Concurrent Negotiation Protocol for an Elevator Group Controller

Temporal Logic of Actions Used to Specify a Concurrent Negotiation Protocol within a Multi-agent System Controlling a Group of Elevators

Cristian Masalagiu and Vasile Alaiba

Faculty of Computer Science
University "Al.I.Cuza"
Gen. Berthelot 16, Iasi, Romania
mcristy@infoiasi.ro, alaiba@ozland.ro

Abstract. We have created a framework for the development of a multi-agent system to control a group of elevators. This control system, besides real time useful response, must fulfill stability, massive concurrency and correctness requirements. These lead to the necessity of a highly evolved inter-agent negotiation protocol. We introduced a "contract net" like protocol, enabling an unlimited number of concurrent bidding processes to occur at any time. The decentralized architecture and the uniform distribution of the decision makers allow good behavior even when the most severe breakdown occurs. It is outside the scope of our work to specify the way the agents are being programmed and also the specification of the proper behavior of the elevators being under control within the system. We focus on proving the consistency of the protocol specification and soundness properties: the protocol does not lead to inter-agent deadlock, every request is completed within a reasonable amount of time and the system is tolerant on failure. We also show that the system is capable of running an theoretically unlimited number of simultaneous requests and also does load-balancing between the agents. We used temporal logic of actions to specify the protocol. The actual specification makes use of the TLA+ language, and the various properties of the system were checked with TLC.

1 Introduction

The total control of complex real systems requires the use of more and more elaborate techniques. The new technologies to sensitive areas of real life, the need of stability, safety and soundness as desired properties for systems implies new responsibilities for designers. A design flaw can lead to serious accidents and loss of human lives. Thus the importance of formal specifications and modelling becomes obvious.

One of the most sensitive real systems is the elevator, considered as a means for (human) transportation inside a building. In this paper we build a framework for
the development of a multi-agent system used to control a group of elevators. This system must respect requirements as stability, robustness or massive concurrency and soundness. Additionally, it must provide useful "answers" using a limited amount of time. The necessity of developing an inter-agent negotiation protocol that can ideally guarantee all the requirements mentioned above seems to be desirable.

2 Temporal Logic of Actions and the TLA+ Specification Language

We chose the TLA, *temporal logic of actions* [LAM1] as the foundation for our protocol specification. TLA is a logic for specifying and reasoning about concurrent systems. It is a combination of temporal logic and the logic of actions, enriched with modularization and other useful features. A specification in TLA is a logical formula. The various properties of the specification can be given as TLA formulae as well. To prove that some property holds true we prove that a formula of the type "specification implies property" is true in some interpretation.

TLA formulae contain *variables* and *constant symbols*. The meaning, $\llbracket S \rrbracket$, of a formula $S$ is a boolean function on behaviors, where a behavior is an infinite sequence of *states* and a state is an *assignment* of variables to values.

**State functions** are non-boolean expressions built from variables and constant symbols. They correspond to expressions in ordinary programming languages. For example, if $x$ is a variable and $s$ a state, then $x + 1$ is a state function, that assigns the value $s[x + 1] = s[x] + 1$ to the state $s$. A postfix functional notation is used: if $f$ is a state function then $s[\llbracket f \rrbracket]$ denotes the value that $\llbracket f \rrbracket$ assigns to state $s$.

**State predicates** are boolean expressions built from variables and constant symbols. The meaning, $\llbracket P \rrbracket$, of a predicate $P$ is a mapping from states to tobooleans, so $s[\llbracket P \rrbracket]$ equals true or false for every state $s$. We say that a state $s$ *satisfies* a predicate $P$ iff $s[\llbracket P \rrbracket]$ equals true.

The pool of unfulfilled requests that will be used later is introduced as a finite set $req_a$. The initial state of $req_a$ can be given by the state predicate $req_a = \emptyset$. A state $s$ can be a initial state of the set above only if $s[\llbracket req_a = \emptyset \rrbracket]$ equals true. A primed variable denotes the new state of the corresponding unprimed variable. An action describes a relation (transition) between "old" states (that is, unprimed variables) and new states (primed variables). Actions are boolean-valued expressions formed with variables, primed variables and constant symbols. An atomic operation of a concurrent program will be represented in TLA by an action.

The meaning, $\llbracket A \rrbracket$, of an action $A$ is a relation between two states $s$ and $t$. The pair of states $s$ and $t$ is called an "$A$ step" iff $s[\llbracket A \rrbracket]t = true$ (that is, $s$ and $t$ are "related" by $\llbracket A \rrbracket$).

Let $r(l, j, t, id)$ a request labeled $id$ issued at moment $t$ by a passenger wishing to travel from level $l$ going up ($j = 1$) or down ($j = -1$). This request enters
into the pool for further processing. The action describing this step might be written: \( \text{req}_a' = \text{req}_a \cup \{ r(l, j, t, id) \} \).

We will not go further into details with temporal logic of actions. The reader may consult [LAMI] for a complete presentation of TLA logic.

*TLA*+ is a formal (programming) language based on TLA, first order logic and ZFC set theory [LAM3]. As a direct transcription of the TLA logic, it can be used to specify concurrent and reactive systems. It is very expressive, yet simple enough to be able to write easy to understand specifications. Besides the language itself some very useful tools are now available. Maybe the most valuable of all is TLC, a model checker for TLA+ specifications [LAM2]. Another one is the TLATeX Typesetter, a \LaTeX{} program for typesetting TLA+ modules [LAM3].

3 The Elevator Group Problem (EGP)

Scheduling a group of elevators is a very difficult real time decision problem. In fact, it is a multiplication of the dial-a-ride problem (DARP), proved to be NP-hard in [COJ]. Many variants of the elevator problem may be considered (including different services offered), as described in [KOE]. We chose to split the decision between the agents, in such a way that every one of them ”gets” a DARP problem, and ”emits” an estimative cost for the request, based on their experience and knowledge. In our approach we shall not consider auxiliary services the system may be required to offer, such as non-stop travel. We focus on the inter-agent protocol and the way it can guarantee a real time response of the system, independently of the individual behavior.

3.1 Problem Statement.

Let \( L \) be the nonvoid, finite set of levels of a building, and \( E \) the set of elevator (cages). Let \( J = \{ \text{up}, \text{down} \} \) be the set of movement directions and \( T \) a discrete set of moments of time equally dispersed (\( \forall t, t' \in T, |t' - t| = c \) a fixed constant).
The requests addressed to the system are tuples over $\mathcal{R}_a = \mathcal{L} \times \mathcal{J} \times \mathcal{T} \times \mathcal{I}$, where $\mathcal{L}, \mathcal{J}, \mathcal{T}$ as defined before and $\mathcal{I}$ a set of natural numbers. The tuple $r(l, j, t, id)$ is created when the system is demanded to transport some person(s) from level $l$, with direction $j$. The moment of occurrence is stored in $t$. We choose a different natural number $i \in \mathcal{I}$ to be assigned to each request at the moment of its occurrence. Simultaneous negotiation of many different requests may thus take place without confusions.

A request $r(l, j, t, id)$ is considered fulfilled when an elevator $e \in \mathcal{E}$ controlled by the corresponding agent $a \in \mathcal{A}$ stops at level $l$ to load and transport in the given direction $j$. Let $\mathcal{R}_f = \mathcal{R}_a \times \mathcal{A} \times \mathcal{T}$ be the set of fulfilled requests. If $r \in \mathcal{R}_a$ then $rf(r, a, t_f) \in \mathcal{R}_f$, where $t_f$ is the moment when $r$ gets fulfilled and $a \in \mathcal{A}$ is the agent controlling the elevator $e$.

The goal of the system is to fulfill all requests while minimizing the difference between the creation and fulfill moments (minimize the value $|t_f - t|$ for each request). Another period of time to minimize is the duration of the travel - the time spent by a person inside the elevator, but this can be solved only at the agent level and can be ignored at the protocol level.

4 Our approach to solving the EGP

4.1 Multi-Agent Systems

Our idea for solving the described problem was a multi-agent system. Thus, every elevator is controlled by an ”intelligent” agent. The agents used in our system have the following properties [WOO]:
1. autonomy;
2. social ability (through negotiation);
3. reactivity (to requests);
4. pro-activeness (they can take their own decisions, e.g. in negotiations) and
5. learning.

They also respect the following conditions:

1. veracity – an agent will not knowingly communicate false information;
2. rationality – an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved – at least insofar as its beliefs permits;
3. they share a common goal – to minimize the waste of people’s time due to the use of the elevators.

Let $\mathcal{A}$ be the (finite, nonvoid) set of agents in the system such as every agent $a \in \mathcal{A}$ has associated an unique elevator $e \in E$ (there exists a one-to-one correspondence between $\mathcal{A}$ and $E$). We assume that all agents have the above properties, and that they use our protocol for negotiation of the requests (described in the next section).

### 4.2 The Inter-Agent Negotiation Protocol (IANP)

A typical negotiation scenario begins with the occurrence of a request $r(l, j, t, id)$. All the agents are notified about this event but, at any time, only one of the agents has the flag. The flag dictates the leader agent of the current bidding process.

Now, let $a$ be the leader agent. Immediately after the request $r$ is received, $a$ passes the flag to another agent $a'$. Thus, as we have pointed out, $a$ is only the
leader of the current bidding process, allowing $a'$ be the leader of the following one. Next, $a$ broadcasts its offer to all agents.

An offer is a tuple $o(a_1, a_2, r, c) \in \mathcal{O} = A^2 \times R_x \times C$, where $a_1$, $a_2$ are agents, $r$ is the corresponding request, and $c$ is the cost. The cost is an estimate of the time agent $a_1$ needs in order to fulfill the request $r$. The offer is sent by from $a_1$ to $a_2$.

All the agents are aware of all the requests. After the leader $a$ broadcasts its offer, the agents having a better estimate cost send their offers back to it. $a$ waits a fixed amount of time $dt$ to collect offers. When the bidding related to the request $r$ is over (i.e. the current time exceeds $t + dt$) $a$ elects the agent with the best offer $b$ and sends it a confirmation message: assign $(a, b, r)$. If none exists then it will assign the request to itself.

The described protocol is summarized in Table 1. It represents in fact a generic state machine, which describes the general behaviors of our multi-agent system (only the macro-issues are taken into account). Any information regarding the internal state of the agents – necessary to describe the protocol – is introduced using assertions. For example, to express the fact that agent $a$ is the flag carrier we state that only $\text{flag}(a)$ is true; that is, $\text{flag}(a) \land \forall x \in \mathcal{A} : (x \neq a) \land \neg \text{flag}(x)$.

<table>
<thead>
<tr>
<th>Event</th>
<th>Allowed if</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r(l, j, t, id)$</td>
<td>$\forall x \in \mathcal{A} : \text{sendRequest}(x, r)$</td>
<td></td>
</tr>
<tr>
<td>$o(a, x, id, c)$</td>
<td>$\text{flag}(a) \land x \neq a \land \exists r \in \mathcal{R}_a : r.id = o.id$</td>
<td>$\text{leader}(a, id) \land \exists a' \in \mathcal{A} : a' \neq a \land \text{sendFlag}(a')$</td>
</tr>
<tr>
<td>$o(a, a, id, c_i)$</td>
<td>$\exists o(a, a_i, id, c) \in \mathcal{O}, r \in \mathcal{R}_a : r.id = o{id} \land a_i &lt; c \land now &lt; t_id + dt$</td>
<td>$\text{addBid}(o, id)$</td>
</tr>
<tr>
<td>$\text{assign}(a, b, id)$</td>
<td>$\text{leader}(a, id) \land (now &gt; t_id + dt) \land b \in \mathcal{A} : o(b).c = \min{o.c : o \in \mathcal{O}_id}$</td>
<td>$\text{fulfill}(id, b, now)$</td>
</tr>
</tbody>
</table>

Table 1. The Inter-agent negotiation protocol

From Table 1 it is obvious that an initial state for the system is needed. Thus, at the beginning no requests are issued and none are fulfilled; consequentially, no offers have been transmitted, i.e. $\mathcal{R}_a = \emptyset \land \mathcal{R}_f = \emptyset \land \mathcal{O} = \emptyset$. There is only one $\text{flag}$ agent and no leader since no bidding processes are running. We give the state of the system before any event occur (the initial state) be the following formula: $(\exists a \in \mathcal{A} : \text{flag}(a)) \land \forall x \in \mathcal{A} : x \neq a \land \neg \text{flag}(x)) \land (\forall id \in \mathcal{I}, x \in \mathcal{A} : \neg \text{leader}(x, id))$.

4.3 Desirable Properties

It is well-known that important problems related to the behaviour of concurrent systems may be expressed in terms of fairness, liveness and deadlock. In fact, the proposed protocol has to guarantee the fairness, liveness and absence of deadlock for the elevator multiagent system.
Fairness and Liveness. There are many ways in which fairness/liveness may be expressed ([LPS]). We shall say that our system is fair and alive if no action can be always enabled but never taken. Applying our protocol we have to get a response to every request. This will cause the introduction of real time restrictions into the specifications and into the protocol itself. The activation predicate of the third and fourth lines in Table 1 contain the special variable now, designating the actual value of the real time. This way their values change over time on sequential evaluations and force the transition of the system from one state to another at specific moments of time. The time reference used is $t_{id}$, that is, the moment of the occurrence of the request identified with id. As long as the leader agent keeps running, the system will generate a response after the $dt$ period of time expires.

Absence of deadlock. In our framework, this means that "no matter what the behavior of the system is, it must change states". An inter-agent deadlock can thus be characterized by the absence of action. Let Next be the formula describing the whole evolution of the system. To prove the absence of deadlock the specification must satisfy the invariance property $\Box(\text{enabled } \text{Next})$. The property, which uses the temporal operator "box" ($\Box$), states that a Next step is always possible, i.e. there can be no infinite sequence of stuttering steps. We do not insist on other general properties of the system such as termination (irrelevant in our case), or correctness (again irrelevant if we accept that all the elements of our multi-agent system are correctly defined).

5 TLA+ Specification of EMAS

A TLA+ specification of a system is mainly a formula. One can group several formulas into a module to increase readability. We shall split our specification into three modules: MLSets, IAgent, MLProtocol. We describe them in the following subsections.

5.1 Sets involved and their properties

We begin by fixing the sets of constants to be used throughout. Notice that we do not have to specify the variables as sets because TLA+ is based on ZFC set theory. The TypeInvariant introduced here will be used later to check the consistency of the specification.

```
MODULE MLSets

EXTENDS Naturals, Reals, FiniteSets

CONSTANTS L, J, T, I, A, C, Dt
ASSUME ValAssump \[ \Delta \wedge L \subseteq \text{Nat} \]
```
\begin{align*}
\land J &= \{-1, 1\} & \text{Directions: } -1 = \text{DOWN}, 1 = \text{UP} \\
\land T &\subseteq \text{Reals} & \text{Time} \\
\land I &\subseteq \text{Nat} & \text{Identifiers} \\
\land \text{IsFiniteSet}(A) & & \text{Agents} \\
\land C &\subseteq \text{Reals} & \text{Cost} \\
\land Dt &\in \text{Reals} & \text{Bidding process time length}
\end{align*}

**VARIABLES**
- `req_a`, The set of addressed requests
- `req_f`, The set of fulfilled requests
- `.offer`, The set of offers passed by between the agents

Will be used to check the consistency of the specification

\[
\text{MLTypeInvariant} \triangleq \land \text{req}_a \subseteq [l : L, j : J, t : T, i : I] \\
\land \text{req}_f \subseteq [l : L, j : J, t : T, i : I, a : A, tf : T] \\
\land \text{offer} \subseteq [a1 : A, a2 : A, i : I, c : C]
\]

The set of issued requests from the level elevator buttons

\[
\text{Req}_i \triangleq [l : L, j : J, t : T]
\]

A tuple with all the variables in the specification

\[
\text{vars} \triangleq \langle \text{req}_a, \text{req}_f, \text{offer} \rangle
\]

### 5.2 Interface to any agent

The assertions needed to be made about agents are grouped in the \textit{IAgent} module. The formulae actually describe the interactions of our system with the outside universe. We say that \textit{an agent runs correctly} if it satisfies all the \textit{IAgent} formulae.

---

**MODULE** \textit{IAgent} specifies the interface to an Agent

**EXTENDS** \textit{MLSets}  

**CONSTANTS** \textit{Eval}  

**ASSUME** \textit{ValAssume} \triangleq \textit{Eval} \subseteq A \times [\text{Req}_i \rightarrow C]

---

**VARIABLES**
- `flag`, indicates the flag agent
- `leader`, indicates the leader agent of a bidding process
- `know`, the set of requests known to an agent

\[
\text{IATypeInvariant} \triangleq \land \text{flag} \in [A \rightarrow \text{BOOLEAN}] \\
\land \text{leader} \subseteq A \times I \\
\land \text{know} \subseteq A \times \text{Req}_i
\]
sendRequest($x, r, id$) $\triangleq \land x \in A$
$\land r \in Req_i$
$\land id \in I$
$\land req\_a' = req\_a \cup \{[r \text{ EXCEPT } !.id = id]\}$

$\text{makeLeader}(a, r)$ $\triangleq \land a \in A$
$\land r \in req\_a$
$\land \text{flag}(a)$
$\land \text{leader}' = \text{leader} \cup \langle a, r.id \rangle$

$\text{passflag}(a)$ $\triangleq \text{choose ap} \in A :$
$\land ap \neq a$
$\land \text{flag}' = \text{flag EXCEPT } ![a, ap] = \text{FALSE, TRUE}$

### 5.3 The specification of IANP

The $\text{MLProtocol}$ module introduces the inter-agent negotiation protocol as described in Section 4.2. The consistency of the specification is checked using the $\text{TypeInvariant}$ defined in the $\text{MLSets}$ module: \textsc{theorem} $\text{MLProtocol} \Rightarrow \text{TypeInvariant}$.

\begin{verbatim}

MODULE MLProtocol

EXTENDS MLSets, IAgent

\end{verbatim}

---

Initial state of the system
\textsc{MLInit} $\triangleq \land \text{TypeInvariant}$
$\land \text{req}\_a = \{\}$
$\land \text{req}\_f = \{\}$
$\land \text{offer} = \{\}$
$\land \text{choos}e\ a \in A : (\text{flag}(a) \land (\forall x \in A : (x \neq a \land \neg\text{flag}(x))))$
$\land \forall id \in I, x \in A : \neg\text{leader}(x, id)$

\textsc{Request}(r) $\triangleq \forall x \in A, ra \in \text{req}\_a, rf \in \text{req}\_f :$
\text{choose id} \in I :
$\land \text{id} \neq ra.id$
$\land \text{id} \neq rf.id$
$\land \text{req}\_a' = \text{req}\_a \cup \{[r \text{ EXCEPT } !.id = id]\}$
$\land \text{sendRequest}(x, r, id)$

\textsc{LeaderBid}(a, r) $\triangleq \land \text{makeLeader}(a, r)$
$\land \text{passFlag}(a)$
$\land \forall x \in A : (x \neq a \land \text{sendBid}(a, x, r))$

\textsc{Next} $\triangleq \lor \exists r \in \text{Req}_i : \text{Request}(r)$
$\lor \exists a \in A, r \in \text{req}\_a :$
\[\begin{align*}
\land \text{flag}[a] \\
\land \text{LeaderBid}(a, r) \\
\lor \exists o_1, o_2 \in \text{offer}, r \in \text{req}_-a : \\
\land o_1.id = r.id \\
\land o_2.id = r.id \\
\land \text{now} < r.t + Dt \\
\land \langle o_2.a1, r.id \rangle \in \text{leader} \\
\land o_1.a2 = o_2.a1 \\
\land o_1.a1 = o_2.a2 \\
\land \text{AddBid}(o_2, id) \\
\lor \exists a \in A, r \in \text{req}_-a : \\
\land \langle a, r.id \rangle \in \text{leader} \\
\land \text{now} > r.t + Dt \\
\land \exists o \in \text{offer} : o.a2 = a \land o.c = \min\{o.c : o \in \text{offer} \land o.id = r.id\} \\
\land \text{Fulfill}(r, o.a1, \text{now})
\end{align*}\]

\[\text{MLProtocol} \triangleq \text{MLInit} \land \Box[\text{Next}]_{\text{vars}}\]

\textbf{6 Conclusions}

Temporal logic of actions is a powerful model for specifying real systems. This is of paramount importance especially if the systems are very sensitive, highly complex or involving high risks. We choose a problem that is both complex and risky, and offered a possible solution.

The multi-agent systems are autonomous, reactive and pro-active communities of artificial intelligence entities. Making them appropriate for real life industrial usage proves by itself a challenge, but the outcome is promising.

We focus on an inter-agent negotiation protocol for a multi-agent system addressing the elevator group problem. We specified the protocol using the temporal logic of actions and were able to prove that it is consistent and generally correct. More exactly we studied the possibility of inter-agent deadlock and the reactivity of the system.

The multi-agent solution to the EGP problem remains open to further research. Our protocol should be formally checked and enhanced to better support random agent failure. Additional services could be introduced and many additions could be necessary. The specification of the agents involved can be another difficult - yet promising - problem.

As the specification gets more and more complete and formally checked a real implementation of the system will be considered. At present we are experimenting with a computer simulation of the multi-agent system based on the protocol described in this paper. Yet the results are far from being complete and further development will continue.
References


