A Delegation-Based Cooperative Robotic Framework

Patrick Doherty and Fredrik Heintz
Dept. of Computer and Information Science, Linköping University, Sweden
{patrick.doherty, fredrik.heintz}@liu.se

***Abstract*— Cooperative robotic systems, such as unmanned aircraft systems, are becoming technologically mature enough to be integrated into civil society. To gain practical use and acceptance, a verifiable, principled and well-defined foundation for interactions between human operators and autonomous systems is needed. In this paper, we propose and specify such a formally grounded collaboration framework. Collaboration is formalized in terms of the concept of delegation and delegation is instantiated as a speech act. Task Specification Trees are introduced as both a formal and pragmatic characterization of tasks and tasks are recursively delegated through a delegation process. The delegation speech act is formally grounded in the implementation using Task Specification Trees, task allocation via auctions and distributed constraint solving. The system is implemented as a prototype on unmanned aerial vehicle systems and a case study targeting emergency service applications is presented.**

I. Introduction

Cooperative robotic systems, such as unmanned aircraft systems, are becoming technologically mature enough to be integrated into civil society. To gain practical use and acceptance, a verifiable, principled and well-defined foundation for interactions between human operators and autonomous systems is needed. This interaction is going to be mixed-initiative in nature. Humans will request help from robotic systems and robotic systems will request help from humans when collaborating to achieve complex missions in unstructured and challenging environments. In developing a principled framework for such sophisticated interactions, many interdependent conceptual and pragmatic issues arise and need clarification both theoretically and pragmatically.

The complexity of developing deployed architectures for realistic cooperative activities among robots that operate in the real world under time and space constraints is very high. We tackle this complexity by working both abstractly at a formal logical level and concretely at a systems building level. More importantly, the two approaches are related to each other by grounding the formal abstractions into actual software implementations. This guarantees the fidelity of the actual system to the formal specification.

This paper presents a principled formal framework for cooperative robotic systems based on delegation. The basis for the principled framework for interaction between human operators and robotic systems is a triad of fundamental, interdependent conceptual issues: delegation, mixed-initiative interaction and adjustable autonomy. These concepts are used to clarify, validate and verify different types of interaction between robotic platforms and human operators. The concept of delegation is particularly important and provides in a sense a bridge between mixed-initiative interaction and adjustable autonomy.

Delegation – In any mixed-initiative interaction, humans may request help from robotic systems and robotic systems may request help from humans. One can model such requests as a form of delegation, $Delegate(A, B, task, constraints)$, where A is the delegating agent, B is the contractor, $task$ is the task being delegated which 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. In our framework, delegation is formalized as a speech act and the delegation process invoked can be recursive.

Adjustable Autonomy – In solving tasks in a mixed-initiative setting, the robotic systems 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. One can begin to develop a principled means of adjusting autonomy through the use of the $task$ and $constraint$ parameters in $Delegate$. 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 robotic system, such as a UAV, 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 UAV, $Delegate(GOP, UAV, task, constraints)$ and in a symmetric manner, the ability of a UAV to be able to delegate tasks to a GOP, $Delegate(UAV, GOP, task, constraints)$. 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.

This work is partially supported by grants from the Swedish Research Council (VR) Linnaeus Center CADICS, VR grant 2009-3857, the ELLIIT Excellence Center at Linköping-Lund in Information Technology, NFFP5 The Swedish National Aviation Engineering Research Program, and the Center for Industrial Information Technology CENIIT.

II. Delegation as a Speech Act

Delegation is central to the conceptual and architectural framework we propose. Consequently, formulating an abstraction of the concept with a formal specification amenable to pragmatic grounding and implementation in a software system is paramount. As a starting point, Falcone & Castelfranchi provide an illuminating, but informal discussion about delegation as a concept from a social perspective [1], [2]. Their approach to delegation builds on a BDI model of agents, that is, agents having beliefs, goals, intentions, and plans [3]. However, their 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 [4]. Strong delegation means that the delegation is explicit. The formal characterization of the speech act is expressed in KARO [5]. The KARO formalism is an amalgam of dynamic logic and epistemic/doxastic logic, augmented with additional modal operators to deal with the motivational aspects of agents.

The target for delegation is a *task*. The Merriam-Webster dictionary definition of a task is "a usually assigned piece of work often to be finished within a certain time". Assigning a piece of work to someone by someone is in fact what delegation is about. In computer science, a *piece of work* in this context is generally represented as a composite action. There is also often a purpose to assigning a piece of work to be done. This purpose is generally represented as a *goal*, where the intended meaning is that a task is a means of achieving a goal. We will require both a formal specification of a task at a high-level of abstraction in addition to a more data-structural specification flexible enough to be used pragmatically in an implementation. For the formal specification, the definition provided by Falcone & Castelfranchi will be used. For the data-structure specification used in the implementation, Task Specification Trees will be defined in a Section III.

Falcone & Castelfranchi define a task as a pair $\tau = (\alpha, \phi)$ consisting of a goal ϕ, and a plan α for that goal, or rather, a plan and the goal associated with that plan. Conceptually, a plan is a composite action. At this abstraction level, the definition of a task is purposely left general. For instance, timing and resource issues are abstracted away although they will be dealt with explicitly in the implementation.

From the perspective of adjustable autonomy, the task definition is quite flexible. If α is a single elementary action with the goal ϕ implicit and correlated with the post-condition of the action, the contractor has little flexibility as to how the task will be achieved. On the other hand, if the goal ϕ is specified and the plan α is not provided, then the contractor has a great deal of flexibility in achieving the goal. There are many variations between these two extremes and these variations capture the different levels of autonomy and trust exchanged between two agents.

Paraphrasing Falcone & Castelfranchi into KARO terms, we consider a notion of strong delegation represented by a speech act $S-Delegate(A,\ B,\ \tau)$ of A delegating a task $\tau = (\alpha, \phi)$ to B, where α is a possible plan and ϕ is a goal.

Preconditions:
(1) $Goal_A(\phi)$
(2) $Bel_A Can_B(\tau)$ (implies $Bel_A Bel_B(Can_B(\tau))$)
(3) $Bel_A(Dependent(A, B, \tau))$
(4) $Bel_B Can_B(\tau)$

Postconditions:
(1) $Goal_B(\phi)$ and $Bel_B Goal_B(\phi)$
(2) $Committed_B(\alpha)$ (also written $Committed_B(\tau)$)
(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) $Intend_A(do_B(\alpha))$
(6) $MutualBel_{AB}$("the statements above" $\wedge\ SociallyCommitted(B, A, \tau)$)[1]

This particular characterization 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 [6], [7].

III. Task Specification Trees

Both the declarative and procedural representation and semantics of tasks are central to the delegation process. The relation between the two representations is also essential if one has the goal of formally grounding the delegation process in the system implementation. A task was previously defined abstractly as a pair (α, ϕ) consisting of a composite action α and a goal ϕ. In this section, we introduce a formal task specification language which allows us to represent tasks as *Task Specification Trees* (TSTs).

For our purposes, the task representation must be highly flexible, sharable, dynamically extendible, and distributed in nature. Tasks need to be delegated at varying levels of abstraction and also expanded and modified because parts of complex tasks can be recursively delegated to different robotic agents which are in turn expanded or modified. Consequently, the structure must also be distributable. Additionally, a task structure is a form of compromise between an explicit plan in a plan library at one end of the spectrum and a plan generated through an automated planner [8], [9] 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.

The task specification formalism should allow for the specification of various types of task compositions, including sequential and concurrent, in addition to more general constructs such as loops and conditionals. The task specification should also provide a clear separation between tasks and platform specific details for handling the tasks. The

[1]A discussion pertaining to the semantics of non-KARO modal operators may be found in [4].

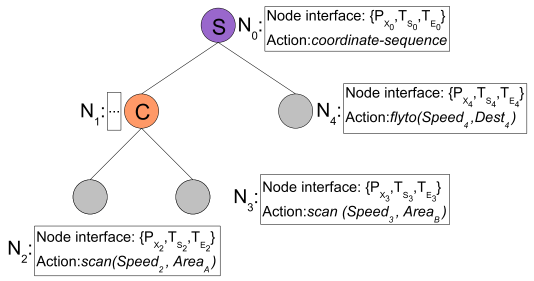
S
N0: Node interface: {P_X0,T_S0,T_E0}
Action:coordinate-sequence
N1: ...
C
N4: Node interface: {P_X4,T_S4,T_E4}
Action:flyto(Speed4,Dest4)
N3: Node interface: {P_X3,T_S3,T_E3}
Action:scan (Speed3, AreaB)
N2: Node interface: {P_X2,T_S2,T_E2}
Action:scan(Speed2, AreaA)

Fig. 1. An example TST.

specification should focus on what should be done and hide the details about how it could be done by different platforms.

In the general case, A TST is a declarative representation of a complex multi-agent task. In the architecture realizing the delegation framework a TST is also a distributed data structure. Each node in a TST corresponds to a task that should be performed. There are six types of nodes: sequence, concurrent, loop, select, goal, and elementary action. All nodes are directly executable except goal nodes which requires some form of expansion or planning to generate a plan for achieving the goal.

Each node has a *node interface* containing a set of parameters, called *node parameters*, that can be specified for the node. The node interface always contains a platform assignment parameter and parameters for the start and end times of the task, usually called P, T_S and T_E. These parameters can be part of the constraints associated with the node 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.

Consider a small scenario where the mission is to first scan $\mathsf{Area_A}$ and $\mathsf{Area_B}$, and then fly to $\mathsf{Dest_4}$. A TST describing this mission is shown in Figure 1. Nodes N_0 and N_1 are composite action nodes, sequential and concurrent, respectively. Nodes N_2, N_3, and N_4 are elementary action nodes. Each node specifies a task and has a node interface containing node parameters and a platform assignment parameter. In this case only temporal parameters are shown representing the intervals tasks should be completed in.

A. TST Syntax

The syntax of a TST specification has the following BNF:

TST ::= NAME ('(' VARS ')')? '=' (**with** VARS)? TASK (**where** CONS)?
TSTS ::= TST | TST ';' TSTS
TASK ::= <elementary action> | <goal>
  | **sequence** TSTS | **concurrent** TSTS
  | **while** <cond> TST | **if** <cond> **then** TST **else** TST
VAR ::= <var name> | <var name> '.' <var name>
VARS ::= VAR | VAR ',' VARS
CONS ::= <constraint> | <constraint> **and** CONS
ARG ::= VAR | <value>
ARGS ::= ARG | ARG ',' ARGS
NAME ::= <node name>

Where <elementary action> is an elementary action $name(p_0, ..., p_N)$, <goal> is a goal $name(p_0, ..., p_N)$, $p_0, ..., p_N$ are parameters, and <cond> is a FIPA ACL query message requesting the value of a boolean expression.

The TST clause introduces the main recursive pattern. 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**) over the variables previously introduced.

Example: Consider the TST depicted in Figure 1. The nodes N_0 to N_4 have the task names τ_0 to τ_4 associated with them. This TST contains two composite actions, *sequence* (τ_0) and *concurrent* (τ_1), and two elementary actions, *scan* (τ_2, τ_3) and *flyto* (τ_4).

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

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

B. TST Semantics

A TST specifies a complex task (composite action) under a set of tree-specific and node-specific constraints which together are intended to represent the context in which a task should be executed in order to meet the task's intrinsic requirements, in addition to contingent requirements demanded by a particular mission. The leaf nodes of a TST represent elementary actions used in the definition of the composite action the TST represents and the non-leaf nodes essentially represent control structures for the ordering and execution of the elementary actions. The semantic meaning of non-leaf nodes is essentially application independent, whereas the semantic meaning of the leaf nodes are highly domain dependent. They represent the specific actions or processes that an agent will in fact execute. The procedural correlate of a TST is a program.

During the delegation process, a TST is either provided or generated to achieve a specific set of goals, and if the delegation process is successful, each node is associated with an agent responsible for the execution of that node.

Informally, the semantics of a TST node will be characterized in terms of whether an agent believes it *can* successfully execute the task associated with the node in a given context represented by constraints, given its capabilities and resources. This can only be a belief because the task will be executed in the future and even under the best of conditions, real-world contingencies may arise which prevent the agent from successfully completing the task. The formal semantics for TST nodes will be given in terms of the logical predicate $Can()$ which we have used previously in the formal definition of the S-Delegate speech act, although in this case, we will add additional arguments. This is not a coincidence since our goal is to ground the formal specification of the S-Delegate speech act into the implementation in a very direct manner.

Recall that in the formal semantics for the speech act S-Delegate (described in Section II), the logical predicate $Can_X(\tau)$ is used to state that an agent X has the capabilities and resources to achieve task τ. An important precondition for the successful application of the speech act is that the delegator (A) believes in the contractor's (B) ability to achieve the task τ, (2): $Bel_A Can_B(\tau)$. Additionally, an important result of the successful application of the speech act is that the contractor actually has the capabilities and resources to achieve the task τ, (4): $Can_B(\tau)$. In order to directly couple the semantic characterization of the S-Delegate speech act to the semantic characterization of TST's, we will assume that a task $\tau = (\alpha, \phi)$ in the speech act characterization corresponds to a TST. Additionally, the TST semantics will be characterized in terms of a Can predicate with additional parameters to incorporate constraints.

In this case, the Can predicate is extended to include as arguments a list $[p_1, \ldots, p_k]$ denoting all node parameters in the node interface together with other parameters provided in the (**with** VARS) construct[2] and an argument for an additional constraint set $cons$ provided in the (**where** CONS) construct.[3] Observe that $cons$ can be formed incrementally and may in fact contain constraints inherited or passed to it through a recursive delegation process. The formula $Can(B, \tau, [t_s, t_e, \ldots], cons)$ then asserts that an agent B has the capabilities and resources for achieving task τ if $cons$, which also contains node constraints for τ, is consistent. The temporal variables t_s and t_e associated with the task τ are part of the node interface which may also contain other variables which are often related to the constraints in $cons$.

Determining whether a fully instantiated TST satisfies its specification, will now be equivalent to the successful solution of a constraint problem in the formal logical sense. The constraint problem in fact provides the formal semantics for a TST. Constraints associated with a TST are derived from a reduction process associated with the $Can()$ predicate for each node in the TST. The generation and solution of constraints will occur on-line during the delegation process. Let us provide some more specific details. In particular, we will show the very tight coupling between the TST's and their logical semantics.

The basic structure of a Task Specification Tree is:

[2]For reasons of clarity, we only list the node parameters for the start and end times for a task, $[t_s, t_e, \ldots]$, in this article.

[3]For pedagogical expediency, we can assume that there is a constraint language which is reified in the logic and is used in the CONS constructs.

TST ::= NAME ('(' VARS$_1$ ')')? '=' (**with** VARS$_2$)? TASK (**where** CONS)?

where VARS$_1$ denotes node parameters, VARS$_2$ denotes additional variables used in the constraint context for a TST node, and CONS denotes the constraints associated with a TST node. Additionally, TASK denotes the specific type of TST node. In specifying a logical semantics for a TST node, we would like to map these arguments directly over to arguments of the predicate $Can()$. Informally, an abstraction of the mapping is

$$Can(ag_1, TASK, VARS_1 \cup VARS_2, CONS) \quad (1)$$

The idea is that for any fully allocated TST, the meaning of each allocated TST node in the tree is the meaning of the associated $Can()$ predicate instantiated with the TST specific parameters and constraints. The meaning of the instantiated $Can()$ predicate can then be associated with an equivalent Constraint Satisfaction Problem (CSP) which turns out to be true or false depending upon whether that CSP can be satisfied or not. The meaning of the fully allocated TST is then the aggregation of the meanings of each individual TST node associated with the TST, in other words, a conjunction of CSPs.

One would also like to capture the meaning of partial TSTs. The idea is that as the delegation process unfolds, a TST is incrementally expanded with additional TST nodes. At each step, a partial TST may contain a number of fully expanded and allocated nodes in addition to other nodes which remain to be delegated. In order to capture this process semantically, one extends the semantics by providing meaning for an unallocated TST node in terms of both a $Can()$ predicate and a $Delegate()$ predicate:

$$\exists ag_2\, Delegate(ag_1, ag_2, TASK, VARS_1 \cup VARS_2, CONS) \quad (2)$$

Either ag_1 can achieve a task, or (exclusively) it can find an agent, ag_2, to which the task can be delegated. In fact, it may need to find one or more agents if the task to be delegated is a composite action.

Given the $S\text{-}Delegate(ag_1, ag_2, TASK)$ speech act semantics, we know that if delegation is successful then as one of the postconditions of the speech act, ag_2 can in fact achieve $TASK$ (assuming no additional contingencies):

$$Delegate(ag_1, ag_2, TASK, VARS_1 \cup VARS_2, CONS) \quad (3)$$
$$\rightarrow Can(ag_2, TASK, VARS_1 \cup VARS_2, CONS)$$

Consequently, during the computational process associated with delegation, as the TST expands through delegation where previously unallocated nodes become allocated, each instance of the $Delegate()$ predicate associated with an unallocated node is replaced with an instance of the $Can()$ predicate. This recursive process preserves the meaning of a TST as a conjunction of instances of the $Can()$ predicate which in turn are compiled into a (interdependent) set of CSPs and which are checked for satisfaction using distributed constraint solving algorithms.

Sequence Node For a *sequence node*, the child nodes should be executed in sequence, from left to right, during the execution time of the sequence node.

$Can(B, S(\alpha_1, ..., \alpha_n), [t_s, t_e, \ldots], cons) \leftrightarrow$
$\exists t_1, \ldots, t_{2n}, \ldots \bigwedge_{k=1}^{n} (Can(B, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k)$
$\quad \vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k))$
$\wedge\, consistent(cons)$[4]
where $cons = \{t_s \leq t_1 \wedge (\bigwedge_{i=1}^{n} t_{2i-1} < t_{2i})$
$\quad \wedge (\bigwedge_{i=1}^{n-1} t_{2i} \leq t_{2i+1}) \wedge t_{2n} \leq t_e\} \cup cons'$

Concurrent Node For a *concurrent node*, the child nodes should be executed during the time interval of the concurrent node.

$Can(B, C(\alpha_1, ..., \alpha_n), [t_s, t_e, \ldots], cons) \leftrightarrow$
$\exists t_1, \ldots, t_{2n}, \ldots \bigwedge_{k=1}^{n} (Can(B, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k)$
$\quad \vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k))$
$\wedge\, consistent(cons)$
where $cons = \{\bigwedge_{i=1}^{n} t_s \leq t_{2i-1} < t_{2i} \leq t_e\} \cup cons'$.

Observe that the constraint sets $cons_k$ in the semantics for the concurrent and sequential nodes are simply the constraint sets defined in the (**where** CONS) constructs for the child nodes included with the sequential or concurrent nodes, respectively. Additionally, the definition of the constraint set $cons$ in the semantics for the concurrent and sequential nodes contains the structural temporal constraints which define sequence and concurrency, respectively, together with possibly additional constraints, denoted by $cons'$ that one may want to include in the constraint set. Note also, that we are assuming that scoping and overloading issues for variables in embedded TST structures are dealt with appropriately in the recursive expansion of the $Can()$ predicates in the definitions.

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. When expanded at runtime, the net result can be any of the node types.

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. When expanded at runtime, the net result is a sequence node.

Goal A *goal node* is a leaf node which can not be directly executed. Instead it has to be expanded by using an automated planner or related planning functionality (such as [8], [9]). After expansion, a TST branch representing the generated plan is added to the original TST.

$Can(B, Goal(\phi), [t_s, t_e, \ldots], cons) \leftrightarrow$
$\exists \alpha\, (GeneratePlan(B, \alpha, \phi, [t_s, t_e, \ldots], cons)$
$\quad \wedge\, Can(B, \alpha, [t_s, t_e, \ldots], cons)) \wedge consistent(cons)$

Observe that the agent B can generate a partial or complete plan α and then further delegate execution or completion of the plan recursively via the $Can()$ statement in the second conjunct.

Elementary Action An *elementary action node* is a leaf node that specifies a domain-dependent action. The semantics of Can for an elementary action is platform dependent.

$Can(B, \tau, [t_s, t_e, \ldots], cons, \ldots) \leftrightarrow$
$\quad Capabilities(B, \tau, [t_s, t_e, \ldots], cons)$
$\wedge\, Resources(B, \tau, [t_s, t_e, \ldots], cons) \wedge consistent(cons)$

There are two parts to the definition of Can for an elementary action node. These are defined in terms of a *platform specification* which is assumed to exist for each agent potentially involved in a cooperative mission. The platform specification has two components.

The first, specified by the predicate $Capabilities(B, \tau, [t_s, t_e, \ldots], cons)$ is intended to characterize all static capabilities associated with platform B that are required as capabilities for the successful execution of τ. If platform B has the necessary static capabilities for executing task τ in the interval $[t_s, t_e]$ with constraints $cons$, then this predicate will be true.

The second, specified by the predicate $Resources(B, \tau, [t_s, t_e, \ldots], cons)$ is intended to characterize dynamic resources such as fuel and battery power, which are consumable, or cameras and other sensors which are borrowable. Since resources generally vary through time, the semantic meaning of the predicate is temporally dependent.

Resources for an agent are represented as a set of parameterized resource constraint predicates, one per task. The parameters to the predicate are the task's parameters, in addition to the start time and the end time for the task. For example, assume there is a task $flyto(dest, speed)$. The resource constraint predicate for this task would be $flyto(t_s, t_e, dest, speed)$. The resource constraint predicate is defined as a conjunction of constraints, in the logical sense. As an example, consider the task $flyto(dest, speed)$ with the corresponding resource constraint predicate $flyto(t_s, t_e, dest, speed)$. The constraint model associated with the task for a particular platform P_1 might be:

$$t_e = t_s + \frac{distance(pos(t_s, P_1), dest)}{speed} \wedge (S_{Min} \leq speed \leq S_{Max})$$

IV. Allocating Tasks to Platforms

The Delegate speech act requires that the delegating agent believes that the contractor has the ability to achieve the task. One central problem is therefore to find an agent which can achieve a particular, potentially very complex, task. When a task becomes complex it is highly likely that no single agent can achieve it alone. Since an agent can achieve a task by delegating parts of it to another agent, recursive delegation can solve the problem. The problem is therefore to find a set of agents who together can achieve a complex task with time, space and resource constraints through recursive delegation. This can be seen as a task allocation problem. The problem is to allocate tasks to platforms and assign values to parameters such that each task can be carried out by its assigned platform and all the constraints are satisfied.

When a platform is assigned an elementary action node in a TST, the constraints associated with that action

[4]The predicate $consistent()$ has the standard logical meaning and checking for consistency would be done through a call to a constraint solver which is part of the architecture.

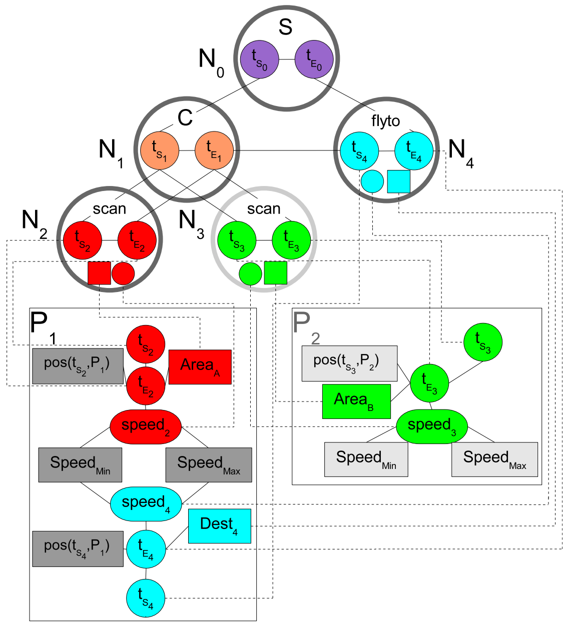

Fig. 2. The completely allocated and reduced TST showing the interaction between the TST constraints and the platform dependent constraints.

are instantiated and added to the constraint store of the platform. The resource constraint is connected to the constraint problem defined by the TST through the node parameters. A platform can be allocated more than one node in a TST. This may introduce implicit dependencies between the actions since each allocation adds resource constraints to the constraint problem of the platform. There can for example be a shared resource that both actions uses.

A *complete allocation* is an allocation which allocates every node in a TST to a platform. A completely allocated TST defines a constraint problem that represents all the constraints for this particular allocation of the TST. As the constraints are distributed among the platforms it is a distributed constraint problem. If the constraint problem is consistent then a *valid allocation* has been found and each solution can be seen as a potential execution schedule of the TST. The consistency of an allocation can be checked by a distributed constraint satisfaction problem (DisCSP) solver such as the Asynchronous Weak Commitment Search (AWCS) algorithm [10] or ADOPT [11].

However, solving the task allocation problem as a single DisCSP problem is currently not possible since the problem is too large even for modest TSTs. Instead we have developed a heuristic search approach for allocating tasks to platforms which uses marginal cost auctions to guide the search [12]. This allows reasonably large TSTs to be allocated.

Example The constraint problem for a TST is derived by recursively reducing the Can predicate statements associated with each task node with formally equivalent expressions, beginning with the top-node τ_0 until the logical statements reduce to a constraint network. Below, we show the reduction of the complex task α_0 represented by the TST in Figure 1 when there are three platforms, P_0, P_1 and P_2, with the appropriate capabilities. P_0 has been delegated the composite action α_0 and has recursively delegated α_2 and α_4 to P_1 and α_3 to P_2 while keeping α_1. α_i is the composite action described by the TST rooted in node τ_i.

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

Let us focus on the reduction of first element in the sequence, α_1. Since P_0 has not delegated α_1 we expand the Can predicate one more step:

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

Since P_0 has recursively delegated α_2 to P_1 and α_3 to P_2 the $Delegate$ predicates can be reduced to Can predicates:

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

Since P_0 has recursively delegated α_4 to P_1 we can complete the reduction and end up with the following:

$$
\begin{aligned}
&Can(P_0, \alpha_0, [t_{s_0}, t_{e_0}], cons) = \\
&Can(P_0, S(C(\alpha_2, \alpha_3), \alpha_4), [t_{s_0}, t_{e_0}], cons) \leftrightarrow \\
&\exists t_{s_1}, t_{e_1}, t_{s_4}, t_{e_4} \\
&\quad \exists t_{s_2}, t_{e_2}, t_{s_3}, t_{e_3}(Can(P_1, \alpha_2, [t_{s_2}, t_{e_2}], cons_{P_1}) \\
&\qquad\qquad \wedge Can(P_2, \alpha_3, [t_{s_3}, t_{e_3}], cons_{P_2})) \\
&\quad \wedge Can(P_1, \alpha_4, [t_{s_4}, t_{e_4}], cons_{P_1})
\end{aligned}
$$

The remaining tasks are elementary actions and consequently the definition of Can for these are platform dependent. When a platform is assigned an elementary action node the resource constraints for that action is added to the local constraint store. The local constraints are connected to the distributed constraint problem through the node parameters of the assigned node. All remaining Can predicates in the recursion are replaced with constraint sub-networks associated with specific platforms as shown in Figure 2. To check that distributed constraint problem is consistent we use local CSP solvers together with a DisCSP solver.

V. A Cooperative UAS Case Study

One important application area for unmanned aircraft systems is to assist emergency services. Here we consider an emergency services assistance scenario where an unmanned aircraft system (UAS) should scan a disaster area for injured people and deliver relief packages to them. In the first part of the scenario, the UAS scans the disaster area and creates a map over the locations of the identified survivors [13].

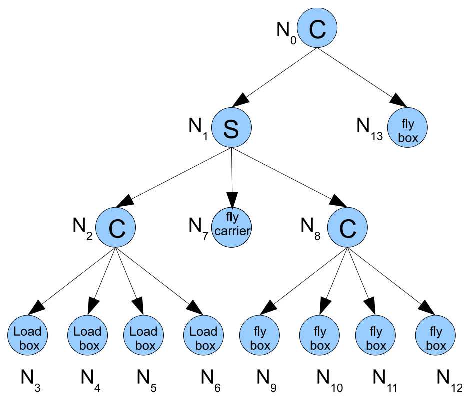

Fig. 3. The TST for the supply delivery case study.

In the second part, the UAS delivers boxes with supplies to the survivors. To transport a box it can either be carried directly by an unmanned aircraft or it can be loaded onto a carrier which is transported to a key position from where the boxes can be distributed to their final locations.

In this particular scenario, there is a UAS consisting of two platforms (P_1 and P_2), an operator (OP_1) which has found five survivors (S_1–S_5), and a carrier. Both platforms have the capability to transport a single box while only platform P_1 has the capability to transport a carrier. Both platforms also have the capabilities to coordinate sequence and concurrent tasks. At the same time another operator, OP_2, is performing a mission with the platforms P_3 and P_4 north of the survivors. P_3 is currently idle and OP_1 is therefore allowed to borrow it if necessary.

From the map, a TST is created that will achieve the goal of distributing relief packages to all survivors (Figure 3). The TST contains a sub-TST (N_1–N_{12}) for loading the carrier with four boxes (N_2–N_6), delivering the carrier (N_7), and unloading the packages from the carrier and delivering them to the survivors (N_8–N_{12}). A package must also be delivered to a survivor (S_5) far away from where most of the survivors were found (N_{13}). The delivery of packages can be done concurrently to save time, while the loading, moving, and unloading of the carrier is a sequential operation.

To achieve the mission, OP_1 delegates the TST to P_1. P_1 is now responsible for N_0 and for recursively delegating the nodes in the TST that it is not able to do itself. The allocation algorithm traverses the TST in depth-first order and uses marginal cost auctions as a heuristic to guide the allocation. P_1 starts with node N_1 and when the entire sub-TST rooted in N_1 is allocated then it will find an allocation for node N_{13}. Nodes N_1 and N_2 are composite action nodes which have the same marginal cost for all platforms. P_1 therefore allocates N_1 and N_2 to itself. The constraints from nodes N_0–N_2 are added to the constraint network of P_1. The network is consistent since the composite action nodes describe an unrestricted schedule.

Below node N_2 are four elementary action nodes. Since P_1 is responsible for N_2, it tries to allocate them one at the time. For elementary action nodes, the choice of platform is the key to a successful allocation. The candidates for node N_3 are platforms P_1 and P_2 so an auction is created where each of them may bid on N_3. P_1 is closest to the package depot and therefore gives the lowest bid. P_1 is allocated to N_3. For node N_4, platform P_1 is still the best choice, and it is allocated to N_4. Given the new position of P_1 after being allocated N_3 and N_4, P_2 is now closest to the depot resulting in the lowest bid and is allocated to N_5 and N_6. The schedule initially defined by nodes N_0–N_2 is now also constrained by how long it takes for P_1 and P_2 to carry out action nodes N_3–N_6. The constraint network is distributed among platforms P_1 and P_2.

The next node for P_1 to allocate is N_7, the carrier delivery node. P_1 is the only platform that has the fly carrier capability and is allocated the node. Continuing with nodes N_8–N_{12}, the platform with the lowest bid in the auction for each node is platform P_1, since it is in the area after delivering the carrier. P_1, is therefore allocated all the nodes N_8–N_{12}. The last node, N_{13}, is allocated to platform P_2 and the allocation is complete.

The only non-local information used by P_1 was the capabilities of the available platforms which was gathered through a broadcast. Everything else is local. The bids are made by each platform based on local information and the consistency of the constraint network is checked through distributed constraint satisfaction techniques.

The total mission time is 58 minutes, which is much longer than the operator expected. Since the allocation of the TST results in a constraint problem, it is possible for the operator to modify it by adding more constraints, and thereby potentially changing the task allocation. The operator puts a time constraint on the mission, restricting the total time to 30 minutes.

The added time constraint makes the current allocation inconsistent. The last allocated node must therefore be re-allocated. However, no platform for N_{13} can make the allocation consistent, not even the unused platform P_3. Backjumping starts. Platform P_1 is in charge, since it is responsible for allocating node N_{13}. The N_1 sub-network is disconnected. Trying different platforms for node N_{13}, P_1 discovers that N_{13} can be allocated to P_2. Since removing all constraints due to the allocation of node N_1 and its children made the problem consistent, the backjump point is in the sub-TST rooted in N_1. Removing the allocations for sub-tree N_8 does not make the problem consistent so further backjumping is necessary. Notice that with a single consistency check the algorithm could deduce that no possible allocation of N_8 and its children can lead to a consistent allocation of N_{13}. Removing the allocation for node N_7 does not make a difference either. However, removing the allocations for the sub-TST N_2 makes the problem consistent. When finding an allocation of N_{13} after removing the constraints from N_6 the allocation process continues from N_6. When a consistent allocation is found, P_1 informs the operator. The operator inspects the allocation and approves it, thereby confirming the delegation and starting the execution of the mission.

Based on this scenario we have started to evaluate the performance of our task allocation algorithm. In a first

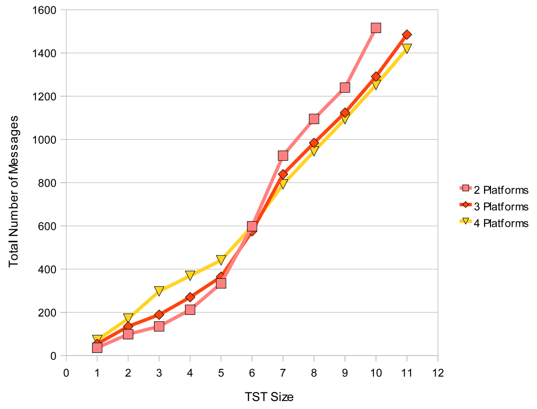

Fig. 4. The total number of messages sent when allocating a TST to a group of platforms while varying the size (number of carriers) of the TST.

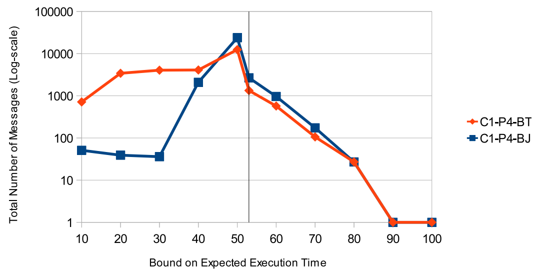

Fig. 5. The total number of messages sent when allocating a TST with 1 carrier to 4 platforms using either chronological backtracking (BT) or backjumping (BJ) while varying the bound on the expected execution time.

experiment we increased the size of the TST by repeating the sub-tree rooted in N_1 corresponding to increasing the number of carriers and boxes to be transported. Figure 4 shows how the number of messages varies as the number of carriers increases from 1 to 11, where each carrier corresponds to 12 TST nodes. In a second experiment we took the original TST and added a bound on the total expected execution time of the mission. This makes finding a valid allocation much harder. Figure 5 shows how the number of messages varies with the bound on the expected execution time for the whole mission when allocating a TST with 1 carrier (13 TST nodes) to 4 platforms. The figure shows that it is expensive to determine that there is no valid allocation and that the cost decreases exponentially as the bound moves away from the lowest possible bound (which is 53).

VI. Related Work

Due to the multi-disciplinary nature of the work considered here, there is a vast amount of related work too numerous to mention. In addition to the work referenced in the article, we instead consider a number of representative references from the areas of cooperative multi-robot systems and task allocation from a robotic perspective.

Cooperative multi-robot systems have a long history in robotics, multi-agent systems and AI in general. One early study presented a generic scheme based on a distributed plan merging process [14]. Another early work is ALLIANCE [15], which is a behavior based framework for instantaneous task assignment of loosely coupled subtasks with ordering dependencies. M+ [16] integrates mission planning, task refinement and cooperative task allocation. It uses a task allocation protocol based on the Contract Net protocol with explicit pre-defined capabilities and task costs. The M+CTA framework [17] is an extension of M+, where each robot has an individual plan and tasks are initially decomposed and then allocated. After the planning step, robots negotiate with each other to adapt their plans in the multi-robot context. The MURDOCH system [18] uses a publish/subscribe protocol for communication and an auction mechanism for task allocation. The result is very similar to the Contract Net protocol. Other Contract-Net and auction-based systems similar to those described above are COMETS [19], Hoplites [20] and TAEMS [21].

References

[1] C. Castelfranchi and R. Falcone, "Toward a theory of delegation for agent-based systems," *Robotics and Autonomous Systems*, vol. 24, pp. 141–157, 1998.

[2] R. Falcone and C. Castelfranchi, "The human in the loop of a delegated agent: The theory of adjustable social autonomy," *IEEE Trans. Syst., Man, Cybern. A*, vol. 31, no. 5, pp. 406–418, 2001.

[3] P. Cohen and H. Levesque, "Intention is choice with commitment," *AI*, vol. 42, no. 3, pp. 213–261, 1990.

[4] P. Doherty and J.-J. C. Meyer, "Towards a delegation framework for aerial robotic mission scenarios," in *Proc. International Workshop on Cooperative Information Agents*, 2007.

[5] B. v. L. W. van der Hoek and J.-J. C. Meyer, "An integrated modal approach to rational agents," in *Foundations of Rational Agency*, M. Wooldridge and A. Rao, Eds., 1998.

[6] P. Cohen and H. Levesque, "Teamwork," *Nous*, vol. 25, no. 4, 1991.

[7] E. Davis and L. Morgenstern, "A first-order theory of communication and multi-agent plans," *Journal Logic and Computation*, vol. 15, no. 5, pp. 701–749, 2005.

[8] J. Kvarnström and P. Doherty, "Automated planning for collaborative UAV systems," in *Proc. ICARCV*, 2010.

[9] J. Kvarnström, "Planning for loosely coupled agents using patrial order forward-chaining," in *Proc. ICAPS*, 2011.

[10] M. Yokoo, "Asynchronous weak-commitment search for solving distributed constraint satisfaction problems," in *Proc. CP*, 1995.

[11] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "ADOPT: Asynchronous distributed constraint optimization with quality guarantees," *AI*, vol. 161, 2006.

[12] D. Landén, F. Heintz, and P. Doherty, "Complex task allocation in mixed-initiative delegation: A UAV case study (early innovation)," in *Proc. PRIMA*, 2010.

[13] P. Rudol and P. Doherty, "Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery," in *Proc. IEEE Aerospace Conference*, 2008.

[14] R. Alami, F. Ingrand, and S. Qutub, "A scheme for coordinating multirobot planning activities and plans execution," in *Proc. ECAI*, 1998.

[15] L. E. Parker, "Alliance: An architecture for fault tolerant multi-robot cooperation," *IEEE Trans. Robot. Automat*, vol. 14, no. 2, 1998.

[16] S. Botelho and R. Alami, "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement," in *Proc. ICRA*, 1999.

[17] R. Alami and S. C. Botelho, "Plan-based multi-robot cooperation," in *Advances in Plan-Based Control of Robotic Agents*, 2001.

[18] B. Gerkey and M. Mataric, "Sold!: Auction methods for multi-robot coordination," *IEEE Trans. Robot. Automat.*, vol. 18, no. 5, 2001.

[19] T. Lemaire, R. Alami, and S. Lacroix, "A distributed tasks allocation scheme in multi-UAV context," in *Proc. ICRA*, 2004.

[20] N. Kaldra, D. Ferguson, and A. Stentz, "Hoplites: A market-based framework for planned tight coordination in multirobot teams," in *Proc. ICRA*, 2005.

[21] K. Decker, "TAEMS: A framework for environment centered analysis and design of coordination mechanisms," in *Foundations of Distributed AI*. Wiley Inter-Science, 1996.