SYMBOLIC VERIFICATION OF A STRENGTH BASED MULTI-AGENT ARGUMENTATION SCHEME

Shravan Shetty, Shashi Kiran H.S, Murali Babu Namala, and Sanjay Singh

Department of Information & Communication Technology, Manipal Institute of Technology, Manipal-576104, India

ABSTRACT

Software systems have evolved to the age of Artificial Intelligence (AI), consist- ing of independent autonomous agents interacting with each other in dynamic and unpredictable environments. In this kind of environment it is often very difficult to predict all the interactions between the agents. Hence verification of an interaction between multiple agents has become a key research area in AI. In this paper we model and verify an Automatic Meeting Scheduling (AMS) problem having multiple agent communication.

The AMS problem helps us to emulate a real life scenario where multiple agents can argue over the defined constraints. A weighted strength based argu- mentation scheme is proposed, where each argument is weighed against each other to determine the strongest evidence. The argumentation model described in the paper have six agents: Initiator, Scheduler and four Participant agents. We have formalized the agent interactions using Computation Tree Logic (CTL) and verified the scheme by providing suitable specifications (SPEC) in a symbolic model verifier tool called, NuSMV.

1 INTRODUCTION

There have been tremendous advances in the area of agent based argumentation systems. In such schemes, a common consensus is drawn from an interaction or argumentation between different entities called Agents. Currently agents can be considered as a component with independent behavior. Interactions and argu- mentation can be termed as agent communication. There is however a subtle difference between interaction and argumentation:

– Interaction can be a simple message passing scheme where the agents are exchanging knowledge with each other.

– Argumentation is a complex behavior where each of the agents will refute the other agent by providing its own evidence for a situation.

However both the communication types have a common result oriented approach; commonly termed as Desire in Belief, Desire and Intention (BDI Logic) [1]. This was originally developed so as to emulate human practical reasoning [3]. Agent knowledge can be classified as:

– Internal: Internal knowledge is specific to one agent and its view of reaching the overall system objectives. This is equivalent to Belief of the BDI logic [1].

– Global: Global knowledge is shared by all agents in the system. The overall system objectives is known by each of the agent, hence it becomes part of the global knowledge.

Typically an argumentation takes place based on the individual internal or local knowledge. In an attempt to model such an argumentation scheme, we have introduced a novel concept of argument strength, which will eventually decide the outcome of an argument.

During the last two decades, there have been considerable research on agent interaction as it plays a pivotal role in the area of artificial intelligence. Argu- mentation scheme is a key research area in majority of the upcoming intelligent applications as there is an increasing need of independent system modules to reach a common goal. This argumentation not only makes the interaction effi- cient but also improves the outcome of the dialogue between the agents. Katie Atkinson [9] has given an argumentation scheme based on Command Dialogue Protocol

DOI : 10.5121/ijaia.2011.2210

(CDP). This scheme uses dialogue interactions between commander- agent and the receiver-agent. An argumentation in this scheme is based on the evidence that is provided. In our paper we employ a similar approach but rely on the strength of an argument for verification and simulation via model checking.

Similarly in [2] presented a dialogue framework that allows different agents with some expertise to inquire about the beliefs of others and also share the knowledge about the effect of actions. In this they have given two argumentation models which allows defeasible reasoning about what to believe and defeasible reasoning about what to do. In our approach, we do not consider that any of the agent know about the belief of others. Each agent has an independent view of the overall objective, so we have considered that each of the interaction in an argument, to have a strength associated with it. This enables us to determine the agent with the strongest argument. The evidence or the knowledge provided by this agent can be accepted.

We put forward a multi-agent modeling scheme using Argument Strength. We illustrate this argumentation scheme using a multi-agent meeting scheduling problem. We have formalized the agent interactions using Computation Tree Logic (CTL) [7] and verified the scheme by providing suitable specifications (SPEC) in a symbolic model verifier tool called, NuSMV [6]. The user manual for using this tool can be found in [5].

The remaining paper is organized as follows. Section 2 gives a theoretical background of agents and their interaction schemes. Section 3 gives the method- ology employed in the AMS problem and its constraints. Section 4 expose us to a model checking environment used to verify the specifications identified. The verification of the model developed using NuSMV along with its trace results are mentioned in Section 5. Finally conclusion and areas of future work has been drawn in Section 6.

2 THEORETICAL BACKGROUND

2.1 Agents

An Agent is a component capable of performing independent actions. It can be thought of as a simple software module or even as complicated as a robot with artificial intelligence. Every agent is associated with attribute, behavioral rules, small memory resources, decision making sophistication and rules to modify behavioral rules [8]. An agent is autonomous and can function independently in its environment and in its interactions with other agents where as the objective of all agents are considered to be the similar. Agents represent entities in a model and agent relationships represent entity interaction.

2.2 Agent Based Modeling

Agent Based Modeling (ABM) is synonymous to Individual Based Modeling (IBM) [8]. We can consider any interaction of two modules as an agent com- munication. The typical sender-receiver model in computer networks can be modeled as two agents in itself and the message passed between them can be modeled as an agent communication.

The Buyer-Seller model [4] serves as an example for explaining a simple argumentation scheme. The seller-agent tries to sell a car based on the features available e.g. air bag. Both buyer and seller-agent have a global knowledge about existence of an air bag and its use. The internal knowledge of the seller would be the research technicalities about the air bag performance, whereas the buyers internal knowledge would be limited to articles in newspapers. However we can consider the sale of the car as the common goal. The interaction between the two agents can be modeled as an argument over the quality and need of an airbag. An argument ends when both the agents are convinced by the evidence provided, leading to the goal or desire as in the BDI model [1]. An argument posed by each is weighted against each other in the strength based scheme. For instance, the strength of a seller-agent argument is more, as research data has empirical proof over its claim as compared to the newspaper as that of the buyer-agent.

2.3 Applications with Multi-Agent Communication

An argumentation in the real world is much complex as independent behaviors of each agent is included. Multi-Agent modeling is being used in wide applica- tion domains. One critical areas is the Air Traffic Control (ATC), where various agents communicate with each other with a common objective of safe and se- cure air traffic handling. They also evaluate the performance of an air traffic management system by interacting with each other.

Another pivotal role of multi-agent system is in Biomedical research. An agent based model is used in research work related to biological science for e.g. immunity, health etc. Agents in Market analysis can communicate stock prices and can be used to predict share prices based on relevant past knowledge on an argumentation framework.

3 METHODOLOGY

To understand an Agent interaction scheme, we have identified a Multi-Agent meeting scheduler problem [10]. This scheme will have many agents interacting with each other to setup a meeting. It can be visualized as a real life meeting, where an agents such as initiator, scheduler and the various participants of the meeting will look forward to schedule a meeting based on individual constraints. For initiating the meeting the initiator-agent might pick prerequisites like the date, time, location and the resources needed. This is supplied to the scheduler without keeping in mind any other conflicting meeting. The conflict can be with the unavailability of any of the prerequisites of the meeting. Under such conflicts, an argumentation takes place between the agents. We propose a scheme for modeling such argumentation based on the Strength of an Argument.

Argumentation is relatively easy to model when one of the argument is true and the other is false. We can consider the outcome of an argument based on the true argument. But when we consider a scheme where both the arguments are true, it is often difficult to model. Here we consider strength of an arguments for modeling the right outcome. In the Initiator-Scheduler argumentation, the strength of an argument by the scheduler-agent with respect to the meeting constraints are more as compared to the initiator-agent.

Table 1. Notations Used

Sl.No.	Notation	Entity
1	I	Initiator-agent
2	S	Scheduler-agent
3	P	Participant-agent
4	I argument	Initiator-argument
5	S argument	Scheduler-argument
6	P argument	Participant-

The meeting scheduling problem [10] gives us a real life scenario where an agent based model can be used in order to schedule a meeting. An argumentation based on individual agent knowledge is identified and strength associated for each is considered for modeling. The agents included in the AMS scenario are:

– Initiator

– Scheduler

– Participants

Ideally there can be many participants taking part in the meeting, we consider only the following to understand a simple strength based argumentation scheme:

– Active Participant: Can be considered as the chair of the meeting.

– Important Participant: Can be considered as Active participant's boss.

– Audience1 and Audience2: Participants who are part of the audience.

The Table 1 gives the various notations used henceforth, to identify each agent or their communication.

The interaction between the agents are as shown in Fig.1. I sends a request for the meeting by sending a message to S. S communicates between an I and P. It is responsible for establishing a meeting based on the responses of each participants.

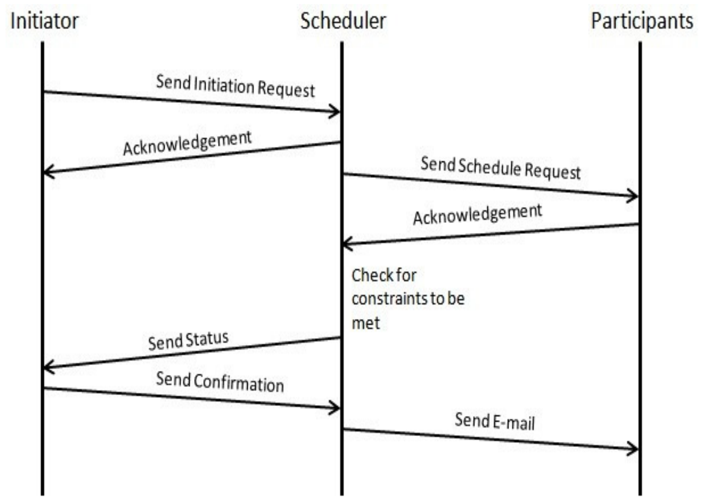


Fig. 1. Agent Interaction Diagram

An argumentation is modeled over the meeting constraints such as date, time, location and availability of other resources (e.g. projectors, pointers etc). There will be an argument between any of the two agents whenever there is a conflict in the meeting constraints. For instance, the initiator-agent might pick a meeting date without looking into any conflicting constraints. The scheduler- agent checks the date and passes an argument to refute the initial request. The strength of the S argument can be considered to be more than that of the I argument. An argument between the S and P are also similar. The strength of P argument over the meeting constraints is more than that of I or S. The pseudo-code in Algorithm 1 explain the argumentation scheme. Line 1 is used to initiate the meeting with the constraints date, venue and the intended participant list. These details are sent to S, any argumentation hence formed is weighed against each other and the stronger argument is accepted as shown in Line 3. The argumentation scheme proceeds only if the I argument is stronger, otherwise the meeting cannot be scheduled. The S argument is modeled to be stronger in situation such as unavailable dates or meeting rooms (venue). Line 7 shows a possible argumentation between S and P, a corresponding acceptance in Line 8. If P argument is stronger then the meeting is canceled. This situation can be considered when the participants are busy, the venue is far, etc. Algorithm 2 is used to assess any two arguments between agents by referring to Algorithm 3, which computes the strength of each argument. The argument having higher score will equivalently have a higher strength.

Algorithm 1 ScheduleMeeting(*I*, *S*, *P*)

```
 1: Initiate(Date, Venue, Participant List)
 2: for each argument exchanged between I and S do
 3:   Accepted←AssessStrength(I argument, S argument)
 4: end for
 5: if I argument is accepted then
 6:   send Schedule request to all participants
 7:   for each argument exchanged between S and P do
 8:     Accepted←AssessStrength(S argument, P argument)
 9:   end for
10:   if S argument is accepted then
11:     send meeting request to the participants
12:   else
13:     cancel the meeting
14:   end if
15: else
16:   cancel the meeting
17: end if
```

Algorithm 2 AssessStrength(*Argument1*, *Argument2*)

```
1: CalculateStrength(Argument1, Argument2)
2: Accept Argument with higher score
3: return Accepted Argument
```

4 MODEL CHECKING

Over the years, model checking has evolved greatly into the software domain rather than being confined to hardware such as electronic circuitries. Model checking is one of the most successful approach to verification of any model against formally expressed requirements. It is a technique used for verifying finite state transition system. The specification of system model can be formalized in

Algorithm 3 CalculateStrength(*Argument1*, *Argument2*)

```
1: Score1 := Argument1.agent_weight + $\sum_{i=1}^{N} ConstraintWeight_i$
2: Score2 := Argument2. _weight + $\sum_{i=1}^{N} ConstraintWeight_i$
   /* where N : is the number of constraints */
3: return max(Score1, Score2)
```

temporal logic, which can be used to verify if a specification holds true in the model.

Model checking has a number of advantages over traditional approaches which are based on simulation, testing and deductive reasoning. In particular, model checking is an automatic, fast tool to verify the specification against the model. If any specification is false, model checker will produce a counter-example that can be used to trace the source of the error.

We have modeled the multi-agent meeting schedule and verified against few specifications in NuSMV. We chose NuSMV because it allows easy tracing of the state variables. This is equivalent to a debugging process in any program- ming language like C or Java. We have considered weights for each agent, which randomly picks a value from a range. The strength of an argument can then be assessed based on the higher score considering the meeting constraints. A simple method would be to assign range values to each constraint and a weight for each agent. The overall argument strength can be modeled as a score equal to the product of the agent weight and the sum of the constraint weights. We have considered higher weight for the participant-agents.

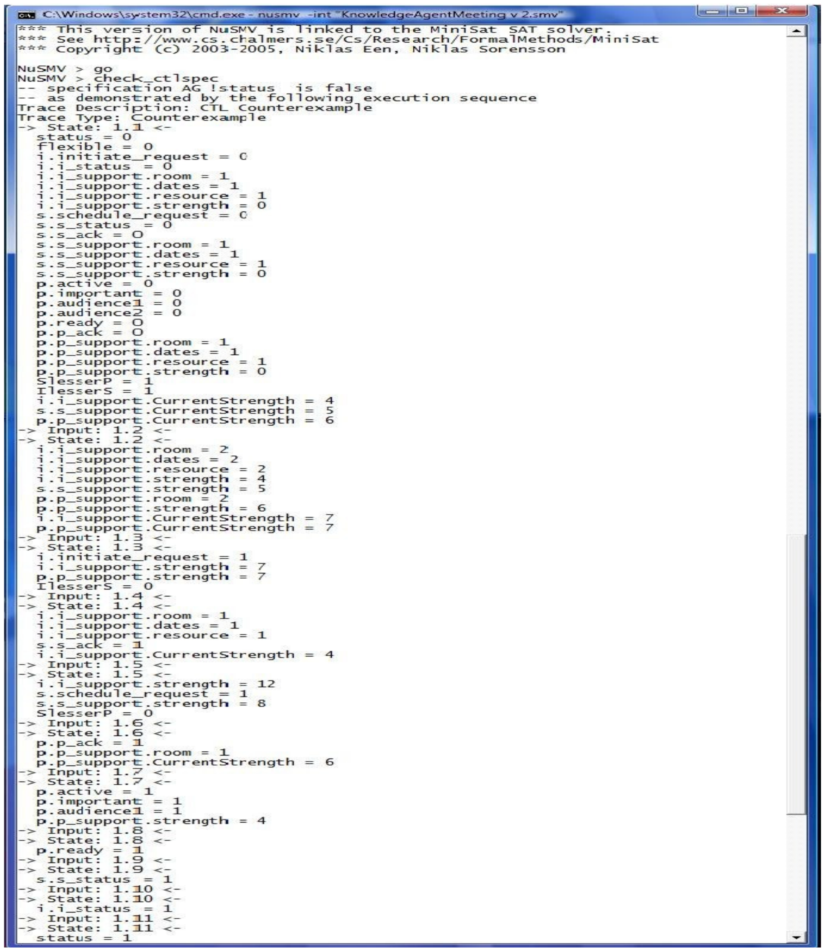
We have modeled two types of meeting scheduler:

– Static meeting schedule

– Dynamic meeting schedule

A static schedule is one where the meeting is fixed as soon as the meeting constraints are met. Confirmed participants cannot change the decisions, but the non-confirmed participants can decide to attend the meeting after the meeting is fixed too.

In case of dynamic schedule, the meeting is fixed just before the date if the constraints are met. The participants can change their responses till the date issued by the scheduler. The conformance with the constraints are checked only just before the meeting date.



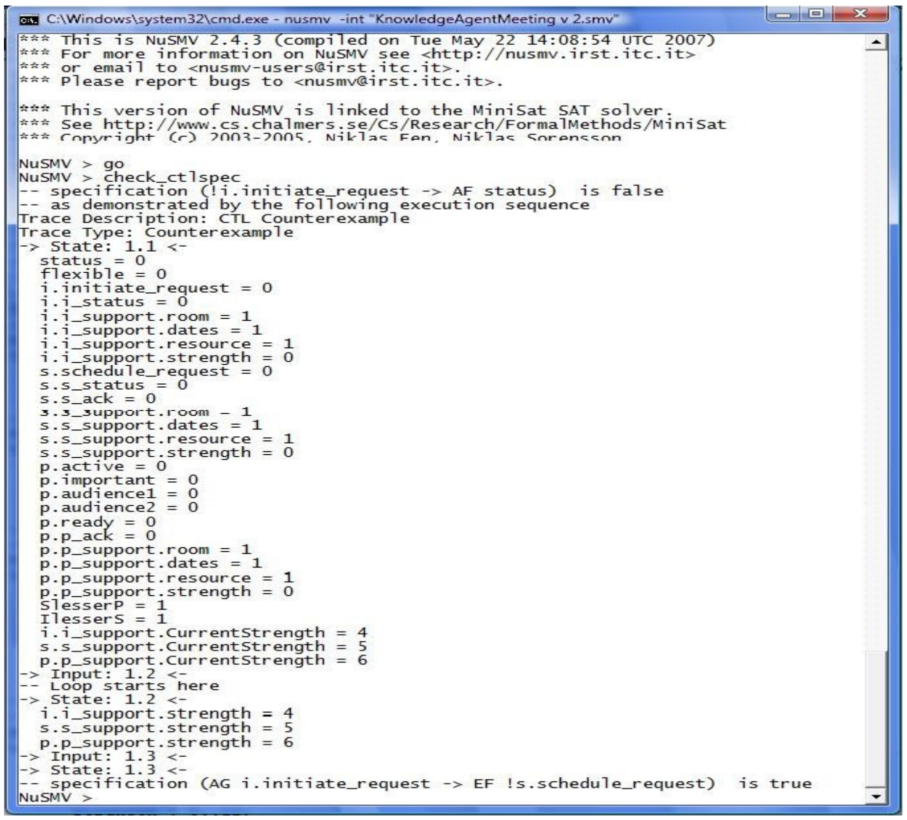
```
C:\Windows\system32\cmd.exe - nusmv  -int "KnowledgeAgentMeeting v 2.smv"
*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > check_ctlspec
-- specification AG !status  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  status = 0
  flexible = 0
  i.initiate_request = 0
  i.i_status = 0
  i.i_support.room = 1
  i.i_support.dates = 1
  i.i_support.resource = 1
  i.i_support.strength = 0
  s.schedule_request = 0
  s.s_status = 0
  s.s_ack = 0
  s.s_support.room = 1
  s.s_support.dates = 1
  s.s_support.resource = 1
  s.s_support.strength = 0
  p.active = 0
  p.important = 0
  p.audience1 = 0
  p.audience2 = 0
  p.ready = 0
  p.p_ack = 0
  p.p_support.room = 1
  p.p_support.dates = 1
  p.p_support.resource = 1
  p.p_support.strength = 0
  SlesserP = 1
  IlesserS = 1
  i.i_support.CurrentStrength = 4
  s.s_support.CurrentStrength = 5
  p.p_support.CurrentStrength = 6
-> Input: 1.2 <-
-> State: 1.2 <-
  i.i_support.room = 2
  i.i_support.dates = 2
  i.i_support.resource = 2
  i.i_support.strength = 4
  s.s_support.strength = 5
  p.p_support.room = 2
  p.p_support.strength = 6
  i.i_support.CurrentStrength = 7
  p.p_support.CurrentStrength = 7
-> Input: 1.3 <-
-> State: 1.3 <-
  i.initiate_request = 1
  i.i_support.strength = 7
  p.p_support.strength = 7
  IlesserS = 0
-> Input: 1.4 <-
-> State: 1.4 <-
  i.i_support.room = 1
  i.i_support.dates = 1
  i.i_support.resource = 1
  s.s_ack = 1
  i.i_support.CurrentStrength = 4
-> Input: 1.5 <-
-> State: 1.5 <-
  i.i_support.strength = 12
  s.schedule_request = 1
  s.s_support.strength = 8
  SlesserP = 0
-> Input: 1.6 <-
-> State: 1.6 <-
  p.p_ack = 1
  p.p_support.room = 1
  p.p_support.CurrentStrength = 6
-> Input: 1.7 <-
-> State: 1.7 <-
  p.active = 1
  p.important = 1
  p.audience1 = 1
  p.p_support.strength = 4
-> Input: 1.8 <-
-> State: 1.8 <-
  p.ready = 1
-> Input: 1.9 <-
-> State: 1.9 <-
  s.s_status = 1
-> Input: 1.10 <-
-> State: 1.10 <-
  i.i_status = 1
-> Input: 1.11 <-
-> State: 1.11 <-
  status = 1
```

Fig. 2. Screen shot depicting the NuSMV implementation of CTL SPEC No. 2

5 NuSMV SIMULATION RESULTS

5.1 Specifications

Like in most object oriented programming languages, NuSMV also uses . (dot) operator to access state variables of a module instance. For example in our im- plementation, i is an instance of the module Initiator. The state variables of the module Initiator can be accessed using i. (state variables). Hence i.i support.strength



```
C:\Windows\system32\cmd.exe - nusmv -int "KnowledgeAgentMeeting v 2.smv"
*** This is NuSMV 2.4.3 (compiled on Tue May 22 14:08:54 UTC 2007)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > check_ctlspec
-- specification (!i.initiate_request -> AF status)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  status = 0
  flexible = 0
  i.initiate_request = 0
  i.i_status = 0
  i.i_support.room = 1
  i.i_support.dates = 1
  i.i_support.resource = 1
  i.i_support.strength = 0
  s.schedule_request = 0
  s.s_status = 0
  s.s_ack = 0
  s.s_support.room = 1
  s.s_support.dates = 1
  s.s_support.resource = 1
  s.s_support.strength = 0
  p.active = 0
  p.important = 0
  p.audience1 = 0
  p.audience2 = 0
  p.ready = 0
  p.p_ack = 0
  p.p_support.room = 1
  p.p_support.dates = 1
  p.p_support.resource = 1
  p.p_support.strength = 0
  SlesserP = 1
  IlesserS = 1
  i.i_support.CurrentStrength = 4
  s.s_support.CurrentStrength = 5
  p.p_support.CurrentStrength = 6
-> Input: 1.2 <-
-- Loop starts here
-> State: 1.2 <-
  i.i_support.strength = 4
  s.s_support.strength = 5
  p.p_support.strength = 6
-> Input: 1.3 <-
-> State: 1.3 <-
-- specification (AG i.initiate_request -> EF !s.schedule_request)  is true
NuSMV >
```

Fig. 3. Screen shot depicting the NuSMV implementation of CTL SPEC No. 3 and 4

Table 2. Specifications for the Meeting schedule

Sl.No.	Specification	Satisfiability
1	(AG i.initiate request → EF	True
2	AG(!status	False(Counter-
3	(!i.initiate request → EF	False(Counter-
4	(AG i.initiate request→EF	True

implies the state variable strength of the instance i support which in turn is a state variable of the instance i. The Table 3 shows the list of variables with their corresponding meaning.

We have verified the multi-agent scheme by many suitable specifications, few of them has been listed in Table 2. Each of the specification is explained below:

Table 3. List of State Variables used

Sl.No.	Variable	Meaning
1	i	Instance of the module
2	s	Instance of the module
3	p	Instance of the module
4	i.initiate	Initiator request flag
5	s.s ack	Scheduler acknowledgment flag
6	s.schedule	Scheduler request flag
7	p.p ack	Participant acknowledgment
8	p.ready	Participant ready flag

– AG i.initiate request → EF status

This checks the property that when a meeting is initiated, does there exist a path such that the meeting is fixed? This returns TRUE.

– AG(!status)

This specification is used to simulate a trace of all the states for which the meeting gets fixed. The property can be interpreted as, is it always the case that the meeting is fixed? This specification is FALSE and a counter-example simulation is generated by NuSMV showing the state transitions needed for fixing the meeting.

– !i.initiate request → EF status

This specification verifies the property, whenever a meeting is not initiated, is there any path that will fix the meeting? This is FALSE and hence generates a counter-example but in no path the status of the meeting is set to true when the meeting is actually not initialized.

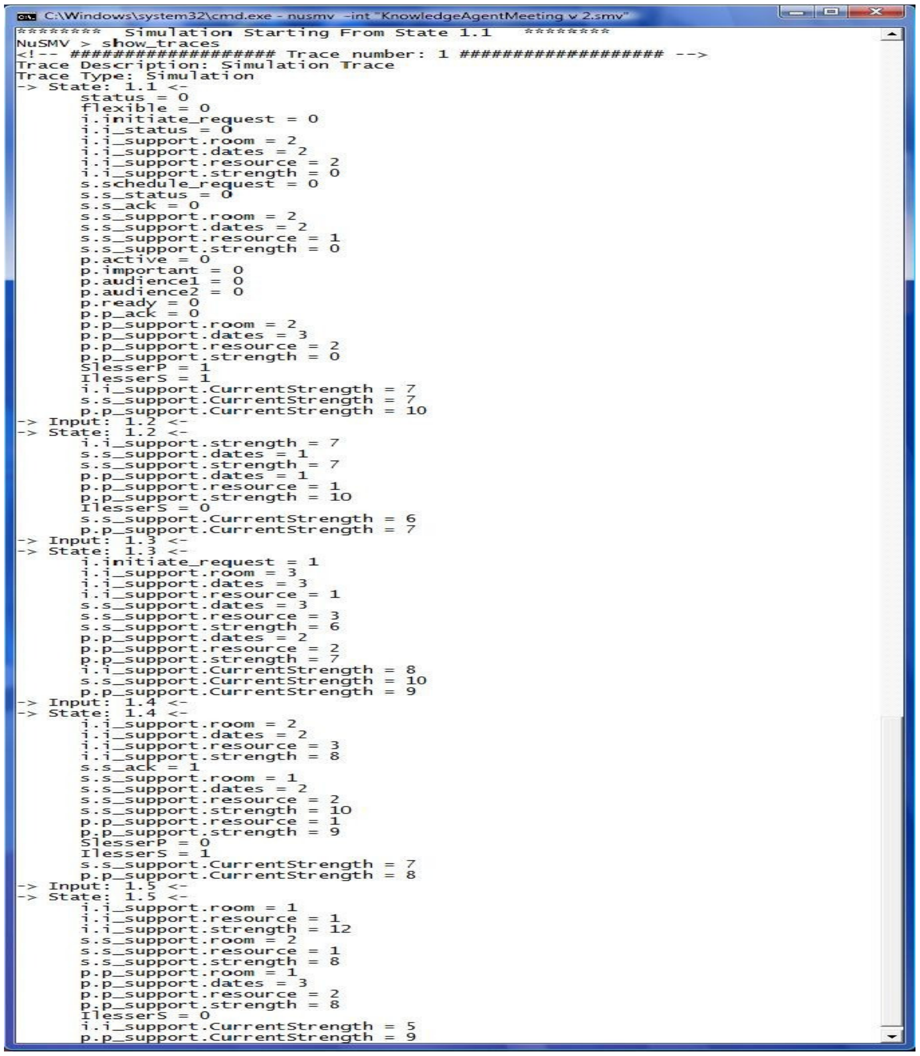
– AG i.initiate request→EF !schedule

The last specification verifies if there can be any path where the meeting is initiated but the scheduler does not forward this request to the participants. This is TRUE, as a request by the initiator can be canceled by the scheduler directly when the meeting constraints are not met (e.g. date or venue etc).

5.2 Trace Results

Trace in NuSMV demonstrates the transitions in state variables without the need of any specifications mentioned in the model. Each trace is identified by a trace number and state numbers corresponding to each state. Each state is hence represented by State:(T raceN umber.StateN umber). For example, State:1.4 refers to the fourth state of the first generated trace. In every state only those variables having an updated value as compared to the previous state, are reflected and the rest are not displayed.

In addition to the simulation results to verify the specification, we have also generated a trace result of the AMS problem discussed. Trace can be generated by using the show traces command in NuSMV after the .smv code has been built. Figure 4 shows the trace result from state 1.1 to 1.5 and Figure 5 continues the trace from 1.6 to 1.14.

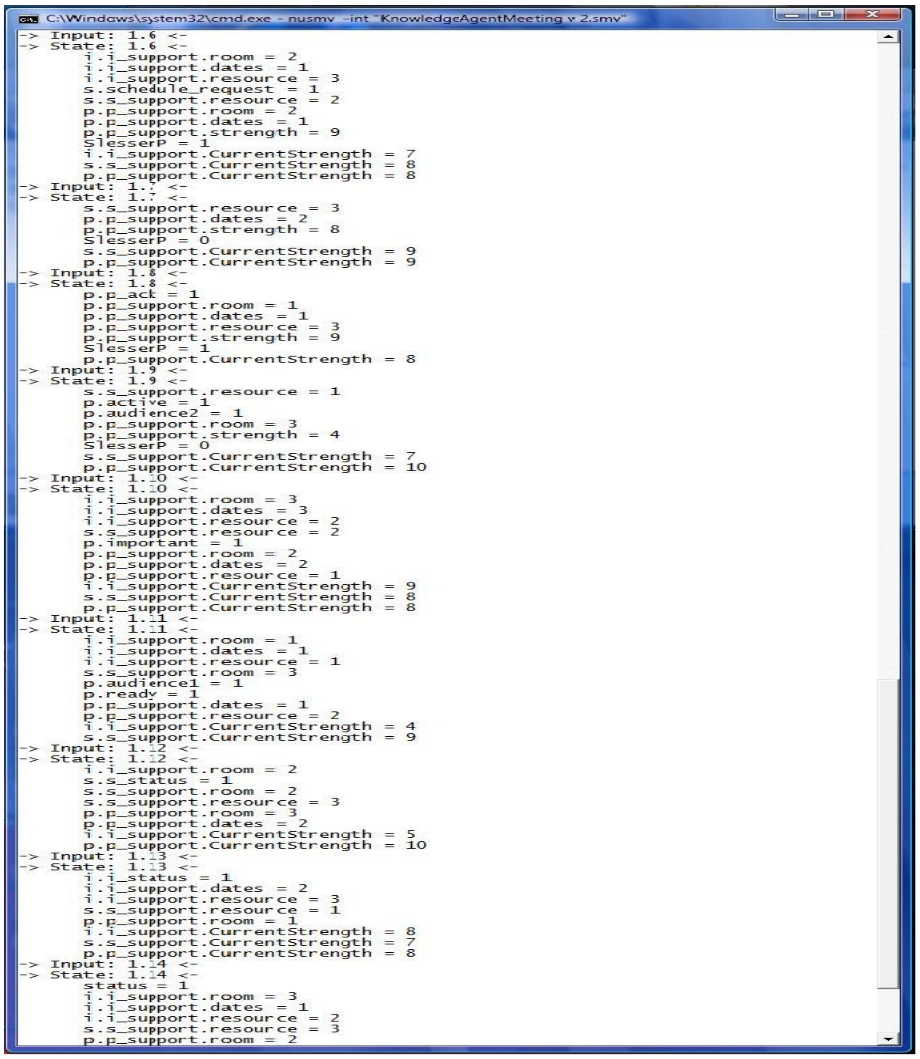


```
C:\Windows\system32\cmd.exe - nusmv  -int "KnowledgeAgentMeeting v 2.smv"
********   Simulation Starting From State 1.1    ********
NuSMV > show_traces
<!-- ################### Trace number: 1 ################### -->
Trace Description: Simulation Trace
Trace Type: Simulation
-> State: 1.1 <-
    status = 0
    flexible = 0
    i.initiate_request = 0
    i.i_status = 0
    i.i_support.room = 2
    i.i_support.dates = 2
    i.i_support.resource = 2
    i.i_support.strength = 0
    s.schedule_request = 0
    s.s_status = 0
    s.s_ack = 0
    s.s_support.room = 2
    s.s_support.dates = 2
    s.s_support.resource = 1
    s.s_support.strength = 0
    p.active = 0
    p.important = 0
    p.audience1 = 0
    p.audience2 = 0
    p.ready = 0
    p.p_ack = 0
    p.p_support.room = 2
    p.p_support.dates = 3
    p.p_support.resource = 2
    p.p_support.strength = 0
    SlesserP = 1
    IlesserS = 1
    i.i_support.CurrentStrength = 7
    s.s_support.CurrentStrength = 7
    p.p_support.CurrentStrength = 10
-> Input: 1.2 <-
-> State: 1.2 <-
    i.i_support.strength = 7
    s.s_support.dates = 1
    s.s_support.strength = 7
    p.p_support.dates = 1
    p.p_support.resource = 1
    p.p_support.strength = 10
    IlesserS = 0
    s.s_support.CurrentStrength = 6
    p.p_support.CurrentStrength = 7
-> Input: 1.3 <-
-> State: 1.3 <-
    i.initiate_request = 1
    i.i_support.room = 3
    i.i_support.dates = 3
    i.i_support.resource = 1
    s.s_support.dates = 3
    s.s_support.resource = 3
    s.s_support.strength = 6
    p.p_support.dates = 2
    p.p_support.resource = 2
    p.p_support.strength = 7
    i.i_support.CurrentStrength = 8
    s.s_support.CurrentStrength = 10
    p.p_support.CurrentStrength = 9
-> Input: 1.4 <-
-> State: 1.4 <-
    i.i_support.room = 2
    i.i_support.dates = 2
    i.i_support.resource = 3
    i.i_support.strength = 8
    s.s_ack = 1
    s.s_support.room = 1
    s.s_support.dates = 2
    s.s_support.resource = 2
    s.s_support.strength = 10
    p.p_support.resource = 1
    p.p_support.strength = 9
    SlesserP = 0
    IlesserS = 1
    s.s_support.CurrentStrength = 7
    p.p_support.CurrentStrength = 8
-> Input: 1.5 <-
-> State: 1.5 <-
    i.i_support.room = 1
    i.i_support.resource = 1
    i.i_support.strength = 12
    s.s_support.room = 2
    s.s_support.resource = 1
    s.s_support.strength = 8
    p.p_support.room = 1
    p.p_support.dates = 3
    p.p_support.resource = 2
    p.p_support.strength = 8
    IlesserS = 0
    i.i_support.CurrentStrength = 5
    p.p_support.CurrentStrength = 9
```

FIG. 4. TRACE RESULT 1

In the trace result at state 1.3 the values of i.i support.strength and s.s support.strength is 7 and 6 respectively. As argument strength of I is greater than or equal to that of S (i.e. 7 ≥ 6), the meeting is initiated and s.s ace is set to 1.Once this is set to 1, the values of i.i support.strength and s.s support.strength are fixed to 12 and 8 respectively.

In state 1.6, s.s support.strength and p.p support.strength have values 8 and 9 respectively. As the strength of the P is higher, p ack is not set to 1 in the next



C:\Windows\system32\cmd.exe - nusmv -int "KnowledgeAgentMeeting v 2.smv"

```
-> Input: 1.6 <-
-> State: 1.6 <-
     i.i_support.room = 2
     i.i_support.dates = 1
     i.i_support.resource = 3
     s.schedule_request = 1
     s.s_support.resource = 2
     p.p_support.room = 2
     p.p_support.dates = 1
     p.p_support.strength = 9
     SlesserP = 1
     i.i_support.CurrentStrength = 7
     s.s_support.CurrentStrength = 8
     p.p_support.CurrentStrength = 8
-> Input: 1.7 <-
-> State: 1.7 <-
     s.s_support.resource = 3
     p.p_support.dates = 2
     p.p_support.strength = 8
     SlesserP = 0
     s.s_support.CurrentStrength = 9
     p.p_support.CurrentStrength = 9
-> Input: 1.8 <-
-> State: 1.8 <-
     p.p_ack = 1
     p.p_support.room = 1
     p.p_support.dates = 1
     p.p_support.resource = 3
     p.p_support.strength = 9
     SlesserP = 1
     p.p_support.CurrentStrength = 8
-> Input: 1.9 <-
-> State: 1.9 <-
     s.s_support.resource = 1
     p.active = 1
     p.audience2 = 1
     p.p_support.room = 3
     p.p_support.strength = 4
     SlesserP = 0
     s.s_support.CurrentStrength = 7
     p.p_support.CurrentStrength = 10
-> Input: 1.10 <-
-> State: 1.10 <-
     i.i_support.room = 3
     i.i_support.dates = 3
     i.i_support.resource = 2
     s.s_support.resource = 2
     p.important = 1
     p.p_support.room = 2
     p.p_support.dates = 2
     p.p_support.resource = 1
     i.i_support.CurrentStrength = 9
     s.s_support.CurrentStrength = 8
     p.p_support.CurrentStrength = 8
-> Input: 1.11 <-
-> State: 1.11 <-
     i.i_support.room = 1
     i.i_support.dates = 1
     i.i_support.resource = 1
     s.s_support.room = 3
     p.audience1 = 1
     p.ready = 1
     p.p_support.dates = 1
     p.p_support.resource = 2
     i.i_support.CurrentStrength = 4
     s.s_support.CurrentStrength = 9
-> Input: 1.12 <-
-> State: 1.12 <-
     i.i_support.room = 2
     s.s_status = 1
     s.s_support.room = 2
     s.s_support.resource = 3
     p.p_support.room = 3
     p.p_support.dates = 2
     i.i_support.CurrentStrength = 5
     p.p_support.CurrentStrength = 10
-> Input: 1.13 <-
-> State: 1.13 <-
     i.i_status = 1
     i.i_support.dates = 2
     i.i_support.resource = 3
     s.s_support.resource = 1
     p.p_support.room = 1
     i.i_support.CurrentStrength = 8
     s.s_support.CurrentStrength = 7
     p.p_support.CurrentStrength = 8
-> Input: 1.14 <-
-> State: 1.14 <-
     status = 1
     i.i_support.room = 3
     i.i_support.dates = 1
     i.i_support.resource = 2
     s.s_support.resource = 3
     p.p_support.room = 2
```

Fig. 5. Trace Result 2

state 1.7. This is equivalent to a scenario where there is an argument between S and P with the latter having a stronger argument. But in state 1.7, both s.s support.strength and p.p support.strength have values 8. Hence in the next state 1.8, p.p ack is set to 1. This shows a typical argumentation scheme in which each agent argues twice. In the first argument, P poses a stronger argument due to which an acknowledgment (i.e. p.p ack) is not sent. This leads to a second argument where S refutes with a stronger argument, eventually leading to an acknowledgment from P.

6 CONCLUSION AND FUTURE WORK

In this paper we have used an argumentation scheme based on strength for a multi-agent scenario. The meeting scheduler problem gives us a simple yet con- vincing model for designing an argumentation scheme. It also shows and explains the steps to identify the agents and their communication scheme. The strength based scheme is effectively used in modeling this communication scheme. It can be straightaway extended to a more complex scenario having many agents. We have also seen the formalization of the model and its corresponding specifica- tions been verified and simulated using NuSMV. It is also easy to check the counter-examples generated by NuSMV simulation, so any modifications in the implementation can be easily done.

As future enhancements we would like to model strength of argument dy- namically by incremental development of their knowledge.

REFERENCES

1. A.S.Rao, Georgeff, M.: Bdi agents: From theory to practice. In: First International Conference on Multiagent System. pp. 312–319. AAAI (1995)
2. Black, E., Atkinson, K.: Dialogues that account for different perspectives in col- laborative rgumentation. In: Decker, Sichman, Sierra, Castelfranchi (eds.) 8th Int. Conf. on Autonomous Agents and Multiagent Systems. AAMAS, Budapest, Hungary (2009)
3. Bratman, M.E.: Intentions, Plans, and Practical Reason.CSLI Publication, Stanford (1987),available via http://csli- publications.stanford.edu/site/1575861925.shtml
4. Caminada, M., Prakken, H.: Argumentation in agent system part 2: Dialogue.In: Argumentation in Agent System. EASSS, Universiteit of Utrecht, Netherlands(2007)
5. Cavada, R., Cimatti, A., et al.: Nusmv 2.4 user manual (2009), available via http://nusmv.irst.itc.it
6. Cimatti, A., Clarke, E., et al.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: CAV '02: Proceedings of the 14th International Conference on Com- puter Aided Verification. pp. 359–364. Springer-Verlag (2002)
7. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, Cambridge, United Kingdom (2005)
8. MACAL, C., NORTH, M.: AGENT-BASED MODELING AND SIMULATION. IN: PROCEEDINGS OF THE 2009 WINTER SIMULATION CONFERENCE. PP. 86–98. IEEE (2009)
9. MEDELLIN, R., ATKINSON, K., MCBURNEY, P.: MODEL CHECKING COMMAND DIALOGUES. IN: ASSOCIATION FOR THE ADVANCEMENT OF ARTIFICIAL INTELLIGENCE. PP. 58–63. AAAI (2009)
10. ROLLAND, C., ET AL.: A PROPOSAL FOR A SCENARIO CLASSIFICATION FRAMEWORK. REQUIREMENT ENGINEERING 3, 23–47 (1998)