# Agent Support for Human Team Collaboration in Uncertain Environments*

Daniele Masato, Timothy J. Norman, and Wamberto W. Vasconcelos

Department of Computing Science
University of Aberdeen, AB24 3UE, Scotland, UK
{d.masato,t.j.norman,w.w.vasconcelos}@abdn.ac.uk

**Abstract.** Working collaboratively in a team and formulating a plan of action to achieve a particular goal is often a complex task for humans, especially when the decision-making process is performed in time-stressed situations, in which response teams are assembled in an ad-hoc fashion and operate in dynamic and uncertain environments. We are conducting research towards the development of software agents that will track the activities of human teams and monitor the plan execution in order to offer advice that would enable the humans to avoid problems, resolve them or recognise unexpected options, and in general maintain the synchronization among the different plan components.

**Key words:** Agent Support, Human Team, Plan Synchronisation, Uncertain Environments

## 1 Introduction

Working together in a team and formulating a plan of action to achieve a particular goal is often a complex task for humans. This is mainly because making decisions, taking responsibilities and performing joint actions all involve a number of interdependent activities that need to be coordinated together. These issues are emphasised when the decision-making process is performed in time-stressed situations, in which response teams are assembled in an ad-hoc fashion and might be asked to perform non-standard tasks in dynamic and uncertain environments. In such a context, humans may be overwhelmed by the amount of information they need to process to perform decision-making and coordination of tasks; thus, humans may fail because they do not recognise that initial

* This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation hereon.

assumptions are no longer valid and the actual course of action needs to be revised.

It is accepted that, at the moment, humans are much better than any intelligent software in analogical, spatial and heuristic reasoning, yet they are on the other hand limited in their cognitive ability to process large amounts of information [5]. Thus, when under stress, humans would make better and faster decisions if the *right information* was delivered to the *relevant recipient* at the *right time*, rather than having to process and share large amounts of information on their own. Software agents can support the information-driven decision-making process by constructing and maintaining a model of the plan and the context the team is operating within [8]. The model can be exploited to understand humans' intent and activities, recognise problems in advance and potentially offer advice that would enable the team to avoid problems, resolve them or recognize unexpected options, in this way relieving the team's cognitive burden.

The remainder of this paper is organised as follows: in Section 2 we outline our ideas to design software agents which can provide the type of support mentioned above. In Sections 3 and 4 we describe what kind of knowledge software agents need to acquire in order to monitor the plan execution and how that knowledge can be represented into the agents. In Sections 5 and 6 we present our approach to the agent design, starting with efficient ways to recognise team activities, and map them to the actual team's intent, of course considering plan constraints and the uncertainty of the environment. We finally conclude with some related work and future directions in Sections 7 and 8.

## 2 Research focus

Recognising human teams' intent and activities in a realistic simulation is the first step towards the design of software agents that are aware of the team situation and can monitor the plan execution, providing the kind of up-to-date, timely advice required to enhance team collaboration and human-agent interactions. Figure 1 presents our approach to tackle this problem. We assume that a number of human teams are collaborating together on a plan to achieve an overall goal (left-hand side of the figure). The main goal is divided into multiple subgoals, each tipically assigned to a specific team. An assigned subgoal $Subgoal_i$ is in turn achieved by executing in a number of hierarchical tasks $Task_i$, each of those can be recursively subdivided in simpler subtasks $STask_i$, in a *Hierarchical Task Network (HTN)* [9] representation. The bottom level of a task decomposition consists of a set of group behaviours $B_i$ (i.e. manoeuvres in formation, see Section 3.2) the team is assumed to execute as part of the expected course of action (*Expected COA* in Figure 1).

The achievement of the (sub)goals can be affected by dependencies among the tasks (see Section 4.1) that have to be satisfied during the execution. For example, in Figure 1, $Team_1$ and $Team_2$ are collaborating in parallel, however their subtasks ($STask_i$) need to follow a sequence (arrows between boxes). In a dynamic and uncertain environment, the synchronisation among the different

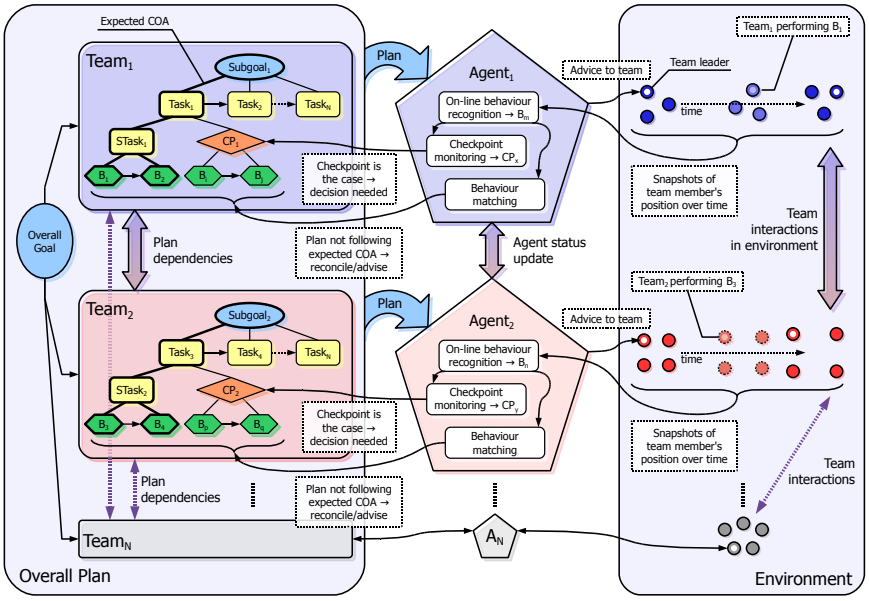


**Fig. 1.** System overview

plan components might be difficult to maintain, or changes in the course of action may become necessary, hence planners usually employ checkpoints $CP_i$ (see Figure 1) to monitor the execution and identify problematic situations (see Section 3.3). Checkpoints can also be used to pinpoint conditions that could positively impact the plan, as shortcuts and new opportunities. Because the teams interacting within the environment (right-hand side on the figure) might not follow the expected course of action, it would therefore be appropriate to reconcile their behaviours with a component of the plan, if possible. For example, because of terrain constraints, $Team_1$ and $Team_2$ might have to swap their roles during the execution of the plan without jeopardising the plan outcome as long as both $Subgoal_1$ and $Subgoal_2$ are achieved. This requires that both teams are aware of the change to avoid losing synchronisation, but this might be prevented because, say, there are visual obstacles or radio communication problems.

We aim to maintain and improve the collaboration between teams by attaching to each of them a software agent performing those monitoring processes that would usually be carried out by humans, but may result problematic in the circumstances mentioned above. By analysing on-line spatio-temporal traces of the teammates' positions in the environment, the software agent can recognise group behaviours and match them with the expected course of action, checking

at the same time whether checkpoints are being satisfied[1]. When the enactment is not following the expected track, we envisage software agents to try and reconcile the evidence with the plan, possibly exchanging status updates among them, or eventually warning the team leader that his group is off-sync and a decision needs to be taken. The same intervention mechanism could be used to offer advice about new opportunities, e.g. estimating whether a goal can be achieved faster because a set of preconditions has become the case.

## 3 Information requirements for software agents

Today, one of the challenges of assisting humans in their activities does not lie in the capabilities or power of the computer or device to be used, but rather in having the humans communicating their intents to a software agent and being confident enough to trust the agent's reply [15]. The risk is indeed that an inappropriate agent's intervention could be dismissed as irrelevant or annoying, defeating the efforts of supporting humans. Nonetheless, in order to understand what is happening in the environment, software agents need to gather information about the plan being pursued and its development. In the following discussion, we examine how much knowledge we can collect both during the plan design and execution.

### 3.1 Plan design phase

When teams operate in an uncertain environment, it might be difficult, if not impossible, to formulate a plan that foresees every possible contingency (e.g. in a military domain the classic assumption *"no plan survives contact with the enemy intact"* is often cited). This is worsened by the lack of time that can arise even during the design phase and because of non-standard tasks that need to be carried out. As a consequence, the plan structure may not cover in detail some of the tasks and several contingencies may be missed, under the hypothesis that an experienced team will be able to cope with these issues as the enactment proceeds. This usually works when human teams are well trained and have been working together before, but may fail for ad-hoc instances, which are formed rapidly and without much time for co-training.

Nevertheless, planners do lay out an expected course of action (see *Expected COA* in Figure 1), which can be represented as the agent initial knowledge and exploited for the monitoring purposes. Other opportunities to gather information arise, even in time-critical situations, when humans are given the chance of incrementally define/modify their plans [1] during the enactment phase, and if agent asked for more input only when such input would be relevant to the decision-making process [15], so we will investigate techniques which will enable us to incrementally modify a designed plan.

[1] For simplicity, we will initially assume the agents know the whole plan with all its dependencies

### 3.2 Plan execution phase

In circumstances in which time is limited and the team is actively involved in the dynamics of the plan execution, speech is probably the most effective and immediate means of communication. Unfortunately, performing speech-to-text analysis of communications in a hasty execution environment, with overlapping voices and noise, is still a big challenge, so we shall not employ any text-to-speech technology for our agents.

On the other hand, there are basic group behaviours that human teams employ to react to the surrounding environment, both in planned and unexpected circumstances (e.g. formations in group sports or military manoeuvres, see Figure 1, right-hand side). These behaviours have distinctive spatio-temporal structures, in terms of the relative position of the team members to each other and to external landmarks, and in terms of the temporal sequence within which the spatial configuration is maintained. These patterns can be exploited to design algorithms that recognize such behaviours [13] (see Section 5). Basic team behaviours also represent what the team is assumed to have learned through previous training or agreements, and can be considered the building blocks for a more complex plan, composed in a hierarchical fashion.

### 3.3 Plan execution monitoring

Once the plan has been designed and its enactment starts, monitoring and assessing this enables humans to react to changes, unexpected failures or new opportunities. In general, checkpoints are put in place to focus the monitoring process and detect whether critical decisions need to be made. They are established in the plan design process (but we could foresee them being added during the execution phase as well), to help performing timely and relevant decisions. A *timely* decision allows overcoming problems of the environment physics, giving enough time to the team to position itself in relation to the environmental constraints (e.g. terrain) and other teams to achieve the desired task. A *relevant* decision allows gaining and maintaining the ability to achive the plan goals. Checkpoints are usually employed to decide whether to employ time sensitive/critical assets in the context of the current plan, deviate from the original plan (alternative courses of action or branches) or initiate the transition to the next plan task.

A way to define a checkpoint consists in a *set of conditions* that, when they hold, triggers the decision-making process. The preconditions can be specified as conjunctions of observations, based on the information coming from the environment or from the plan execution status. A checkpoint also defines a *deadline* by which the information required to evaluate the conditions needs to be obtained, in order to perform a timely decision. After that, the decision would no longer be relevant. Note that a checkpoint may not define the precise details of the course of action to be taken (they could be specified when that particular situation occurs), nonetheless it identifies a point of contact between what the team leader would like to know about the plan while it is being executed and what a software agent can provide to try and satisfy these requirements.

*Example.* In military planning, checkpoints go under the name of *Decision Points*. The Commander sets a number of Decision Points during the planning process and for each he specifies the *Commander's Critical Information Requirements (CCIR)*, the requirements which *"allow the commander to track potential decisions, recognize that the time is right and that the decision is relevant to overall success"*[2]. A *Latest Time Information Is Of Value (LTIOV)*, referred above as the deadline, is also attached to a Decision Point.

## 4 A minimal model of context

As introduced in Section 1, software agents can support the decision-making process by constructing and maintaining a model of the plan and the context the team is operating within. We will now describe a minimal model of context that would allow us to design such agents.

### 4.1 Plan representation

n our research we will initially represent a plan in a hierarchical way using the concepts and control construct provided by the OWL-S[3] ontology. There is a straightforward link between *Hierarchical Task Network*-style plans [9] and the OWL-S process model, such that most of OWL-S concepts can be directly mapped to HTN constructs [16, 12] which an HTN planner can process. The plan hierarchy will include both the recursive plan decomposition into atomic tasks and the temporal relationships between the decomposed tasks (ordering, synchronization, etc.) made possible by the following OWL-S control structures[4]:

- **Sequence**: defines a list of tasks that need to be executed in order.
- **Any-Order**: defines a list of operations that can be done in any order.
- **Split**: defines a set of tasks that should be executed concurrently.
- **Split+Join**: defines a set of concurrent tasks that will terminate only when all the components have completed.

Each atomic task generated by the hierarchical plan decomposition process ($B_i$ in Figure 1), should be defined in terms of:

- **What**: the group behaviour expected from the team (the atomic task itself).
- **Who**: the team who is scheduled to perform the *What*. This parameter may be optional, as sometimes the decision about who will do what is made during the actual execution.
- **Where**: the location in which the *What* is to be performed.

[2] See `http://www.armchairgeneral.com/tactics-101-014-decision-making-and-the-power-of-commanders-critical-information-requirements.htm`

[3] See `http://www.w3.org/Submission/OWL-S`

[4] OWL-S also specifies the **Iterate**, **Repeat-While** and **Repeat-Until** control constructs but at the moment we will only consider non-iterative structures.

– **When**: an expected time (interval or point in time) by which the *What* should be completed[5]. This parameter is not required, but it is probably the most important in time-stressed situations.
– **Which**: the resource used to performed the *What* (e.g. a particular vehicle or asset the team will employ). This may be optional as well.
– **Why**: the reason why the *What* needs to be carried out. The justification for a task definitely helps in providing better advice to the team during the execution, especially when the course of action changes.

It is not necessary to specify all of the above parameters at the lowest level of the hierarchy, for each atomic task. Instead, some of them may be specified at the topmost or intermediate levels within the hierarchy, and then inherited by the levels underneath (e.g. *Who* could be specified for a complex task – say, a Sequence – so that all its subtasks would be performed by the same team, if no more specific information are given for the subtasks).

OWL-S also defines the concept of **Choice**[6], a set of tasks from which one should be chosen and executed. OWL-S Choices can be seen as checkpoints in our plan representation (see Section 3.3), where different choices may represent different courses of actions. The preconditions of a choice can be listed as:

– **What** behaviour is being performed in order to consider the checkpoint.
– **Who**, **Where** and **Which** as defined above.
– **When**, that is an expected time (interval or point in time) by which the *What* is expected to happen. This parameter can be optional and is not necessarily related to the deadline, however it must not conflict with it[7].

### 4.2 Information collection during execution

In order to infer the team's intent, a software agent needs also to collect information while the plan is being carried out. Because in some environments it might be difficult to employ extensive instrumentation (e.g. cameras, sensors, radios) to monitor the plan execution, we will initially rely only on spatio-temporal observations of the team members' positions collected at regular intervals over time, with an option to integrate more data sources in the future. Retrieving the position of a moving person or vehicle in a simulated environment is quite easy, but it should be feasible in real-life exercises as well, by simply using a GPS device (which nowadays can be found in many high-end mobile phones).

Similarly to the above discussion, the execution information can be seen as:

– **Where**: the position of a team, can be calculated as the centroid of the team members' positions.

[5] This should of course be consistent with the plan structure as specified by the combination of OWL-S control structures. For example, the second task in a Sequence cannot have an expected time set before the first task's one.
[6] We will consider the OWL-S **If-Then-Else** construct as a special case of **Choice**.
[7] For example, if the *When* is defined as an interval $[a, b]$, then $b < deadline$ must hold, whereas if it is an expected time $t$ then $t < deadline$ must hold.

– **When**: the instant in time in which the position is sampled.
– **What** and **Who**: because team behaviours show distinctive spatio-temporal patterns, the *When* and *Where* will be exploited to recognise what task the team is performing (see also Section 5).
– **Which**: the asset in use by the team. This could be tricky to identify, but we can again assume that it can be fitted with an identification tag.

In conclusion, using a minimal context model consisting of **What**, **Who**, **Where**, **When** and **Which** (5W) it should be possible to characterise the plan tasks, checkpoints and observations from the environment. This model, however, does not capture the whole context in which the plan is being enacted (hence the name "minimal"). For example it does not account for any communication among the team members or how the weather can affect the execution, but it should be simple enough to keep the computational cost low.

## 5 Proposed approach

The problem of monitoring a plan execution and matching recognised team behaviours to steps in the plan can be divided into three steps (see the boxes inside the Agent entities in Figure 1):

1. Acquiring regular "5W snapshots" of the environment using the 5W model.
2. Classify the current snapshot (or the sequence so far) to retrieve the team behaviour over time.
3. Using the results from step 2 as evidence, match it to the current plan execution status and identifying satisfied checkpoints.

Each of these steps is detailed in the following discussion.

### 5.1 Acquiring environment snapshots

We have chosen Battlefield 2[8] as our environment, a first-person shooter game with strategy elements in which players act in a virtual battle space. Players are organized in two teams and can play different military roles, use different types of weapons and particular classes of vehicles, depending on the selected role. We instrumented the game to capture many types of events, such as participants joining the game, interacting with vehicles and other players or entering in pre-defined areas identified by sensors [6]. The instrumentation allows us to collect *Who* is participating in the game, the participants' positions (*Where*) over time (*When*, the sampling interval can be changed), which assets (vehicles) they are using (*Which*), so that only the *What* needs to be inferred from the collected data.

[8] See `http://www.ea.com/official/battlefield/battlefield2/us`

### 5.2 Behaviour Classification

An interesting approach in recognising team behaviours (*What*) is presented in [13]: that work exploits the team members' positions to perform group behaviour recognition using traces collected from simulations. Specifically, an efficient algorithm called *Simultaneous Team Assignment and Behaviour Recognition (STABR)* is introduced. This algorithm analyses the traces in three stages:

- First the static positions of all the subjects in the environment are used to recognise possible formations, identifying in such a way also hypothetical teams.
- Each hypothetical team formation is then retained only it shows sufficient temporal support, that is, when such formation is following a known movement pattern over time (this also enables to identify the behaviour).
- Finally, the surviving hypotheses are used to partition the subjects into the actual teams over the time sequence, according to a cost function that gives higher scores to solutions with fewer changes in composition and behaviour of the teams.

*STABR* has proven to outperform agglomerative clustering, which is usually employed to group subjects into teams, even with noisy observations. One of the limitations in that work, however, is that the algorithm has been evaluated only with fully available (i.e. offline) traces, although real traces, rather than on-line observations that would be continuously gathered in our scenario. We are investigating means to adapt and extend *STABR* in an on-line setting to continuously classify the team's actions.

### 5.3 Plan status monitoring

In dynamic, time-stressed environments, it is inevitable to come across uncertainty, hence this aspect needs to be factored into the design of monitoring software agents which will be operating in those circumstances. Two main types of uncertainty are identified in [10]:

- *Aleatory*, that is uncertainty generated by a system behaving in random ways (also called objective uncertainty).
- *Epistemic*, that is uncertainty generated by the lack of knowledge about the system under analysis (also called subjective uncertainty).

The usual approach to handle aleatory uncertainty is the probabilistic (i.e. Bayesian) model, whereas for epistemic uncertainty the *The Dempster-Shafer Theory (DST)* [11] is often considered. The probabilistic model has been applied to epistemic uncertainty as well, however it has the following disadvantages:

- the a-priori probability of every event (e.g. performing a specific behaviour) is required. When such information is not available, a uniform probability is usually assigned to all events. This means deliberately assigning a probability to events whose characteristics/models are actually not known.

– The sum of all probabilities must add up to 1 and, as a consequence, knowing the probability $P(A)$ of an event $A$ occurring entails knowing the probability $P(\neg A)$ of that event not occurring. This leaves out the concept of ignorance, that is, the fact that one may not know $P(\neg A)$.

The DST, instead, seems more suitable to handle epistemic uncertainty, particularly in situations where:

– It is not straightforward to assign probabilities to single events due to ambiguous, conflicting or lack of information (e.g. sensor readings).
– It is appropriate to consider subjective degrees of belief on sets of events, rather than on objective probabilities about single events.
– Knowing the likelihood of a set of events to happen does not necessarily entails knowing the likelihood of that not happening.

In our case, we can consider $S = (Ev, Pl, Tm)$ to be the system composed by the uncertain environment $Ev$, the plan $Pl$ and a human team $Tm$ executing $Pl$ in $Ev$, where $Ev$ accounts for other teams operating in the same area. Although $Ev$ also consists of random components (e.g. the weather), they are very difficult to model in terms of probability. It is even harder to model how $Tm$ would react to changes in $Ev$ in terms of conditional probabilities. Moreover, $Pl$ is not a random component of the system as it has been laid out beforehand, albeit in an abstract form, and teammates will not behave randomly, rather they will act deliberately both when collaborating with others and facing unexpected events. Therefore, we are of the opinion that it is appropriate to employ the DST in our domain. This translates as the fact that, when performing behaviour recognition, we do not need the probabilities for each possible behaviour, rather we can assign a degree of belief only to a subset of likely behaviours.

## 6 Behaviour matching using the DST

To perform the matching between behaviours detected in the environment and the expected course of action, we will follow closely the method introduced in [2], although we will use a different rule of combination. Given a finite set of hypotheses $\Theta$ whose power set is $2^\Theta$, the DST defines a *basic probability assignment (bpa)* as a function $m : 2^\Theta \mapsto [0, 1]$ where:

$$m(\emptyset) = 0 \quad \text{and} \quad \sum_{X \subseteq \Theta} m(X) = 1 \tag{1}$$

Hypothesis sets $X$ for which $m(X) > 0$ holds are called *focal elements* of the *bpa*; $m(X)$ represents the amount of belief committed in exactly the hypothesis subset $X$, and not to more specific subsets of $X$. Thus, the total belief committed to $X$ and its subsets is given by:

$$Bel(X) = \sum_{Y \subseteq X} m(Y) \quad \text{where} \quad X \subseteq \Theta \tag{2}$$

In our approach, each hypothesis in $\Theta$ will represent one of the atomic behaviours generated by the hierarchical decomposition, whereas a set of hypotheses will represent a branch of the plan (e.g. the expected course of action). For example, according to Figure 1, $\Theta = \{B_1, B_2, ..., B_i, B_j\}$ and $STask_1 = \{B_1, B_2\}$. Given a plan branch $X$, $Bel(X)$ will represent the amount of belief that the team is actually performing one of the atomic tasks in $X$. For example, according to Figure 1, $Bel(Task_1) = Bel(\{B_1, B_2, ..., B_i, B_j\})$ will evaluate whether the team is performing one of the expected behaviours in the actual course of action.

The DST also states how multiple, independent sources of evidence (degrees of belief on sets of events) should be combined to obtain an aggregate event likelihood, based on Dempster's rule of combination [11]. Given two *bpa*s $m_1$ and $m_2$, their combination is calculated as:

$$m_{12}(X) = \frac{\sum_{Y \cap Z = X} m_1(Y) \cdot m_2(Z)}{1 - c} \quad \text{where} \quad c = \sum_{Y \cap Z = \emptyset} m_1(Y) \cdot m_2(Z) \quad (3)$$

In Formula 3, $c$ quantifies the *conflict* between two evidential sets and it is used to normalise the resulting *bpa* $m_{12}$ so that both constraints in Formula 1 hold. This normalisation, however, has been disputed by several people ([10] reviews their work) because it completely removes the conflict and can yield counterintuitive results. Several alternative rules have been proposed, but there seems still to be no general agreement on any particular rule; rather, there are different lines of thought about how to combine new evidence according to the application domain. Given a *bpa* $m_C$ describing the current assessment of the plan execution status and an environment snapshot (evidence $E$), we have identified three cases that can arise when the agent tries and detect a behaviour[9]. With reference to $Team_1$ in Figure 1, $Agent_1$ may:

1. Recognise a behaviour $B_i$ which can be mapped to $Team_1$'s subplan (e.g. $B_1$, $B_2$). In this case the agent will produce a *bpa* $m_E$, based on the evidence $E$, where $m_E(B_i) = k$, $m_E(\Theta) = 1 - k$.
2. Recognise a behaviour $B_p$ which cannot be mapped to $Team_1$'s subplan (e.g. $B_3$, $B_4$, assuming $Team_1$ and $Team_2$ have swapped their assignments). The agent will produce a *bpa* $m_E$ where $m_E(B_p) = k$, $m_E(\Theta) = 1 - k$.
3. Not recognise any behaviour at all. In this case the agent will produce a *bpa* $m_E$ where $m_E(\emptyset) = k$, $m(\Theta) = 1 - k$.

After the recognition, $m_C$ will be combined with $m_E$ to obtain a new assessment $m_{CE}$. There are a few considerations to make about the above situations:

– The parameter $k$, $0 \leq k \leq 1$, accounts for the accuracy of the *STABR* algorithm. *STABR* needs to perform several iterations to recognise a behaviour

[9] The issue of how to initialise $m_C$ when no evidence has been collected yet needs to be considered as well. Although the approach in [2] suggests using the vacuous hypothesis $m_C(\Theta) = 1$ when no evidence is available, we are investigating other initialisation methods. These will assign initial values to $m_C(B_i)$ according to how atomic behaviours are structured and linked in the expected course of action.

with a given accuracy (e.g. 99%, $k = 0.99$), but a lower number might be required in order to increase the recognition speed in a real-time simulation.
- In all three cases, assignign a degree of belief of $1-k$ to the set $\Theta$ accounts for our ignorance about what behavior is being (or not) performed, according to the selected accuracy for *STABR*. Moreover, associating degrees of belief either to a singleton set or the set $\Theta$ of all hypotheses, reduces the computational complexity of combination rules from exponential time to polynomial time in the number of focal elements in the *bpa*s to be combined (see the *Singleton DST* introduced in [3]).
- In case 3, we want the resulting $Bel_{CE}(X)$ function calculated over $m_{CE}$ to show a value below 1 for every $X \subseteq \Theta$, including the case $X = \Theta$. This would be indeed a strong signal that something is going wrong during the execution. To achieve this, however, we need to relax the definition of *bpa* (see Formula 1) by allowing $m(\emptyset) \geq 0$, such that $Bel(\Theta)$ will not account for some of the total belief mass, as it has be assigned to the empty set.

Among the several combination rules for the DST reported in [10], none seems to satisfy all the above requirements. For example, *Yager's rule* requires Formula 1 to hold and assigns the amount of conflict $c$ to the set $\Theta$ (effectively transforming conflict in ignorance), so we cannot deal with situations covered by case 3. On the other hand *Smets' rule* always assigns conflict to the empty set, as a consequence if $Team_1$ starts performing $B_1$ but then switches to $B_2$ for any reason, the belief mass intially assigned to $B_1$ will be transferred to the empty set, resulting in $Bel_{CE}(\Theta)$ evaluating to less than 1 even if in fact $Team_1$ is still operating within the expected course of action. For these reason, we will employ the *Combination by Compromise (CBC) rule* [17]. Given two *bpa*s $m_1$ and $m_2$, this rule, based on Equation 3 with $c$ set to 0, assigns a part of $m_1(Y) \cdot m_2(Z)$ to $C = Y \cap Z$, and distributes the remaining part to $Y_Z = Y \cap \neg Z$ and $Z_Y = \neg Y \cap Z$, depending on the values of $m_1(Y)$ and $m_2(Z)$, respectively. The combined *bpa* $m_{12}$ is calculated as:

$$m_{12}(X) = r_1(X) + r_2(X) + r_3(X) \tag{4}$$

where

$$\begin{aligned}
r_1(X = C) &= \sum_{C=Y\cap Z} \left[ n_1(C) \cdot n_2(C) + \frac{n_1(Y_Z) \cdot n_2(C)^2}{n_1(Y_Z) + n_2(C)} + \frac{n_1(C)^2 \cdot n_2(Z_Y)}{n_1(C) + n_2(Z_Y)} \right] \\
r_2(X = Y_Z) &= \sum_{Y_Z=Y\cap\neg Z} \left[ \frac{n_1(Y_Z)^2 \cdot n_2(C)}{n_1(Y_Z) + n_2(C)} + \frac{n_1(Y_Z)^2 \cdot n_2(Z_Y)}{n_1(Y_Z) + n_2(Z_Y)} \right] \\
r_3(X = Z_Y) &= \sum_{Z_Y=\neg Y\cap Z} \left[ \frac{n_1(C) \cdot n_2(Z_Y)^2}{n_1(C) + n_2(Z_Y)} + \frac{n_1(Y_Z) \cdot n_2(Z_Y)^2}{n_1(Y_Z) + n_2(Z_Y)} \right]
\end{aligned} \tag{5}$$

and

$$\begin{aligned}
n_1(C) &= \alpha_1 \cdot m_1(Y) \quad \text{and} \quad n_1(Y_Z) = (1 - \alpha_1) \cdot m_1(Y) \\
n_2(C) &= \alpha_2 \cdot m_2(Z) \quad \text{and} \quad n_2(Z_Y) = (1 - \alpha_2) \cdot m_2(Z)
\end{aligned} \tag{6}$$

In Formula 6, $n_1$ and $n_2$ perform an even distribution of the belief mass between $C$ and $Y_Z$, and $C$ and $Z_Y$; if $C = \emptyset$ then $\alpha_1 = \alpha_2 = 0$, if $Y_Z = \emptyset$ or $Z_Y = \emptyset$ then $\alpha_1 = 1$ or $\alpha_2 = 1$ respectively, otherwise $\alpha_1 = \alpha_2 = \frac{1}{2}$.

## 7 Related work

Much research has been done in how software agents can be designed to effectively support humans teams (e.g. [1, 15, 4] to name a few), but most of this work involves human participants interacting with software agents through interfaces that represented a simplified operational environment (e.g. 2D maps, switches and buttons on the screen), and where every single parameter governing the simulation could be monitored and controlled by the experimenters (no uncertainty). In general participants interacted with the interface in a point-and-click fashion or using vocal commands, but there was no physical action simulation.

On the other hand, in a number of domains such as military operations or emergency response, human teams are physically involved in the plan execution while a number of decisions needs to be made, for example, because of unexpected situations coming along or because a contingency course of action needs to be initiated. As a consequence, being already cognitively committed to perform an activity, they are less willing to spend time in processing new information, especially if it contradicts their actual beliefs about the plan development (known also as confirmation bias [7]). We are thus following an approach similar to that reported in [14, 6], where a first-person computer game is employed (and possibly adapted) in order for human teams to act in more realistic setting that reflects better the scenarios in which both humans and software agents will eventually operate together. This approach may present some shortcomings, because, as there are no real-life consequences from their actions, participants might not behave exactly how they would in a real-world exercise, sometimes becoming bored or reckless [6], but it is in our opinion more realistic than point-and-click interfaces.

## 8 Conclusions

This paper presents research in progress concerning monitoring and synchronising human teams during the enactment of a plan of action in an uncertain environment, by supporting them through software agents. The design of such agents is not an easy task, because it requires them to be aware of what is happening in the environment in a way similar to what humans do, however we believe that by combining behavior recognition techniques and evidentail reasoning, this can be achieved. We have proposed a way in which a plan can be encoded into the agent design and the minimal amount of knowledge the agent needs to gather both during the design and the execution phases in order to effectively monitor the enactment. Finally we have outlined a three-step approach to our design. We are now working towards an implementation and we expect to test our implementation using a first-person game as a realistic simulation environment.

## References


1. J. Allen and G. Ferguson. Human-Machine Collaborative Planning. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, October 2002.
2. M. Bauer. A Dempster-Shafer approach to modeling agent preferences for plan recognition. User Modeling and User-Adapted Interaction. *User Modeling and User-Adapted Interaction*, pages 317–348, 1995.
3. N. J. Blaylock. *Towards tractable agent-based dialogue.* PhD thesis, Department of Computer Science, University of Rochester, August 2005.
4. X. Fan, S. Sun, M. McNeese, and J. Yen. Extending the Recognition-Primed Decision Model to Support Human-Agent Collaboration. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.
5. X. Fan and J. Yen. Realistic Cognitive Load Modeling for Enhancing Shared Mental Models in Human-Agent Collaboration. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.
6. D. Masato, T. J. Norman, S. Poltrock, H. Bowyer, and P. Waggett. Adaptive Military Behaviour in a Collaborative Simulation. In *SISO European Simulation Interoperability Workshop*, July 2008.
7. R. S. Nickerson. Confirmation Bias: A Ubiquitous Phenomenon in Many Guises. *Review of General Psychology*, 2:175–220, 1998.
8. S. Poltrock, M. Handel, H. Bowyer, and P. Waggetg. A Dynamic Model of Mission Context. In *Procceedings of the Second Annual Conference of the International Technology Alliance in Network and Information Science*, September 2008.
9. S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey, USA, second edition, 2003.
10. K. Sentz and S. Ferson. Combination of Evidence in Dempter-Shafer Theory. Technical Report SAND2002–0835, Sandia National Laboratories, 2002.
11. G. Shafer. *A mathematical theory of evidence.* Princeton University Press, Princeton, New Jersey, USA, 1976.
12. E. Sirin and B. Parsia. Planning for Semantic Web Services. In *Semantic Web Services Workshop at the Third International Semantic Web Conference*, 2004.
13. G. Sukthankar and K. Sycara. Simultaneous Team Assignment and Behavior Recognition from Spatio-temporal Agent Traces. In *Proceedings of Twenty-First National Conference on Artificial Intelligence*, July 2006.
14. G. Sukthankar, K. Sycara, J. A. Giampapa, C. Burnett, and A. Preece. Towards a Model of Agent-Assisted Team Search. In *Procceedings of the First Annual Conference of the International Technology Alliance in Network and Information Science*, September 2007.
15. K. Sycara and M. Lewis. *Team Cognition: Understanding the Factors That Drive Process and Performance*, chapter 10, pages 203–233. American Psychological Association, 2004.
16. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proceedings of the Second International Semantic Web Conference*, October 2003.
17. K. Yamada. A New Combination of Evidence Based on Compromise. *Fuzzy Sets and Systems*, 159(13):1689–1708, 2008.