TOWARDS A COMPREHENSIVE TEAMWORK MODEL FOR HIGHLY DYNAMIC DOMAINS

Hendrik Skubch, Michael Wagner, Roland Reichle, Stefan Triller, Kurt Geihs
Distributed Systems Group, Kassel University, Wilhelmshöher Allee 73, Kassel, Germany
{*skubch, wagner, reichle, triller, geihs*}*@vs.uni-kassel.de*

Keywords: Mutli-Agent Systems, Teamwork, Coordination, Coorperation, Dynamic Domains

Abstract: Cooperative behaviour of agents within highly dynamic and nondeterministic domains is an active field of research. In particular establishing responsive teamwork, where agents are able to react to dynamic changes in the environment while facing unreliable communication and sensory noise, is an open problem. Unexpectedly changing situations force agents to react and adapt under tight time-constraints. Hence they often cannot communicate or agree upon their decisions before acting upon them. We present a novel model for cooperative behaviour geared towards such domains. In our approach, the agents estimate each other's decision and correct these estimations once they receive contradictory information. We aim at a comprehensive approach for agent teamwork featuring intuitive modelling capabilities for multi-agent activities, abstractions over activities and agents, and clear operational semantics for the new model. We show experimentally that the resulting behaviour stabilises towards teamwork and can achieve a cooperative goal.

1 INTRODUCTION

Highly dynamic and nondeterministic domains impose a number of challenges for the realisation of responsive yet coherent teamwork of autonomous agents.Teams of agents operating in such domains have to be robust against sensory noise, breakdown of individual agents, and unexpectedly changing situations. Such changes in the environment require the agents to react and adapt under tight time-constraints. This entails that it is often impossible to explicitly communicate – or even agree upon – a decision before acting on it. Maintaining a highly responsive and coherent teamwork is even more difficult, if communication is unreliable.

In this paper, we introduce a novel approach for cooperative behaviour, focusing on teams of agents acting in highly dynamic domains. The model consists of the agent oriented language ALICA (A Language for Interactive Cooperative Agents), which provides modelling facilities for cooperative behaviour, and clear operational semantics, that determine in detail how agents act.

ALICA is closely related to STEAM (Tambe, 1997), a model for teamwork based on both *Joint Intentions* (Levesque et al., 1990) and *Shared Plans* (Grosz and Kraus, 1996). In contrast to STEAM, ALICA agents in general do not establish joint intentions before acting towards a cooperative goal. Instead, each agent estimates the decisions of its teammates and acts upon this prediction. The resulting internal states are communicated periodically, thus allowing for individual decisions to be corrected dynamically. ALICA provides language elements to enforce an explicit agreement, resulting in a joint intention, for activities that require time critical synchronisations, such as lifting an object cooperatively.

We present operational semantics for ALICA programs based on 3APL's semantics (Hindriks et al., 1999). ALICA programs are modelled from a global perspective, such that team activities are described directly instead of being the result of interacting single agent programs. The operational semantics explicitly dictates how to execute an ALICA program. The complete semantics does not fit in the scope of this paper, but can be referred to in (Skubch et al., 2009).

ALICA features a two-layered abstraction between concrete activities and agents. *Roles* abstract

over agents, describing capabilities and resulting preferences for specific tasks. *Tasks*, on the other hand, abstract from specific activities, and relate similar activities within different contexts, called *plans*. This allows for plans to be specified independently of the team executing them and teams to be specified independently of the plans they might execute.

The state of the art is enhanced by providing a rich language to specify cooperative behaviour, a programming model with clear operational semantics, and elaborate coordination mechanisms for responsive teamwork at the same time. Thus, our approach can be considered as a step towards a comprehensive team-work model for highly dynamic domains.

The next section discusses related work and describes ALICA's relation to STEAM and 3APL in more detail. Section 3 formally introduces the language, whose semantics is discussed in Section 4. In Section 5, we present evaluation results within the robotic soccer domain. Finally, Section 6 concludes the paper and hints at future work.

2 RELATED WORK

Many research activities tackled the problem of describing agent behaviour and addressed the challenge to establish coherent teamwork of autonomous agents. As a result there exist several teamwork theories. One of them is the Joint Intentions Theory (Levesque et al., 1990), founded on BDI (Bratman, 1987). The framework focuses on a team's joint mental state, called a 'joint intention'. A team jointly intends a team action if team members jointly commit to an action while in a specified mental state. In order to enter a joint commitment, team members have to establish appropriate mutual beliefs and individual commitments. Although the Joint Intentions Theory does not mandate communication and several techniques are available to establish mutual beliefs about actions from observations, communication currently seems to be the only feasible way to attain joint commitments. However, a single joint intention for a high-level goal seems not appropriate to model team behaviour in detail and to ensure coherent teamwork.

The Shared Plans Theory (Grosz and Kraus, 1996; Grosz and Sidner, 1990) helps to overcome this shortcoming. In contrast to joint intentions, the Shared Plans Theory is not based on a joint mental attitude but on an intentional attitude called 'intending that'. 'Intention that' is defined by a set of axioms that guide an individual to take actions, enabling or facilitating its teammates to perform assigned tasks. A SharedPlan for group action specifies beliefs about how to do an action and sub actions.

STEAM (Tambe, 1997) builds on both teamwork theories and tries to combine their benefits. It starts with joint intentions, but then builds up hierarchical structures that parallel the Shared Plan Theory. So STEAM formalises commitments by building and maintaining joint intentions and uses Shared Plans to treat a team's attitudes in complex tasks, as well as unreconciled tasks.

In contrast to STEAM, ALICA agents do not establish joint intentions before acting towards a cooperative goal. Instead, each agent estimates the decisions of its team mates and acts upon this prediction. Conflicting individual decisions are detected and corrected using periodically communicated internal states of teammates. Although STEAM provides approaches for selective communication and tracking of mental attitudes of teammates, we argue that for highly dynamic domains and time-critical applications the strict requirement to establish or estimate a joint commitment before a joint activity is started has to be skipped. We deem agents that decide and act until contradictory information is available to be much more suitable for such applications. The assignment of agents to teams and teams to operators employed by STEAM seems to be too static for highly dynamic domains. In a soccer game, for example, a defending robot should also be able to carry out an attack if it obtains the ball and the game situation seems to be promising to do so. In order to facilitate such behaviour, we provide a different definition of roles and incorporate the concept of tasks. With its coordination approach, ALICA also abandons the concept of a 'team-leader' which STEAM assumes for different purposes in team coordination.

All previously described teamwork models have in common that they provide mechanisms to reason about or to establish teamwork, but they do not specify the internals of plans or operators. STEAM, and its implementation TEAMCORE (Pynadath et al., 1999), just assume reactive or situated plans, but do not specify the internal control cycle of an agent. Here, agent programming languages inspired the design of ALICA, in particular 3APL. ALICA incorporates many concepts of 3APL, e.g., the definition of the belief base, substitution of variables and the interpretation of goals as 'goals-to-do', which are not described declaratively but via plans that are directed towards achieving a goal. Furthermore, ALICA defines its operational semantics similar to 3APL through a transition system. However, in contrast to ALICA, 3APL also facilitates explicit specification of goals. It introduces rule sets and beliefs to allow reasoning about both, goals and plans. 3APL understands itself

as an abstract language, allowing to specify the deliberative cycle itself. ALICA is specifically aimed at modelling multi-agent strategy from a global perspective. This is impossible in 3APL. Moreover, ALICA's transitional rule system is geared towards cooperative agents, and hence is much more specific then the rule system employed by 3APL.

Considering the aspects above, we argue that ALICA enhances the state of the art in team-work models, as it provides a further step towards a comprehensive approach that provides support for all aspects of team coordination and also for explicit programming of team behaviour from a global perspective at the same time. With its approach to allow agents to decide and act towards a certain team-goal without explicit establishment of a joint commitment, it is also very suitable for highly dynamic domains that require fast decisions and actions and do not allow explicit communication and negotiations beforehand.

3 THE LANGUAGE

The central notion within ALICA are *plans*. A plan describes specific activities, a team of agents has to execute in order to achieve a certain goal. Plans are modelled similar to petri-nets, containing *states* and *transitions* between the states. Each transition is guarded by a condition, which must be believed to hold by an agent in order to progress along it.

The set of all plans in an ALICA program is denoted by $\mathcal{P}$. $\mathcal{Z}$ denotes all states. The belief base of each agent is described in a logic with language $\mathcal{L}$, hence all conditions are elements of $\mathcal{L}$. Each transition τ is an element of $\mathcal{Z} \times \mathcal{Z} \times \mathcal{L}$.

ALICA abstracts two-fold from agents, by *tasks* and *roles*. A role is assigned to an agent based on its capabilities. This assignment is treated to be compartively static, i.e., it holds until the team composition changes. This is the case if an agent joins or leaves the team, e.g., due to being incapacitated.

Tasks on the other hand abstract from specific paths within plans. As such, they denote similar activities in different plans. If a group of agents is to execute a plan, the corresponding tasks are allocated to the available agents based on the situation at hand and the roles the agents are assigned. This two-layered abstraction allows for programs to be specified independently of the team composition that will be on hand during execution. A team composition can be described solely by the roles each agent is assigned, while plans can be described without referring to roles. Each role has a numeric preference towards tasks, which allows roles to be mapped onto tasks dynamically during runtime (see Section 4.1).

Since plans describe activities for multiple agents, they have multiple initial states, each of which is annotated by a task. Hence, a task abstracts from specific activities within plans, and multiple plans can be annotated with the same tasks. For instance, a plan to build a tower and one to build a bridge might both contain the task of moving building blocks.Since tasks might require multiple agents, each task-plan pair (p, τ) is annotated by a cardinality, i.e., by an interval over $\mathbb{N}_0 \cup \{\infty\}$, denoting how many agents must at least and may at most be allocated to τ in p.

The applicability of a plan in a situation is measured in two ways. Firstly, each plan p is annotated by a precondition $\mathrm{Pre}(p)$, which has to hold when the agents start to execute it, and a runtime condition $\mathrm{Run}(p)$, which has to hold throughout its execution. Secondly, each plan p is annotated by a utility function $\mathcal{U}_p$, which is used to evaluate the plan together with a potential allocation of agents to tasks within p wrt. a situation. A utility function maps believed or potential situations onto the real numbers. Both conditions and utility functions solely refer to the belief base of an agent, which contains believed allocations.

Plans can be grouped together in *plan types*, providing the agents with sets of alternatives for solving a particular problem. Choosing a specific plan from a plan type is done autonomously by the agents in question. This selection is guided by the utility functions and conditions of the plans involved. Each state contains an arbitrary number of plan types. For each plan type, the agents involved have to choose a plan and execute it, i.e., multiple plans, one from each plan type, are executed in parallel. Additionally, each state contains an arbitrary number of *behaviours*. Behaviours are atomic single-agent action programs. The set of all behaviours is denoted by $\mathcal{B}$. Each behaviour within a state is meant to be executed by all agents inhabiting the corresponding state.

The relationship between states, plans and plan types spans a hierarchical structure, called the *plan tree* of an ALICA program.

4 SEMANTICS

The semantics of ALICA is given by a transitional rule system, which describes how the internal states of agents change over time. These internal states are referred to as *agent configurations*. Additionally, a set of axioms, Σ_B, constrains these configurations.

Definition 4.1 (Agent Configuration)**.** An agent configuration is a tuple $(B, \Upsilon, E, \theta, R)$, where B is the agent's belief base, Υ its plan base, $E \subseteq \mathcal{B} \times \mathcal{Z}$ the

set of behaviours b the agent executes together with the state in which b occurs, θ a substitution, and R is a set of roles assigned to the agent.

The plan base contains triples (p,τ,z), indicating that the agent is committed to task τ in plan p and currently inhabits state z within p. Each triple is reflected by a literal in the belief base, $\text{In}(a,p,\tau,z)$, representing the belief that (p,τ,z) is an element of agent a's plan base. Similarly, $\text{HasRole}(a,r)$ captures the belief that role r is assigned to a.

4.1 Task Allocation and Plan Selection

Whenever an agent enters a state, it has to take care that a plan out of each plan type within that state is executed. This does not entail that the agent is required to execute the plan, it is sufficient to come to the conclusion that there are agents executing it. This gives rise to the *allocation problem*.

An allocation done by agent a for a plan p is a set of literals of the form $\text{In}(a',p,\tau,z)$, referred to as $\text{TAlloc}(a,p|\mathcal{F})$. The allocation is calculated under the set of assumptions $\mathcal{F}$. Normally, it equals the belief base of a. In case of planning, $\mathcal{F}$ can refer to a potential situation in the future. A task allocation is subject to several constraints, such as the pre- and runtime conditions of plan p, $\text{Pre}(p)$ and $\text{Run}(p)$. Moreover, it must be consistent with the assumptions $\mathcal{F}$ under which it is computed, and the axioms Σ_B. E.g., Σ_B rule out agents taking on multiple tasks within the same plan. Definition 4.2 captures these constraints.

Definition 4.2 (Valid Task Allocation)**.** A Task Allocation $\text{TAlloc}(a,p|\mathcal{F})$ for agent a with the configuration $(B,\Upsilon,E,\theta,\text{R})$ is valid iff

- $\Sigma_B \cup \mathcal{F} \cup \text{TAlloc}(a,p|\mathcal{F}) \not\models \bot$
- $\Sigma_B \cup \mathcal{F} \cup \text{TAlloc}(a,p|\mathcal{F}) \models (\text{Pre}(p) \wedge \text{Run}(p))\theta$
- $\Sigma_B \cup \mathcal{F} \cup \text{TAlloc}(a,p|\mathcal{F}) \models \text{TeamIn}(p)$

where $\text{TeamIn}(p)$ denotes that the given team of agents $\mathcal{A}$ executes plan p. Formally:

$$\begin{aligned}\text{TeamIn}(p) \stackrel{def}{=}\ & (\forall\tau \in \text{Tasks}(p))(\exists n_1,n_2)\\ & \xi(p,\tau) = (n_1,n_2) \wedge\\ & (\exists A')A' \subseteq \mathcal{A} \wedge n_1 \leq |A'| \leq n_2 \wedge\\ & (\forall a' \in \mathcal{A})a' \in A' \leftrightarrow (\exists z)\,\text{In}(a',p,\tau,z)\end{aligned}$$

where $\xi(p,\tau)$ refers to the cardinalities of task τ in plan p.

The set of agents that can be referred to in an allocation is limited by Σ_B to the ones believed to be in the respective parent state z, i.e., for each agent a' occuring in $\text{TAlloc}(a,p|\mathcal{F})$, $\text{In}(a',p',\tau',z)$ must hold in $\mathcal{F}$. Among all valid task allocations for p, agent a has to choose the one that maximises the utility function of plan p. Various algorithms can be used for this maximisation problem; we use A^* in our implementation. Since p is an element of a plantype P, the optimisation extends to all plans in P. In other words, there must not be a plan p' in P for which a valid task allocation T' wrt. $\mathcal{F}$ exists such that $\mathcal{U}_p(\mathcal{F} \cup \text{TAlloc}(a,p|\mathcal{F})) < \mathcal{U}_{p'}(\mathcal{F} \cup T')$. This problem can trivially be integrated into the maximisation by extending the search space to all elements of P.

If the task allocation of agent a allocates a to task τ in p, a enters a state identified by p and τ, denoted as $\text{Init}(p,\tau)$. Intuitively, this is the initial state for task τ in plan p. Since $\text{Init}(p,\tau)$ can contain plan types again, new allocation problems arise. For every plan type P' in $\text{Init}(p,\tau)$, a plan p' in P' has to be chosen, such that $\text{TAlloc}(a,p'|\mathcal{F} \cup \text{TAlloc}(a,p|\mathcal{F}))$ is valid. So, the task allocation is recursive. This recursion is be captured by a backtracking algorithm, backtracking whenever a valid allocation cannot be found.

Note, the recursion applies only to those states the calculating agent enters. As such, agents only deal with a local view of the complete allocation. Figure 1 depicts an example of the backtrack search and this local view, where agent a allocates agents a, b, and c. Figure 1(a) shows agent a's result of the recursive task allocation. Agent a has to allocate itself within p_4. If it cannot find a valid task allocation for p_4, backtracking occurs (see Figure 1(b)). Figure 1(c) shows the final task allocation, where agent a has swapped places with agent c within plan p_1. Note, that a has not allocated b and c within p_3. This reflects the local view of a on this plan tree.

After an allocation is computed, the belief base is updated accordingly and the agent acts based on the calculated assumptions. In order to deal with potential inconsistencies, each agent communicates its plan base periodically. Whenever an agent receives a message, its belief base is updated. Conflicts can be detected and are dealt with by specific transitional rules. They entail a reallocation, an abortion of the plan, or the pursuit of an alternative course of action. Here, we focus on reallocation, which an agent performs if it believes this will improve the corresponding utility function. In particular, the utility of a plan with an invalid allocation is defined to be 0, and greater or equal 0 in case of a valid one.

4.2 Transitional Rule System

Transition rules define how an agent's configuration changes during a single deliberation cycle wrt. the executed ALICA program. Thus, they describe the operational semantics of ALICA. Each rule is guarded

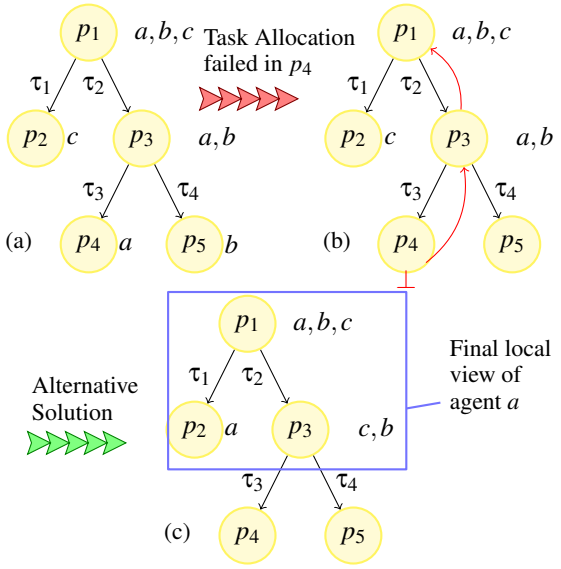

Figure 1: Recursive Task Allocation

by a condition and transforms the current configuration of an agent into a new one.

The complete rule system (Skubch et al., 2009) is out of scope of this paper. Essentially, each rule governs a reaction to specific situations an agent faces with respect to an ALICA program. Rules are partitioned into two sets. First, so-called operational rules, describing the normal operation of a program. Second, repair rules, which handle unexpected failures.

The operational rules define how an agent executes an ALICA program. They monitor transitions, describe how an agent leaves a state, and enforce task allocations for newly entered states. Furthermore, operational rules react to successful execution of behaviours or plans and propagate this information up the plan tree. A precedence order over rules is used to break ties between multiple applicable rules.

Repair rules handle the cases where something went wrong. Depending on the kind of failure, the corresponding behaviour or plan is first aborted, and afterwards either retried, replaced by an alternative or, if both is impossible, the failure is propagated up the plan tree. Each of these reactions is handled by a particular rule. In case two repair rules can be applied, precedence is given to the one that requires the smallest change. For example, retrying a particular plan is less of an effort than propagating the failure upwards.

Repair rules are always overruled by operational rules. Thereby, unnecessary repairs are avoided, and domain specific failure handling can be modelled by declaring transitions referring to believed failures.

The transitional rule system is not only used to update agent configurations, but it allows the agents to track each other's internal state. For instance, if an agent follows a transition, it will assume that all other agents in the corresponding state do the same. This assumption is corrected once an agent receives a message containing the plan base of another.

A specific repair rule is used to handle cases where the utility of a plan in execution is believed to be comparatively low. In this case, the current utility of the plan is compared with the potential best utility achievable by reallocating all involved agents. We present this rule in more detail as it is of major importance to the evaluation in Section 5. Given the utility function $\mathcal{U}_p$ of the plan in question, the belief base B of the agent containing the allocation T of p, the agent adopts a new allocation $T' = \text{TAlloc}(a, p|B \setminus T)$ if

$$\mathcal{U}_p((B \setminus T) \cup T') - \mathcal{U}_p(B) - Dif(p,T,T') > t_p$$

holds, where t_p is a plan specific threshold and Dif is a similarity measurement over allocations.

$$Dif(p,T,T') = 1 - \frac{|\{a|\,\text{In}(a,p,\tau,z) \in T \wedge (\exists z')\,\text{In}(a,p,\tau,z') \in T'\}|}{|T'|}$$

Both the threshold and the similarity measurement are used to stabilise an assignment once it is adopted.

5 EVALUATION

We integrated ALICA into a software framework for autonomous robots and use it to control a robotic team in the RoboCup Midsize league. The execution loop of each robot has a frequency of $30Hz$. Each cycle, a robot has to process new sensory data and incoming messages, update its belief base, check and apply transitional rules, and execute behaviours. Every robot peridiocally broadcasts its plan base and parts of its belief base, more specifically, believed positions of all objects the utility functions depend on. This information is sent with a frequency of $10Hz$[1].

Message delay has a direct impact on the cooperative behaviour, since received data are integrated into the belief base regardless of the message's age. This is done intentionally, to foster the conclusions we can draw from experimental data. The initial allocations of newly entered plans are protected by discarding contradicting messages for $250ms$. This ensures that a calculated allocation has a minimal life cycle, and is not immediately overwritten by delayed messages.

Our approach was evaluated in the robotic soccer domain. The evaluation presented here is two-fold.

[1]The frequency is chosen as to keep the network load low.

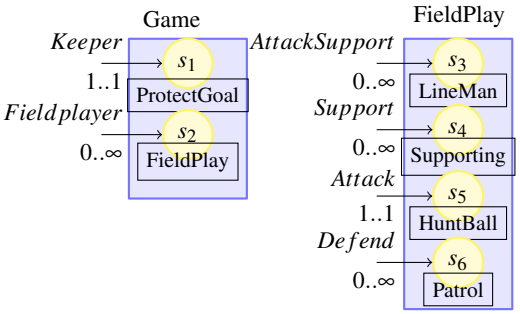

Figure 2: Simple Example Plan in the Soccer Domain

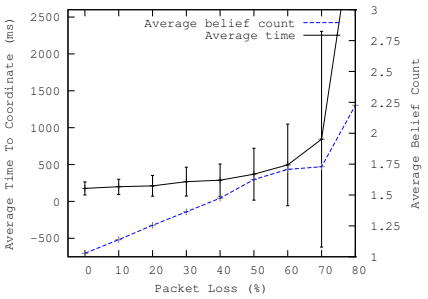

Figure 3: Average Times To Coordinate and Average Belief Count with Packet Loss

First we present results obtained in a simulated environment, with unreliable network due to packet loss or delay. Delay and packet loss were artificially added for clean results. Second, we present results obtained under real-world conditions during the RoboCup '09.

In the simulation, six agents play soccer cooperatively. In order to have reproducible results, there is no opponent team. The sole plan with a dynamic utility function ($\mathcal{U}_{FP}$) in this test is depicted in Figure 2.

$$\begin{aligned}\mathcal{U}_{FP} &= 0.1Pri + 1.0 \max_{x \in Attack}\left(1 - \frac{Dist(x, Ball)}{FieldSize}\right) \\ &\quad + 0.2 \max_{y \in Defend}\left(1 - \frac{Dist(y, OwnGoal)}{FieldSize}\right)\end{aligned}$$

where $FieldSize$ refers to the maximal possible distance on the football field. Pri sums over the preferences of each agent's roles towards their tasks.

$$Pri = \frac{\sum_{\tau \in \text{Tasks}(p)} \sum_{\text{In}(x,p,\tau,z)} \max_{\text{HasRole}(x,r)} \text{Pref}(r,\tau)}{\sum_{\tau \in \text{Tasks}(p)} \sum_{\text{In}(x,p,\tau,z)} 1}$$

We use two measures, "Time To Coordinate" (TTC) refers to the average length of the time intervals, the team was in disagreement about the allocation within plan FieldPlay. "Belief Count" (BC) refers to the average number of different allocations believed by at least one robot. In the used data, the average frequency of events requiring an allocation change was about 7.8 per minute. The simulator caused these events by resetting the ball's position.

Figure 3 illustrates the influence of packet loss on the two measures. Under perfect conditions, TTC is 177ms, with a standard deviation of 89ms. BC is 1.028 with standard deviation of 0.22. TTC slowly increases till packet loss is at 50%, at which point TTC is at 370ms with standard deviation of 352ms. Beyond this point, the curve steepness increased. At 80% packet loss, TTC is already at 3894ms and coordination hardly occurs any more. Note that the standard deviation of TTC follows the same pattern. This is due to the increasing likelihood that two disagreement intervals overlap and the increasing noise, which introduces additional small disagreement periods. The belief count on the other hand proves to be more resistant against packet loss, it grows linear with the amount of packet loss. The same holds for its standard deviation, which reaches 1.0 at 60% loss.

The relationship between packet delay and the two measures is shown in Figure 4. Packet delay was introduced with a uniformly distributed noise in $[-25ms, 25ms]$. After an initial steep ascend of both TCC and BC in the interval from 0ms to 50ms delay, both measures grow only slowly with additional delay. At 50ms, TTC is 263ms with a standard deviation of 177ms, and BC equals 1.34 with standard deviation of 0.7. Afterwards, deviation of BC stays almost constant while the deviation of TTC continues to rise. Again, this is due to intervals overlapping and delayed messages inducing short periods of disagreement.

The most interesting fact here is the initial ascend of the two measures. This indicates a benefit of discarding messages older than a certain time span, as the data indicate a delay of 100ms is as bad as packet loss of 40%. Note that, when in disagreement, the robots still act, at no point in time a robot did not try to fulfil a task. In summary, with disagreement periods of 177ms under perfect conditions, 370ms under 50% packet loss, and 361ms under 200ms delay, we can state that the agents not only react quickly to changes in the environment but also achieve agreement about the task allocation quickly and hence act and cooperate according to the modelled plans.

Figure 5 shows the average belief count during actual soccer matches. The data are normalised by the number of robots involved. For comparison, simulation results are shown as well. The data was gathered from 226.6 minutes of game play. Unfortunately, the network quality fluctuated to quickly to draw meaningful TTC values from the data collected. In average, a disagreement lasted for 300ms, i.e., 3 communication cycles. They occurred 22.8 times per minute. Hence, in 11% of the time, a disagreement over the al-

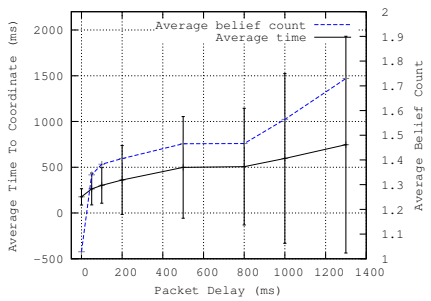

Figure 4: Average Times To Coordinate and Average Belief Count with Packet Delay

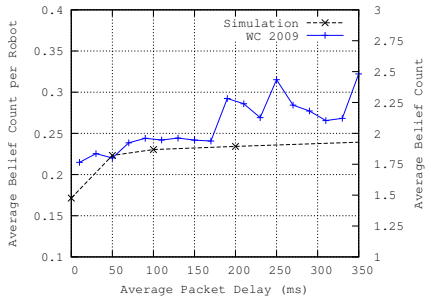

Figure 5: Normalised Average Belief Count vs Packet Delay during RC 2009

location existed. Given the dynamic character of the domain, reflected by the fact that each robot modified its plan base to accommodate a new situation 17.5 times per minute, we deem this acceptable.

Compared with the simulation, the conditions are harder, as the game is faster and much more dynamic. Also, sensor noise degrades the coherence of information used for team coordination. Especially in close-quarter situations, our robots disregarded communicated data in favour for local sensor data. This led to breakdowns in the teamwork. Hence, it is imperative to use fused information only for task allocations in order to guarantee resolution of such conflicts.

However, the average belief count during the RoboCup corresponds to the simulation data, even though the plans employed during the RC were more complex. Not only allocated the robots themselves within one plan, but also did they choose autonomously the same plan out of a plan type. In general, three plans were available on the level of FieldPlay, two defensive and one aggressive. This is a small indicator that the approach scales well with the complexity of the employed plans. A situation from an actual game displaying dynamic allocations can be found under `http://www.youtube.com/watch?v=3V_vedh95kc`.

6 CONCLUSION

In this paper, we sketched a new language for describing multi-agent plans together with an operational semantics. This language is geared towards highly dynamic domains in which agents have to take decisions under tight time constraints. Such domains often occur in robotic scenarios. We presented how agents can estimate their team mates' decisions, act upon them, and correct them when new information is available. We showed experimentally that this approach is robust against a high degree of packet loss and delay, common properties of communication in robotic scenarios.

Several open questions remain. It is still unclear how coordination is affected if several domain properties, which are not observable by all agents, are used for utility and condition calculations. We believe that the approach employed by STEAM, namely communicating failures together with their reason, is a suitable method to foster coordination in this case. Finally, an embedding of ALICA into 3APL is work in progress.

REFERENCES

Bratman, M. (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press.

Grosz, B. J. and Kraus, S. (1996). Collaborative plans for complex group action. *ARTIFICIAL INTELLIGENCE*, 86(2):269–357.

Grosz, B. J. and Sidner, C. L. (1990). Plans for discourse. In *Intentions in Communication*. MIT Press.

Hindriks, K. V., Boer, F. S. D., Hoek, W. V. D., and Meyer, J.-J. C. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

Levesque, H. J., Cohen, P. R., and Nunes, J. H. T. (1990). On Acting Together. In *Proc. of AAAI-90*, pages 94–99, Boston, MA.

Pynadath, D. V., Tambe, M., and Chauvat, N. (1999). Toward team-oriented programming. In *Intelligent Agents VI: Agent Theories, Architectures, and Languages*, pages 233–247.

Skubch, H., Wagner, M., and Reichle, R. (2009). A language for interactive cooperative agents.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124.