A Distributed Task Specification Language for Mixed-Initiative Delegation*

Patrick Doherty
Linköping University
Linköping, Sweden
patrick.doherty@liu.se

David Landén
Linköping University
Linköping, Sweden
david.landen@liu.se

Fredrik Heintz
Linköping University
Linköping, Sweden
fredrik.heintz@liu.se

ABSTRACT

In the next decades, practically viable robotic/agent systems are going to be mixed-initiative in nature. Humans will request help from such systems and such systems will request help from humans in achieving the complex mission tasks required. Pragmatically, one requires a distributed task specification language to define tasks and a suitable data structure which satisfies the specification and can be used flexibly by collaborative multi-agent/robotic systems. This paper defines such a task specification language and an abstract data structure called Task Specification Trees which has many of the requisite properties required for mixed-initiative problem solving and adjustable autonomy in a distributed context. A prototype system has been implemented for this delegation framework and has been used practically with collaborative unmanned aircraft systems.

Keywords

Multi-Agent Architectures, Cooperation, Delegation, Distributed Systems, Unmanned Aerial Vehicles

1. INTRODUCTION

In the past decade, the Unmanned Aircraft Systems Technologies Lab[1] at the Department of Computer and Information Science, Linköping University, has been involved in the development of autonomous unmanned aircraft systems (UAS's) and associated hardware and software technologies [8]. The size of our research platforms range from the RMAX helicopter system [9, 21] (Figure 1) developed by Yamaha Motor Company, to smaller micro-size rotor based systems such as the LinkQuad (Figure 2)[2] and LinkMAV [11] (Figure 1), in addition to a fixed wing platform, the PingWing [6] (Figure 1). The latter three have been designed and developed by the Unmanned Aircraft Systems Technologies Lab. Previous work has focused on the development of robust autonomous systems for UAS's which seamlessly integrate control, reactive and deliberative capabilities that meet the requirements of hard and soft realtime constraints [9, 16].

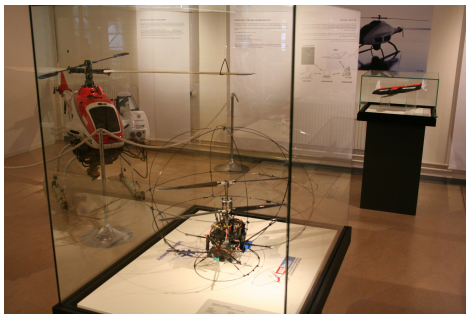

Figure 1: The UASTech RMAX (left), LinkMAV (center) and the PingWing (right)

Figure 2: The UASTech LinkQuad Quadrotor Helicopter

More recently, our research efforts have begun to focus on applications where UAS's with heterogeneous unmanned aircraft are required to collaborate not only with each other but also with diverse human resources [10].

As UAS's become more autonomous, mixed-initiative interaction between human operators and such systems will be central in mission planning and tasking. In the near future, the practical use and acceptance of UAS's will have to be

*This work is partially supported by grants from the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII, the Swedish Research Council (VR), the VR Linnaeus Center CADICS, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, and the Center for Industrial Information Technology CENIIT.

[1] www.ida.liu.se/divisions/aiics/

[2] www.uastech.com

based on a verifiable, principled and well-defined interaction foundation between one or more human operators and one or more autonomous systems. In developing a principled framework for such complex interaction between UAS's and humans in complex scenarios, a great many interdependent conceptual and pragmatic issues arise and need clarification both theoretically, but also pragmatically in the form of demonstrators.

1.1 A Conceptual Triad

In our current research, we have targeted a triad of fundamental, interdependent conceptual issues: delegation, mixed-initiative interaction and adjustable autonomy. The triad of concepts is being used as a basis for developing a principled and well-defined framework for interaction that can be used to clarify, validate and verify different types of interaction between human operators and UAS's both theoretically and practically in experimentation with our deployed platforms. The concept of delegation is particularly important and in some sense provides a bridge between mixed-initiative interaction and adjustable autonomy.

Delegation – In any mixed-initiative interaction, humans request help from robotic systems and robotic systems may request help from humans. One can abstract and concisely model such requests as a form of delegation, $Delegate(A, B, task, constraints)$, where A is the delegating agent, B is the potential contractor, *task* is the task being delegated and consists of a goal and possibly a plan to achieve the goal, and *constraints* represents a context in which the request is made and the task should be carried out.

Adjustable Autonomy – In solving tasks in a mixed-initiative setting, the robotic system involved will have a potentially wide spectrum of autonomy, yet should only use as much autonomy as is required for a task and should not violate the degree of autonomy mandated by a human operator unless agreement is made. One can begin to develop a principled means of adjusting autonomy through the use of the *task* and *constraint* parameters in the $Delegate(A, B, task, constraints)$ predicate. A task delegated with only a goal and no plan, with few constraints, allows the robot to use much of its autonomy in solving the task, whereas a task specified as a sequence of actions and many constraints allows only limited autonomy.

Mixed-Initiative Interaction – By mixed-initiative, we mean that interaction and negotiation between a UAS and a human will take advantage of each of their skills, capacities and knowledge in developing a mission plan, executing the plan and adapting to contingencies during the execution of the plan. Mixed-initiative interaction involves a very broad set of issues, both theoretical and pragmatic. One central part of such interaction is the ability of a ground operator (GOP) to be able to delegate tasks to a UAS, $Delegate(GOP, UAS, task, constraints)$ and in a symmetric manner, the ability of a UAS to be able to delegate tasks to a GOP, $Delegate(UAS, GOP, task, constraints)$. Issues pertaining to safety, security, trust, etc., have to be dealt with in the interaction process and can be formalized as particular types of constraints associated with a delegated task.

1.1.1 The Central Role of Tasks and their Specification and Semantics

An important conceptual and pragmatic issue which is central to the three concepts and their theoretical and pragmatic integration is that of a task and its representation and semantics in practical systems. The task representation must be highly flexible, distributed and dynamic. Tasks need to be delegated at varying levels of abstraction and also expanded and modified as parts of tasks are recursively delegated to different UAS agents. Consequently, the structure must also be distributable. Additionally, a task structure is a form of compromise between a compiled plan at one end of the spectrum and a plan generated through an automated planner [14] at the other end of the spectrum. The task representation and semantics must seamlessly accommodate plan representations and their compilation into the task structure. Finally, the task representation should support the adjustment of autonomy through the addition of constraints or parameters by agents and human resources.

1.2 Paper Structure

The first part of the paper sets the broader context by providing a short summary in Section 2 of a formal delegation framework based on the use of speech acts in addition to a short summary about the pragmatics of implementing such a system on UAS's in Section 3. The second part of the paper described in Section 4 is specifically about task specification and provides details about task representation and semantics through the use of Task Specification Trees. This section also provides an example. The paper then concludes with related work and conclusions.

2. SEMANTIC PERSPECTIVE

In [3, 13], Falcone & Castelfranchi provide an illuminating, but informal discussion about delegation as a concept from a social perspective. Their approach to delegation builds on a BDI model of agents, that is, agents having beliefs, goals, intentions, and plans [4], but the specification lacks a formal semantics for the operators used. Based on intuitions from their work, we have previously provided a formal characterization of their concept of strong delegation using a communicative speech act with pre- and post-conditions which update the belief states associated with the delegator and contractor, respectively [10]. In order to formally characterize the operators used in the definition of the speech act, we use KARO [20] to provide a formal semantics. The KARO formalism is an amalgam of dynamic logic and epistemic / doxastic logic, augmented with several additional (modal) operators in order to deal with the motivational aspects of agents.

First, we define the notion of a task as a pair consisting of a goal and a plan for that goal, or rather, a plan and the goal associated with that plan. Paraphrasing Falcone & Castelfranchi into KARO terms, we consider a notion of strong/strict delegation represented by a speech act S-Delegate(A, B, τ) of A delegating a task $\tau = (\alpha, \phi)$ to B, where α is a possible plan and ϕ is a goal. It is specified as follows:

S-Delegate(A, B, τ), where $\tau = (\alpha, \phi)$

Preconditions:

(1) $Goal_A(\phi)$

(2) $Bel_A Can_B(\tau)$ (Note that this implies $Bel_A Bel_B(Can_B(\tau))$)

(3) $Bel_A(Dependent(A, B, \alpha))$

Postconditions:

(1) $Goal_B(\phi)$ and $Bel_B Goal_B(\phi)$

(2) $Committed_B(\alpha)$.

(3) $Bel_B Goal_A(\phi)$

(4) $Can_B(\tau)$ (and hence $Bel_B Can_B(\tau)$, and by (1) also $Intend_B(\tau)$)

(5) $MutualBel_{AB}$("the statements above" $\wedge$ $SociallyCommitted(B, A, \tau)$)[3]

Informally speaking this expresses the following: the preconditions of the S-delegation act of A delegating task τ to B are that (1) ϕ is a goal of delegator A (2) A believes that B can (is able to) perform the task τ (which implies that A believes that B himself believes that he can do the task!) (3) A believes that with respect to the task τ he is dependent on B.

The postconditions of the delegation act mean: (1) B has ϕ as his goal and is aware of this (2) he is committed to the task (3) B believes that A has the goal ϕ (4) B can do the task τ (and hence believes it can do it, and furthermore it holds that B intends to do the task, which was a separate condition in Falcone & Castelfranchi's set-up), and (5) there is a mutual belief between A and B that all preconditions and other postconditions mentioned hold, as well as that there is a contract between A and B, i.e. B is socially committed to A to achieve τ for A. In this situation we will call agent A the delegator and B the contractor.

Typically a social commitment (contract) between two agents induces obligations to the partners involved, depending on how the task is specified in the delegation action. This dimension has to be added in order to consider how the contract affects the autonomy of the agents, in particular the contractor's autonomy. We consider a few relevant forms of delegation specification below.

2.1 Closed vs Open Delegation

Falcone & Castelfranchi furthermore discuss the following variants of task specification:

- closed delegation: the task is completely specified: both goal and plan should be adhered to.
- open delegation: the task is not completely specified: either only the goal has to be adhered to while the plan may be chosen by the contractor, or the specified plan contains 'abstract' actions that need further elaboration (a 'sub-plan') to be dealt with by the contractor.

So in open delegation the contractor may have some freedom to perform the delegated task, and thus it provides a large degree of flexibility in multi-agent planning, and allows for truly distributed planning.

The specification of the delegation act in the previous subsection was in fact based on closed delegation. In case of open delegation, α in the postconditions can be replaced by an α', and τ by $\tau' = (\alpha', \phi)$. Note that the fourth clause, viz. $Can_B(\tau')$, now implies that α' is indeed believed to be an alternative for achieving ϕ, since it implies that $Bel_B[\alpha']\phi$ (B believes that ϕ is true after α' is executed). Of course, in the delegation process, A must agree that α' is indeed viable. This would depend on what degree of autonomy is allowed.

This particular specification of delegation follows Falcone & Castelfranchi closely. One can easily foresee other constraints one might add or relax in respect to the basic specification resulting in other variants of delegation [5, 7]. In [10], we also provide an instantiation of the delegation framework using 2APL, a popular agent programming language.

3. PRAGMATIC PERSPECTIVE

From a semantic perspective, delegation as a speech act provides us with conceptual insight and an abstract specification which can be used as a basis for a more pragmatic implementation on actual UAS platforms. There is a large gap between semantics and pragmatics which one would like to reduce in a principled manner. To do this, we have chosen to also work from a bottom-up perspective and have developed a prototype software system that implements the delegation framework using a JADE-based architecture specified in the next section. This system has been tested using a number of complex collaborative scenarios described in [14, 17].

One particularly interesting result of approaching the complex characterization of delegation from a top-down abstract semantic perspective and a bottom-up implementation perspective is that one can ground the semantic insights into the implementation in a very direct manner. A central component in the speech-act based characterization of delegation is the use of $Can()$ in the pre-conditions to the speech act. It turns out that verifying the truth of the $Can()$ preconditions becomes equivalent to checking the satisfiability of a distributed constraint network generated through recursive calls to the delegation operator in the implementation. This will be shown in Section 4.

3.1 An Agent-Based UAS Architecture

Our RMAX helicopters use a CORBA-based distributed architecture [9]. For our experimentation with collaborative UASs, we view this as a legacy system and extend it with what is conceptually an additional outer layer in order to leverage the functionality of JADE [12]. "JADE (Java Agent Development Framework) is a software environment to build

[3]A discussion pertaining to the semantics of all non-KARO modal operators may be found in [10].

agent systems for the management of networked information resources in compliance with the FIPA specifications for interoperable multi-agent systems." [12]. The reason for using JADE is pragmatic. Our formal characterization of the *Delegate()* operator is as a speech act. We also use speech acts as an agent communication language and JADE provides a straightforward means for integrating the FIPA ACL language which supports speech acts with our existing systems. The outer layer may be viewed as a collection of JADE agents that interface to the legacy system. We are currently using four agents in the outer layer:

1. **Interface agent** - This agent is the clearinghouse for communication. All requests for delegation and other types of communication pass through this agent. Externally, it provides the interface to a specific robotic system.
2. **Delegation agent** - The delegation agent coordinates delegation requests to and from other UAS systems, with the Executor, Resource and Interface agents. It does this essentially by verifying that the pre-conditions to a *Delegate*() request are satisfied. In particular, calls to the Resource agent determine whether *Can*() assertions are satisfiable.
3. **Executor agent** - After a task is contracted to a particular UAS, it must eventually execute that task relative to the constraints associated with it. The Executor agent coordinates this execution process.
4. **Resource agent** - The Resource agent determines whether the UAS of which it is part has the resources and ability to actually do a task as a potential contractor. Such a determination may include the invocation of schedulers and planners to satisfy goal requests and use of constraint solvers in order to determine whether local and global constraints can be satisfied.

A prototype implementation of this system has been tested both in the field with RMAX helicopters and in-the-loop simulation.

4. TASK SPECIFICATION TREES

We require a formal specification for a task representation which we call *task specification trees* (TST's). This representation has to be implicitly sharable, dynamically extendable, and distributed in nature. Such a task structure is passed from one agent to another and possibly extended in more detail as the delegation process is invoked recursively among agents and humans. If the delegation process is successful, the resulting shared structure is in fact executable in a distributed manner. The Delegation agents associated with specific UAS's are responsible for passing and extending such structures in order to meet the requirements of goal specifications or instantiations of abstract task specifications. The Executor agents associated with specific UAS's have the capacity to execute specific nodes in a shared task specification tree that have been delegated to them. The Resource agents include constraint solvers which are used to check the consistency of local and global constraints, and they sometimes participate in distributed constraint solving structures across UAS platforms.

4.1 TST Concepts

In this section, we describe the syntax and semantics of a task specification format for specifying constraint-based task trees. We call a task-tree specified in the task specification format a task specification tree.

A TST is a distributed data structure with a declarative representation that describes a complex multiagent task. Each node in a TST corresponds to a task that should be performed. Each node has a *node interface* containing a set of parameters that can be specified for the node. The parameter *platform assignment* determines the platform that should execute the node, if specified. The other parameters of the node interface, called *node parameters*, determine task specific details of the node. Instantiating the platform assignment and node parameters partially determines the degree of autonomy for the task the TST specifies.

Nodes in a TST either specify actions or goals. Action nodes can be executed when instantiated, whereas goal nodes require a plan to be generated when instantiated. The plan then becomes a new TST branch that in turn can be instantiated and executed. Nodes can also be removed and added during execution, for example, to repair a TST after a failure. When a TST has been executed, the resulting TST structure is in fact a representation of the history of what has been done, including concrete task instantiations, errors, and repairs.

A TST has a constraint model for each node. The constraints associated with a node are called *node constraints*. A TST also has *tree constraints*, expressing precedence and organizational relations between the nodes in the TST. Together the constraints form a constraint network covering the TST. In fact, the node parameters function as constraint variables in a constraint network, and setting the value of a node parameter constrains not only the network, but implicitly, the degree of autonomy of an agent.

A primitive action node is also called an *elementary task*. What defines a task as elementary is application dependent, although node constraints do integrate with the larger tree constraint network (see Figure 4). A composite action node is also called a composite task.

A platform that can be assigned a task in a TST is called a *candidate*. Whether a platform can be assigned or not is determined by the *capabilities* and resources required for a node. Only platforms with the required capabilities can be candidates. The TST concepts are shown in Figure 3.

Nodes N_0 and N_1 are composite action nodes, sequential and concurrent, respectively. Nodes N_2, N_3 and N_4 are primitive action nodes (also called elementary tasks). Each node specifies a task and has a node interface containing node parameters and a platform assignment variable. In this case only temporal parameters are shown representing the respective intervals a task should be completed in. The parameters in the node interface correspond to global variables that can be used in tree constraints relating the nodes in the tree.

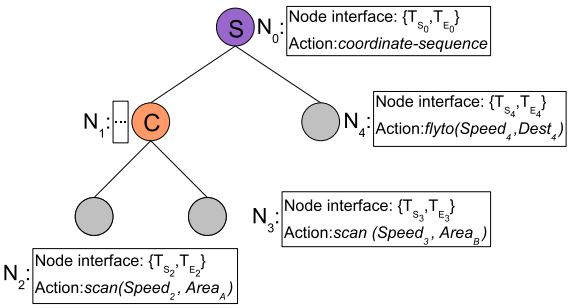

Figure 3: A TST with the nodes N_0–N_4.

4.2 TST Syntax

A TST specification is a TST node. A TST node consists of local variables, a task type and constraints. The task type is either a primitive action, a goal or a composite action. Composite actions apply their composition relation to one or more child tasks.

The syntax of a TST specification has the following BNF:

SPEC ::= TST

TST ::= NAME ('(' VARS ')')? '='
(**with** VARS)? TASK (**where** CONS)?

TSTS ::= TST | TST ';' TSTS

TASK ::= ACTION | GOAL |
(NAME '=')? NAME ('(' ARGS ')')? |
while COND TST | **if** COND **then** TST **else** TST |
sequence TSTS | **concurrent** TSTS

VAR ::= <variable name> |
<variable name> '.' <variable name>

VARS ::= VAR | VAR ',' VARS

CONSTRAINT ::= <constraint>

CONS ::= CONSTRAINT | CONSTRAINT **and** CONS

ARG ::= VAR | VALUE

ARGS ::= ARG | ARG ',' ARGS

VALUE ::= <value>

NAME ::= <node name>

COND ::= <ACL query>

GOAL ::= <goal statement>

ACTION ::= <primitive action>

The TST clause in the BNF introduces the main recursive pattern in the specification language.The right hand side of the equality provides the general pattern of providing a variable context for a task (using **with**) and a set of constraints (using **where**) which may include the variables previously introduced. An example of a TST specification for the TST depicted in Figure 3 is provided in Section 4.4.

4.3 TST Semantics

A TST defines what to do as well as constraints on how to do it. In this section we describe informally the complex action defined by a TST and formally what it means for a fully instantiated TST to satisfy its specification. This is defined as a constraint problem. The constraint problem is formed by the capabilities needed for the TST and the actual platforms assigned to the task nodes.

In the delegation framework described in Section 2, the logical predicate $Can_B(\tau)$ states that platform B has the capability to achieve or execute task τ. One precondition for delegation is that the delegator believes in the contractor's capability to achieve or execute τ, and one postcondition for successful delegation is that the contractor has the capability to achieve or execute the task τ.

In this section, it is shown how our previous formal specification of delegation is connected to the dynamic formation of a constraint problem for an assigned TST, where a TST corresponds to a task $\tau_k = [\alpha_k, \phi_k]$ [10]. In this case, the *Can* predicate is extended to include a node interface $[t_s, t_e]$ representing begin and end times for a node/task and an additional constraint set *cons*. Observe that *cons* can be formed incrementally and may in fact contain constraints inherited or passed to it through a recursive delegation process.

$Can(B, \tau, [t_s, t_e], cons)$ then asserts that an agent B has the capability for achieving or executing task τ if *cons* together with the node constraints for τ are consistent. The task $\tau_k = [\alpha_k, \phi_k]$ consists of a tuple containing α_k, a plan or action(complex or elementary) and ϕ_k, a goal (which may be empty). The temporal variables $[t_s, t_e]$ associated with the task τ are part of the node interface which may also contain other variables and are often related to *cons*.

For each type of composite action (sequence, concurrent, loop, selector), we can now describe the meaning of its associated task as part of a larger constraint problem which is the union of the constraints for the participating sub-nodes in the composite action. The tree constraints associated with composite action nodes are independent of the platform assigned to the node in the sense that they are domain independent and their execution is standard. In contrast to composite action nodes, the meaning of a primitive action node is expressed as a platform dependent constraint problem and execution is dependent on the platform involved. We describe one primitive action node for the capability *flyto* for a UAS platform as an example (the definition of *flyto* may be different for different platforms). The meaning of a goal node is the constraint problem associated with the plan that is generated for achieving the goal.

Sequential

- In a *sequence node*, the child nodes should be executed in sequence (from left to right) during the execution time of the sequence node.

- $Can(B, S(\alpha_1, ..., \alpha_n), [t_s, t_e], cons) \leftrightarrow$
$\exists t_1, \ldots, t_{2n} \bigwedge_{k=1}^{n} (Can(B, \alpha_k, [t_{2k-1}, t_{2k}], cons_B)$
$\vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}], cons_B)) \wedge$
$consistent(cons_B)$[4]

- $cons_B =$
$\{t_s \leq t_1 \wedge (\bigwedge_{i=1}^{n} t_{2i-1} < t_{2i} \wedge t_{2i} \leq t_{2i+1}) \wedge t_{2n+1} \leq t_e\} \cup cons$

[4]The predicate *consistent*() has the standard logical meaning.

Concurrent

- In a *concurrent node* each child node should be executed during the time interval of the concurrent node. [5]

- $Can(B, C(\alpha_1, ..., \alpha_n), [t_s, t_e], cons) \leftrightarrow$
  $\exists t_1, \ldots, t_{2n} \bigwedge_{k=1}^{n} (Can(B, \alpha_k, [t_{2k-1}, t_{2k}], cons_B)$
  $\vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}], cons_B)) \wedge$
  $sconsistent(cons_B)$

- $cons_B =$
  $\{\bigwedge_{i=1}^{n} t_s \leq t_{2i-1} < t_{2i} \leq t_e\} \cup cons$

Elementary Action

- An *elementary action node* specifies a domain-dependent action (flyto in this example). An elementary action node is a leaf node.

- $Can(B, flyto(stime, etime, speed, pos), [t_s, t_e], cons) \leftrightarrow$
  $consistent(cons_B)$

- $cons_B = \{t_e = t_s + dist(location(t_s), pos)/speed \wedge$
  $min\_speed \leq speed \leq max\_speed \wedge t_s < t_e\} \cup cons$ [6]

Selector Node

- Compared to a sequence or concurrent node, only one of the *selector node*'s children will be executed, which one is determined by a test condition in the selector node. The child node should be executed during the time interval of the selector node. A selector node is used to postpone a choice which can not be known when the TST is specified.

Loop Node

- A *loop node* will add a child node for each iteration the loop condition allows. In this way the loop node works as a sequence node but with an increasing number of child nodes which are dynamically added. Loop nodes are similar to selector nodes, they describe additions to the TST that can not be known when the TST is specified.

Goal

- A *goal node* is a leaf node which can not directly be executed, instead it has to be expanded through a call to an automated planner. After expansion a TST branch representing the generated plan is added to the original TST.

- $Can(B, Goal(\phi), [t_s, t_e], cons) \leftrightarrow$
  $\exists \alpha\, (GeneratePlan(B, \alpha, \phi, [t_s, t_e], cons) \wedge$
  $Can(B, \alpha, [t_s, t_e], cons)) \vee$
  $\exists a_k Delegate(B, a_k, Goal(\phi), [t_s, t_e], cons)$

[5] Concurrency in this case does not necessarily mean that the execution of the child nodes should be concurrent, they can be in sequence, but concurrency is *allowed*, so $S \subseteq C$.Comparing S and C to Allen's interval algebra [1], C can cover all 13 relations, and S can cover 4 of the relations (b,m,bi,mi).

[6] Additional constraints would be required in general to model resource allocation, etc. For clarity, we leave these nuances outside the example.

4.4 Example

Assume we have a TST structure as depicted in Figure 3. Let's assume that nodes N_0 to N_4 have the task names τ_0 to τ_4 associated with them, respectively. This TST contains two composite actions, with the capabilities *sequence* (τ_0) and *concurrent* (τ_1). The primitive action nodes τ_2 and τ_3 require a *scan* capability and τ_4 requires a *flyto* capability. We can assume that the delegation process together with a platform allocation algorithm [2] has found three UAS platforms, P_0, P_1 and P_2, with the appropriate capabilities. P_0 has been assigned and/or delegated the composite actions τ_0 and τ_1. P_0 has recursively delegated parts of these tasks to P_1 (τ_2 and τ_4) and $P_2(\tau_3)$.

The specification for this TST is:

$\tau_0(T_{S_0}, T_{E_0}) =$
with $T_{S_1}, T_{E_1}, T_{S_4}, T_{E_4}$ **sequence**
$\tau_1(T_{S_1}, T_{E_1}) =$
with $T_{S_2}, T_{E_2}, T_{S_3}, T_{E_3}$ **concurrent**
$\tau_2(T_{S_2}, T_{E_2}) = \text{scan}(T_{S_2}, T_{E_2}, Speed_2, Area_A);$
$\tau_3(T_{S_3}, T_{E_3}) = \text{scan}(T_{S_3}, T_{E_3}, Speed_3, Area_B)$
where $cons_{\tau_1}$;
$\tau_4(T_{S_4}, T_{E_4}) = \text{flyto}(T_{S_4}, T_{E_4}, Speed_4, Dest_4)$
where $cons_{\tau_0}$

- $cons_{\tau_0} = \{T_{S_0} \leq T_{S_1} \wedge T_{S_1} \leq T_{E_1} \wedge T_{E_1} \leq T_{S_4} \wedge$
  $T_{S_4} \leq T_{E_4} \wedge T_{E_4} \leq T_{E_0}\}$
- $cons_{\tau_1} = \{T_{S_1} \leq T_{S_2} \wedge T_{S_2} \leq T_{E_2} \wedge T_{E_2} \leq T_{E_1} \wedge$
  $T_{S_1} \leq T_{S_3} \wedge T_{S_3} \leq T_{E_3} \wedge T_{E_3} \leq T_{E_1}\}$

The constraint problem for the TST is derived by recursively reducing the Can predicates for each task with "semantically" equivalent expressions, beginning with the top-node τ_0 until the logical statements reduce to a constraint network:

$Can(P_0, \alpha_0, [t_{s_0}, t_{e_0}], cons) =$
$Can(P_0, S(\alpha_1, \alpha_4), [t_{s_0}, t_{e_0}], cons) \leftrightarrow$
$\exists t_{s_1}, t_{e_1}, t_{s_4}, t_{e_4}\, ((Can(P_0, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0})$
$\vee \exists a_1 (Delegate(a_1, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0})))$
$\wedge (Can(P_0, \alpha_4, [t_{s_4}, t_{e_4}], cons_{P_0}) \vee$
$\exists a_2 (Delegate(a_2, \alpha_4, [t_{s_4}, t_{e_4}], cons_{P_0}))))$

Let's continue with a reduction of the 1st element in the sequence α_1 (The 1st conjunct in the previous formula on the right-hand side of the biconditional):

$Can(P_0, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0})$
$\vee \exists a_1 (Delegate(a_1, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0}))$

We can assume that P_0 has been allocated α_1, so the 2nd disjunct is false.

$Can(P_0, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0}) =$
$Can(P_0, C(\alpha_2, \alpha_3), [t_{s_1}, t_{e_1}], cons_{P_0}) \leftrightarrow$
$\exists t_{s_2}, t_{e_2}, t_{s_3}, t_{e_3}\, ((Can(P_0, \alpha_2, [t_{s_2}, t_{e_2}], cons_{P_0}) \vee$
$\exists a_1\, Delegate(a_1, \alpha_2, [t_{s_2}, t_{e_2}], cons_{P_0})) \wedge$
$(Can(P_0, \alpha_3, [t_{s_3}, t_{e_3}], cons_{P_0}) \vee$
$\exists a_2\, Delegate(a_2, \alpha_3, [t_{s_3}, t_{e_3}], cons_{P_0})))$

The node constraints for τ_0 and τ_1 are then added to P_0's constraint store. What remains to be done is a reduction of tasks τ_2 and τ_4 associated with P_1 and τ_3 associated with P_2. We can assume that P_1 has been delegated α_2 and P_2 has been delegated α_3 as specified. Consequently, we can reduce to

$$
\begin{aligned}
&Can(P_0, \alpha_1, [t_{s_1}, t_{e_1}], cons_{P_0}) = \\
&Can(P_0, C(\alpha_2, \alpha_3), [t_{s_1}, t_{e_1}], cons_{P_0}) \leftrightarrow \\
&\exists t_{s_2}, t_{e_2}, t_{s_3}, t_{e_3} \, (Can(P_1, \alpha_2, [t_{s_2}, t_{e_2}], cons_{P_0}) \wedge \\
&Can(P_2, \alpha_3, [t_{s_3}, t_{e_3}], cons_{P_0}))
\end{aligned}
$$

These remaining tasks are primitive or elementary tasks and consequently platform dependent. The definition of *Can* for a primitive action node is platform dependent. When a platform is assigned to a primitive action node a local constraint problem is created on the platform and then connected to the global constraint problem through the node parameters of the assigned node's node interface. In this case, the node parameters only include temporal constraints and these are coupled to the internal constraint variables associated with the primitive actions.The completely allocated and reduced TST is shown in Figure 4. The reduction of *Can* for any primitive action node contains no further *Can* predicates, since a primitive action only depends on the platform itself. All remaining *Can* predicates in the recursion are replaced with constraint sub-networks associated with specific platforms as shown in Figure 4.

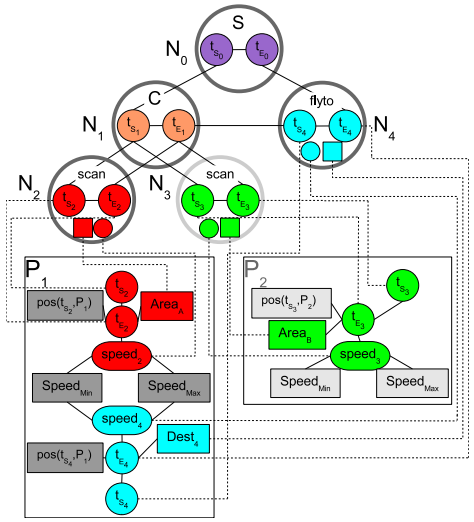

Figure 4: The completely allocated and reduced TST showing interaction between the global and internal platform dependent constraints.

The net result is that a logical specification of collaborative capabilities required for successful achievement of a complex set of distributed tasks is formally reduced to a distributed constraint problem. So, in effect, the semantic basis for a TST and what couples it to the abstract delegation speech act, corresponds to a constraint problem and provides a pragmatic implementation correlate of the formal abstraction. The satisfiability of which provides real-world grounding to the abstract delegation speech act.

An unassigned composite action node and an unassigned primitive action node have different implications. The constraints of composite actions nodes are global and related to the tree shape created by the composition of the composite actions. For global constraints, the choice of platform does not matter, it still results in a similar constraint problem. Primitive action nodes on the other hand create a unique local constraint problem for the node platform combination, implying that the choice of platform in such cases must be handled with greater care. If we arbitrarily assign platforms to τ_2, τ_3 and τ_4 in the example, this may result in a constraint network that is inconsistent, but it does not mean that there exist no platform assignment combinations to τ_2, τ_3 and τ_4 where the resulting constraint network actually is consistent. This is why solving the task allocation problem entails a search among possible assignment combinations and the use of distributed constraint algorithms to determine proper allocation. A number of allocation algorithms for dealing with this problem are reported in [2].

5. RELATED WORK

Two related task specification languages which are representative of state of the art in this area are the Configuration Description Language [15], used in *MissionLab* and the task description language (TDL) [18].

CDL has a recursive composition of configurations, similar to our TST task structure. In CDL a behavior and a set of parameters creates an agent. Agents can be composed into larger entities, called assemblages, that function as macro-agents. Assemblages can in turn be part of larger assemblages. CDL has been used as the basis for MissionLab, a tool for mission specification using case based reasoning. Task-allocation is done using a market-based paradigm with contract-nets. Task allocation can be done together with mission specification, or at run time [19].

With TDL it is possible to specify task decomposition, synchronization, execution monitoring, and exception handling. TDL is an extension to C++, meaning the specification is compiled and executed on the robots. Task are in the form of task-trees. A task has parameters and is either a goal or a command, where a command is similar to an action node in a TST. Goal nodes can have both goal and command nodes as children, but commands nodes have no goal children. An action can perform computations dynamically and add child nodes or perform some physical action in the world. An action can contain conditional, iterative and recursive code.

Both CDL and TDL are similar to TST, but with the difference that the specification of a TST is not precompiled and therefore allow more dynamic handling of tasks in the case of changing circumstances. The specification remains through the stages of task-allocation (delegation) and execution. Each node in a TST has parameter values which are restricted by constraints. Each node has an executor object (for each platform) that can be instantiated with the parameter values determined in the task allocation stage. Since we have this separation between specification and execution of a task, connected as a constraint problem of the node parameters and platform assignments, we can go back and forth from the task-allocation and execution stage, which must be done when monitoring formulas fails and an error is de-

tected, or when the mission is changed with mixed-initiative input. The loose coupling between specification and execution is needed for combining the adjustable autonomy and mixed-initiative features.

6. CONCLUSIONS

The complexity of developing deployed architectures for realistic collaborative activities among agents that operate in the real world and under time and space constraints is extreme when compared to much existing formal work which tackles parts of the larger problem at very high levels of abstraction. We have tried to show the benefits of using both strategies, working abstractly at a formal logical level and also concretely at a system building level. More importantly, we have shown how one might relate the two approaches to each other by *grounding* the formal abstractions into actual software implementations. This of course guarantees the fidelity of the actual system to the formal specification.

We proposed TSTs as a vehicle for representing tasks and showed how they relate to the formal delegation abstraction, how its semantics can be described as a constraint model and how that model is used in an actual implemented system to give meaning to the ability of an agent to be able to do or execute a task. There is much future work to be done in this complex research area, but work in this direction can continue based on the foundations provided in this work.

7. REFERENCES

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[2] D.Landén and F. Heintz and P. Doherty. Complex task allocation in mixed-initiative delegation: A UAV case study (Early Innovation). The 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA-2010), 2010.

[3] C. Castelfranchi and R. Falcone. Toward a theory of delegation for agent-based systems. In *Robotics and Autonomous Systems*, volume 24, pages 141–157, 1998.

[4] P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[5] P. Cohen and H. Levesque. Teamwork. *Nous, Special Issue on Cognitive Science and AI*, 25(4):487–512, 1991.

[6] G. Conte, M. Hempel, P. Rudol, D. Lundström, S. Duranti, M. Wzorek, and P. Doherty. High accuracy ground target geo-location using autonomous micro aerial vehicle platforms. In *Proceedings of the AIAA-08 Guidance,Navigation, and Control Conference*, 2008.

[7] E. Davis and L. Morgenstern. A first-order theory of communication and multi-agent plans. *Journal Logic and Computation*, 15(5):701–749, 2005.

[8] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, 2000.

[9] P. Doherty, P. Haslum, F. Heintz, T. Merz, T. Persson, and B. Wingman. A distributed architecture for intelligent unmanned aerial vehicle experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, 2004.

[10] P. Doherty and J.-J. C. Meyer. Towards a delegation framework for aerial robotic mission scenarios. In *Proceedings of the 11th International Workshop on Cooperative Information Agents*, 2007.

[11] S. Duranti, G. Conte, D. Lundström, P. Rudol, M. Wzorek, and P. Doherty. LinkMAV, a prototype rotary wing micro aerial vehicle. In *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace*, 2007.

[12] G. C. F. Bellifemine, F. Bergenti and A. Poggi. JADE – a Java agent development framework. In J. D. R. H. Bordini, M. Dastani and A. Seghrouchni, editors, *Multi-Agent Programming - Languages, Platforms and Applications*. Springer, 2005.

[13] R. Falcone and C. Castelfranchi. The human in the loop of a delegated agent: The theory of adjustable social autonomy. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 31(5):406–418, 2001.

[14] J. Kvarnström and P. Doherty. Automated planning for collaborative systems. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010.

[15] D. C. MacKenzie, R. Arkin, and J. M. Cameron. Multiagent mission specification and execution. *Auton. Robots*, 4(1):29–52, 1997.

[16] T. Merz, P. Rudol, and M. Wzorek. Control System Framework for Autonomous Robots Based on Extended State Machines. In *Proceedings of the International Conference on Autonomic and Autonomous Systems*, 2006.

[17] P.-M. Olsson, J. Kvarnström, P. Doherty, O. Burdakov, and K. Holmberg. Generating UAV communication networks for monitoring and surveillance. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010.

[18] R. Simmons and D. Apfelbaum. A task description language for robot control. In *in Proceedings of the Conference on Intelligent Robots and Systems (IROS*, 1998.

[19] P. Ulam, Y. Endo, A. Wagner, and R. C. Arkin. Integrated mission specification and task allocation for robot teams - design and implementation. In *ICRA*, pages 4428–4435, 2007.

[20] B. v. L. W. van der Hoek and J.-J. C. Meyer. An integrated modal approach to rational agents. In M. Wooldridge and A. Rao, editors, *Foundations of Foundations of Rational Agency*, volume 14 of *Applied Logic Series*. An Integrated Modal Approach to Rational Agents, 1998.

[21] M. Wzorek, G. Conte, P. Rudol, T. Merz, S. Duranti, and P. Doherty. From motion planning to control – a navigation framework for an unmanned aerial vehicle. In *Proceedings of the 21st Bristol International Conference on UAV Systems*, 2006.