A Delegation-Based Architecture for Collaborative Robotics [⋆]

Patrick Doherty, Fredrik Heintz, and David Landén
{patrick.doherty, fredrik.heintz}@liu.se

Linköping University
Dept. of Computer and Information Science
581 83 Linköping, Sweden

Abstract. Collaborative robotic systems have much to gain by leveraging results from the area of multi-agent systems and in particular agent-oriented software engineering. Agent-oriented software engineering has much to gain by using collaborative robotic systems as a testbed. In this article, we propose and specify a formally grounded generic collaborative system shell for robotic systems and human operated ground control systems. 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 implemented in the collaborative system shell. The delegation speech act is formally grounded in the implementation using Task Specification Trees, task allocation via auctions and distributed constraint problem solving. The system is implemented as a prototype on Unmanned Aerial Vehicle systems and a case study targeting emergency service applications is presented.

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 aerial vehicles (UAV's) and associated hardware and software technologies [14–16]. The size of our research platforms range from the RMAX helicopter system (100kg) [8, 17, 59, 66, 69] developed by Yamaha Motor Company, to smaller micro-size rotor based systems such as the LinkQuad[2] (1kg) and LinkMAV [28, 60] (500g) in addition to a fixed wing platform, the PingWing [9] (500g). These UAV platforms are shown in Figure 1.The latter three have been designed and developed by the Unmanned Aircraft Systems Technologies Lab. All four platforms are fully autonomous and have been deployed.

[⋆] This work is partially supported by grants from the Swedish Research Council (VR) Linnaeus Center CADICS, VR grant 90385701, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, NFFP5-The Swedish National Aviation Engineering Research Program, and the Center for Industrial Information Technology CENIIT.

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

[2] www.uastech.com

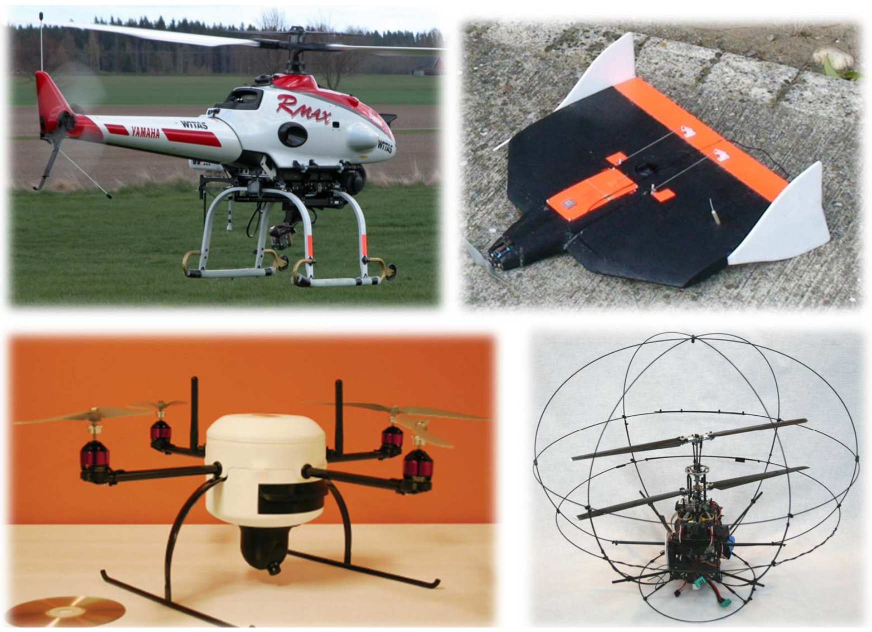

Fig. 1. The UASTech RMAX (upper left), PingWing (upper right), LinkQuad (lower left) and LinkMAV (lower right).

Previous work has focused on the development of robust autonomous systems for UAV's which seamlessly integrate control, reactive and deliberative capabilities that meet the requirements of hard and soft realtime constraints [17, 55]. Additionally, we have focused on the development and integration of many high-level autonomous capabilities studied in the area of cognitive robotics such as task planners [18, 19], motion planners [66–68], execution monitors [21], and reasoning systems [20, 23, 54], in addition to novel middleware frameworks which support such integration [40, 42, 43]. Although research with individual high-level cognitive functionalities is quite advanced, robust integration of such capabilities in robotic systems which meet real-world constraints is less developed but essential to introduction of such robotic systems into society in the future. Consequently, our research has focused, not only on such high-level cognitive functionalities, but also on system integration issues.

More recently, our research efforts have transitioned toward the study of systems of UAV's. The accepted terminology for such systems is Unmanned Aircraft Systems (UAS's). A UAS may consist of one or more UAV's (possibly heterogenous) in addition to one or more ground operator systems (GOP's). We are interested in applications where UAV's are required to collaborate not only with each other but also with diverse human resources [22, 24, 25, 41, 52]. UAV's are now becoming technologically mature enough to be integrated into civil society. Principled interaction between UAV's and human resources is an essential component in the future uses of UAV's in complex emergency services or bluelight scenarios. Some specific target UAS scenario examples are search and rescue missions for inhabitants lost in wilderness regions and assistance in guiding them to a safe destination; assistance in search at sea scenarios; assistance in more devastating scenarios such as earthquakes, flooding or forest fires; and environmental monitoring.

As UAV's become more autonomous, mixed-initiative interaction between human operators and such systems will be central in mission planning and tasking. By mixed-initiative, we mean that interaction and negotiation between one or more UAV's and one or more humans 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.

In the future, the practical use and acceptance of UAV's will have to be 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 UAV's and humans in complex scenarios, a great many interdependent conceptual and pragmatic issues arise and need clarification not only theoretically, but also pragmatically in the form of demonstrators. Additionally, an iterative research methodology is essential which combines foundational theory, systems building and empirical testing in real-world applications from the start.

The complexity of developing deployed architectures for realistic collaborative 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. Bridging this conceptual gap robustly is an important area of research and given the complexity of the systems being built today demands new insights and techniques.

The conceptual basis for the proposed collaboration framework includes a triad of fundamental, interdependent conceptual issues: delegation, mixed-initiative interaction and adjustable autonomy (Figure 2). The concept of delegation is particularly important and in some sense provides a bridge between mixed-initiative interaction and adjustable autonomy.

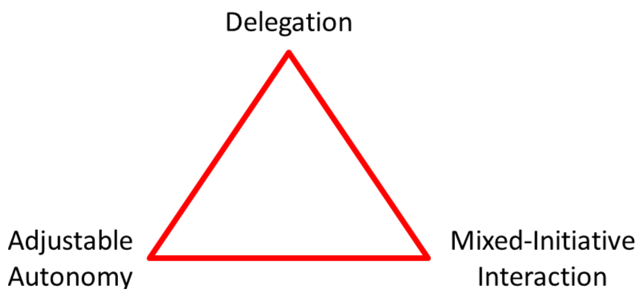

Fig. 2. A conceptual triad of concepts.

Delegation – In any mixed-initiative interaction, humans may 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 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. 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 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 $Delegate(A, B, task, constraints)$. 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. It may even be the case that the delegator does not allow the contractor to recursively delegate.

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 article is intended to provide a description of a relatively mature iteration of a principled framework for collaborative robotic systems based on these concepts which combines both formal theories and specifications with an agent-based software architecture which is guided by the formal framework. As a test case, the framework and architecture will be instantiated using a UAS involved in an emergency services application. A prototype software system has been implemented and has been used and tested both in simulation and on UAV systems.

1.1 Outline

In Section 2, we propose and specify a formal logical characterization of delegation in the form of a speech act. This speech act will be grounded in the software architecture proposed. In Section 3, an overview of the software architecture used to support collaboration via delegation is provided. It is an agent-based, service oriented architecture consisting of a generic shell that can be integrated with physical robotics systems. In Section 4, a formal characterization of tasks in the form of Task Specification Trees is proposed. Task Specification Trees are tightly coupled to the the Delegation speech act and to the actual software processes that instantiate the speech act in the software architecture. In Section 5, the important topic of allocating tasks in a Task Specification Tree to specific platforms is considered. Additionally, we show how the semantic characterization of Task Specification Trees is grounded in a distributed constraint problem whose solution drives the actual execution of the tasks in the tree. In Section 6,we turn our attention to describing the computational process that realizes the speech act on a robotic platform. In Section 7, we describe how that computational process is pragmatically realized in the software architecture by defining a number of agents, services and protocols which drive the process. In Section 8, we put the formal and pragmatic aspects of the approach together and show how the collaboration framework can be used in a relatively complex real-life emergency services scenario consisting of a number of UAV systems. In Section 9, we describe some of the representative related work and in Section 10 we conclude with a summary and future work.

2 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, in [5, 31], 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 [6]. 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 [25]. In order to formally characterize the operators used in the definition of the speech act, we use KARO [48] 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.

The target for delegation is a *task*. A dictionary definition of a task is ”a usually assigned piece of work often to be finished within a certain time”.[3] 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 (TST's) will be defined in a Section 4. Falcone & Castelfranchi define a task

[3] Merriam-Webster free on-line dictionary. `m-w.com`

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. We extend the definition of a task to a tuple $\tau = (\alpha, \phi, cons)$, where $cons$ represents additional constraints associated with the plan α, such as timing and resource constraints. At this level of abstraction, the definition of a task is purposely left general but will be dealt with in explicit detail in the implementation using TST's and constraints.

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. These extremes loosely follow Falcone & Castelfranchi's notions of closed and open delegation described below.

Using KARO to formalize aspects of Falcone & Castelfranchi 's work, we consider a notion of *strong delegation* represented by a speech act Delegate(A, B, τ) of A delegating a task $\tau = (\alpha, \phi, cons)$ to B, where α is a possible plan, ϕ is a goal, and $cons$ is a set of constraints associated with the plan ϕ. Strong delegation means that the delegation is explicit, an agent explicitly delegates a task to another agent. It is specified as follows:

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

Preconditions:

(1) $Goal_A(\phi)$
(2) $Bel_A Can_B(\tau)$ (Note that this implies $Bel_A Bel_B(Can_B(\tau))$)
(3) $Bel_A(Dependent(A, B, \tau))$
(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" $\wedge$ $SociallyCommitted(B, A, \tau)$)[4]

Informally speaking this expresses the following: the preconditions of the delegate 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 itself believes that it can do the task) (3) A believes that with respect to the task τ it is dependent on B. The speech act S-Delegate is a communication command and can be viewed as a request for a synchronization (a "handshake") between sender and receiver. Of course,

[4] A discussion pertaining to the semantics of all non-KARO modal operators may be found in [25].

this can only be successful if the receiver also believes it can do the task, which is expressed by (4).

The postconditions of the strong delegation act mean: (1) B has ϕ as its goal and is aware of this (2) it is committed to the task τ (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 formalization), (5) A intends that B performs α (so we have formalized the notion of a goal to have an acheivement in Falcone & Castelfranchi's informal theory to an intention to perform a task) and (6) 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. Falcone & Castelfranchi discuss the following variants:

- Closed delegation: the task is completely specified and both the goal and the 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.

In open delegation the contractor may have some freedom in how 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 above is based on closed delegation. In case of open delegation, α in the postconditions can be replaced by an α', and τ by $\tau' = (\alpha', \phi, cons')$. Note that the fourth clause, $Can_B(\tau')$, 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', together with constraints $cons'$, 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 [7, 11, 27]. It is important to keep in mind that this formal characterization of delegation is not completely hierarchical. There is interaction between both the delegators and contractors as to how goals can best be achieved given the constraints of the agents involved. This is implicit in the formal characterization of open delegation above, although the process is not made explicit. This aspect of the process will become much clearer when the implementation is described.

There are many directions one can take in attempting to close the gap between this abstract formal specification and grounding it in implementation. One such direction taken in [25] is to correlate the delegate speech act with plan generation rules in

2APL [10], which is an agent programming language with a formal semantics. In this article, a different direction is taken which attempts to ground the important aspects of the speech act specification in the actual processes used in our robotic systems. Intuitions will become much clearer when the architectural details are provided, but let us describe the approach informally based on what we have formally specified.

If a UAV system A has a goal ϕ which it is required to achieve, it first introspects and determines whether it is capable of achieving ϕ given its inherent capabilities and current resources in the context it is in, or will be in, when the goal has to be achieved. It will do this by accessing its capability specification (assumed) and determine whether it believes it can achieve ϕ, either through use of a planning and constraint solving system (assumed) or a repertoire of stored actions. If not, then the fundamental preconditions in the S-Delegate speech act are the second, $Bel_A Can_B(\tau)$ and the fourth, $Bel_B Can_B(\tau)$. Agent A must find another agent it believes *can* achieve the goal ϕ implicit in τ. Additionally, B must also believe it can achieve the the goal ϕ implicit in τ. Clearly, if A can not achieve ϕ itself and finds an agent B that it believes can achieve ϕ and B believes it can achieve ϕ, then it is dependent on B to do that (precondition 3: $Bel_A(Dependent(A, B, \alpha))$). Consequently, all preconditions are satisfied and the delegation can take place.

From a pragmatic perspective, determining (in an efficient manner) whether an agent B *can* achieve a task τ (in an efficient) manner, is the fundamental problem that has to be not only implemented efficiently, but also grounded in some formal sense. The formal aspect is important because delegation is a recursive process which may involve many agents, automated planning and reasoning about resources, all in the context of temporal and spatial constraints. One has to have some means of validating this complex set of processes relative to a highly abstract formal specification which is convincing enough to trust that the collaborative system is in fact doing what it is formally intended to do.

The pragmatic aspects of the software architecture through which we ground the formal specification include the following:

– An agent layer based on the FIPA Abstract Architecture will be added on top of existing platform specific legacy systems such as our UAV's. This agent layer allows for the realization of the delegation process using speech acts and protocols from the FIPA Agent Communication Language.
– The formal specification of tasks will be instantiated pragmatically as Task Specification Trees (TST's), which provide a versatile data structure for mapping goals to plans and plans to complex tasks. Additionally, the formal semantics of tasks is defined in terms of a predicate Can which can be directly grounded above to the semantics of the S-Delegate speech act and below to a constraint solving system.
– Finding a set of agents who together can achieve a complex task with time, space and resource constraints through recursive delegation can be defined as a very complex distributed task allocation problem. Explicit representation of time, space and resource constraints will be used in the delegation process and modeled as a distributed constraint satisfaction problem (DCSP). This allows us to apply existing DCSP solvers to check the consistency of partial task assignments in the delegation process and to formally ground the process. Consequently, the Can predicate used

in the precondition to the S-Delegate speech act is both formally and pragmatically grounded into the implementation.

3 Delegation-Based Software Architecture Overview

Before going into details regarding the implementation of the delegation process and its grounding in the proposed software architecture, we provide an overview of the architecture itself.

Our RMAX helicopters use a CORBA-based distributed architecture [17]. For our experimentation with collaborative UAV's, we view this as a legacy system which provides sophisticated functionality ranging from control modes to reactive processes, in addition to deliberative capabilities such as automated planners, GIS systems, constraint solvers, etc. Legacy robotic architectures generally lack instantiations of an agent metaphor although implicitly one often views such systems as agents. Rather than redesign the legacy system from scratch, the approach we take is to agentify the existing legacy system in a straightforward manner by adding an additional agent layer which interfaces to the legacy system. The agent layer for a robotic system consists of one or more agents which offer specific functionalities or services. These agents can communicate with each other internally and leverage existing legacy system functionality. Agents from different robotic systems can also communicate with each other if required.

Our collaborative architectural specification is based on the use of the FIPA (Foundation for Intelligent Physical Agents) Abstract Architecture [32]. The FIPA Abstract Architecture provides the basic components for the development of a multi-agent system. Our prototype implementation is based on the FIPA compliant Java Agent Development Framework (JADE) [29, 62] which implements the abstract architecture. "JADE (Java Agent Development Framework) is a software environment to build agent systems for the management of networked information resources in compliance with the FIPA specifications for interoperable multi-agent systems." [30].

The FIPA Abstract Architecture provides the following fundamental modules:

- An Agent Directory module keeps track of the agents in the system.
- A Directory Facilitator keeps track of the services provided by those agents.
- A Message Transport System module allows agents to communicate using the FIPA Agent Communication Language (FIPA ACL) [33].

The relevant concepts in the FIPA Abstract Architecture are agents, services and protocols. All communication between agents is based on exchanging messages which represent speech acts encoded in an agent communication language (FIPA ACL). Services provide functional support for agents. There are a number of standard global services including agent-directory services, message-transport services and a service-directory service. A protocol is a related set of messages between agents that are logically related by some interaction pattern.

JADE provides base classes for agents, message transportation, and a behavior model for describing the content of agent control loops. Using the behavior model, different agent behaviors can be constructed, such as cyclic, one-shot (executed once), sequential, and parallel behavior. More complex behaviors can be constructed using the basic behaviors as building blocks.

From our perspective, each JADE *agent* has associated with it a set of *services*. Services are accessed through the Directory Facilitator and are generally implemented as behaviors. In our case, the communication language used by agents will be FIPA ACL which is speech act based. New protocols will be defined in Section 7 to support the delegation and other processes.

The purpose of the Agent Layer is to provide a common interface for collaboration. This interface should allow the delegation and task execution processes to be implemented without regard to the actual realization of elementary tasks, capabilities and resources which are specific to the legacy platforms.

We are currently using four agents in the agent 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 or ground control station.
2. **Delegation agent**- The delegation agent coordinates delegation requests to and from other UAV systems and ground control stations, with the Executor, Resource and Interface agents. It does this essentially by verifying that the pre-conditions to a $Delegate()$ request are satisfied.
3. **Execution agent** - After a task is contracted to a particular UAV or ground station operator, 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 UAV or ground station 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, planners and constraint solvers in order to determine this.

Figure 3 provides an overview of an agentified robotic or ground operator system.

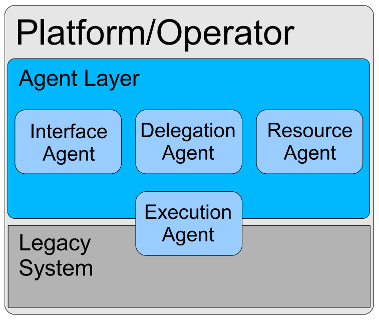

Fig. 3. Overview of an agentified platform or ground control station.

The FIPA Abstract Architecture will be extended to support delegation and collaboration by defining an additional set of services and a set of related protocols.The interface agent, resource agent and delegation agent will have an interface service, resource service and delegation service associated with it, respectively, on each individual robotic or ground station platform. The executor service is implemented as a non-JADE agent that understands FIPA protocols and works as a gateway to a platform's legacy system. Additionally, three protocols, the Capability-Lookup, Delegation and Auction protocols, will be defined and used to drive the delegation process.

Human operators interacting with robotic systems are treated similarly by extending the control station or user interface functionality in the same way. In this case, the control station is the legacy system and an agent layer is added to this. The result is a collaborative human robot system consisting of a number of human operators and robotic platforms each having both a legact system and an agent layer as shown in Figure 4.

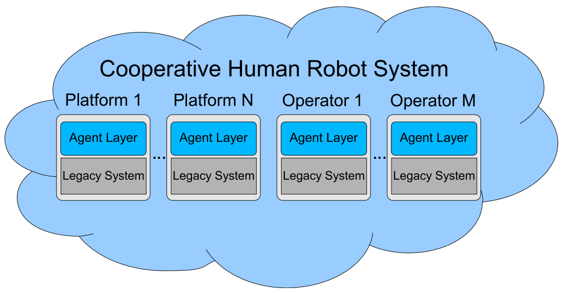

Fig. 4. An overview of the collaborative human robot system.

The reason for using the FIPA Abstract Architecture and JADE is pragmatic. The focus of our research is not to develop new agent middleware, but to develop a formally grounded generic collaborative system shell for robotic systems. 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.

Further details as to how the delegation and related processes will be implemented based on additional services and protocols will be described in Section 7. Before doing this, the processes themselves will be specified in Section 6. We begin by providing a formal characterization of Tasks in the form of Task Specification Trees.

4 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 tuple $(\alpha, \phi, cons)$ consisting of a composite action α, a goal ϕ and a set of constraints $cons$, associated with α . In this section, we introduce a formal task specification language which allows us to represent tasks as *Task Specification Trees* (TST's). The task specification trees map directly to procedural representations in our proposed system implementation.

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 [51] 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 flexibility allows for the use of both central and distributed planning, and also to move along the scale between these two extremes. At one extreme, the operator plans everything, creating a central plan, while at the other extreme the agents are delegated goals and generate parts of the distributed plan themselves. Sometimes neither completely centralized nor completely distributed planning is appropriate. In those cases the operator would like to retain some control of how the work is done while leaving the details to the agents. Task Specification Trees provide a formalism that captures the scale from one extreme to the next. This allows the operator to specify the task at the point which fits the current mission and environment.

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 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 require 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 denoted P, T_S and T_E, respectively. 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.

4.1 TST Syntax

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 ::= <elementary action>

Where

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

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.

Example Consider a small scenario where the mission is to first scan $\mathsf{Area_A}$ and $\mathsf{Area_B}$, and then fly to $\mathsf{Dest_4}$ (Figure 5). A TST describing this mission is shown in Figure 6. 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 variable. In this case only temporal parameters are shown representing the respective intervals a task should be completed in.

In the TST depicted in Figure 6. The 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, *sequence* (τ_0) and *concurrent* (τ_1) and three elementary actions *scan* (τ_2, τ_3) and *flyto* (τ_4). The resulting TST specification is:

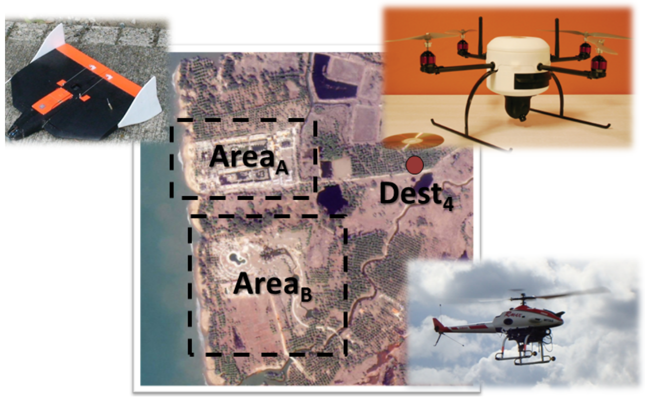

Fig. 5. Example mission of first scanning Area$_A$ and Area$_B$, and then fly to Dest$_4$.

```
$\tau_0(T_{S_0},T_{E_0})$ =
  **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** $cons_{\tau_0}$
```

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

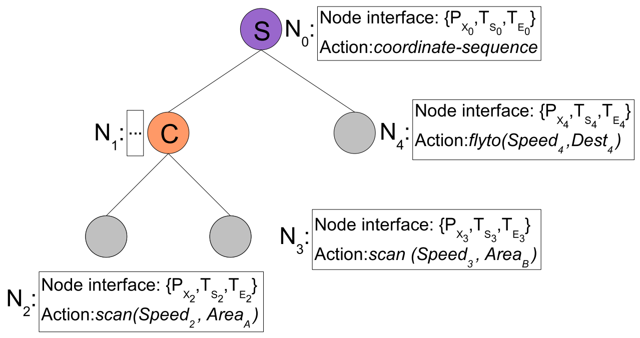
S
N$_0$: Node interface: {P$_{X_0}$,T$_{S_0}$,T$_{E_0}$}
Action:*coordinate-sequence*
N$_1$: ···
C
N$_4$: Node interface: {P$_{X_4}$,T$_{S_4}$,T$_{E_4}$}
Action:*flyto(Speed$_4$,Dest$_4$)*
N$_3$: Node interface: {P$_{X_3}$,T$_{S_3}$,T$_{E_3}$}
Action:*scan (Speed$_3$, Area$_B$)*
N$_2$: Node interface: {P$_{X_2}$,T$_{S_2}$,T$_{E_2}$}
Action:*scan(Speed$_2$, Area$_A$)*

Fig. 6. A TST for the mission in Figure 5.

4.2 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 semantics of a TST will be the aggregation of the semantics for each individual node in the tree.

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 2, 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 $\tau = (\alpha, \phi, cons)$ 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 explicitly.

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[5] and an argument for an additional constraint set

[5] 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.

$cons$ provided in the (**where** CONS) construct.[6] 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 $Can(B, \tau, [t_s, t_e, \ldots], cons)$[7] 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:

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

$$Can(agent_1, TASK, VARS_1 \cup VARS_2, CONS) \tag{1}$$

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 dependent 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 CSP's.

One would also like to capture the meaning of partial TST's. 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:

$$\exists agent_2\, Delegate(agent_1, agent_2, TASK, VARS_1 \cup VARS_2, CONS) \tag{2}$$

[6] 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.

[7] Note that we originally defined $\tau = (\alpha, \phi, cons)$ as a tuple consisting of a plan, a goal and a set of constraints for reasons of abstraction when defining the delegation speech act. Since we now want to explicitly use $cons$ as an argument to the Can predicate in the implementation, we revert to defining $\tau = (\alpha, \phi)$ as a pair instead, where the constraints $cons$ are lifted up as an argument to Can.

Either $agent_1$ can achieve a task, or (exclusively) it can find an agent, $agent_2$, 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 $S\text{-}Delegate(agent_1, agent_2, TASK)$ speech act semantics, we know that if delegation is successful then as one of the postconditions of the speech act, $agent_2$ can in fact achieve $TASK$ (assuming no additional contingencies):

$$\begin{aligned} &Delegate(agent_1, \mathsf{agent}_2, TASK, VARS_1 \cup VARS_2, CONS) \\ &\quad \rightarrow Can(\mathsf{agent}_2, TASK, VARS_1 \cup VARS_2, CONS) \end{aligned} \tag{3}$$

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

- In a *sequence node*, the child nodes should be executed in sequence (from left to right) during the execution time of the sequence node.
- $Can(B, S(\alpha_1, ..., \alpha_n), [t_s, t_e, \ldots], cons) \leftrightarrow$
  $\exists t_1, \ldots, t_{2n}, \ldots \bigwedge_{k=1}^{n}[(Can(B, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k)$
  $\quad \vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k))]$
  $\quad \wedge\, consistent(cons)$[8]
- $cons = \{t_s \leq t_1 \wedge (\bigwedge_{i=1}^{n} t_{2i-1} < t_{2i}) \wedge (\bigwedge_{i=1}^{n-1} t_{2i} \leq t_{2i+1}) \wedge t_{2n} \leq t_e\} \cup cons'$[9]

Concurrent Node

- In a *concurrent node* each child node should be executed during the time interval of the concurrent node.
- $Can(B, C(\alpha_1, ..., \alpha_n), [t_s, t_e, \ldots], cons) \leftrightarrow$
  $\exists t_1, \ldots, t_{2n}, \ldots \bigwedge_{k=1}^{n}[(Can(B, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k)$
  $\quad \vee \exists a_k Delegate(B, a_k, \alpha_k, [t_{2k-1}, t_{2k}, \ldots], cons_k))]$
  $\quad \wedge\, consistent(cons)$
- $cons = \{\bigwedge_{i=1}^{n} t_s \leq t_{2i-1} < t_{2i} \leq t_e\} \cup cons'$

Selector Node

[8] 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.

[9] In addition to the temporal constraints, other constraints may be passed recursively during the delegation process. $cons'$ represents these constraints.

– 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 legal 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. After expansion, a TST branch representing the generated plan is added to the original TST.
– $Can(B, Goal(\phi), [t_s, t_e, \ldots], cons) \leftrightarrow$
$\exists \alpha\, (GeneratePlan(B, \alpha, \phi, [t_s, t_e, \ldots], cons) \wedge Can(B, \alpha, [t_s, t_e, \ldots], cons))$
$\wedge\, consistent(cons)$

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 Node

– An *elementary action node* specifies a domain-dependent action. An elementary action node is a leaf node.
– $Can(B, \tau, [t_s, t_e, \ldots], cons) \leftrightarrow$
$Capabilities(B, \tau, [t_s, t_e, \ldots], cons) \wedge Resources(B, \tau, [t_s, t_e, \ldots], cons)$
$\wedge\, consistent(cons)$

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 collaborative mission. The platform specification has two components.

The first, specified by the predicate $Capabilities(B, \tau, [t_s, t_e], cons)$ is intended to characterize all static capabilities associated with platform B that are required as capabilities for the successful execution of τ. These will include a list of tasks and/or services the platform is capable of carrying out. 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 $Resources(B, \tau, [t_s, t_e], cons)$ are 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. The general pattern for this conjunction is:

$t_e = t_s + F, C_1, ..., C_N$, where

- F is a function of the resource constraint parameters and possibly local resource variables and
- $C_1, \ldots, C_N$ is a possibly empty set of additional constraints related to the resource model associated with the task.

Example 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:

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

Depending on the platform, this constraint model may be different for the same task. In that sense, it is platform dependent.

5 Allocating Tasks in a TST to Platforms

Given a TST representing a complex task, an important problem is to find a set of platforms that can execute these tasks according to the TST specification. 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 of the TST are satisfied.

For a platform to be able to carry out a task, it must have the capabilities and the resources required for the task as described in the previous section. A platform that can be assigned a task in a TST is called a *candidate* and a set of candidates is a *candidate group*. The capabilities of a platform are fixed while the available resources will vary depending on its commitments, including the tasks it has already been allocated. These commitments are generally represented in the constraint stores and schedulers of the platforms in question. The resources and the commitments are modeled with constraints. Resources are represented by variables and commitments by constraints. These constraints are local to the platform and different platforms may have different constraints for the same action. Figure 7 shows the constraints for the scan action for platform P_1.

When a platform is assigned an action node in a TST, the constraints associated with that action are instantiated and added to the constraint store of the platform. The

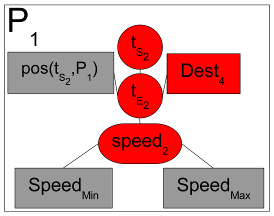

Fig. 7. The parameterized platform constraints for the scan action. The red/dark variables represent node parameters in the node interface. The gray variables represent local variables associated with the platform P1's constraint model for the scan action. These are connected through dependencies.

platform constraints defined in the constraint model for the task are connected to the constraint problem defined by the TST via the node parameters in the node interface for the action node. Figure 8 shows the constraint network after allocating node N_2 from the TST in Figure 6 (on page 14) to platform P_1.

A platform can be allocated to more than one node. This may introduce implicit dependencies between actions since each allocation adds constraints to the constraint store of the platform. For example, there could be a shared resource that both actions use. Figure 9 shows the constraint network of platform P_1 after it has been allocated nodes N_2 and N_4 from the example TST. In this example the position of the platform is implicitly shared since the first action will change the location of the platform.

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 in effect a distributed constraint problem. If a consistent solution for this constraint problem is found then a *valid allocation* has been found and verified. Each such 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 (DCSP) solver such as the Asynchronous Weak Commitment Search (AWCS) algorithm [70] or ADOPT [56].

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 TST from Figure 6 (on page 14) when there are three platforms, P_0, P_1 and P_2, with the appropriate capabilities. P_0 has been 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 $P_2(\tau_3)$.

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

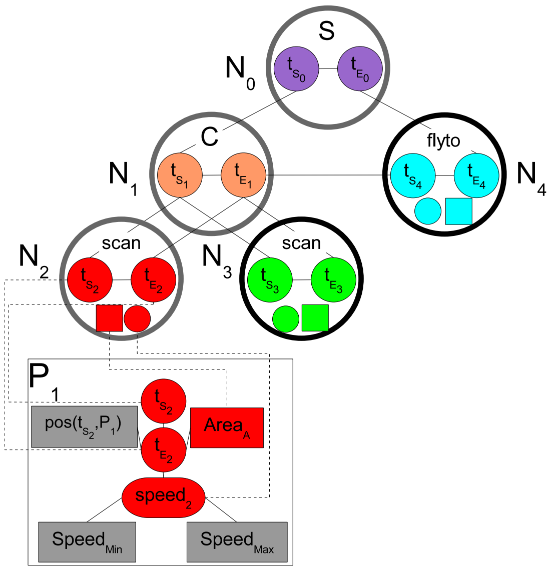

Fig. 8. The combined constraint problem after allocating node N_2 to platform P_1.

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):

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

Since P_0 has been allocated α_1, the 2nd disjunct is false.

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

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

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

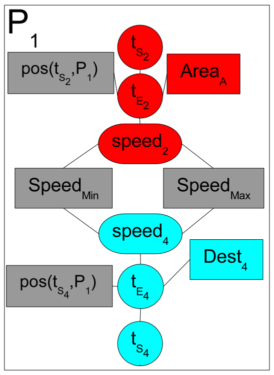

Fig. 9. The parameter constraints of platform P_1 when allocated node N_2 and N_4.

Since P_0 has recursively delegated α_4 to P_1 (the 2nd conjunct in the original formula on the right-hand side of the biconditional) we can complete the reduction and end up with the following:

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

These remaining tasks are elementary actions and consequently the definitions of Can for these action nodes are platform dependent. When a platform is assigned to an elementary 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 elementary actions. The completely allocated and reduced TST is shown in Figure 10. The reduction of Can for an elementary action node contains no further Can predicates, since an elementary 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 10.

In summary, the delegation process, if successful, provides a TST that is both valid and completely allocated. During this process, a network of distributed constraints is generated which if solved, guarantees the validity of the multi-agent solution to the original problem, provided that additional contingencies do not arise when the TST is actually executed in a distributed manner by the different agents involved in the collaborative solution. This approach is intended to ground the original formal specification of the S-Delegate speech act with the actual processes of delegation used in the implementation. Although the process is pragmatic in the sense that it is a computational process, it in effect strongly grounds this process formally, due to the reduction of the collaboration to a distributed constraint network which is in effect a formal representation. This results in real-world grounding of the semantics of the Delegation speech act via the Can predicate.

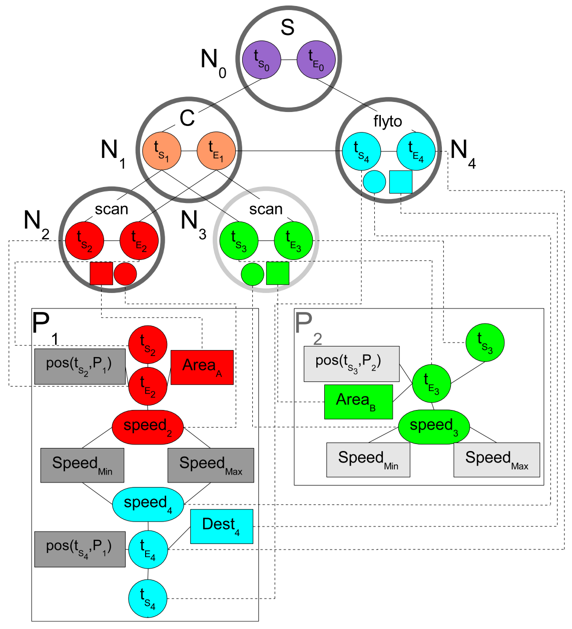

Fig. 10. The completely allocated and reduced TST showing the interaction between the TST constraints and the platform dependent constraints.

6 The Delegation Process

Now that the S-Delegate speech act, the Task Specification Tree representation, and the formal relation between them has been considered, we turn our attention to describing the computational process that realizes the speech act in a robotic platform.

According to the semantics of the Delegate(A,B,$\tau = (\alpha, \phi)$) speech act the delegator A must have ϕ as a goal, believe that there is an agent B that is able to achieve τ, and believe that it is dependent on B for the achievement of τ via action α. In the following, we assume that the agent A already has ϕ as a goal and that it is dependent on some other agent to achieve the task. Consequently, the main issue is to find an agent B that is able to achieve the task τ.

This could be done in at least two ways. Agent A could have a knowledge base encoding all its knowledge about what other agents can and can not do and then reason about which agents could achieve τ. This would be very similar to a centralized form of multi-agent planning since the assumption is that τ is a complex task. This is problematic because it would be difficult to keep such a knowledge base up-to-date and it would be quite complex given the heterogeneous nature of the platforms involved. Additionally, the pool of platforms accessible for any given mission at a given time is not known since platforms come and go.

As an alternative, the process of finding agents to achieve tasks will be done in a more distributed manner through communication among agents and an assumption that elementary actions are platform dependent and the details of such actions are not required in finding appropriate agents to achieve the tasks at hand.

The following process takes as input a complex task represented as a TST. The TST is intended to describe a complex mission. The process will find an appropriate agent or set of agents capable of achieving the mission possibly through the use of recursive delegation. If the allocation of agents in the TST is approved by the delegators recursively, then the mission can then be executed. Note that the mission schedule will be distributed among the group of agents that have been allocated tasks and the mission may not necessarily start immediately. This will depend on the temporal constraints used in the TST specification. But commitments to the mission will have been made in the form of constraints in the constraint stores and schedulers of the individual platforms. Note also, that the original TST given as input does not have to be completely specified. It may contain goal nodes which require expansion of the TST with additional nodes.

The process is as follows:

1. Allocate the complex task through an iterative and recursive process which finds a platform to whom the task can be delegated to. This process expands goals into tasks, assigns platforms to tasks, and assigns values to task parameters. The input is a TST and the output is a fully expanded, assigned and parameterized TST.
2. Approve the mission or request the next consistent instantiation. Repeat 1 until approved or no more instantiations.
3. If no approved instantiated mission is found then fail.
4. Otherwise, execute the approved mission until finished or until constraints associated with the mission are violated during execution. While executing the mission, constraints are monitored and their parameterization might be changed to avoid violations on the fly.
5. If constraints are violated and can not be locally repaired goto 1 and begin a recursive repair process.

The first step of the process corresponds to finding a set of platforms that satisfy the preconditions of the S-Delegate speech act for all delegations in the TST. The approval corresponds to actually executing the speech act where the postconditions are implicitly represented in the constraint stores and schedulers of the platforms. During the execution step, the contractors are committed to the constraints agreed upon during the approval of the tasks. They do have limited autonomy during execution in the form of being able to modify internal parameters associated with the tasks as long as they do not violate those constraints externally agreed upon in the delegation process.

6.1 An Algorithm for Allocating Complex Tasks Specified by TSTs

The most important part of the Delegation Process is to find a platform that satisfies the preconditions of the S-Delegate speech act. This is equivalent to finding a platform which is able to achieve the task either itself of through recursive delegation. This can be viewed as a task allocation problem where each task in the TST should be allocated to an agent.

Multi-robot task allocation (MRTA) is an important problem in the multi-agent community [38, 39, 53, 63, 71, 72]. It deals with the complexities involved in taking a description of a set of tasks and deciding which of the available robots should do what. Often the problem also involves maximizing some utility function or minimizing a cost function. Important aspects of the problem are what types of tasks and robots can be described, what type of optimization is being done, and how computationally expensive the allocation is.

This section presents a heuristic search algorithm for allocating a fully expanded TST to a set of platforms. A successful allocation allocates each node to a platform and assigns values to parameters such that each task can be carried out by its assigned platform and all the constraints of the TST are satisfied. During the allocation, temporal variables will be instantiated resulting in a schedule for executing the TST.

The algorithm starts with an empty allocation and extends it one node at a time in a depth-first order over the TST. To extend the allocation, the algorithm takes the current allocation, finds a consistent allocation of the next node, and then recursively allocates the rest of the TST. Since a partial allocation corresponds to a distributed constraint satisfaction problem, a DCSP solver is used to check whether the constraints are consistent. If all possible allocations of the next node violate the constraints, then the algorithm uses backtracking with backjumping to find the next allocation.

The algorithm is both sound and complete. It is sound since the consistency of the corresponding constraint problem is verified in each step and it is complete since every possible allocation is eventually tested. Since the algorithm is recursive the search can be distributed among multiple platforms.

To improve the search, a heuristic function is used to determine the order platforms are tested. The heuristic function is constructed by auctioning out the node to all platforms with the required capabilities. The bid is the marginal cost for the platform to accept the task relative to the current partial allocation. The cost could for example be the total time required to execute all tasks allocated to the platform.

To increase the efficiency of the backtracking, the algorithm uses backjumping to find the latest partial allocation which has a consistent allocation of the current node. This preserves the soundness as only partial allocations that are guaranteed to violate the constraints are skipped.

AllocateTST The AllocateTST algorithm takes a TST rooted in the node N as input and finds a valid allocation of the TST if possible. To check whether a node N can be allocated to a specific platform P the TryAllocateTST algorithm is used. It tries to allocate the top node N to P and then tries to recursively find an allocation of the sub-TSTs.

AllocateTST(Node N)

```
1. Find the set of candidates $C$ for $N$.
2. Run an auction for $N$ among the candidates in $C$ and order $C$ according to the bids.
3. For each candidate $c$ in the ordered set $C$:
   (a) If TryAllocateTST($c$, $N$) then return success.
4. Return failure.
```

TryAllocateTST(Platform P, Node N)

```
1. AllocateTST $P$ to $N$.
2. If the allocation is inconsistent then undo the allocation and return false.
3. For each sub-TST $n$ of $N$ do
   (a) If AllocateTST($n$) fails then undo the allocation and do a backjump.
4. An allocation has been found, return true.
```

Node Auctions Broadcasting for candidates for a node N only returns platforms with the required capabilities for the node. There is no information about the usefulness or cost of allocating the node to the candidate. Blindly testing candidates for a node is an obvious source of inefficiency. Instead, the node is auctioned out to the candidates. Each bidding platform bids its marginal cost for executing the node. I.e., taking into account all previous tasks the platform has been allocated, how much more would it cost the platform to take on the extra task. The cost could for example be the total time needed to complete all tasks. To be efficient, it is important that the cost can be computed by the platform locally. We are currently only evaluating the cost of the current node, not the sub-TST rooted in the node. This leaves room for interesting extensions. Low bids are favorable and the candidates are sorted according to their bids. The bids are used as a heuristic function that increases the chance of finding a suitable platform early in the search.

7 Extending the FIPA Abstract Architecture for Delegation

In Section 3, we provided an overview of the software architecture being used to support the delegation-based collaborative system. It consists of an agent layer added to a legacy system. There are four agents in this layer with particular responsibilities, the Interface Agent, the Resource Agent, the Delegation Agent and the Executor Agent. In previous sections, we described the delegation process which includes recursive delegation, the generation of TSTs, allocation of tasks in TST's to agents, and the use of distributed constraint solving in order to guarantee the validity of an allocation and solution of a TST. This complex set of processes will be implemented in the software architecture by extending the FIPA Abstract Architecture with a number of application dependent services and protocols:

– We will define a Interface Service, Resource Service, Delegation Service and Executor Service, associated with each Interface, Resource, Delegation, and Executor Agent, respectively, on each platform. These services are local to agents and not global.
– We will also define three interaction protocols, the Capability Lookup Protocol, Auction Protocol, and Delegation Protocol. These protocols will be used by the agents to guide the interaction between them as the delegation process unfolds.

7.1 Services

To implement the Delegation Process the Directory Facilitator and four new services are needed. The *Delegation Service* is responsible for coordinating delegations. The Delegation Service uses the *Interface Service* to communicate with other platforms, the Directory Facilitator to find platforms with appropriate capabilities, the *Resource Service* to keep track of local resources and the *Executor Service* to execute tasks using the legacy system.

Directory Facilitator The Directory Facilitator (DF) is part of the FIPA Abstract Architecture. It provides a registry over services where a service name is associated with an agent providing that service. In the collaborative architecture the DF is used to keep track of the capabilities of platforms. Every platform should register the names of the tasks that it has the capability to achieve. This provides a mechanism to find all platforms that have the appropriate capabilities for a particular task. To check that a platform also has the necessary resources a more elaborate procedure is needed which is provided by the Resource Service. The Directory Facilitator also implements the Capability Lookup protocol described below.

The Interface Service The Interface Service, implemented by an Interface Agent, is a clearinghouse for communication. All requests for delegation and other types of communication pass through this service. Externally, it provides the interface to a specific robotic system. The Interface Service does not implement any protocols, rather it forwards approved messages to the right internal service.

The Resource Service The Resource Service, implemented by a Resource Agent, is responsible for keeping track of the local resources of a platform. It determines whether the platform has the resources to achieve a particular task with a particular set of constraints. It also keeps track of the bookings of resources that are required by the tasks the platform has committed to. When a resource is booked a *booking constraint* is added to the local constraint store. During the execution of a complex task the Resource Service is responsible for monitoring the resource constraints of the task and detecting violations as soon as possible. Since resources are modeled using constraints this reasoning is mainly a constraint satisfaction problem (CSP) which is solved using local solvers that are part of the service.

In the prototype implementation, constraints are expressed in ESSENCE' which is a sub-set of the ESSENCE high-level language for specifying constraint problems [35]. The idea behind ESSENCE is to provide a high-level, solver independent, language which can be translated or compiled into solver specific languages. This opens up the possibility for different platforms to use different local solvers. We use the translator Tailor [37] which can compile ESSENCE' problems into either Minion [36] or ECLiPSe [65]. We currently use Minion as the local CSP solver. The Resource Service implements the Auction protocol described below.

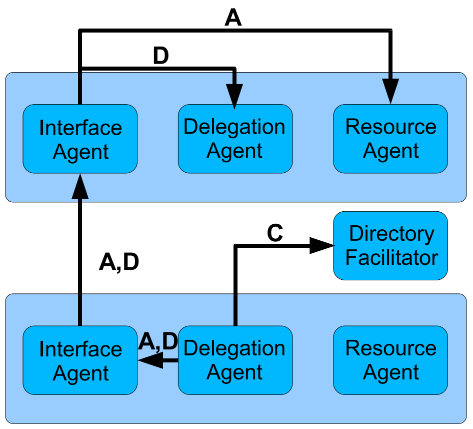

Fig. 11. An overview of the agents involved in the Auction (A), Capability Lookup (C), and Delegation (D) protocols.

The Delegation Service The Delegation Service, implemented by a Delegation Agent, coordinates delegation requests to and from the platform using the Executor, Resource and Interface Services. It does this by implementing the Delegation Process described in Section 6. The Delegation Service implements the Delegation Protocol described below.

The Executor Service The Executor Service, implemented by a Executor Agent, is responsible for executing tasks using the legacy system on the platform. In the simplest case this corresponds to calling a single function in the legacy system while in more complicated cases the Executor Service might have to call local planners to generate a local plan to achieve a task with a particular set of constraints.

7.2 Protocols

This section describes the three main protocols used in the collaboration framework: the Capability Lookup Protocol, the Auction Protocol, and the Delegation Protocol. An overview of the agents involved in the protocols is shown in Figure 11.

The Capability Lookup Protocol The Capability Lookup Protocol is based on the FIPA Request Protocol. This protocol is used to find all platforms that have the capabilities for a certain task. The content of the *request* message is the name of the task. The reply is an *inform* message with the platforms that have the capabilities required for the task.

The Auction Protocol The Auction Protocol is based on the FIPA Request Protocol. The protocol is used to request bids for tasks from platforms. The bid should reflect the cost for the platform to accept the task and is calculated by an *auction strategy*. An auction strategy could for instance be the marginal cost strategy where the bid is the marginal cost (in time) for a platform to take on the task. The content of the *request* message is the task that is being auctioned out. If the platform makes a bid, then the reply is an *inform* message containing the task and the bid. Otherwise, a *refuse* message is returned. One reason for not making a bid could be that the platform lacks the capabilities or resources for the task.

The Delegation Protocol The Delegation Protocol, which is an extension of the FIPA Contract Net protocol [34, 61], implements the Delegation Process described in Section 6. The Delegation Protocol, like the Contract Net Protocol, has two phases, each containing the sending and receiving of a message. The first phase allocates platforms to tasks satisfying the preconditions of the S-Delegate speech act and the second phase executes the task satisfying the postconditions of the S-Delegate speech act.

In the first phase a *call-for-proposal* message is sent from the delegator, and a *propose* or *refuse* message is returned by the potential contractor. The content is a declarative representation of the task in the form of a TST and a set of constraints. When a potential contractor receives a *call-for-proposal* message, an instance of the Delegation Protocol is started. When the first phase is completed, if successful, the preconditions for the S-Delegate speech act are satisfied and all the sub-tasks in the TST have been allocated to platforms such that all the constraints are satisfied.

In the second phase an *accept-proposal* is sent from the delegator to the contractor. This starts the execution of the task. If the execution is successful, then the contractor returns an *inform* message otherwise a *failure* message. Such failure messages will invoke repair processes that will not be described in this article.

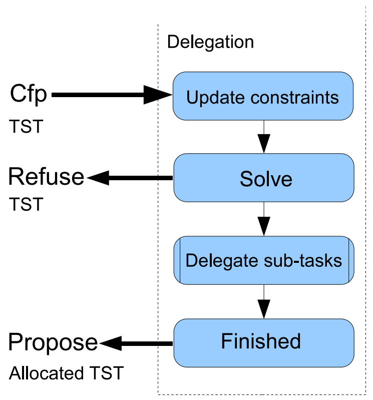

Fig. 12. An overview of the Delegation Protocol.

An overview of the steps in the Delegate Protocol is shown in Figure 12. When a Delegation Agent receives a *call-for-proposal* message with a TST the platform becomes a potential contractor. To check if the platform can accept the delegation it first updates that part of the its constraint network representing all the constraints related to the TST. This is done by instantiating the platform specific resource constraints for the action associated with the top node of the TST. If the resulting constraint problem is inconsistent, then a *refuse* message is returned to the delegator. Otherwise, the resources required for the node are booked through the Resource Service and the sub-tasks of the TST are recursively delegated. When a platform books its resources, it places commitments in the form of constraints in its constraint stores and schedulers which reserve resources and schedule activities relative to the temporal constraints which are part of the TST solution.

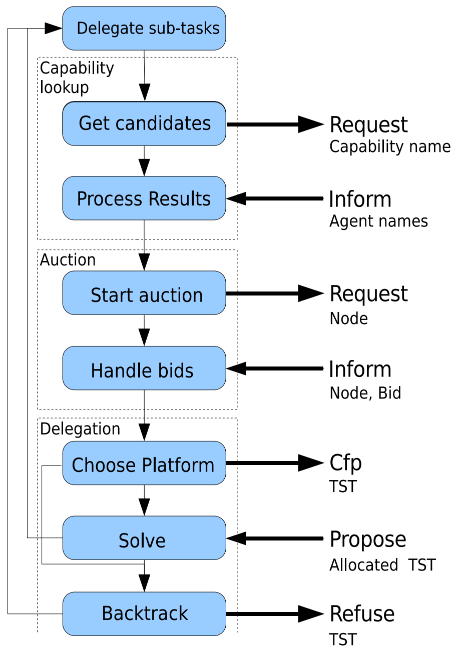

Fig. 13. An overview of the recursive delegation of sub-tasks part of the Delegation Protocol.

For each sub-task of the TST the Delegation Protocol goes through the steps shown in Figure 13. First, it will use the Capability Lookup Protocol to find all the platforms that have the capabilities, but not necessarily the resources, to achieve the task. Then it will use the Auction Protocol to request bids from these platforms in parallel. The bids are used to decide the order in which the platforms are tried. The platform with the lowest bid, i.e. the lowest cost, will be allocated the task first. If that allocation fails, then the platform with the next lowest bid will be allocated the task. Allocating a task to a platform involves sending a *call-for-proposal* message with the task to the platform. This will trigger the Delegation Protocol on that platform. If an allocation fails, then backtracking starts. If backtracking has tested all the choices, then the potential contractor returns a *refuse* message to the delegator.

If all sub-tasks can either be allocated to the platform or delegated to some other platform, then a *propose* message with the allocated TST is returned to the delegator.

8 A Collaborative UAS Case Study

On December 26, 2004, a devastating earthquake of high magnitude occurred off the west coast of Sumatra. This resulted in a tsunami which hit the coasts of India, Sri Lanka, Thailand, Indonesia, and many other islands. Both the earthquake and the tsunami caused great devastation. During the initial stages of the catastrophe, there was a great deal of confusion and chaos in setting into motion rescue operations in such wide geographic areas. The problem was exacerbated by a shortage of manpower, supplies, and machinery. The highest priorities in the initial stages of the disaster were searching for survivors in many isolated areas where road systems had become inaccessible and providing relief in the form of delivery of food, water, and medical supplies. Similar real-life scenarios have occurred more recently in China and Haiti where devastating earthquakes have caused tremendous material and human damage.

Let us assume that one has access to a fleet of autonomous unmanned helicopter systems with ground operation facilities. How could such a resource be used in the real-life scenario described?

A prerequisite for the successful operation would be the existence of a multi-agent software infrastructure for assisting emergency services. At the very least, one would require the system to allow mixed-initiative interaction with multiple platforms and ground operators in a robust, safe, and dependable manner. As far as the individual platforms are concerned, one would require a number of different capabilities, not necessarily shared by each individual platform, but by the fleet in total. These capabilities would include: the ability to scan and search for salient entities such as injured humans, building structures, or vehicles; the ability to monitor or survey these salient points of interest and continually collect and communicate information back to ground operators and other platforms to keep them situationally aware of current conditions; and the ability to deliver supplies or resources to these salient points of interest if required. For example, identified injured persons should immediately receive a relief package containing food, water, and medical supplies.

To be more specific in terms of the scenario, we can assume there are two separate legs or parts to the emergency relief scenario in the context sketched previously.

Leg I In the first part of the scenario, it is essential that for specific geographic areas, the unmanned aircraft platforms cooperatively scan large regions in an attempt to identify injured persons. The result of such a cooperative scan would be a saliency map pinpointing potential victims and their geographical coordinates and associating sensory output such as high resolution photos and thermal images with the potential victims. The saliency map could then be used directly by emergency services or passed on to other unmanned aircrafts as a basis for additional tasks.

Leg II In the second part of the scenario, the saliency map from Leg I would be used for generating and executing a plan for the UAS to deliver relief packages to the injured. This should also be done in a cooperative manner.

We will now consider a particular instance of the emergency services assistance scenario. In this instance there is a UAS consisting of two platforms (P_1 and P_2) and an operator (OP_1). In the first part of the scenario the UAS is given the task of searching two areas for victims. The main capability required by the platforms is to fly a search pattern scanning for people. In this scenario, both platforms have this capability. It is implemented by looking for salient features in the fused video streams from color and thermal cameras [59]. In the second part the UAS is given the task to deliver boxes with food and medical supplies to the identified victims. To transport a box it can either be carried directly by an unmanned aircraft or it can be loaded onto a carrier which is then transported to a key position from where the boxes are distributed to their final locations. In this scenario, 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 sequential and concurrent tasks.

8.1 Leg I: The Victim Search Case Study

The victim search case study covers the first part of the emergency services assistance scenario. In this particular scenario, see Figure 5 on page 14, the UAS should first scan $\mathsf{Area_A}$ and $\mathsf{Area_B}$ for survivors, and then fly to $\mathsf{Dest_4}$ to be ready to load emergency supplies. The TST for this mission is shown in Figure 6 on page 14.

To carry out the mission, the operator needs to delegate the TST to one of the platforms. This is done by invoking the Delegation Protocol in the operator ground station. The protocol will find a platform that can achieve the complex task and then give the operator the option to approve the choice. If the choice is approved, then the mission will be carried out.

The Delegation Agent of OP_1 starts the process of finding a platform that can achieve the TST by finding all platforms that have the capabilities for the top node N_0, which is both platforms. It then auctions out N_0 to both platforms to find the best initial choice. In this case, the marginal cost is the same for both platforms, so the first platform, P_1, is chosen. The Delegation Agent of OP_1 then sends a *call-for-proposal* message with the TST to the winner, P_1. This invokes the Delegation Protocol on P_1.

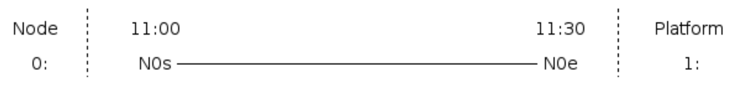

Fig. 14. The schedule after assigning node N_0 to P_1.

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. See Figure 14 for the schedule. The allocation algorithm traverses the TST in depth-first order. P_1 will first find a platform for node N_1. When the entire sub-TST rooted in N_1 is allocated then it will find an allocation for node N_4. Node N_1 is a composite action node which has the same marginal cost for all platforms. P_1 therefore allocates N_1 to itself. The extended schedule is shown in Figure 15. The constraints from nodes N_0–N_1 are added to the constraint network of P_1. The network is consistent because the composite action nodes describe a schedule without any restrictions.

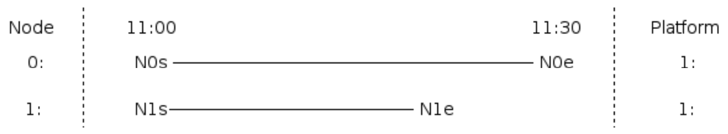

Fig. 15. The schedule after assigning node N_1 to P_1.

Platform P_1 should now allocate the elementary action nodes N_2 and N_3. A capability lookup operation followed by an auction of node N_2 determines the candidates P_1 and P_2. A *call-for-proposal* message containing N_2 is sent to platform P_2.

P_2 receives the *call-for-proposal* message, loads and instantiates the platform's resource constraint for the *scan* action. The constraint network is connected to the constraint network of the TST. The network is then checked for consistency. Since the network is consistent, node N_2 is now allocated to platform P_2. P_2 returns a *propose* message to P_1. The constraint network now involves both platforms. Figure 16 shows the schedule.

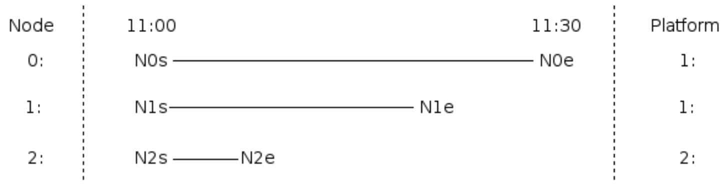

Fig. 16. The schedule after assigning node N_2 to P_2.

Continuing with node N_3, platform P_1 searches for candidates for the node. The capability lookup and auctioning determines platform P_1 as a better choice than P_2 for the second scan node. P_1 delegates the node to itself since the extended constraint network is consistent. Figure 17 shows the extended schedule.

The remaining node, N_4 is delegated to platform P_2. The entire TST is now allocated. The complete schedule is shown in Figure 18.

The operator approves the allocation and starts the mission. An *accept-proposal* message is sent to P_1. P_1 recursively traverses the TST marking the nodes as ready for

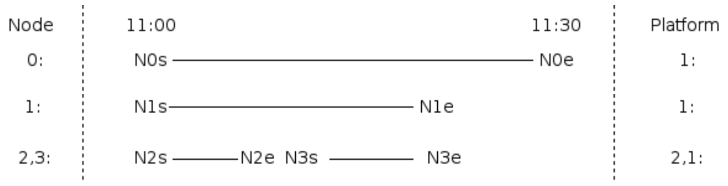

Fig. 17. The schedule after assigning node N_3 to P_1.

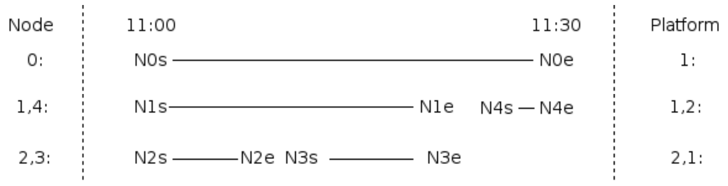

Fig. 18. The schedule after assigning node N_4 to P_2, which is the complete schedule.

execution in depth-first order. Nodes allocated to another platform are marked by sending a *accept-proposal* to the platform. P_1 therefore sends *accept-proposal* messages to P_2 for node N_2 and N_4. The execution starts and the platforms scans the area creating the saliency map shown in Figure 19.

8.2 Leg II: The Supply Delivery Case Study

The supply delivery case study covers the second part of the emergency services assistance scenario. One approach to solving this type of logistics problems is to use a task planner to generate a sequence of actions that will transport each box to its destination. Each action must then be executed by a platform. We have previously shown how to generate pre-allocated plans and monitor their execution [21, 51]. In this paper we show how a plan without explicit allocations expressed as a TST can be cooperatively allocated to a set of unmanned aircraft platforms which where not known at the time of planning.

In this particular scenario, shown in Figure 19, five survivors (S_1–S_5) are found in Leg I, and there are two platforms (P_1–P_2) and one carrier available. At the same time another operator OP_2 is performing a mission with the platforms P_3 and P_4 north of the area in Figure 19. P_3 is currently idle and OP_1 is therefore allowed to borrow it if necessary.

To start Leg II, the operator creates a TST, for example using a planner, that will achieve the goal of distributing relief packages to all survivor locations in the saliency map. The resulting TST is shown in Figure 20. The TST contains a sub-TST (N_1–N_{12}) for loading a 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 the survivor in the right uppermost part of the

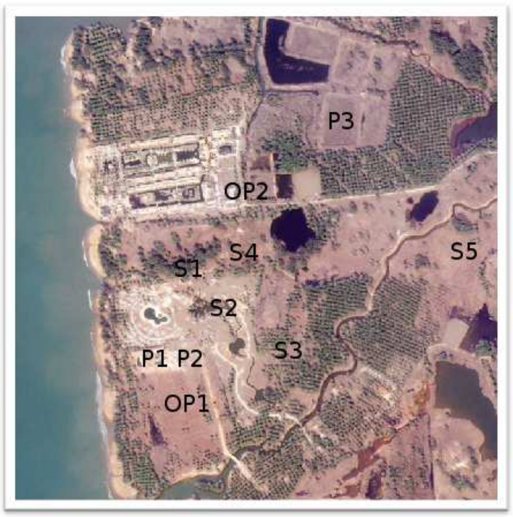

Fig. 19. The disaster area with platforms P_1–P_3, survivors S_1–S_5, and operators OP_1 and OP_2.

region, 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 delegate the TST, the Delegation Agent of OP_1 searches for a platform that can achieve the TST. It starts by finding all platforms that have the capabilities for the top node N_0, which is both platforms. It then auctions out N_0 to both platforms to find the best initial choice. In this case, the marginal cost is the same for both platforms and the first platform, P_1 is chosen. The Delegation Agent of OP_1 then sends a *call-for-proposal* message with the TST to the winner, P_1. This invokes the Delegation Protocol on 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. P_1 will first find a platform for node N_1. 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 because the composite action nodes describe a schedule without any restrictions.

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. This is because of each platform's unique state, constraint model for the action, and available resources. The candidates for node N_3 are platforms P_1 and P_2. P_1 is closest to the package depot and therefore gives the best bid for the node. 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

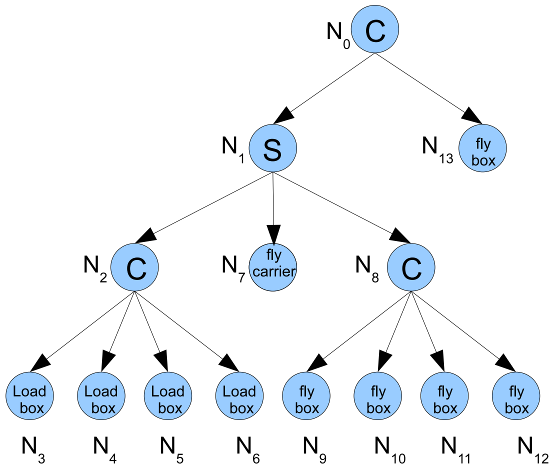

Fig. 20. The TST for the supply delivery case study.

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 to allocate for P_1 is node N_7, the carrier delivery node. P_1 is the only platform that has the capabilities for the fly carrier task and is allocated the node. Continuing with nodes N_8–N_{12}, the platform with the lowest bid 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 final node, N_{13}, is allocated to platform P_2 and the allocation is complete. The resulting schedule is shown in Figure 21.

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 constraint problem defined by the allocation of the TST is distributed between the platforms, it is possible for the operator to modify the constraint problem by adding more constraints, and in this way modify the resulting task allocation. The operator puts a time constraint on the mission, restricting the total time to 30 minutes.

To re-allocate the TST with the added constraint, operator OP_1 sends a *reject-proposal* to platform P_1. 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. Backtracking 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

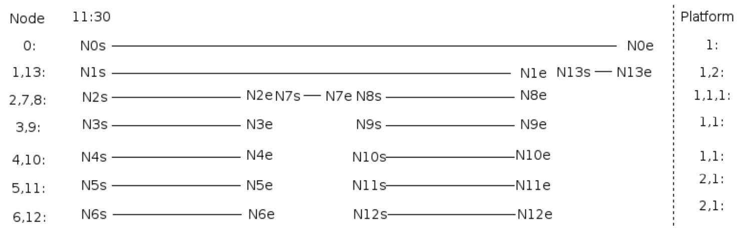

Fig. 21. The complete schedule when using two platforms and no deadline.

that N_{13} can be allocated to P_2. P_1 sends a *backjump-search* message to the platform in charge of the sub-TST with top-node N_1, which happens to be P_1. When receiving the message, P_1 continues the search for the backjump point. 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 and tries the next platform for the node, P_1.

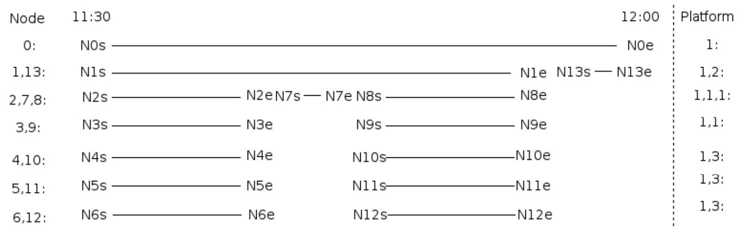

Fig. 22. The resulting schedule after adding the new time constraint.

When the allocation reaches node N_{11} it is discovered that since P_1 has taken on nodes N_3–N_8, there is not enough time left for P_1 to unload the last two packages from the carrier. Instead P_3, even though it makes a higher bid for N_{11}–N_{12}, is allocated to both nodes. Finally platform P_2 is allocated to node N_{13}. It turns out that since platform P_2 helped P_1 loading the carrier, it has not enough time to deliver the final package. Instead, a new backjump point search starts, finding node N_5. The search continues from N_5. This time, nodes N_3–N_9 are allocated to platform P_1, platform P_3 is allocated to node N_{10}–N_{12}, and platform P_2 is allocated to node N_{13}. The allocation is consistent.

The resulting schedule is shown in Figure 22. The allocation algorithm finishes on platform P_1, by sending a *propose* message back to the operator. The operator inspects the allocation and approves it, thereby confirming the delegation and starting the execution of the mission.

9 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 autonomy, cooperative multi-robot systems, task allocation from a robotic perspective, and auctions.

The concept of autonomy has a long and active history in multi-agent systems [44, 47]. One early driving force was space missions that focused on the problem of interaction with autonomous agents and the adjustability of this autonomy [4, 26]. Later, Hexmoor and McLaughlan argue that reasoning about autonomy is an integral component of collaboration among computational units [45]. Hexmoor also argues that trust is essential for autonomy [46]. According to his definition, the autonomy of an agent A with respect to a task t is the degree of self-determination the agent possesses to perform the task. This is similar to the view on autonomy in our approach, where the level of autonomy for an agent is dependent on the strictness of the constraints on the tasks that are delegated to the agent.

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 [2], where robots share plans and coordinates their own plans to produce coordinated plans. In our approach, coordination is achieved by finding solutions to a distributed constraint problem representing the complex task, rather than by sharing and merging plans. Another early work is ALLIANCE [57], which is a behavior-based framework for instantaneous task assignment of loosely coupled subtasks with ordering dependencies. Each agent decides on its own what tasks to do based on its observations of the world and the other agents. Compared to our approach, this is a more reactive approach which does not consider what will happen in the future. M+ [3] integrates mission planning, task refinement and cooperative task allocation. It uses a task allocation protocol based on the Contract Net protocol with explicit, predefined capabilities and task costs. A major difference to our approach is that in M+ there is no temporally extended allocation. Instead, robots make incremental choices of tasks to perform from the set of executable tasks, which are tasks whose prerequisite tasks are achieved or underway. The M+CTA framework [1] is an extension of M+, where a mission is decomposed into a partially ordered set of high-level tasks. Each task is defined as a set of goals to be achieved. The plan is distributed to each robot and task allocation is done incrementally like in M+. When a robot is allocated a task, it creates an individual plan for achieving the task's goals independently of the other agents. After the planning step, robots negotiate with each other to adapt their plans in the multi-robot context. Like most negotiation-based approaches, M+CTA first allocates the tasks and then negotiates to handle coordination. This is different from our approach which finds a valid allocation of all the tasks before committing to the allocation. ASyMTRe [58], uses a reconfigurable schema abstraction for collaborative task execution providing sensor sharing among robots, where connections among the schemas are dynamically formed at runtime. The properties of inputs and outputs of each schema is defined and by determining a valid information flow through a combination of schemas within, and across, robot team members a coalition for solving a particular task can be formed. Like ALLIANCE, this is basically a reactive approach which considers the current task, rather than a set of related tasks as in our approach. Other Contract-Net and auction-based systems similar to those described above are COMETS [53], MURDOCH system [39], Hoplites [49] and TAEMS [12].

Many task allocation algorithms are, as mentioned above, auction-based [13, 39, 49, 63, 71, 72]. There, tasks are auctioned out and allocated to the agent that makes the best bid. Bids are determined by a utility function. The auction concept decentralizes the task allocation process which is very useful especially in multi-robot systems, where centralized solutions are impractical. For tasks that have unrelated utilities, this approach has been very successful. The reason is that unrelated utilities guarantees that each task can be treated as an independent entity, and can be auctioned out without affecting other parts of the allocation. This means that a robot does not have to take other tasks into consideration when making a bid. More advanced auction protocols have been developed to handle dependencies between tasks. These are constructed to deal with complementarities. Examples are sequential single item auctions [50] and combinatorial auctions [64]. These auctions typically handle that different combinations of tasks have different bids, which can be compared to our model where different sets of allocations result in different restrictions to the constraint network between the platforms.

The sequential single item (SSI) auction [50] is of special interest since it is similar to our approach. In SSI auctions, like our task allocation approach, tasks are auctioned out in sequence, one at a time to make sure the new task fits with the previous allocations. The difference is what happens when there is no agent that can accept the next task. In SSI auctions common strategies are to return a task in exchange for the new task or to start exchanging tasks with other agents. This is basically a greedy approach which is incomplete. Our approach on the other hand uses backtracking which is a complete search procedure. Normally SSI auctions are applied to problems where it is easy to find a solution but it is hard to find a good solution. When allocating the tasks in a TST it is often hard to find any solution and SSI auctions are therefore not appropriate.

Combinatorial auctions deal with complementarities by bidding on bundles containing multiple items. Each bidder places bids on all the bundles that are of interest, which could be exponentially many. The auctioneer must then select the best set of bids, called the winner determination problem, which is NP-hard [64]. Since all agents have to bid on all bundles, in our case tasks, they could accept in one round it means that even in the best case there is a very high computational cost involved in using combinatorial auctions. Another weakness is that they do not easily lend themselves to a recursive process where tasks are recursively decomposed and allocated. Our approach, on the other hand, is suitable for recursive allocation and by using heuristic search will try the most likely allocations first which should result in much better average case performance.

10 Conclusions

Collaborative robotic systems have much to gain by leveraging results from the area of multi-agent systems and in particular agent-oriented software engineering. Agent-oriented software engineering has much to gain by using collaborative robotic systems as a testbed. We have proposed and specified a formally grounded generic collaborative system shell for robotic systems and human operated ground control systems. The software engineering approach is based on the FIPA Abstract Architecture and uses JADE to implement the system shell. The system shell is generic in the sense that it can be integrated with legacy robotic systems using a limited set of assumptions. Collaboration is formalized in terms of the concept of delegation and delegation is instantiated as a speech act. The formal characterization of the Speech act has a BDI flavor and KARO, which is an amalgam of dynamic logic and epistemic/doxastic logic, is used in the formal characterization. Tasks are central to the delegation process. Consequently, a flexible, specification language for tasks is introduced in the form of Task Specification Trees. Task Specification Trees provide a formal bridge between the abstract characterization of delegation as a speech act and its implementation in the collaborative system shell. Using this idea, the semantics of both delegation and tasks is grounded in the implementation in the form of a distributed constraint problem which when solved results in the allocation of tasks and resources to agents. We show the potential of this approach by targeting a real-life scenario consisting of UAV's and human resources in an emergency services application. The results described here should be considered a mature iteration of many ideas both formal and pragmatic which will continue to be pursued in additional iterations as future work.

References

1. Alami, R., Botelho, S.C.: Plan-based multi-robot cooperation. In: Advances in Plan-Based Control of Robotic Agents (2001)
2. Alami, R., Ingrand, F., Qutub, S.: A scheme for coordinating multirobot planning activities and plans execution. In: Proc. ECAI (1998)
3. Botelho, S., Alami, R.: M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In: Proc. ICRA (1999)
4. Bradshaw, J., Sierhuis, M., Acquisti, A., Gawdiak, Y., Jeffers, R., Suri, N., Greaves, M.: Adjustable autonomy and teamwork for the personal satellite assistant. In: Proc. IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents (2001)
5. Castelfranchi, C., Falcone, R.: Toward a theory of delegation for agent-based systems. In: Robotics and Autonomous Systems. vol. 24, pp. 141–157 (1998)
6. Cohen, P., Levesque, H.: Intention is choice with commitment. Artificial Intelligence 42(3), 213–261 (1990)
7. Cohen, P., Levesque, H.: Teamwork. Nous, Special Issue on Cognitive Science and AI 25(4), 487–512 (1991)
8. Conte, G., Doherty, P.: Vision-based unmanned aerial vehicle navigation using geo-referenced information. EURASIP Journal of Advances in Signal Processing (2009)
9. Conte, G., Hempel, M., Rudol, P., Lundström, D., Duranti, S., Wzorek, M., Doherty, P.: High accuracy ground target geo-location using autonomous micro aerial vehicle platforms. In: Proceedings of the AIAA-08 Guidance,Navigation, and Control Conference (2008)

10. Dastani, M., Meyer, J.J.C.: A practical agent programming language. In: M. Dastani, K. V. Hindriks, M.P.P., Sterling, L. (eds.) Proc. of the AAMAS07 Workshop on Programming Multi-Agent Systems (ProMAS2007). pp. 72–87 (2007)
11. Davis, E., Morgenstern, L.: A first-order theory of communication and multi-agent plans. Journal Logic and Computation 15(5), 701–749 (2005)
12. Decker, K.: TAEMS: A framework for environment centered analysis and design of coordination mechanisms. In: Foundations of Distributed Artificial Intelligence. Wiley Inter-Science (1996)
13. Dias, M., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: a survey and analysis. Proc. of IEEE 94(1), 1257 – 1270 (2006)
14. Doherty, P.: Advanced research with autonomous unmanned aerial vehicles. In: Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning (2004), extended abstract for plenary talk
15. Doherty, P.: Knowledge representation and unmanned aerial vehicles. In: Proceedings of the IEEE Conference on Intelligent Agent Technolology (IAT 2005) (2005)
16. Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E., Wiklund, J.: The WITAS unmanned aerial vehicle project. In: Proceedings of the 14th European Conference on Artificial Intelligence. pp. 747–755 (2000)
17. Doherty, P., Haslum, P., Heintz, F., Merz, T., Persson, T., Wingman, B.: A distributed architecture for intelligent unmanned aerial vehicle experimentation. In: Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (2004)
18. Doherty, P., Kvarnström, J.: TALplanner: A temporal logic based forward chaining planner. Annals of Mathematics and Artificial Intelligence 30, 119–169 (2001)
19. Doherty, P., Kvarnström, J.: TALplanner: A temporal logic based planner. Artificial Intelligence Magazine (Fall Issue 2001)
20. Doherty, P., Kvarnström, J.: Temporal action logics. In: Lifschitz, V., van Harmelen, F., Porter, F. (eds.) The Handbook of Knowledge Representation, chap. 18, pp. 709–757. Elsevier (2008)
21. Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Journal of Automated Agents and Multi-Agent Systems 19(3), 332–377 (2009)
22. Doherty, P., Landén, D., Heintz, F.: A distributed task specification language for mixed-initiative delegation. In: Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA) (2010)
23. Doherty, P., Łukaszewicz, W., Szałas, A.: Approximative query techniques for agents with heterogenous ontologies and perceptual capabilities. In: Proceedings on the 7th International Conference on Information Fusion (2004)
24. Doherty, P., Łukaszewicz, W., Szałas, A.: Communication between agents with heterogeneous perceptual capabilities. Journal of Information Fusion 8(1), 56–69 (January 2007)
25. Doherty, P., Meyer, J.J.C.: Towards a delegation framework for aerial robotic mission scenarios. In: Proceedings of the 11th International Workshop on Cooperative Information Agents (2007)
26. Dorais, G., Bonasso, R., Kortenkamp, D., Pell, B., Schreckenghost, D.: Adjustable autonomy for human-centered autonomous systems on mars. In: Proc. Mars Society Conference (1998)
27. Dunin-Keplicz, B., Verbrugge, R.: Teamwork in Multi-Agent Systems. Wiley (2010)
28. Duranti, S., Conte, G., Lundström, D., Rudol, P., Wzorek, M., Doherty, P.: LinkMAV, a prototype rotary wing micro aerial vehicle. In: Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace (2007)
29. F. Bellifemine, G.C., Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley and Sons, Ltd (2007)

30. F. Bellifemine, F. Bergenti, G.C., Poggi, A.: JADE – a Java agent development framework. In: R. H. Bordini, M. Dastani, J.D., Seghrouchni, A. (eds.) Multi-Agent Programming - Languages, Platforms and Applications. Springer (2005)
31. Falcone, R., Castelfranchi, C.: 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)
32. Foundation for Intelligent Physical Agents: FIPA Abstract Architecture Specification. http://www.fipa.org
33. Foundation for Intelligent Physical Agents: FIPA Communicative Act Library Specification. http://www.fipa.org
34. Foundation for Intelligent Physical Agents: FIPA Contract Net Interaction Protocol Specification. http://www.fipa.org
35. Frisch, A., Grum, M., Jefferson, C., Hernández, B.M., Miguel, I.: The Design of ESSENCE: A Constraint Language for Specifying Combinatorial Problems. In: IJCAI. pp. 80–87 (2007)
36. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: In: Proceedings of ECAI 2006, Riva del Garda. pp. 98–102 (2006)
37. Gent, I.P., Miguel, I., Rendl, A.: Tailoring solver-independent constraint models: A case study with essence' and minion. In: In Proceedings of the 7th International conference on Abstraction, reformulation, and approximation (SARA2007). pp. 184–199 (2007)
38. Gerkey, B.: On multi-robot task allocation. Ph.D. thesis (2003)
39. Gerkey, B., Mataric, M.: Sold!: Auction methods for multi-robot coordination. IEEE Transactions on Robotics and Automation (2001)
40. Heintz, F., Doherty, P.: DyKnow: A knowledge processing middleware framework and its relation to the JDL fusion model. Journal of Intelligent and Fuzzy Systems 17(4) (2006)
41. Heintz, F., Doherty, P.: DyKnow federations: Distributing and merging information among UAVs. In: Eleventh International Conference on Information Fusion (FUSION-08) (2008)
42. Heintz, F., Kvarnström, J., Doherty, P.: A stream-based hierarchical anchoring framework. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS) (2009)
43. Heintz, F., Kvarnström, J., Doherty, P.: Bridging the sense-reasoning gap: DyKnow - stream-based middleware for knowledge processing. Journal of Advanced Engineering Informatics 24(1), 14–25 (2010)
44. Hexmoor, H., Kortenkamp, D.: Autonomy control software. An introductory article and special issue of Journal of Experimental and Theoretical Artificial Intelligence (2000)
45. Hexmoor, H., McLaughlan, B.: Computationally adjustable autonomy. Journal of Scalable Computing: Practive and Experience 8(1), 41–48 (2007)
46. Hexmoor, H., Rahimi, S., Chandran, R.: Delegations guided by trust and autonomy. Web Intelligence and Agent Systems 6(2), 137–155 (2008)
47. Hexmoor, H., Castelfranchi, C., Falcone, R. (eds.): Agent Autonomy. Springer Verlag (2003)
48. W. van der Hoek, B.v.L., Meyer, J.J.C.: An integrated modal approach to rational agents. In: Wooldridge, M., Rao, A. (eds.) Foundations of Foundations of Rational Agency, Applied Logic Series, vol. 14. An Integrated Modal Approach to Rational Agents (1998)
49. Kaldra, N., Ferguson, D., Stentz, A.: Hoplites: A market-based framework for planned tight coordination in multirobot teams. In: Proc. ICRA (2005)
50. Koenig, S., Keskinocak, P., Tovey, C.: Progress on agent coordination with cooperative auctions. In: Proc. AAAI (2010)
51. Kvarnström, J., Doherty, P.: Automated planning for collaborative systems. In: Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV) (2010)

52. Landén, D., Heintz, F., Doherty, P.: Complex task allocation in mixed-initiative delegation: A UAV case study (early innovation). In: Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA) (2010)
53. Lemaire, T., Alami, R., Lacroix, S.: A distributed tasks allocation scheme in multi-uav context. In: Proc. ICRA (2004)
54. Magnusson, M., Landen, D., Doherty, P.: Planning, executing, and monitoring communication in a logic-based multi-agent system. In: 18th European Conference on Artificial Intelligence (ECAI 2008) (2008)
55. Merz, T., Rudol, P., Wzorek, M.: Control System Framework for Autonomous Robots Based on Extended State Machines. In: Proceedings of the International Conference on Autonomic and Autonomous Systems (2006)
56. Modi, P., Shen, W.M., Tambe, M., Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees. AI 161 (2006)
57. Parker, L.E.: Alliance: An architecture for fault tolerant multi-robot cooperation. IEEE Trans. Robot. Automat 14(2), 220–240 (1998)
58. Parker, L.E., Tang, F.: Building multi-robot coalitions through automated task solution synthesis. Proceeding of the IEEE, Special Issue on Multi-Robot Systems (2006)
59. Rudol, P., Doherty, P.: Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery. In: Proc. of the IEEE Aerospace Conference (2008)
60. Rudol, P., Wzorek, M., Conte, G., Doherty, P.: Micro unmanned aerial vehicle visual servoing for cooperative indoor exploration. In: Proceedings of the IEEE Aerospace Conference (2008)
61. Smith, R.: The contract net protocol. IEEE Transactions on Computers C-29(12) (1980)
62. Telecom Italia Lab: The Java Agent Development Framework (JADE). http://jade.tilab.com
63. Viguria, A., Maza, I., Ollero, A.: Distributed service-based cooperation in aerial/ground robot teams applied to fire detection and extinguishing missions. Advanced Robotics 24, 1–23 (2010)
64. de Vries, S., Vohra, R.: Combinatorial auctions: A survey. Journal on Computing 15(3), 284–309 (2003)
65. Wallace, M.G., Schimpf, J., Novello, S.: A Platform for Constraint Logic Programming. ICL System Journal 12(1), 159–200 (1997)
66. Wzorek, M., Conte, G., Rudol, P., Merz, T., Duranti, S., Doherty, P.: 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)
67. Wzorek, M., Doherty, P.: Reconfigurable path planning for an autonomous unmanned aerial vehicle. In: Proceedings of the 16th International Conference on Automated Planning and Scheduling. pp. 438–441 (2006)
68. Wzorek, M., Kvarnström, J., Doherty, P.: Choosing path replanning strategies for unmanned aircraft systems. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2010)
69. Wzorek, M., Landen, D., Doherty, P.: GSM technology as a communication media for an autonomous unmanned aerial vehicle. In: Proceedings of the 21st Bristol International Conference on UAV Systems (2006)
70. Yokoo, M.: Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In: Proc. CP (1995)
71. Zlot, R.: An auction-based approach to complex task allocation for multirobot teams. Ph.D. thesis (2006)
72. Zlot, R., Stentz, A.: Complex task allocation for multiple robots. In: Proc. ICRA (2005)