for railway consists. Selected elements in the railway consist has been modeled as agents with a simple belief, desires and intentions (BDI) architecture that interact among them to satisfy an overall goal, namely trainset security. This goal can be further refined into consist registration, detection of exception events and control of consist vehicles through command execution.

1 Introduction

Over the past three decades, software engineers have faced the problems and circumstances related with software complexity in concurrent and distributed systems. Among them, it is now widely recognized that interaction is probably the most important single characteristic of complex software.

Since the 1980s, software agents and multi-agent systems have grown to become one of the most active areas of research and development in computing. The agent concept stands for an autonomous system, capable of interacting with other agents in order to satisfy its design objectives. Nowadays, an increasing number of distributed computing systems are being viewed in terms of interacting, semi-autonomous agents.

Having in mind that there is no general agreement about what exactly an agent is, we denote an agent as a system that enjoys [7] *autonomy*, that is, agents encapsulate some state information and take their own decisions based on such state; *reactivity*, because agents are situated in an environment, which they are able to perceive and to respond to its changes; *pro-activeness*, as agents do not simply act in response to their environment, but are able to exhibit goal-directed behaviour; and *social ability*, because agents interact with other agents or humans via some kind of agent communications, and typically have the ability to engage in social activities, such as problem solving or negotiation, in order to achieve a common goal.

The greatest potential of agent architectures relies on the multi-agent concept. Multi-agent systems are ensembles of agents, acting independently from each other to accomplish their own tasks. For this, they have to interact with other agents, humans and with their environment to gather the information or services they need. Additionally, an agent might have to coordinate its activities with other agents in the society to ensure that overall goals can be met. This concept raises the emergence of *agent societies*.

In this paper we describe a case study to illustrate the use of a multi-agent system to design and model an on-board communication system for railway consists. Every agent is modeled using a simple BDI architecture [1] with data structures representing beliefs, desires, and intentions of agents, and functions that represent deliberation and means-end reasoning. Intentions play a central role in the BDI model: they provide stability for decision making, and act to focus the agent's practical reasoning.

The remainder of the paper is structured as follows. Section 2 describes the main characteristics of the system to be modeled. Section 3 introduces an agent society to manage the interactions among the different elements in the consist. Finally, Section 4 presents the conclusions.

2 System Description

The target system is composed of a base station, typically placed inside the engine, and several slave stations, located inside consist units, typically one inside each wagon (see Fig. 1).

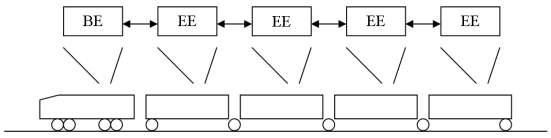

Fig. 1. System outline

The objective (i.e. overall goal of the system) is to guarantee trainset security. On the one side, it permits the engine to keep track of the consist configuration (i.e. which wagons are coupled to the consist, in which order, etc.). On the other side, it supports the activation of consist devices like doors, air conditioning, security equipment or information pannels, and the communication of alarm or exception conditions back to the engine.

The overall behaviour of this system is as follows:

1. Stations communicate with their neighbors through an appropriate communication channel. All intermediate stations have two active bi-directional

Table 1. System messages

Message	Pars.	Orig/Dest	Meaning
REG	–	$B \rightarrow S$	Proceed to register
NEW	id	$S \rightarrow B$	I am station id
$LAST$	id	$S \rightarrow B$	I am the trailing station (id)
TOK	–	$B \rightarrow S$	You are alive?
ACT	id, alm	$S \rightarrow B$	id is alive. Besides, I am sending alarm alm
$LACT$	id, alm	$S \rightarrow B$	id is alive (trailing). Besides, I am sending alarm alm
COM	id, cid	$B \rightarrow S$	id should execute command cid
$RESP$	id, $rinfo$	$S \rightarrow B$	id executed a command. Results are $rinfo$
$LRSP$	id, $rinfo$	$S \rightarrow B$	id (trailing) executed a command. Results/alarms are $rinfo$

connections, whereas the base station in the engine and the slave station in the trailing wagon have a single bi-directional active connection. Messages to be transferred using these communication channels are summrized in Table 1.

2. The base station initiates a registration process for slave stations to register at the base. The process is initiated by a REG message that is propagated along the train. Upon reception of this message, a slave station answers with a NEW message and propagates the REG message to the rest of the consist. NEW messages are propagated back to the engine. The last wagon in the consist responds with a $LAST$ message instead of a NEW message. The trailing position in the train is detected through a $TIMEOUT$ (timeout) perceipt from the environment. This timeout event is signalled by the underlying low-level communications layer.
3. Once all stations are registered, the base station knows the relative position of all components of the consist. From this moment, the base station can:
   - Check the availability of the components of the consist asking for a heartbeat. The base station generates a TOK message that is propagated along the consist. Upon reception of this message, slave stations in wagons respond with an ACT message. The last station responds with an $LACT$ message. ACT and $LACT$ messages in their way to the base station are propagated back by slave stations.
   - Send commands to slave stations. These commands are intended to activate devices in wagons or to check the status of wagon components. For this, the base station generates an (addressed) COM message instead of a TOK. The COM message is propagated downstream to the destination slave station, which responds with a $RESP$ message and generates a TOK message to be forwarded to the rest of the downstream slave stations. Slave stations not in the destination list of the COM message respond to the base station with an $ACT/LACT$ message. If the destination station is the trailing one, it responds with an $LRSP$ message.

The (slave-generated) messages are propagated upstream by intermediate stations until they reach the base station.

4. Slave stations may autonomously generate alarms that are transmitted as parameters in the corresponding $ACT/LACT$ or $RSP/LRSP$ messages.

3 System modelling as an agent society

Along the next paragraphs we model this train management system following an agent-oriented approach. Firstly, we briefly introduce the syntax and semantics of the language used to model agent behaviour. Then, we propose an agent decomposition of this problem to construct a multi-agent system or agent society, identifying inter-agent communication mechanisms. Finally, we describe the behaviour of agents in the society. Due to space restrictions, non-common procedures like on-the-fly coupling and uncoupling of cars are omitted from this description. For a more detailed description of this system see [3].

3.1 Notation

Individual agents follow a model inspired in the Belief-Desire-Intention (BDI) model [1], in the sense that decision making depends on the manipulation of data structures representing the beliefs, desires and intentions of agents. However, compliance to concrete architectures implementing this model has been sacrified in favor of more concise and easier to implement agent specifications.

To simplify system analysis and description, bi-directional communication through physical channels has been modelled using two uni-directional channels. Channel operations are defined using the usual constructors available in languages like Promela [4] or LOTOS [5]. Channels pass messages in first-in first-out order:

- The statement *qname*!*expr* sends the value *expr* (variable or constant) to the chanel *qname*.
- The statement *qname*?*msg* retrieves a message from the head of the channel, and stores it in variable *msg*. The receive operation is executable only when the channel addressed is nonempty. A receive operation on an empty channel gets blocked until data is available in the channel.
- The statement *qname*?*msg*[*cond_expr*] is executable only if the chanel is nonempty and the message at the head of the channel matchs expression *cond_expr*, i.e. [*cond_expr*] represents a guard for the receive operation.

Individual agents are defined inside an **agent** / **endagent** environment. Flow-control sentences and statement separator ";" have their usual meanings, as do the usual boolean and arithmetic operators. Besides, special functions are defined to represent specifics of the BDI model. To manage belief we define:

- *UpdateKB*(KB, Expr). This function updates an agent's knowledge base. The first parameter references the knowledge base to be updated. The second parameter expresses a fact as a first order expression.

- *GetKB*(KB, Expr). This function is used to extract information from the knowledge base passed as a parameter. Extracted information is kept in the local store of the invoking agent.
- *Known*(KB, Expr). This function returns TRUE if the fact expressed by its second parameter can be deducted from the referenced knowledge base (first parameter).

To manage desires and intentions, the functions below are available:

- *SetGoal*(Gid, Expr). Agents call this function to state their goals. The first parameter assigns an identifier to the goal. The second parameter is an expression that defines the goal, the result to be accomplished to fulfill the goal. For goals involving channels (i.e. agent perceptions and actions), two additional operators are available:
  - Receive(chanID, Expr). The goal is to receive specific information through a given channel. This operator has two parameters: the first one identifies the channel involved, and the second one the information that should be received to fulfill the goal.
  - Send(chanID, Expr). The goal is to successfully send information through a channel. Its parameters are defined in a similar way to Receive.
- *CheckGoal*(Gid). This function returns TRUE if the goal referenced has been accomplished, and FALSE otherwise.
- *DropGoal*(Gid). The goal referenced is removed from the goal list.

3.2 Agent Model

The system described above will be modeled by the agent hierarchy depicted in Fig. 2. These hierarchy is composed by the the following elements:

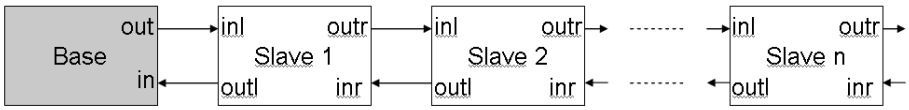

Fig. 2. Multi-agent system for on-board communications

- An agent acting on behalf of the base station, which implements engine's behaviour. This agent is composed by several cooperating agents, each of them in charge of one of the distinct procedures defining the communications protocol (registration, command generation, activity control, alarm reception).
- Several agents acting on behalf of their respective consist components (i.e. slave stations). As for the base station agent, these agents are composed by several cooperating agents implementing distinct aspects of the communication system.

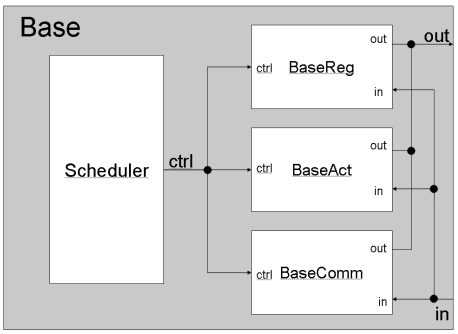

Fig. 3. Multi-agent system for on-board communications

The agent hierarchy for the base station is outlined in figure 3. The hierarchy for slave stations is defined in a similar way. For each station in the trainset, overall agent behaviour is organized by the *Scheduler*. This agent is a goal-based agent whose behaviour is outlined in figure 4 [6].

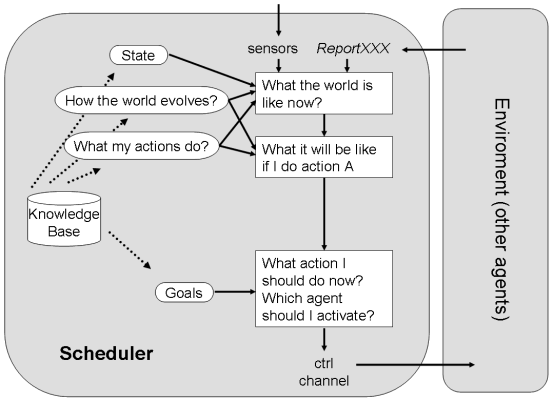

Fig. 4. Outline of agent *Scheduler*

Along the next paragraphs we describe agent's behaviour and interactions to successfully accomplish the main procedures for which this system has been designed, namely consist registration, activity control, and command execution.

For the sake of clarity, we assume that the base station is the leftmost element in the system, and slave stations are coupled to its right. Then, the trailing station will be the rightmost one.

3.3 Consist Registration

Consist registration is achieved through the cooperation of agents *BaseReg* at the base station and active *SlaveReg* placed at slave stations in the consist. These agents are activated by the corresponding *Scheduler* agent when a registration process is required.

Behaviour of agent *BaseReg* is outlined in figure 5. This agent is dormant until it perceives a *Scheduler*-generated $SYNCBR$ message through the control channel. Then, it sets as its goal to perceive a $LAST$ message from its environment, more specifically through its input channel. It generates a REG message to be sent to the adjacent station and waits for messages coming through it input channel. On arrival of NEW messages, the knowledge base is updated with new stations IDs. The process continues until a $LAST$ message is received or an unexpected event occurs (**default** clause). Before becoming dormant again, it checks its goal and raises an exception if the goal has not been fulfilled. Exceptions are captured by the *Scheduler*, and may be due to events like coupling and uncoupling of cars, failures in the underlying communication channel, etc.

```
agent BaseReg(chan in, out, ctrl) is
while (TRUE) do
    ctrl?msg[msg == SYNCBR];
    UpdateKB(Base, {slist := []});
    SetGoal(RegAll, Receive(in, LAST));
    out!REG; in?(msg, id); finish := FALSE;
    while (!finish) do
        case msg of
            NEW: UpdateKB(Base, {slist := slist++[id]})
            LAST: finish := TRUE
            default finish := TRUE
        endcase endwhile
    if (!CheckGoal(RegAll) then raise(ExReg)
endwhile
endagent
```

Fig. 5. Registration. Base station

Behaviour of slave stations is described in figure 6. As in the previuos case, this agent is activated by the scheduler. Its first task is to update slave station's knowledge base stating its IDs as defined by the scheduler. Then, it becomes dormant until a registration process is needed. This agent's goal is to send back to its peer at the left a $LAST$ message. Once stated its goal, it waits for incoming messages from the left. When the agent perceives a REG message, generates a base station-bound NEW message and propagates the REG message to the right. Then, it forwards all incoming messages to the left. If this agent perceives

that its station is the trailing one, updates the knowledge base stating this fact and generates a *LAST* message.

Once the process is finished, it checks its goal and raises an exception if the goal has not been fulfilled. Then, it becomes dormant again.

```
agent SlaveReg(int addr; chan inl, outl, inr, outr, ctrl) is
UpdateKB(Slave, {my_addr := addr});
while (TRUE) do
    ctrl?msg[msg == SYNCSR];
    SetGoal(RegAll, Send(outl, LAST);
    inl?msg[msg == REG];
    outl!(NEW, addr); outr!REG;
    inr?(msg, id); finish := FALSE;
    while (!finish) do
        case msg of
            NEW: outl!(msg, id)
            LAST: UpdateKB(Slave, {imlast := FALSE});
                outl!(msg, id); finish := TRUE
            TIMEOUT: UpdateKB(Slave, {imlast := TRUE});
                outl!(LAST, addr); finish := TRUE
            default finish := TRUE
        endcase endwhile
    if (!CheckGoal(RegAll) then raise(ExReg)
endwhile
endagent
```

Fig. 6. Registration. Slave station

Note that individual agents participating in the process do not explicitly worry about agent registration. As for other adequately defined multi-agent systems, we can say that this agent society will achieve a more complex goal (i.e. consist registration) than the mere aggregation of individual agent's goals (i.e. perceiving a designated message and generating the corresponding answer). In other words, agent collaboration is a key issue in this system.

3.4 Heartbeat activity control

Activity control is performed through the cooperation of agents *BaseAct* and active *SlaveAct* placed in the consist. The procedure is similar to that of registration. In this case, *BaseAct*'s goal is to construct a list of active stations that matchs that of registered ones (see Fig. 7).

Slave stations' goal is to send back to the base station a *LAST* message. First, slave stations autonomously decide, after consulting their knowledge base, if they will send an alarm to the base station. This proactive action will trigger specific messages that will in turn modify the behaviour of the base station. Function

```
agent BaseAct(chan in, out, ctrl)
while (TRUE) do
    ctrl?SYNCBA;
    actl := []; SetGoal(CheckAct, {actl == list});
    out!TOK; in?(msg,id,pars); finish := FALSE;
    while (not(finish)) do
        case msg of
            ACT: actl := actl++[id]; ReportAlm(id, pars)
            LACT: actl := actl++[id]; ReportAlm(id, pars); finish := TRUE
            default finish := TRUE
        endcase endwhile
    if !CheckGoal(CheckAct) then raise(ExAct)
endwhile
endagent
```

Fig. 7. Activity control. Base station

ReportAlarm will affect the behaviour of the *Scheduler* at the base station and cooperating agents *BaseReg*, *BaseAct* and *BaseComm* through synchronization channel *ctrl*.

Then, *SlaveAct* states its position (last, middle) consulting the corresponding knowledge base. Its behaviour depends on this: a trailing station will not forward messages to the right, but generate a $LAST$ message itself.

3.5 Command Execution

Command execution is initiated by the base station through agent *BaseComm* when activated by the *Scheduler*. This latter agent decides to send a command to a designated slave station depending on its perceptions (engineer interaction thorugh the command interface, BDI status, etc.). Then, it activates *BaseComm* through the control channel.

Then, *BaseComm* fetches the command to be sent and discover the position of the addressee in the trainset. With this information, it sets as its goal the reception of the appropriate response message. Once the command is sent through the output channel, this agents fetches all incoming messages and classifies them as alarms, heartbeat messages or responses. Once the polling cycle is completed with the reception of a last-type message, the status of the goal is checked before becoming inactive again.

4 Conclusion

We have described a case study to illustrate the use of a multi-agent system to design and to model an on-board communication system for railway consists. Every element on the railway consist has been depicted as an agent that interacts with the rest of the components to satisfy an overall goal. Such agents are modeled using a simple belief, desires and intentions architecture.

This approach has several benefits with respect to other approaches. On the one side, implementation is drastically simplified. The agent society is composed of agents implemented as simpler pieces of code that interact through communication channels. Individual agent's goals are straightforward and easy to check, and the system functionality is a consequence of agent interaction and cooperation. Overall system goals are not explicitly implemented by agent's code (i.e. agents do not need the whole picture for the system to perform its duties). This reduces the requirements on the supporting platform. For example, slave stations could be implemented in a low-end microcontroller/single schip computer with reduced memory and computational power. On the other side, the modular approach together with agent orientation directly supports design for testability. Individual agents can be tested and validated independently, because goals, behaviour and communications for each agent are defined keeping at a minimum references to the global behaviour. Once individual components have been validated, global system testing can be specifically aimed to inter-agent communications, reducing the resources needed for testing. Besides, this model supports new approaches to testing like heuristic-driven testing [2].

In the case study presented, every agent coordinates its activities with the other agents to achieve the proposed goals: consist registration, heartbeat activity control and command execution. The overall system can be considered as an agent society wherein every agent only knows a piece of the global problem, which globally managed by means of communication and cooperation. In this context, we may see an example showing that the composed system has added value with respect to the simple aggregation of its parts.

References

1. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
2. J.C. Burguillo-Rial, M.J. Fernández-Iglesias, and Martín Llamas-Nistal. Heuristic-driven test case selection from formal specifications. a case study. In *Procs. of Formal Methods Europe 2002*, Lecture Notes on Computer Science. Springer Verlag, 2002. To appear.
3. M.J. Fernández-Iglesias and J.C. Burguillo-Rial. An agent model for a railway consist communication system. Technical report, Grupo de Ingeniería de Sistemas Telemáticos. Universidade de Vigo, ETS de Ingenieros de Telecomunicación. Campus Universitario S/N. 36200 Vigo (SPAIN), 2002. mailto:manolo@det.uvigo.es.
4. G. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
5. ISO. Lotos a formal description technique based on the temporal ordering of observational behaviour. International Standard, 1989.
6. Stuart Rusell and Peter Norvig. *Artificial Intelligence. A Modern Approach*, chapter 2, page 31. Prentice-Hall Inc., 1995.
7. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Konwledge Engineering Review*, 10(2):115–152, 1995.