

Linköping Studies in Science and Technology

Thesis No. 1506

Complex Task Allocation for Delegation: From Theory to Practice

by

David Landén



Linköping University
INSTITUTE OF TECHNOLOGY

Submitted to Linköping Institute of Technology at Linköping University in partial
fulfilment of the requirements for degree of Licentiate of Engineering

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Linköping 2011

Copyright © David Landén 2011

ISBN 978-91-7393-048-2

ISSN 0280-7971

Printed by LiU Tryck 2011

Electronic version available at:

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-70536>

Complex Task Allocation for Delegation: From Theory to Practice

by

David Landén

Oktober 2011

ISBN 978-91-7393-048-2

Linköping Studies in Science and Technology

Thesis No. 1506

ISSN 0280-7971

LiU-Tek-Lic-2011:45

ABSTRACT

The problem of determining who should do what given a set of tasks and a set of agents is called the task allocation problem. The problem occurs in many multi-agent system applications where a workload of tasks should be shared by a number of agents. In our case, the task allocation problem occurs as an integral part of a larger problem of determining if a task can be delegated from one agent to another.

Delegation is the act of handing over the responsibility for something to someone. Previously, a theory for delegation including a delegation speech act has been specified. The speech act specifies the preconditions that must be fulfilled before the delegation can be carried out, and the postconditions that will be true afterward. To actually use the speech act in a multi-agent system, there must be a practical way of determining if the preconditions are true. This can be done by a process that includes solving a complex task allocation problem by the agents involved in the delegation.

In this thesis a constraint-based task specification formalism, a complex task allocation algorithm for allocating tasks to unmanned aerial vehicles and a generic collaborative system shell for robotic systems are developed. The three components are used as the basis for a collaborative unmanned aircraft system that uses delegation for distributing and coordinating the agents' execution of complex tasks.

This work has been supported by The Swedish National Aeronautics Research Program NFFP04-S4203 and NFFP05-01263, Linklab (www.linklab.se), the Linnaeus Center for Control, Autonomy, Decision-making in Complex Systems (CADICS), the Center for Industrial Information Technology CENIIT (grant number 06.09), the ELLIT Excellence Center at Linköping-Lund for Information Technology, the Swedish Foundation for Strategic Research (SSF) and the Swedish Research Council (VR).

Acknowledgements

This work was carried out at the division of Artificial Intelligence & Integrated Computer Systems (AIICS) at Linköping university. I wish to express my sincere gratitude to all those who have contributed to the work leading to this thesis. In particular, I would like to thank:

My supervisor Patrick Doherty for giving me the opportunity to join AIICS and for continuous support over these years. John-Jules Charles Meyer, for together with Patrick Doherty developing the theoretical framework which is the basis for the work in my thesis.

Special thanks go to Fredrik Heintz for his tremendous help during the writing of this thesis. My thesis would never be finished without his encouragement and guidance.

I would like to thank my colleagues at AIICS: Olov Andersson, Gianpaolo Conte, Jonas Kvarnström, Martin Magnusson, Per Nyblom, Per-magnus Olsson, Mikael Nilsson, Piotr Rudol, Håkan Warnqvist, Mariusz Wzorek, for proofreading and help in other projects during this time. Especially, I would like to thank Tommy Persson for his help with the UASTech system.

The administrative and technical personal for the efficient and qualified work.

Last but not least, my family and friends for their encouragement and support. Ning for your happiness and love.

Contents

1	Introduction	1
1.1	Constraints and Multiagent Systems	4
1.2	Problem Statement and Motivation	5
1.3	Contributions	6
1.4	Publications	7
1.5	Thesis Outline	7
2	Background	9
2.1	Delegation as a Speech Act	9
2.2	Delegation-Based Software Architecture Overview	13
3	Task Specification Trees	18
3.1	TST Concepts	19
3.2	TST Syntax	20
3.3	TST Semantics	22
3.4	Related Work	29
4	Allocating Tasks in a TST to Platforms	31
4.1	Deriving the Constraint Problem of a TST	33
4.2	The Delegation Process	35
4.3	Classifying the TST Allocation Problem	37
4.3.1	Basic Task Allocation Problem Properties	38
4.3.2	Complex Task Allocation Problem Properties	39
4.3.3	Summary	41
4.4	The TST Allocation Problem Definition	41
4.5	A TST Allocation Algorithm	42
4.5.1	Distributed Backjumping	44
4.6	Alternative Approaches	47
4.6.1	An Alternative DisCSP Approach	47
4.6.2	An Integer or Linear Programming Approach	48
4.7	Related Work	48

5	Extending the FIPA Abstract Architecture for Delegation	51
5.1	The FIPA Abstract Architecture	51
5.2	An Extended Service Model	54
5.3	The Agent Layer	57
5.3.1	Services	57
5.3.2	Protocols	61
5.4	Task Allocation Algorithm Implementation	64
5.4.1	Capability Lookup	65
5.4.2	Auction Handler	65
5.4.3	Delegation Handler	66
5.4.4	Capability Loader	67
5.4.5	DisCSP Node Interface	69
5.4.6	AWCS	69
5.4.7	AWCS Handler	69
5.4.8	Resource Database Interface	70
5.4.9	Backtrack Handler	70
5.5	Related Work	71
5.6	Summary	73
6	UAS Case Studies: Assisting Emergency Services	74
6.1	Introduction	74
6.2	The Victim Search Scenario	75
6.2.1	The TST for Victim Search Scenario	76
6.2.2	Allocating the Victim Search TST	77
6.3	The Supply Delivery Scenario	79
6.3.1	The TST for the Supply Delivery Scenario	79
6.3.2	Allocating the Supply Delivery TST	79
6.4	The Communication Relay Scenario	82
6.4.1	The TST for the Communication Relay Scenario	83
6.4.2	Allocating the Communication Relay TST	83
6.5	Summary	91
7	Performance Evaluation	92
7.1	Comparison Metrics	93
7.2	The Purpose of the Experiments	95
7.3	Scalability of AllocateTST – Unbounded Allocation	96
7.4	Scalability of AllocateTST – Bounded Allocation	97
7.4.1	Performance for Bounded Allocation of Cx-P1	99
7.4.2	Performance for Bounded Allocation of C1-P2	100
7.4.3	Performance for Bounded Allocation of C1-P3	101
7.4.4	Performance for Bounded Allocation of C1-P4	102
7.4.5	Performance for Bounded Allocation of C2-P2 – C4-P4	103

7.4.6	Discussion – Bounded Allocation Results	104
7.5	A CSP Formulation	111
7.6	An Alternative DisCSP Formulation	120
7.7	Discussion	121
8	Conclusions	124
8.1	Summary	124
8.2	Future Work	126
8.3	Conclusions	128

Chapter 1

Introduction

In the past decade, the Unmanned Aircraft Systems Technologies Lab (UAS-Tech [24]) at the Department of Computer and Information Science, Linköping University, has been involved in the development of autonomous unmanned aerial vehicles (UAVs) and associated hardware and software technologies [26, 27, 28]. The size of our research platforms range from the UAS-Tech RMAX helicopter system (100 kg) [15, 29, 83, 97, 100], a modified version of the RMAX platform developed by Yamaha Motor Company, to smaller micro-size rotor based systems such as the LinkQuad [25] (1 kg) and LinkMAV [43, 84] (500 g) in addition to a fixed wing platform, the PingWing [16] (500 g). These UAV platforms are shown in Figure 1.1. 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 in various flight tests.

Previous work has focused on the development of robust autonomous systems for UAVs which seamlessly integrate control, reactive, and deliberative capabilities that meet the requirements of hard and soft real-time constraints [29, 76]. Additionally focus has been on the development and integration of many high-level autonomous capabilities studied in the area of cognitive robotics such as task planners [32, 33], motion planners [97, 98, 99], execution monitors [35], and reasoning systems [34, 38, 74], in addition to novel middleware frameworks which support such integration [60, 62, 63]. 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 at the same time essential to the introduction of such robotic systems into society in the future. Consequently, the research has focused, not only on such high-level cognitive functionalities, but also on system integration issues.

More recently, the research efforts have transitioned toward the study of systems of UAVs. The accepted terminology for such systems is Unmanned Aircraft Systems (UAS's). A UAS consists of one or more UAVs



Figure 1.1: The UASTech RMAX (upper left), PingWing (upper right), LinkQuad (lower left) and LinkMAV (lower right).

(possibly heterogeneous) in addition to one or more ground operator systems. Of specific interest are collaborative UAS applications where UAVs are required to collaborate not only with each other but also with diverse human resources [37, 39, 40, 61, 69]. With the term *collaborative UAS* we mean a UAS consisting of one or more UAVs and one or more operators that collaborate to achieve tasks. Principled interaction between UAVs and human resources is an essential component in the future uses of UAVs in complex emergency services or blue-light scenarios. Some specific target UAV scenarios, for example are search and rescue missions for people 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 UAVs become more autonomous, mixed-initiative interaction between human operators and such systems will be central in mission planning and tasking. More elaborate collaboration methods, such as mixed-initiative interaction are needed since more autonomy means less direct control.

In the future, to gain practical use and acceptance of UAVs, a verifiable, principled and well-defined interaction foundation between one or more human operators and one or more autonomous systems is required. In developing a principled framework for complex interactions between UAVs and humans in complex scenarios, many interdependent conceptual and pragmatic issues arise and need clarification not only theoretically, but also prag-

matically 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. This complexity is tackled 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 in 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 collaboration framework includes a triad of fundamental, interdependent conceptual issues: delegation, mixed-initiative interaction and adjustable autonomy (Figure 1.2). The concept of delegation is particularly important and in some sense provides a bridge between mixed-initiative interaction and adjustable autonomy.

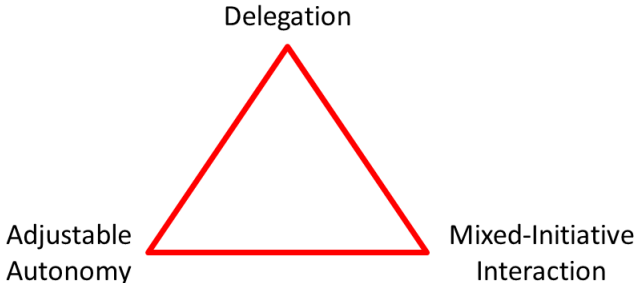


Figure 1.2: The core concepts of the collaborative framework.

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 or not to delegate sub-tasks.

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 an 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.

1.1 Constraints and Multiagent Systems

In this section we will describe a number of constraint problem formulations. Those constraint problem formulations are important because we will state the task allocation problem as a distributed constraint satisfaction problem.

A constraint problem contains a number of variables connected by different types of relations called constraints. The constraints restrict the domains of possible values for the variables. A solution to a constraint satisfaction problem (CSP) is an assignment of a value to each variable so that all constraints are satisfied.

Example: A graph coloring problem, with three nodes and three colors. In the graph coloring problem two related variables may not have the same color. The variables and their domains are $x, y, z, = \{1, 2, 3\}$, the constraints are $x \neq y, y \neq z, x \neq z$. A possible solution is $x = 1, y = 2, z = 3$.

A relatively new sub-area in the constraint research field is the area of distributed constraint satisfaction problem solving, which is basically a distributed version of constraint satisfaction problem solving. This method introduces the concept of agents to the constraint problem. In a distributed constraint satisfaction problem (DisCSP), each agent owns some of the variables. An agent can only change the values of its own variables. The constraints are either *local*, if only involving a single agent's variables, or *global*, if involving more than one agent's variables. Returning to the previous example, in a distributed version of the graph coloring problem, each variable is owned by an agent, meaning there are three agents. Each agent has two

global constraints. There are no local constraints in this example. To find a solution the agents must agree on values for their variables so that all constraints are satisfied.

A related problem formulation is distributed constraint optimization, where a solution is not only about satisfying the constraints but also to minimize or maximize the value of some expression involving one or more constraint variables. For example, each constraint has the value 0 if satisfied, 1 otherwise. The sum of all such constraint values is minimized, meaning a solution should have as few unsatisfied constraints as possible.

Formulating a problem as a distributed constraint satisfaction problem (DisCOP) is natural when solving problems related to multi-agent systems because the problem formulation assumes a multi-agent system of (autonomous) agents.

1.2 Problem Statement and Motivation

The basis for this thesis is the theoretical framework for delegation developed by Doherty and Meyer in [36, 40] and briefly introduced above.

The core of the framework is a speech act for delegation, which is a type of action for delegating a task from one agent to another. The formal semantics of the pre- and postconditions of the speech act are specified in the KARO logic [95]. The framework also describes a number of services needed to carry out delegations and how those services interact during this process.

The goal of the thesis is to answer the research questions involved in realizing the framework in a collaborative UAS. The delegation theory and the proposed framework only provide an abstract definition of a task and do not describe exactly how the pre- and post-conditions of delegation can be fulfilled. These concepts must be made concrete and explicit before a collaborative UAS can be realized. Each part can be stated as a research question:

- R1 - How can a task τ in the delegation theory be specified to make the realization of the delegation speech act possible?
- R2 - How can the preconditions of the delegation speech act be assured, so that a task τ specified according to R1 can be delegated?
- R3 - How can R1 and R2 be realized in a collaborative UAS?

The three research questions R1–R3 are together the problem statement and the topic of this thesis. The answer to R1 is needed for expressing what should be delegated. The answer to R2 is needed to carry out delegations. The answer to R3 is needed for creating a concrete instance of the collaborative UAS.

The main motivation behind the theoretical framework and the work in this thesis is to create a collaborative UAS where collaboration between agents is formed through delegations. By using the delegation concept from the delegation theory together with adjustable autonomy and mixed initiative interaction, the operator will be relieved of micro-managing the UAVs while still retaining the ability to update the restrictions on how the UAS may achieve the tasks. Simplifying the work of the operator becomes increasingly important as the number of UAVs in the UAS increases. Increasing the UAVs' autonomy can ease this problem, but more autonomy is not always the answer. For safety-critical missions it is important that the operator can add restrictions on how the UAS may achieve a delegated task and thus decrease the platforms' autonomy, to avoid dangerous situations.

In a collaborative UAS realizing the concepts of the theoretical framework for delegation it would be possible for an operator to collaborate with the UAVs in a way that fits the current mission. The operator can put restrictions on the execution of a mission, determine the autonomy of the platforms in a mission, the degree of operator involvement during execution of a mission, etc. Such a system would be very useful, for example in assisting emergency services in dangerous and time-critical disaster situations.

1.3 Contributions

The main contributions of this thesis are the answers to the research questions R1–R3, in the form of a task specification formalism (*task specification trees*), a complex task allocation algorithm for allocating the tasks in a task specification tree to UAV platforms (*AllocateTST*), and a generic collaborative system shell for robotic systems (*an extension of the FIPA Abstract Architecture*).

A task specification tree is a declarative, constraint-based representation of a complex multi-agent task. A task can only be delegated to a platform if the delegated platform is able to carry out the task. A prerequisite for delegation is therefore the ability to determine who can be delegated a task, which is a form of task allocation. The task allocation problem occurs because the delegator can not by itself determine if the contractor can be allocated to the task. The AllocateTST algorithm, which is a part of the delegation process, finds suitable platforms for the tasks in a task specification tree. Task allocation is formulated as a distributed constraint satisfaction problem, that is extended for each individual allocation of a task in a task specification tree.

The collaborative robotic shell includes an agent layer with functionality for delegation and a representation of the platforms' capabilities and resources. The implementation builds on a multi-agent system infrastructure compliant with the FIPA Abstract Architecture specification [48]. The original specification has no representation of capabilities and resources, which are needed for the allocation of task specification trees. Efficient resource

usage is an important issue for the types of robotic platforms we use, where each platform carries its own resources. That is why we require a capability and resource model to more accurately model how resources are used in delegated tasks. This model makes it possible to describe the time needed to carry out tasks and how resources are used in the process.

1.4 Publications

The work in this thesis is the result of joint work with Patrick Doherty, Fredrik Heintz and John-Jules Meyer. Parts of this thesis have been published in the following publications:

- [36] Doherty, P., Kvarnström, J., Heintz, F., Landén, D., Olsson, P-M: Research with Collaborative Unmanned Aircraft Systems. In Proceedings of the Dagstuhl Workshop on Cognitive Robotics, (2010).
- [37] 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).
- [69] 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).
- [30] Doherty, P., Heintz, F., Landén, D.: A Delegation-Based Collaborative Robotic Framework. In Proceedings of Collaborative Agents – REsearch and development (CARE), (2011).
- [31] Doherty, P., Heintz, F., Landén, D.: A Delegation-Based Architecture for Collaborative Robotics. In D. Weyns and M.-P. Gleizes (Eds.): Agent Oriented Software Engineering (AOSE) 2010, LNCS 6788, pp. 205-247, (2011).

1.5 Thesis Outline

The next chapter, Chapter 2, provides a background to the work in this thesis, and is previously published in [30, 31, 40]. The next three chapters deal with each of the three research questions stated in Section 1.2. A task specification format is described in Chapter 3. The work in this chapter was previously published in [30, 31, 37]. The inherent task allocation problem of delegation is defined and a solution is presented in Chapter 4. An earlier version of the work in this chapter was published in [69]. An implementation of the collaborative UAS is described in Chapter 5. Part of the work in this chapter was previously published in [31]. The use of the collaborative UAS is

exemplified with a number of emergency service scenarios in Chapter 6. Part of the work in this chapter was previously published in [31, 69]. The task allocation algorithm is evaluated in Chapter 7. The conclusions including future work are presented in Chapter 8.

Chapter 2

Background

The work presented in this chapter was previously introduced by Patrick Doherty and John-Jules Meyer in [36, 40]. The chapter will serve as a background and starting point for the work done in this thesis.

2.1 Delegation as a Speech Act

Delegation is central to the conceptual and architectural framework we propose. Consequently, it is important to formulate a formal specification of the delegation concept that can be grounded in an implementation of the collaborative UAS. As a starting point, in [10, 46], Falcone & Castelfranchi provide an illuminating, but informal discussion about delegation as a concept from a social perspective. Their approach to delegation builds on a Belief-Desire-Intention (BDI) model of agents, that is, agents having beliefs, goals, intentions, and plans [12]. 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 [40]. In order to formally characterize the operators used in the definition of the speech act, we use KARO [95] 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".¹ 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

¹Merriam-Webster free on-line dictionary. m-w.com

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. 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 are required.

For the formal specification, the definition provided by Falcone & Castelfranchi will be used. For the data-structure specification used in the implementation, Task Specification Trees will be defined in Chapter 3. Falcone & Castelfranchi define a task as a pair $\tau = (\alpha, \phi)$ consisting of a goal ϕ , and a plan α for that goal, or rather, a plan and the goal associated with that plan. Conceptually, a plan is a composite action. 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.

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 $\text{Delegate}(A, B, \tau)$ 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) $\text{Goal}_A(\phi)$
- (2) $\text{Bel}_A \text{Can}_B(\tau)$ (Note that this implies $\text{Bel}_A \text{Bel}_B(\text{Can}_B(\tau))$)
- (3) $\text{Bel}_A(\text{Dependent}(A, B, \tau))$
- (4) $\text{Bel}_B \text{Can}_B(\tau)$

Postconditions:

- (1) $\text{Goal}_B(\phi)$ and $\text{Bel}_B \text{Goal}_B(\phi)$
- (2) $\text{Committed}_B(\alpha)$ (also written $\text{Committed}_B(\tau)$)
- (3) $\text{Bel}_B \text{Goal}_A(\phi)$

- (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) $MutualBel_{AB}(\text{"the statements above"} \wedge SociallyCommitted(B, A, \tau))^2$

Informally speaking this expresses the following: the pre-conditions 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, this can only be successful if the receiver also believes it can do the task, which is expressed by (4).

The post-conditions 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 achievement 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 pre-conditions and other post-conditions 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.

²A discussion pertaining to the semantics of all non-KARO modal operators may be found in [40].

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 [13, 20, 42]. 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 [40] is to correlate the delegate speech act with plan generation rules in 2APL [19], which is an agent programming language with a formal semantics. In this thesis, 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 pre-conditions 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 goal ϕ implicit in τ . Clearly, if A cannot 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 (pre-condition 3: $Bel_A(Dependent(A, B, \alpha))$). Consequently, all pre-conditions 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 fun-

damental 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 ours. 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 (TSTs), 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 DisCSP. This allows us to apply existing DisCSP 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 pre-condition to the S-Delegate speech act is both formally and pragmatically grounded into the implementation.

2.2 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.

The UASTech UAV platform [41] is a slightly modified Yamaha RMAX helicopter (see Figure 2.1). It has a total length of 3.6 m (including main rotor) and is powered by a 21 hp two-stroke engine with a maximum takeoff weight of 95 kg. The helicopter has a built-in attitude sensor (YAS) and an

attitude control system (YACS). The helicopter platform has been used for spraying pesticides over crop fields and also in many research projects.

The hardware platform (the box on the left side of the RMAX in Figure 2.1) which has been developed and integrated with the UASTech RMAX contains three PC104 embedded computers. The primary flight control (PFC) has a wireless Ethernet bridge, a RTK GPS receiver, and several additional sensors including a barometric altitude sensor. The PFC is connected to the YAS (attitude sensors) and YACS (attitude controller), an image processing computer (IPC) and a computer for deliberative capabilities (DRC). The image processing system runs on the second PC104 embedded computer and includes a thermal camera and a color CCD camera mounted on a pan/tilt unit, a video transmitter and a recorder (miniDV). The deliberative/reactive (D/R) systems runs on the third PC104 embedded computer and executes all high-level autonomous functionality. Network communication between computers is physically realized with serial line RS232C and Ethernet.



Figure 2.1: The UASTech RMAX Platform.

Our RMAX helicopters use a CORBA-based distributed architecture [29]. For our experimentation with collaborative UAVs, 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 re-design the legacy system from scratch, the approach we take is to agentify the existing legacy system in a straightforward manner by adding an 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 [48]. The FIPA Abstract Architecture provides the basic components for the development of a multi-agent system. Our collaborative UAS implementation is based on the FIPA compliant Java Agent Development Framework (JADE) [7, 92] 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." [6].

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) [50].

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 Chapter 5 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.

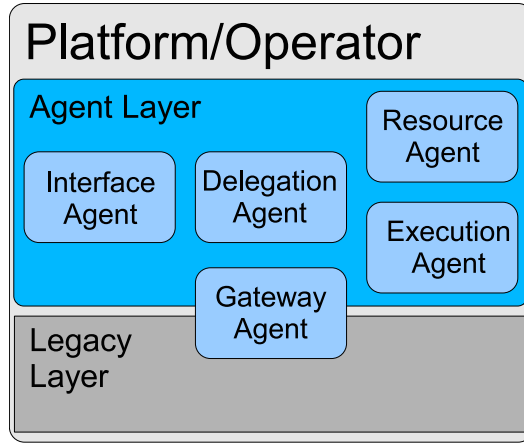


Figure 2.2: Overview of an agentified platform or ground control station.

The agent layer currently contains four agents:

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 Execution, 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 Execution 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 2.2 provides an overview of an agentified robotic or ground operator system.

The FIPA Abstract Architecture is extended (in Chapter 5) to support delegation and collaboration by defining an additional set of services and a set of related protocols. The interface agent, resource agent, execution agent and delegation agent will have an interface service, resource service,

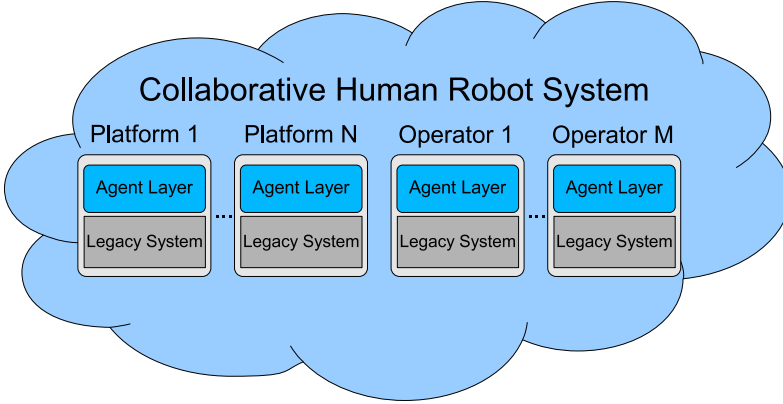


Figure 2.3: An overview of a collaborative human robot system.

execution service and delegation service associated with it, respectively, on each individual robotic or ground station platform. The gateway agent 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 legacy system and an agent layer as shown in Figure 2.3.

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. Speech acts are used 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 is implemented based on additional services and protocols is described in Chapter 5. Before doing this, the processes themselves are specified in Section 4.2. The formal characterization of tasks in the form of task specification trees is the topic of the next chapter.

Chapter 3

Task Specification Trees

“R1 - How can a task τ in the delegation theory be specified to make the realization of the delegation speech act possible?”

A task was previously defined abstractly as a tuple $(\alpha, \phi, cons)$ (on page 10) consisting of a composite action α , a goal ϕ and a set of constraints $cons$, associated with α . In this chapter, we introduce a formal task specification language which allows us to represent tasks as *Task Specification Trees* (TSTs).

We need a declarative representation of tasks that can cover both abstract goals and more concrete plan structures in the same formalism. A representation with procedural features would make the implementation straightforward. Both the declarative and the 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. 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 where they 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 [68] 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 other. 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.

3.1 TST Concepts

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. The nodes with child nodes (i.e. all nodes except goal nodes and elementary action nodes) have a special role in that they model the structure of a sub-task, containing all the child nodes and all the nodes below them. All nodes are directly executable except goal nodes which require some form of expansion or planning to generate a plan for achieving the goal. The plan is translated to a sub-TST and attached to the original TST. For example, the TST in Figure 3.1 contains a sequence node, a concurrent node and three elementary actions nodes.

Each node has a *node interface* containing a set of parameters, called *node parameters*, that can be specified for the node (see the text box attached to each node in Figure 3.1). 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*. For example, in Figure 3.1, the node parameters for node N_3 are P_3, T_{S_3}, T_{E_3} . 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

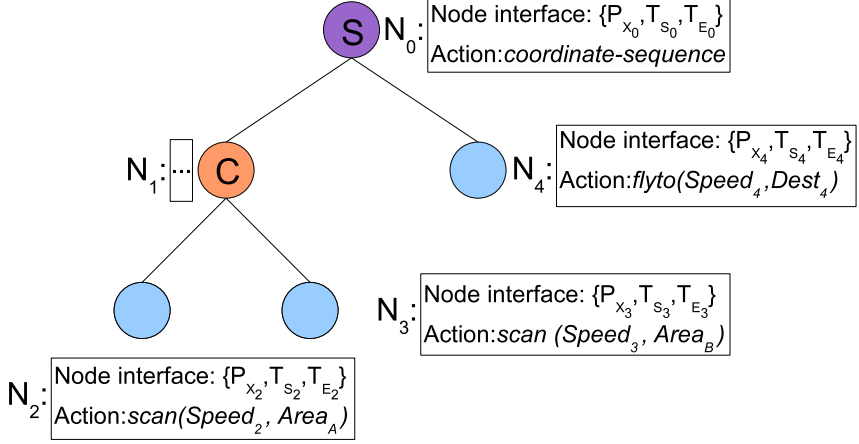


Figure 3.1: A TST with the nodes N_0 – N_4 . Nodes N_0 – N_1 are composite action nodes. The nodes N_2 , N_3 and N_4 are elementary action nodes. Each node specifies a task and has a node interface containing node parameters and the platform assignment variable. The parameters in the node interface corresponds to global variables that can be used in global constraints relating the nodes in the tree.

of autonomy of an agent.

In a TST where all nodes are allocated, each node has a corresponding executor object that, besides executing the task, updates information in the TST node during execution to display the progress. In this way, a TST node can be seen as a black-board where executor objects and the operator can read and update information.

A TST describes relations between composite and elementary actions, and how those relations and actions forms a complex task. A *compound task* is a task where all its parts should be allocated to the same platform. Related to our TSTs, a compound task is a connected part of a TST, where each node should be assigned to the same platform, which is specified by assigning a unifying value to the role parameter of each node in the compound task. A *decomposable task* consists of compound parts, and the parts can be assigned to different platforms.

3.2 TST Syntax

A TST can be represented both graphically as a hierarchical tree data structure and textually as a TST specification. The syntax of a TST specification has the following BNF:

TST ::= NAME ('(' VARS ')')? '='

$(\mathbf{with} \text{ VARS})? \text{ TASK } (\mathbf{where} \text{ CONS})?$
 $\text{TSTS} ::= \text{TST} \mid \text{TST } ';\text{' TSTS}$
 $\text{TASK} ::= \langle \text{elementary action} \rangle \mid \langle \text{goal} \rangle \mid$
 $\quad \mathbf{sequence} \text{ TSTS} \mid \mathbf{concurrent} \text{ TSTS} \mid$
 $\quad \mathbf{while} \langle \text{cond} \rangle \text{ TST} \mid \mathbf{if} \langle \text{cond} \rangle \mathbf{then} \text{ TST} \mathbf{else} \text{ TST}$
 $\text{VAR} ::= \langle \text{var name} \rangle \mid \langle \text{var name} \rangle \text{'.'} \langle \text{var name} \rangle$
 $\text{VARS} ::= \text{VAR} \mid \text{VAR } ';\text{' VARS}$
 $\text{CONSTRAINT} ::= \langle \text{constraint} \rangle$
 $\text{CONS} ::= \text{CONSTRAINT} \mid \text{CONSTRAINT} \mathbf{and} \text{ CONS}$
 $\text{ARG} ::= \text{VAR} \mid \langle \text{value} \rangle$
 $\text{ARGS} ::= \text{ARG} \mid \text{ARG } ';\text{' ARGS}$
 $\text{NAME} ::= \langle \text{node name} \rangle$

Where $\langle \text{elementary action} \rangle$ is an elementary action $name(p_0, \dots, p_N)$, $\langle \text{goal} \rangle$ is a goal $name(p_0, \dots, p_N)$, p_0, \dots, p_N are parameters, and $\langle \text{cond} \rangle$ is a FIPA ACL query message requesting the value of a boolean expression. $\langle \text{constraint} \rangle$ is a general constraint involving the variables in the same scope as the constraint. $\langle \text{name} \rangle$ is any alpha numerical combination, with the purpose of naming a task. The first *VARS* in the TST rule denotes node parameters, the second *VARS* denotes additional variables used in the constraint context for the top-node of the TST and *CONS* denotes the constraints associated with this node.

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 set of variables and their scope 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 Area_A and Area_B , and then fly to Dest_4 (Figure 3.2). A TST describing this mission is shown in Figure 3.1. Nodes N_0 and N_1 are composite action nodes, sequential and concurrent, respectively. Nodes N_2 , N_3 and N_4 are elementary action nodes. Each node specifies a task and has a node interface containing node parameters and a platform assignment 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 3.1. 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:

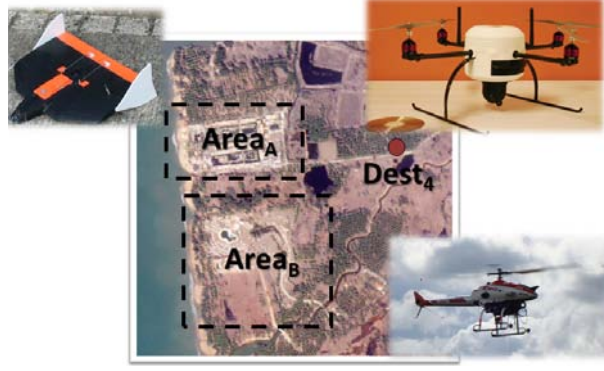


Figure 3.2: Example mission of first scanning $Area_A$ and $Area_B$, and then flying to $Dest_4$.

$$\begin{aligned}
 \tau_0(T_{S_0}, T_{E_0}) = & \\
 \text{with } T_{S_1}, T_{E_1}, T_{S_4}, T_{E_4} \text{ sequence} & \\
 \tau_1(T_{S_1}, T_{E_1}) = & \\
 \text{with } T_{S_2}, T_{E_2}, T_{S_3}, T_{E_3} \text{ concurrent} & \\
 \tau_2(T_{S_2}, T_{E_2}) = \text{scan}(T_{S_2}, T_{E_2}, \text{Speed}_2, \text{Area}_A); & \\
 \tau_3(T_{S_3}, T_{E_3}) = \text{scan}(T_{S_3}, T_{E_3}, \text{Speed}_3, \text{Area}_B) & \\
 \text{where } \text{cons}_{\tau_1}; & \\
 \tau_4(T_{S_4}, T_{E_4}) = \text{flyto}(T_{S_4}, T_{E_4}, \text{Speed}_4, \text{Dest}_4) & \\
 \text{where } \text{cons}_{\tau_0} & \\
 \text{cons}_{\tau_0} = \{T_{S_0} \leq T_{S_1} \wedge T_{S_1} \leq T_{E_1} \wedge T_{E_1} \leq T_{S_4} \wedge T_{S_4} \leq T_{E_4} \wedge T_{E_4} \leq T_{E_0}\} & \\
 \text{cons}_{\tau_1} = \{T_{S_1} \leq T_{S_2} \wedge T_{S_2} \leq T_{E_2} \wedge T_{E_2} \leq T_{E_1} \wedge T_{S_1} \leq T_{S_3} \wedge T_{S_3} \leq T_{E_3} \wedge & \\
 T_{E_3} \leq T_{E_1}\} &
 \end{aligned}$$

3.3 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. 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 2.1, 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 pre-condition 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 , (pre-condition 2, on page 10): $Bel_A Can_B(\tau)$. 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 , (pre-condition 4, on page 10): $Can_B(\tau)$. In order to directly couple the semantic characterization of the S-Delegate speech act to the semantic characterization of TSTs, we will assume that a task $\tau = (\alpha, \phi)$ in the speech act characterization corresponds to a TST. Additionally, the TST semantics will be characterized in terms of a Can predicate with additional parameters to incorporate constraints.

In this case, the Can predicate is extended to include as arguments a list $[p_1, \dots, p_k]$ denoting all node parameters in the node interface together with other parameters provided in the (**with** $VARS$) construct¹ and an argument for an additional set $cons$ provided in the (**where** $CONS$) construct². 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 $Can(B, \tau, [t_s, t_e], cons)$ then asserts that an agent B has the capabilities and resources for achieving task τ if $cons$ together with the node constraints for τ are consistent. The temporal variables t_s and t_e associated

¹For reasons of clarity, we only list the node parameters for the start and end times for a task, $[t_s, t_e]$.

²For pedagogical expediency, we can assume that there is a constraint language, which is reified in the logic and is used in the $CONS$ constructs.

with the task τ are parts of the node interface which may also contain other variables that 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 TSTs and their logical semantics.

The basic structure of a task specification tree is:

$$TST ::= NAME('(' VAR_1 ')')? = '(\mathbf{with} \text{ } VAR_2)? TASK \\ (\mathbf{where} \text{ } CONS)?$$

where VAR_1 denotes node parameters, VAR_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

$$Can(agent_1, TASK, VAR_1 \cup VAR_2, CONS) \quad (3.1)$$

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

$$\exists agent_2 Delegate(agent_1, agent_2, TASK, VAR_1 \cup VAR_2, CONS) \quad (3.2)$$

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 $S - Delegate(agent_1, agent_2, TASK)$ speech act semantics, we know that if delegation is successful, then as one of the post-conditions of

the speech act, $agent_2$ can in fact achieve $TASK$ (assuming no additional contingencies):

$$\begin{aligned} Delegate(agent_1, agent_2, TASK, VARS_1 \cup VARS_2, CONS) \rightarrow \\ Can(agent_2, TASK, VARS_1 \cup VARS_2, CONS) \end{aligned} \quad (3.3)$$

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. These are in turn compiled into an (interdependent) set of CSPs that are checked for satisfaction using distributed constraint solving algorithms.

Sequence Node

The child nodes of a *sequence node* should be executed in sequence (from left to right) during the execution time of the sequence node. A sequence node is shown in Figure 3.3.

Semantics of a Sequence Node

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

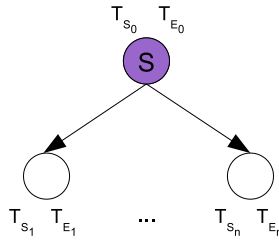


Figure 3.3: A sequence node.

Concurrent Node

In a concurrent node each child node should be executed during the time interval of the concurrent node (see Figure 3.4). Concurrency in this case

does not necessarily mean that the execution of the child nodes *must* be concurrent, they can be in sequence, but concurrency is *allowed*, so $S \subseteq C$.³ Concurrent nodes are used to make more efficient use of the platforms, by allowing for executing tasks in parallel. If two tasks must be executed in parallel, this can be assured with a user constraint that demands the tasks to be assigned to different platforms.

Semantics of a Concurrent Node

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

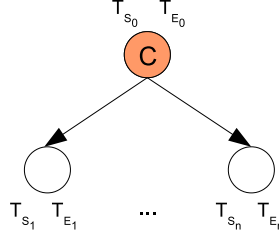


Figure 3.4: A concurrent node.

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

Selector Node

- Compared to a sequence or concurrent node, only one of the *selector node's* children will be executed, which one is determined by a test condition in the selector node. The child node should be executed

³Comparing S and C to Allen's interval algebra [3], C can cover all 13 relations, and S cover 4 of the relations (b,m,bi,mi).

during the time interval of the selector node. A selector node is used to postpone a choice which cannot be known when the TST is specified. When expanded at runtime, the net result can be any of the node types.

Loop Node

- A *loop node* will add a child node for each iteration the loop condition allows. In this way the loop node works as a sequence node, but with an increasing number of child nodes which are dynamically added. Loop nodes are similar to selector nodes, they describe additions to the TST that cannot 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 cannot 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. A goal node is shown in Figure 3.5.
- $$\begin{aligned} Can(B, Goal(\phi), [t_s, t_e, \dots], cons) &\leftrightarrow \\ \exists \alpha (GeneratePlan(B, \alpha, \phi, [t_s, t_e, \dots], cons) & \\ \wedge Can(B, \alpha, [t_s, t_e, \dots], cons)) & \\ \wedge consistent(cons) & \end{aligned}$$

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.



Figure 3.5: A goal node.

Elementary Action Node

- An *elementary action node* is a leaf node that specifies a domain-dependent action. The semantics for Can for an elementary action is platform dependent. Figure 3.6 shows an example of an elementary action node.
- $$\begin{aligned} Can(B, \tau, [t_s, t_e, \dots], cons, \dots) &\leftrightarrow \\ Capabilities(B, \tau, [t_s, t_e, \dots], cons) &\wedge Resources(B, \tau, [t_s, t_e, \dots], cons) \\ \wedge consistent(cons) & \end{aligned}$$

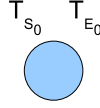


Figure 3.6: An elementary action node.

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

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

$$t_e = t_s + \frac{\text{distance}(\text{pos}(t_s, P_1), \text{dest})}{\text{speed}}$$

$$\text{Speed}_{Min} \leq \text{speed} \leq \text{Speed}_{Max}$$

The corresponding constraint network is in Figure 3.7. The general pattern for this conjunction is:

$$t_e = t_s + F, C_1, \dots, C_N, \text{ where}$$

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

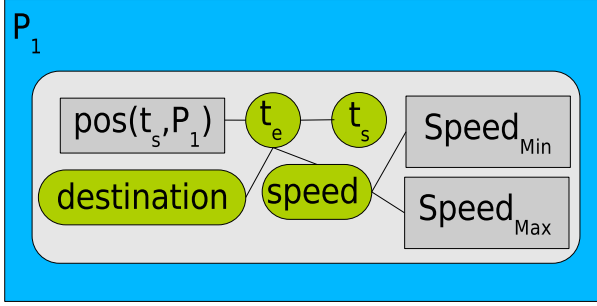


Figure 3.7: Illustrating the constraint network associated with the task *flyto*. Nodes with rounded edges are parameters of the resource predicate for this task. Edges and variables together describes constraints, and an edge between two variables denotes that they are related. The capability resides on platform P_1 .

3.4 Related Work

Related work for this chapter are task/mission specification languages and distributed planning systems. Two important task specification languages are the Configuration Description Language (CDL) [73], used in *MissionLab*, and the Task Description Language (TDL) [87].

CDL has a recursive composition of configurations, similar to our TST task structure. In CDL, a behavior and a set of parameters creates an agent. Agents can be composed into larger entities, called assemblages that work as macro-agents. Assemblages can in turn be part of larger assemblages. CDL has been used as the basis for *MissionLab*, a tool for mission specification using case based reasoning. Task-allocation is done according to the market-based paradigm with contract-nets. Task allocation can be done together with mission specification, or at run time [93].

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

Both CDL and TDL are similar to TST, but with the difference that the specification of a TST is not pre-compiled and therefore allows for more dynamic handling of tasks in the case of changing circumstances. The spec-

ification remains through the stages of task-allocation (delegation) and execution. Each node in a TST has parameter values which are restricted by constraints. Each node has also an executor object (for each platform) that can be instantiated with the parameters determined in the task allocation stage. Since we have this separation between specification and execution of tasks connected as a constraint problem of the node parameters and platform assignments, we can go back and forth from the task-allocation and execution stage. Such transitions are needed when an error is detected, or when the mission is changed with mixed-initiative input. The loose coupling between specification and execution is needed for supporting adjustable autonomy and mixed-initiative.

One of the requirements of the task specification format is to allow the framework for collaborative robotics to work both with central and distributed planning. In distributed planning, many agents are involved in the planning. The main difference from centralized planning is that the agents combine their efforts and plans, instead of one agent doing all the planning. Examples are RETSINA [89] and STEAM [91]. In distributed planning, the coordination structure emerges from the agents' combined work, whereas in our framework for collaborative robotics, the TST expresses the coordination structure.

Another important distributed planning system is partial global planning (PGP) [44, 45]. The PGP framework is a system similar to ours, but also different. A major difference is that PGP is “worth-oriented”, i.e., the system is used to optimize task allocation where a certain number of all tasks are allocated. There is no black or white outcome to the allocation, compared to our system that either fails or succeeds. The TÆMS data structure that describes tasks in PGP is similar to our TSTs. Both task representations can be used to express task trees, but the types of available structure nodes differ. TSTs have sequence, concurrent, loop and select nodes, whereas the corresponding structure nodes in TÆMS are AND/OR nodes. PGP was originally developed for The Distributed Vehicle Monitoring Testbed (DVMT) [72].

Chapter 4

Allocating Tasks in a TST to Platforms

“How can the preconditions of the delegation speech act be assured, so that a task τ specified according to R1 can be delegated?”

Given a TST representing a complex task, an important problem is to find a set of platforms that can execute this complex task according to the TST specification. The problem is to allocate tasks to platforms and assign values to parameters so 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 chapter. Each node in a TST specifies the capabilities required of platforms to be allocated to the node. A platform that can be assigned a task in a TST is called a *candidate* and a set of candidates is called a *candidate group*. The capabilities of a platform are usually fixed while the available resources keep 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. 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 task. Figure 4.1 shows an abstract representation of the constraints for the `scan` action for platform P_1 .

When a platform is assigned an elementary action node in a TST, the constraints associated with that action are instantiated and added to the constraint store of the platform. The 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 4.2 shows the constraint network after allocating node N_2 from the TST in Figure 3.1 (on page 20) to platform P_1 .

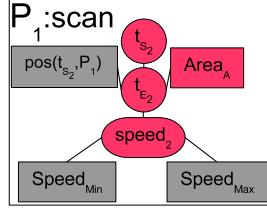


Figure 4.1: The parameterized platform constraints for the `scan` action. The red/dark variables represent node parameters in the node interface. The gray variables represent local variables associated with the platform P_1 's constraint model for the `scan` action. These are connected through dependencies.

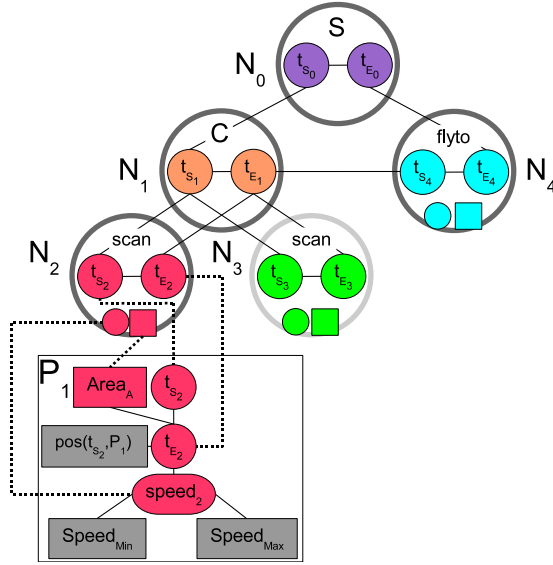


Figure 4.2: The combined constraint problem after allocating node N_2 to platform P_1 .

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 4.3 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 prob-

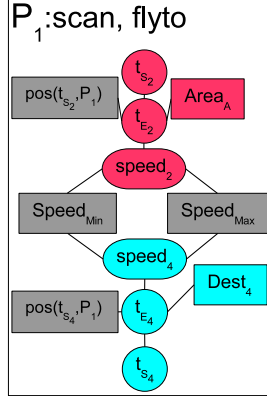


Figure 4.3: The parameter constraints of platform P_1 when allocated node N_2 and N_4 .

lem 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. The interval variables of 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 algorithm such as the Asynchronous Weak Commitment Search (AWCS) algorithm [101] or the The Distributed Breakout Algorithm [65].

4.1 Deriving the Constraint Problem of a TST

The constraint problem for a TST is derived by recursively reducing the *Can* predicate statements associated with each task node to formally equivalent expressions, beginning with the top-node τ_0 and continuing until the logical statements reduce to a constraint network. Below, we show the reduction of the complex task α_0 represented by the TST in Figure 3.1 when there are three platforms, P_0 , P_1 and P_2 , with the appropriate capabilities, P_0 has been delegated the composite action α_0 and P_0 has recursively delegated α_2 and α_4 to P_1 and α_3 to P_2 while keeping α_1 . α_i is the composite action described by the TST rooted in node τ_i .

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

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

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

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

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

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

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

The remaining tasks are elementary actions and consequently the definition of *Can* for these are platform-dependent. When a platform is assigned an elementary action node the resource constraints for that action are added to the local constraint store. The local constraints are connected to the distributed constraint problem through the node parameters of the assigned node. All remaining *Can* predicates in the recursion are replaced with constraint sub-networks associated with specific platforms as shown in Figure 4.4. To check that the resulting distributed constraint problem is consistent we use local CSP solvers together with a DisCSP solver.

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. This is under the assumption 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

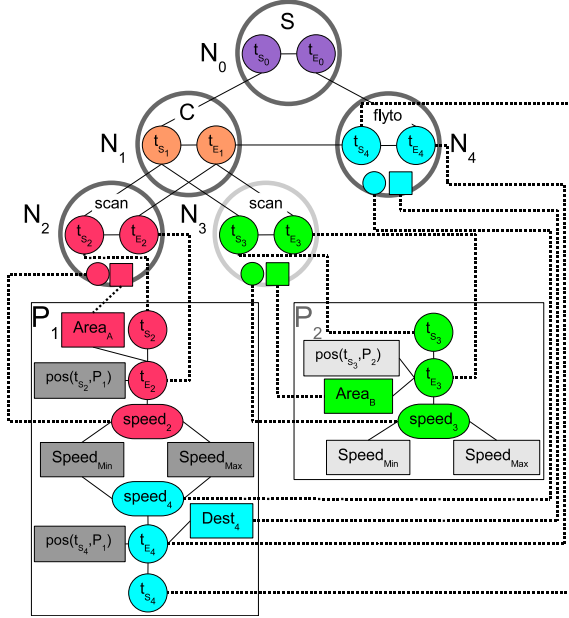


Figure 4.4: The completely allocated and reduced TST showing the interaction between the TST constraints and the platform dependent constraints.

a distributed constraint network, which is in effect a formal representation. This results in a real-world grounding of the semantics of the Delegation speech act via the *Can* predicate.

4.2 The Delegation Process

Now that the S-Delegate speech act, the task specification tree representation, and the formal relation between them have 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 S-Delegate($A, B, \tau = (\alpha, \phi, cons)$) 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 cannot 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. 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 finds 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 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 depends on the temporal constraints used in the TST specification. However, commitments to the mission 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 cannot 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 satisfies the pre-conditions of the S-Delegate speech act for all delegations in the TST. The approval corresponds to actually executing the speech

act. 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.

The most important part of the Delegation Process is to find a platform that satisfies the pre-conditions of the S-Delegate speech act. This is equivalent to finding a platform, which is able to achieve the task either itself or 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 [57, 58, 71, 94, 105, 106]. 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. The characteristics of the task allocation problem will be described in the following section.

4.3 Classifying the TST Allocation Problem

The task allocation problem can be traced back to the Optimal Assignment Problem (OAP) [54]. In OAP, a number of workers m should be assigned to a number of jobs n , with only one worker per job. The worker - job combinations have different utilities, depending on how well suited each worker is for its job. The problem is to find the optimal allocation.

The following assumptions are made in OAP: A worker can only have one job at a time. A job needs only one worker. The assignment is instantaneous. There are no more jobs to take care of later or plan for taking care of later. The jobs are atomic in the sense that they do not relate to each other. Both the utilities and jobs are independent. Assigning one worker to a job does not change other workers' utilities for their jobs. One job does not have to be assigned before another job is assigned. One can see that the problem has three dimensions / parameters that affects its difficulty: worker capacity, job complexity and allocation horizon. Changing one or more of these results in a harder assignment problem.

The OAP problem and its derived variants have been used in the research field on multi-agent systems as a basic formulation for how to allocate tasks to agents. Task allocation is a fundamental problem for achieving coordination in a multi-agent system. The multi-robot task allocation problem is in its simplest form equal to the OAP. The problem also has seven harder variants [57]. In his thesis, Gerkey [57] modifies the following parameters of the problem. Single-task robots (ST) vs. multi-task robots (MT), i.e. can a robot execute one or more tasks at the same time (robot capacity).

The second parameter is single-robot tasks (SR) vs multi-robot tasks (MR), (task complexity). SR means one task needs at most one robot, for MR a task may need more than one robot. The final parameter is instantaneous assignment (IA) vs. time-extended assignment (TE). In IA there is no information to plan for further allocations, instead an allocation can be done directly. For TE there is more information such as information about all tasks that need to be assigned or a model over how tasks are expected to arrive in time.

If we compare our problem to Gerkey's MRTA problem classes, it fits into the MT-SR-TE class. Our problem is MT because each platform can do more than one task at a time. This is because a platform's commitments are only restricted by the platform's resources. The problem is SR, because only one platform is needed per task. Here we view the tasks as the nodes in the TST, if we regard the entire TST as a task, then it is a MR problem. Our problem is also TE because we have a model (TST + constraints) over all tasks that should be assigned and how they relate to each other.

4.3.1 Basic Task Allocation Problem Properties

A task allocation problem is classified by its robot capacity, task complexity and allocation horizon. In this section, we describe more carefully those aspects of our problem.

Robot Capacity

In OAP, a worker can only do one task at a time. Most task allocation algorithms make this assumption. In our case we do not want to unnecessarily restrict the platform's capabilities partly because we often only have a few platforms. A platform can do more than one task at a time, which is modeled by the use of resources. It is the resources that are the limiting factor rather than the platforms. Modeling the problem in this way makes the problem more complex (MT) but allows for more efficient use of the platforms' resources.

Task Complexity

Task complexity describes the number of platforms needed to allocate a single task. We view a TST as a collection of tasks. With this view we have single-robot tasks (ST) because only one robot is needed for each node in a TST.

Allocation Horizon

A TST may describe an undefined schedule, in which case it should be made concrete during task allocation. During the task allocation, constraints derived from the resource predicates are connected to the extended constraint

network on a node per node basis. The complete constraint problem is thus not known at the beginning of the allocation. Instead, more information about the task allocation problem becomes available during the allocation, i.e. we have a time-extended allocation (TE).

4.3.2 Complex Task Allocation Problem Properties

In his thesis, Gerkey points out that the classification does not really apply to task allocation problems where the tasks have dependencies between each other such as interrelated utilities or constraints [57]. Our TSTs have at least task constraints and sometimes also interrelated utilities. A TST is held together by tree constraints, e.g. it has constrained tasks. In a TST there can also exist dependencies between allocations and the utility values for those allocations, caused by resource usage, the updating of the platform's position during sequential allocations of the same platform, etc.

Gerkey's classification is still very useful, since it shows how our problem relates to other MRTA problems. We extend the classification model with the parameters: utility dependencies, unrelated utilities (UU) vs interrelated utilities (IU), and task dependencies, independent tasks (IT) vs constrained tasks (CT), to cover our problem.

Another aspect of the task allocation problem is *who* is making the task allocation. In the OAP, there is no allocator worker responsible for solving the OAP, instead the problem is given and it should be solved by an external allocator. The allocation itself is not seen as work to be done by one of the workers, instead it is an external process. If the allocation is done by one of the workers, we have a much different situation. Both the tasks and the task allocations are tasks for the multi-agent system. Making the task allocation a task for the agents themselves is captured by the delegation concept. A delegation is a task allocation involving at least one allocator agent and one allocated agent. We call the parameter that captures this concept the allocation view and it can take the value external allocation view (EV) or internal allocation view (IV). Whether IV is harder than EV depends on how much information on the task allocation problem the allocator can retrieve. In the EV it is more or less assumed that all information can be given to the external allocator. This does not have to be the case for IV, and if not, then task allocation of the IV type is more constrained.

Related to, but not included in the task allocation problem, is the task allocation environment parameter. A task allocation environment can be even harder than TE, if the task-allocator does not only have to take into account future tasks to allocate, but also that the task allocation problem can change unexpectedly. Changes could include addition or removal of agents and changes to variable domains. In such an environment we have the additional problem of task re-allocation. We call this parameter allocation environment and it can take the value static allocation environment (SA) or dynamic allocation environment (DA).

In the following section we describe the utility-dependence, task-dependency, allocation view and allocation environment in more detail.

Task-Dependencies

Either there are relations between the tasks in the task allocation problem (constrained tasks) or there are not (independent tasks). The former makes the task allocation problem harder, but is also needed in domains where the tasks to be allocated are parts of a larger task structure, such as our TSTs.

Utility-Dependencies

A common method to determine the value of an allocation, so that it can be compared with other allocations is to use utility values. Allocating a platform to a task derives a value describing how well the allocation fits into the entire allocation problem. A possible utility function in our case is the marginal cost of allocating a task to a platform (a high utility is in this case a low cost of allocation). A task allocation problem where the utility value of an allocation does not depend on or influence the utility values of other allocations, has unrelated utilities. The opposite is interrelated utilities.

For example, if platform P_0 is allocated to $Task_A$, then there is an allocation utility $U_{0,A}$. If platform P_0 is allocated to $Task_B$, then there is an allocation utility $U_{0,B}$. If P_0 is allocated to both $Task_A$ and $Task_B$, there is a third allocation utility $U_{0,AB}$, and $U_{0,A} + U_{0,B}$ may not necessarily be equal to $U_{0,AB}$. For cases where independence does not hold, utility values for all such combinations must be returned to provide a correct view of the situation.

Interrelated utilities arises under certain circumstances when the same platform is allocated more than once in the solution to the task allocation problem. For instance, allocating a platform to a task with the meaning of moving the platform to a certain position will have different utilities depending on what other tasks the platform previously has taken on (i.e. where the platform was positioned before). Interrelated utilities can also occur when a resource is used by several allocated tasks. Our missions typically involve less platforms than TST nodes and the resources used are carried on the platforms and therefore severely limited. In our missions, interrelated utilities will occur frequently and there is no reasonable measure we could take to assure that interrelated utilities cannot occur. To make that happen we would have to restrict a platform to only appear once in a TST (only taking into account the elementary tasks), which is a severe limitation in our problem domain. The missions we could express would not be realistic.

Interrelated utilities is the most difficult feature of our task allocation problem. Task allocation problems with this property cannot be allocated with a divide and conquer approach. Such task allocation problems are similar to problems solved with combinatorial auctions [21].

Task Allocation View

Both tasks and allocation of tasks are done by the platforms themselves in our task allocation problem. An internal task allocation is expressed as a delegation. In order to carry out a delegation from platform A to platform B of task τ , B is required to do a task allocation of τ for A . In our case the delegator (A) is a platform or the operator, and the contractor (B) is a platform. This view can be compared to OAP, where task allocation is an external process and none of the workers is responsible for solving the allocation problem.

In our case, the internal allocation view together with the limited knowledge about other platforms, makes task allocation a significantly harder problem.

Task Allocation Environment

The task allocation environment can be either static (SA) or dynamic (DA). In the dynamic case, the presumptions of the task allocation problem can change, whereas in the SA case they do not. In the DA case, the task allocator must be prepared to change or update the allocation when needed. For example, execution of tasks might require more resources than expected, or an allocated platform must leave the collaborate UAS for some reason.

4.3.3 Summary

The TST allocation problem has the internal task allocation view (IV) at the same time as the platforms have limited knowledge about each others capabilities. The TST allocation problem has strong similarities with the problems studied in the DisCSP field. A DisCSP is a distributed constraint satisfaction problem, where a number of variables, connected by constraints should be given values so that all constraints are consistent. The distributed part means that each variable is owned by an agent. In the original constraint satisfaction problem there are no agents and no ownership relations exists. The agent concept, and the constraint-based representation of TSTs makes the task allocation problem suitable for a DisCSP approach. A TST is in the form of a task tree and describes a schedule (TC). The problem has time extended allocation (TE), which means that a platform can be assigned more than once in a TST. Goal nodes also add to the TE aspect because the extent of the task allocation problem is not known before the goal nodes are expanded.

4.4 The TST Allocation Problem Definition

In this and the previous chapter we described aspects of our task allocation problem. We noted that our version of the task allocation problem can be

stated as a distributed constraint satisfaction problem. In this section we define the problem formally.

The Task Allocation Problem: *Given a TST with nodes N_0, \dots, N_n and platform P_0, \dots, P_m , for every node $n_i \in N_0, \dots, N_n$ allocate it to a platform $p_i \in P_0, \dots, P_m$, so that all constraints in the constraint network formed by the TST and the allocated platform's resource constraints $(P_i, N_0), \dots, (P_j, N_n)$ are satisfied.*

In the general case, the task allocation problem can be viewed as a conditional distributed constraint satisfaction problem. A conditional constraint satisfaction problem is a constraint problem whose content (constraints and variables) depends on a number of conditional variables. Different choices for those variables, creates different constraint problems. A conditional distributed constraint satisfaction problem is a distributed version of the conditional constraint satisfaction problem. In our case, the DisCSP is formed depending on the choices of platforms to TST nodes that are made. A node in a TST can be allocated to all platforms that have the required capability and sufficient resources. The platforms are heterogeneous, meaning that the DisCSP will be different depending on which platform that is chosen. The task allocation problem has two layers, a higher conditional layer that determines the content of the formed DisCSP, and a lower layer that determine consistency in the DisCSP. The two layers interact, because only certain choices of platforms to TST nodes result in DisCSPs that are consistent.

4.5 A TST Allocation Algorithm

This section presents a heuristic search algorithm for allocating a fully expanded TST to a set of platforms. A successful allocation assigns each node to a platform and assigns values to all 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 DisCSP 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 in which platforms are tested. The heuristic function is constructed by auctioning out each 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 back-jumping to find the latest partial allocation which has a consistent allocation of the current node. This preserves soundness, as only partial allocations that are guaranteed to violate the constraints are skipped. The algorithm is complete because with backtracking or backjumping the algorithm will test all possible unique configurations.

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 (add the constraints to the platform's DisCSP Node).
2. If the allocation is inconsistent (check by running the DisCSP algorithm) 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 back-jump (see Section 4.5.1).
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. That is taking into account all previous tasks the platform has been allocated, and calculating 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 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.

4.5.1 Distributed Backjumping

A dead-end is reached when a platform is trying to allocate a node N_k but there is no consistent allocation. The platform must then undo previous allocations until a partial allocation is found where N_k can be allocated. This is the backjump point where the backtracking will start.

More formally, the current partial allocation can be seen as the assignment A_1, \dots, A_k of platforms to each node in the sequence N_1, \dots, N_k . Instead of backtracking over the next allocation for N_1, \dots, N_{k-1} as in normal chronological backtracking, the algorithm finds the node N_j with the highest index j such that a consistent allocation for N_k can be found given the partial allocation A_1, \dots, A_j . The node N_j is called the *backjump point*. Using the fact that N_k must be allocated we can skip all partial allocations of N_{j+1}, \dots, N_{k-1} that do not lead to a consistent allocation of N_k .

The backjump point is found by disconnecting parts of the DisCSP network and then trying all possible allocations for N_k . When the node can be allocated with parts of the network disconnected, it means that the backjump point resides in the disconnected part of the network. The localization of the backjump point continues in the previously disconnected network by recursively dividing it into smaller parts. Each new partial allocation is checked by trying to extend it with an allocation of N_k . Since the task allocation process is distributed the backjump process must also be distributed.

To describe the algorithm, the following definitions are used. A platform is *in charge* of all nodes below a node it has been allocated. The node that could not be allocated is called the *failure point*. The platform trying to find an allocation for the failure point is called *failure point allocator*. *Disconnecting* a network means temporarily removing the variables in the network from the DisCSP, which is equivalent to removing the corresponding allocations. When a platform disconnects networks and checks for consistency, an

activation message is sent from the platform to the failure point allocator. The failure point allocator will then try applicable platforms for the failure point until a new allocation is found or none exists. The failure point allocator sends an *allocation succeeded* if an allocation is found, otherwise an *allocation failed* message.

The procedures **Search Upwards** and **Search Downwards** are used to find the backjump point, beginning with the *Search Upwards* procedure. Two different search procedures are necessary since we first have to find which platform is in control over the backjump point, and then have to find the actual backjump point.

Search Upwards

1. Disconnect all child branches (that have been allocated) except the branch that contains the failure point. Signal the failure point allocator to start finding an allocation for the failure point.
 - (a) If the failed node can be allocated, reconnect all child branches and start searching for the backjump point by calling **Search Downwards**.
 - (b) If no allocation can be found, then do a **Search Upwards** starting from the parent of the node. If the node has no parent, then there is no allocation.

Search Downwards

1. Disconnect child branches one at the time in the reverse order they were allocated and check the consistency. If the network is consistent, then the backjump point is in that branch.
2. When a branch containing the backjump point is located, check if the child branch has a composite action node as the top-node. In that case, do a recursive **Search Downwards** starting at that node. Otherwise, the backjump point has been found.

Example

Consider the TST in Figure 3.1 (on page 20). Assume that the operator has put a time bound of 30 minutes on the TST. **AllocateTST** works with the top-node N_0 on platform P_0 . The candidates for N_0 are platform P_0 and P_1 . P_0 auctions out the node N_0 to the platforms, and each platform returns their bid on the node. Let us assume the order is P_1 , P_0 based on the bids. Platform P_0 will now try to allocate the node to platform P_1 . If it does not work, it will try with itself instead, before it gives up.

When platform P_1 is chosen to be allocated to node N_0 , the TST with top-node N_0 is delegated from the delegator P_0 to the contractor P_1 . P_1 allocates itself to N_0 . The constraint network is formed on P_1 and it is

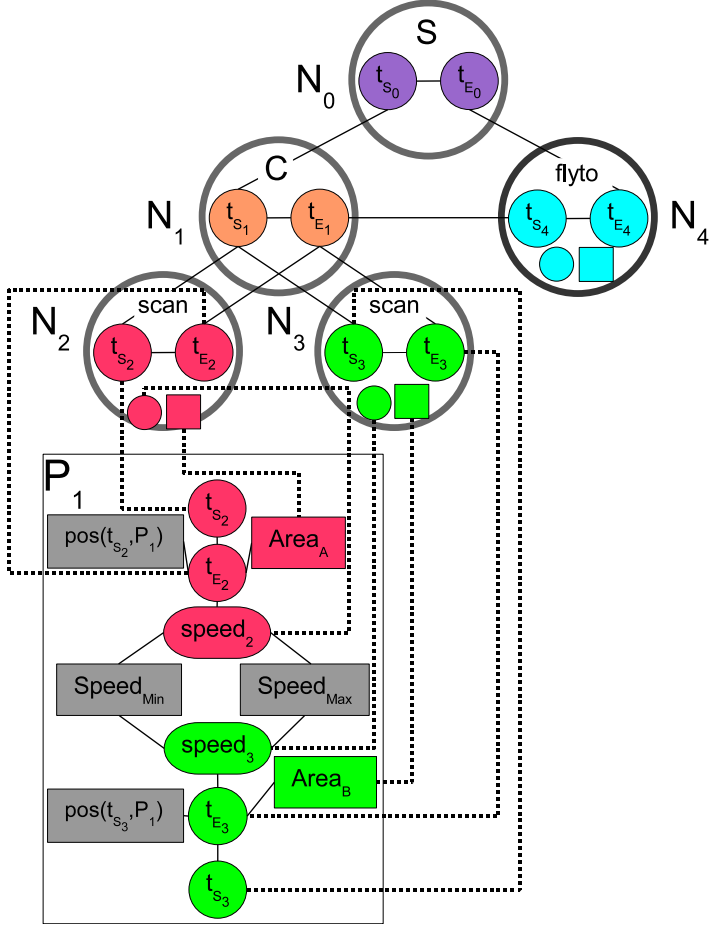


Figure 4.5: Nodes N_0 – N_3 are allocated to platform P_1 , node N_4 remains.

consistent. Since the node N_0 has two child nodes, P_1 has to allocate them before concluding the delegation from P_0 . For each of the child nodes, N_1 and N_4 , P_1 will now work as the delegator in the same way as P_0 worked as delegator for N_0 .

Assume that all nodes except N_4 have been allocated (see Figure 4.5). Platform P_1 has two candidates for N_4 , itself and P_0 . P_1 cannot be allocated to N_4 within the time bound. P_0 is tested instead, with the same result. P_0 must now backtrack to find the point where it can make a new choice and from there on allocate the remaining nodes in the TST. Backtracking to node N_3 and choosing the next candidate in the list for that allocation makes it possible to allocate N_4 within the time bound, see Figure 4.6.

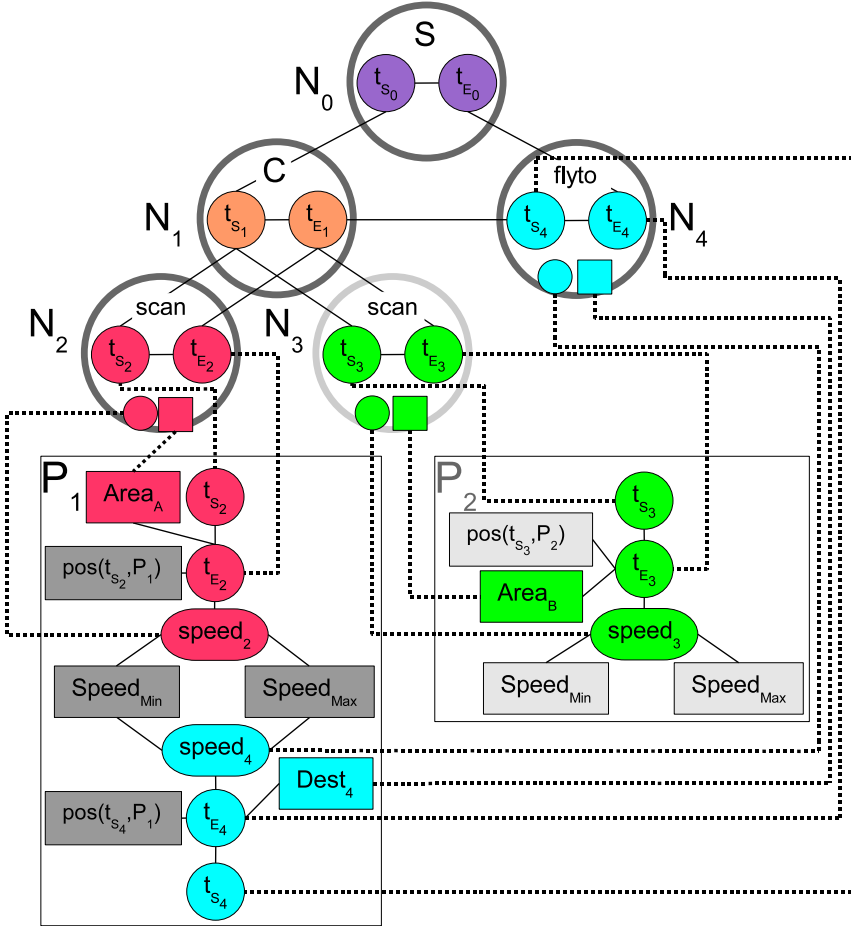


Figure 4.6: The complete allocation after backtracking.

4.6 Alternative Approaches

In this section we consider other approaches to the task allocation problem.

4.6.1 An Alternative DisCSP Approach

An alternative DisCSP approach to the task allocation problem would simply involve all possible candidate platforms for all tasks the platforms are capable of. The resulting DisCSP would be much larger than the DisCSP solved by AllocateTST, but on the other hand it will not need to add and remove constraints. This approach is explored in Chapter 7, in Section 7.6.

4.6.2 An Integer or Linear Programming Approach

The task allocation problem is similar to the winner determination problem for combinatorial auctions. This problem has been solved with Integer Programming and Linear Programming [21], which makes those approaches interesting for our problem. One approach could be to replace the local CSP solver with a dedicated Integer Programming and Linear Programming solver and run the AWCS algorithm on top. Another possibility is to use a complete distributed Linear or Integer programming approach [5] instead of a DisCSP algorithm. Evaluating and comparing those approaches to our chosen approach is left for future work.

4.7 Related Work

Task allocation algorithms suitable for a comparison with AllocateTST are algorithms that can handle more of the hard features in the task allocation classification (such as time extended allocation (TE), constrained task (CT) and interrelated utilities (IU)). Most task allocation algorithms can only handle problems with independent tasks (IT) for task dependencies and unrelated utilities (UU) for utility dependencies. But there are exceptions, such as Zlot's and Gerkey's task allocation algorithms described in [57, 106]. Zlot's task allocation algorithm [106] can handle complex tasks, which in this case is a form of constrained task trees. The task tree consists of AND/OR nodes and elementary nodes, similar to our TSTs with composite and elementary action nodes. The main difference between complex tasks and our TSTs is the interrelated utilities. None of the referred task allocation algorithms can solve problems with this property. Trying to allocate a TST using the task allocation algorithm described in [57] would not work since the algorithm cannot handle task structures. The schedule might not be correct and resources could be overbooked, leading to a failures during execution. Trying to solve the problem with the auction-based algorithm from [106] could produce an allocation, but since interrelated utilities are not taken into account, the allocation may contain overbooked resources and not result in a working solution. This is because the task allocation algorithms [57, 106] assume that it is easy to find a solution, but hard to find the best solution or a good solution.

Many task allocation algorithms are auction-based, for instance [23, 105]. The tasks are auctioned out to the agent most fit for the work, which is 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. The auction-based approach has been successful for tasks that have unrelated utilities. This is because UU ensures that a task or a sub-task tree can be treated as an independent entity, and can be auctioned out without affecting other parts of the allocation. When a robot puts a bid on a task it only has to take that task

into consideration. In our problem, possible resource conflicts and the fact that the allocation is IU means we do not have this isolation of sub-tasks. A bid can be different depending on the other commitments of the platforms. An auction-based solution for our problem would need to include combinatorial auctions or changing or replacing tasks to deal with interrelated utilities. Such interrelated utilities are called *complementarities* in auction terminology.

Distributed constraint optimization, which is basically a DisCSP with an objective function for optimization, is becoming a popular problem solving method [81]. This problem solving method is not necessarily a useful approach to our problem because it is not obvious what should be optimized. If we optimize on execution time, the platforms might waste too much resources compared to the gain in time. If we optimize on the number of violated constraints, we might get solutions that can not be executed even if only a few constraints are violated. For our task allocation problem we are interested in finding *a* solution not necessarily the optimal one, since it is not clear what is an optimal solution.

Various modifications of the DisCSP introduces different dynamic aspects to the problem. One example is the network consistency problem [14]. There has been some work on extending the DisCSP problem to take into account addition and removal of constraints, which relates to our conditional DisCSP problem. Typical DisCSP and DisCOP algorithms such as AWCS and ADOPT [77] assumes a static constraint network, which is determined from the start. But there are extensions to the algorithms that allow additions of constraints and variables [47, 85].

Dynamic DisCSP approaches have been applied to dynamic versions of the task allocation problem. In the original task allocation problem it is assumed that there is an evaluation function that can be used to measure the utility of a given allocation. In dynamic task allocation, the value of an allocation can vary during the assignment process. Variations depend on changing circumstances in the environment, changes in robot behavior and changing the capabilities and resources of robots [86]. In the dynamic version of the task allocation problem, called dynamic task-reallocation, tasks must be reallocated as the circumstances change so that the entire problem can be solved. An algorithm for dynamic task-reallocation problems called SOLO has been proposed for problems of this type. The algorithm builds on the Asynchronous Backtracking Algorithm with multiple variables per agent [86].

The task in a dynamic task allocation problem can be described as an abstract task, modeled by a dynamic constraint network which reflects the current configuration of the problem (there is a notion of time both during allocation and execution). A framework for solving this type of problem has been suggested [66]. The framework contains operations for adding and removing constraints to the network, adding and deleting variables and checking consistency continuously. The system is used in the Extendible Uni-

form Remote Operations Planner Architecture (EUROPA), which is used for solving AI planning, scheduling, constraint programming, and optimization problems. EUROPA does not use distributed constraint satisfaction problem solving, but extends the centralized CSP formulation to include dynamic aspects of the problem. A planning problem in EUROPA is mapped to a dynamic CSP (DCSP) [52].

Chapter 5

Extending the FIPA Abstract Architecture for Delegation

“How can $R1$ and $R2$ be realized on a collaborative UAS?”

To find an answer to the third research question, we start by defining the basic requirements of a collaborative multi-robot system. With this as a foundation we describe how the tasks a platform is capable of carrying out can be represented and what functionalities are needed to support the delegation of tasks.

A collaborative multi-robot system is a special type of multi-agent system where the agents are physical robots acting in the real world. A starting point for building the infrastructure of a multi-robot system would then be to study an infrastructure of a corresponding pure software multi-agent system. Much research has already been done in this field by the FIPA organization, resulting in the FIPA Abstract Architecture specification [48] and many other specifications of agent communication and communication protocols.

5.1 The FIPA Abstract Architecture

The aim of the FIPA organization is to promote agent-based technology and provide standards for agent interoperability. The organization has released a number of standards for agent communication and multi-agent system. For agent communication FIPA has introduced the agent communication language FIPA ACL. For multi-agent systems has FIPA introduced the FIPA Abstract Architecture specification. The name refers to a collection of components that provide foundational services for a multi-agent system.

The idea behind the specification is to standardize the functionality that is needed in most multi-agent systems. The components of the FIPA Abstract Architecture contain functionality to:

- Keep track of agents, to know which agents that are in the multi-agent system, and be able to handle events such as agents leaving or entering the system.
- Keep records of agents' abilities. With the assumption that the agents in the multi-agent system have services that they can provide for each other, this functionality keeps track of which agents that can provide a service.
- Allow agents to communicate. In other words providing basic functionality for finding agent addresses and providing the functionality needed for message based communication between agents in the system.
- Provide an agent communication language (high-level speech-act based communication).

The functionalities are provided by the following components:

Agent Directory This component provides “white pages services”, meaning that it maintains a list of all agents in a multi-agent system and can give information about how to contact them. It handles events such as agents entering and leaving the multi-agent system. It can be implemented as an agent or as a service.

Directory Facilitator This component provides “yellow pages services”, meaning it maintains a list of all services provided by agents in a multi-agent system and can provide information about which agents that provide a certain service. It can be implemented as an agent or as a basic service component.

Message Transport System This component is the basis for all communication in a multi-agent system. It handles sending and receiving of messages, network protocols and other low-level communication functionality needed for communication. Agent communication languages and protocols are then built on top of this component.

Agent Communication Language In the FIPA Abstract Architecture, agent communication is message-based and realized using an agent communication language, such as FIPA ACL.

The four components are shown in Figure 5.1. The components constitute an empty multi-agent system, without any agents or application-specific

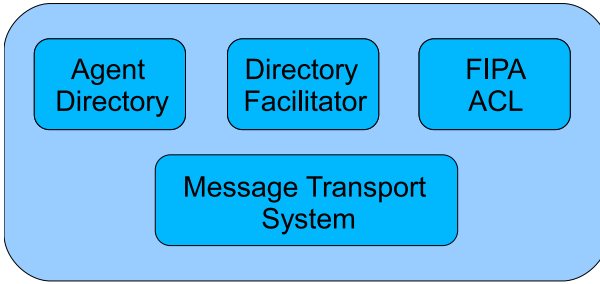


Figure 5.1: The components of the FIPA Abstract Architecture.

functionality. The idea is then to populate the system with agents and implement domain-specific ontologies, protocols and other application-oriented functionality, using the four components as basic functionality.

The Abstract Architecture is a good starting point for our collaborative robotic shell, but unfortunately it is not enough. Since we are developing a robotic system we must handle real-world constraints such as the platforms' locations and resources and how those affect the platforms' capabilities to perform tasks. The *Capabilities* and *Resources* predicates for tasks that platforms are capable of must be tied to the platforms actual services and resources. The information about what tasks the platforms are capable of must also be made available somehow. The Abstract Architecture does not cover all those issues. The main weakness lies in how the FIPA Abstract Architecture specifies services. The following requirements must be met by the collaborative robotic shell to realize the delegation functionality and handle tasks.

- It must be possible to specify how resources are used when a platform carries out a task. This is needed to make sure that resources are used as efficiently as possible and avoid double bookings when allocating TSTs.
- It must be possible to specify and find platforms that are capable of a certain task. It must also be possible to determine if a platform can be delegated a task by involving the platform in a task allocation process.
- It must be possible to implement protocols such as the auction and the delegation protocol.
- It must be possible to implement the delegation and task execution processes without regard to the actual realization of elementary tasks, capabilities and resources which are platform specific.
- Further it must be possible to implement the agent layer with interfaces both to other platforms and to the platform's legacy system.

In Chapter 2, the collaborative robotic shell and its parts were briefly introduced. We repeat some of the important parts here. The collaborative shell is assumed to be built on top of a legacy system, a robot architecture with an already existing software and hardware architecture. The collaborative shell consists of an agent layer that wraps the legacy system, and provides interfaces to the legacy system and to other platforms. The agent layer consists of four services, the Interface Service for platform to platform communication, a Delegation Service for handling delegations, an Execution Service for handling the execution of allocated tasks and communication with the legacy system, and a Resource Service for handling the platforms resources. Most of the previous listed requirements and the functionality in the agent layer is met by the FIPA Abstract Architecture, except the representation of resources and capabilities. The legacy system can provide services and carry out tasks. However those services and tasks must be represented in suitable way to be integrated with the task specification format and the task allocation algorithm. We describe our extended service model for handling such issues in the next section.

5.2 An Extended Service Model

The Directory Facilitator is basically a list of service names and for each entry a list of names of agents providing the service. Agents can ask the Directory Facilitator (using a basic ontology for service queries) for agents providing a service, and get a list of agent names back.

The service concept in the FIPA Abstract Architecture is very wide to cover a broad spectrum of applications. Basically it describes an agent's ability to do something. Platforms in a collaborative UAS can carry out tasks. A task specification tree describes what tasks that should be carried out. The platforms' capabilities in some sense correspond to the FIPA service concept.

With the FIPA service concept it is possible to state the question "*which platform can provide service X?*". Querying the Directory Facilitator provides a map between services and agents. We could use the FIPA service concept to specify what tasks a platform is capable of by storing the name of the task, the parameters of the task, and the agent's name in the Directory Facilitator. The problem with this is that the resource usage aspect of a task is missing, which is necessary for realizing the delegation speech act.

The FIPA Abstract Architecture must therefore be extended with a platform specification that also takes into account the use of resources. The platform specification captures both the static and the dynamic aspects of a task, which make it possible to answer questions such as "*how can a platform carry out a task under the current circumstances?*". The platform specification consists of the two components $Capabilities(B, \tau, [t_s, t_e, \dots], cons)$ and $Resources(B, \tau, [t_s, t_e, \dots], cons)$ for each task a platform is capable of. A composite action node distinguishes itself from an elementary action node

in that it currently only has a *Capabilities()* predicate. For each predicate pair in the platform specification, the platform has a platform-dependent, constraint-based representation of how the platform's static and dynamic resources are used when carrying out the corresponding task.

We extend the Abstract Architecture with a platform specification by introducing a Resource Service that keeps track of a platform's resources, and how they are used when carrying out tasks. The Resource Service is described in Section 5.3.1. For each task, the static part, the *Capabilities()* is represented by the name of the task it is capable of as stored in the Directory Facilitator. Finding a platform that can be delegated a task then becomes a two step process. In the first step, a list of possible candidate platforms is created by asking the Directory Facilitator for platforms capable of a certain task. This shows which platforms the task can be delegated to. The second step is interleaved with the delegation process, where also the dynamic part is evaluated on a platform per platform basis.

How the collaborative robotic shell is built up from legacy system to the agent layer is shown below. Two platforms and their legacy systems (and the legend for this and the following two figures) are shown in Figure 5.2.

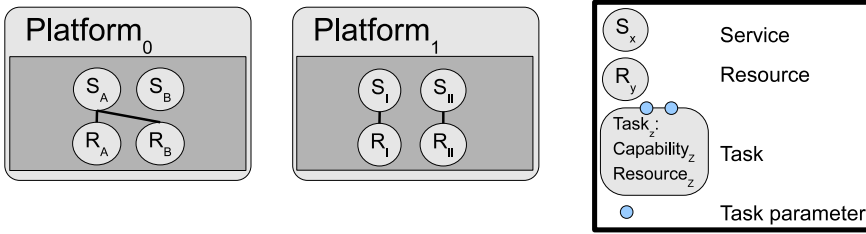


Figure 5.2: Two platforms and their legacy systems are shown in Figure 5.2. Each platform contains its own services and resources.

In the FIPA Abstract Architecture, the services in the system would be registered in the Directory Facilitator as shown in Figure 5.3.

In this case there is no description of how resources are used by the services because there is no resource concept in the Abstract Architecture. Our extension includes an agent layer containing a platform specification that not only registers the capabilities of the platform in the Directory Facilitator, but also adds a capability and resource model to represent how resources are used when carrying out a task, see Figure 5.4. The light-colored circles on a task describes the task's parameters. We need this extension for describing how resources are used when carrying out tasks, and also to connect the functionality in the legacy system to the tasks a platform is capable of performing.

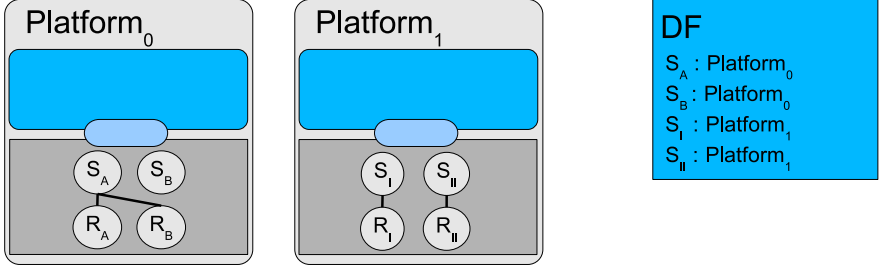


Figure 5.3: A multi-agent system based on the FIPA Abstract Architecture. The agent layer is depicted in light blue. Only the Gateway Agents are shown, but the platforms also contain other agents. The services are registered in the Directory Facilitator.

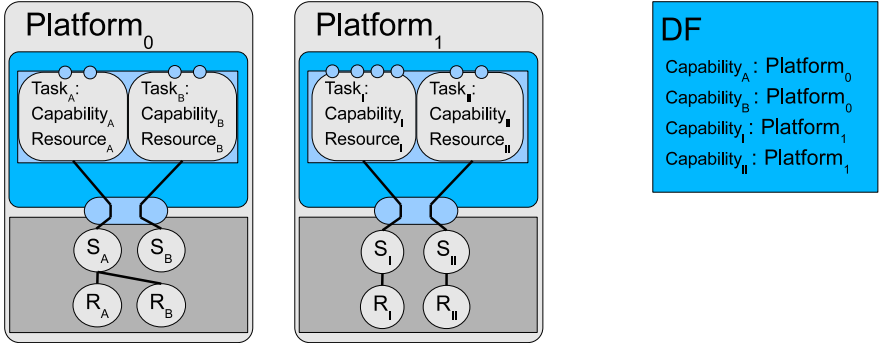


Figure 5.4: A collaborative multi-robot system where each platform has a platform specification, containing the capability and resource predicates, is shown in Figure 5.4. The names of the tasks a platform is capable of performing are stored in the Directory Facilitator.

Example

A platform P_0 has the service `flyto` which contains the procedural code for steering and moving the platform from one position to another. This functionality is part of the legacy system of P_0 . Adding an implementation of the FIPA Abstract Architecture on top of the legacy system introduces a Directory Facilitator. Storing the platform–service tuple in the Directory Facilitator expresses the fact that the platform has the `flyto` service.

However, a collaborative UAS uses task specification trees, not services, to describe the tasks platforms should carry out. A platform specification has *Capabilities*($B, \tau, [t_s, t_e, \dots], cons$) and *Resources*($B, \tau, [t_s, t_e, \dots], cons$) predicates to describe how the tasks use services and resources in the legacy

system. This model must be added to the FIPA Abstract Architecture.

For the task *flyto*, a *Capabilities()* predicate represents that a platform has the static capabilities for the *flyto* task (has the procedural knowledge). The name of the task and the platform's name is stored in the Directory Facilitator, i.e. $P_0 - flyto$. The *Capabilities()* and *Resources()* predicates are implemented as sets of resource constraints that describe the time it takes to carry out the legacy system's *flyto* service and how resources are used in the process. For the *flyto* task, the constraints express the time required to move from one position to another depending on the distance and the speed of the platforms (the constraints are shown in the example on page 28). A more complex model can for instance express how dynamic resources such as fuel are consumed depending on the speed of a platform.

5.3 The Agent Layer

In Chapter 2, 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 the extended FIPA Abstract Architecture on top of a legacy system. The agent layer contains five of the agents in the collaborative robotic shell. The agents are the Interface Agent, the Resource Agent, the Delegation Agent, Execution Agent and the Directory Facilitator, see Figure 5.5. The Gateway Agent belongs to the legacy system. In the previous chapter, we described the delegation process which includes recursive delegation, the generation of TSTs, allocation of tasks in TSTs to platforms, and the use of distributed constraint solving in order to guarantee the validity of an allocation of a TST. This complex set of processes is realized in the software architecture by extending the FIPA Abstract Architecture with a number of application-dependent services and protocols:

- Four services are defined, an Interface Service, a Resource Service, a Delegation Service and an Execution Service, associated with the corresponding Interface, Resource, Delegation, and Execution Agent. These services are local to each platform.
- Three interaction protocols are defined, the Capability Lookup Protocol, the Auction Protocol and the Delegation Protocol. These protocols are used by agents to guide the interactions between them as the delegation process unfolds.

5.3.1 Services

To implement the delegation process the Directory Facilitator and the four previously mentioned 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

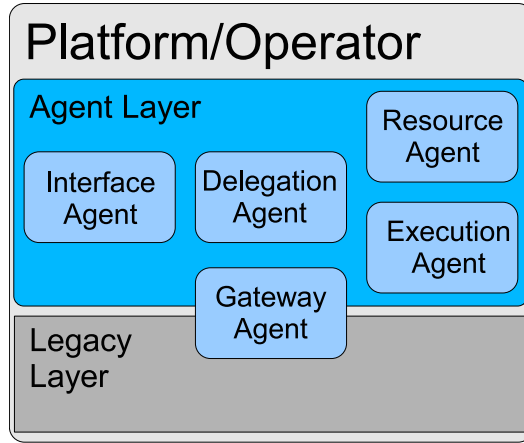


Figure 5.5: Overview of an agentified platform or ground control station.

find platforms with appropriate capabilities, the *Resource Service* to keep track of local resources and the *Execution 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 robotic shell the DF is used to keep track of a platform's capabilities. 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 platform / ground operator control station. 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, which is solved using local solvers that are part of the service.

In the implementation, constraints are expressed in ESSENCE' which is a sub-set of the ESSENCE high-level language for specifying constraint problems [53]. 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 [56] which can compile ESSENCE' problems into either Minion [55] or ECLiPSe [96], and Gecode [90]. We currently use Minion as the local CSP solver.

The Resource Service implements the Auction protocol described in Section 5.3.2.

Using Capabilities

Section 3.3 described the *Capabilities()* and *Resources()* predicates that represent the static and dynamic aspects of a task in the form of constraints, called resource constraints. The resource constraints are formed in the delegation process when a task is going to be allocated to a platform. The process goes through different stages, each using different representations of the resource constraints. In the first stage, the resource constraints of a task are loaded, containing a set of abstract constraints. The abstract constraints are instantiated with the platforms current state and resource status, producing a set of concrete constraints. Finally, booking constraints are added to the constraint set (if any exist). The resulting constraint set will then form a DisCSP Node (or extend an already existing DisCSP Node) on the platform.

A DisCSP Node is the collection of constraints and variables a platform has collected during the task allocation. It is basically the same thing as an agent in the DisCSP terminology. The variables are indexed after the platform's id and the task node's id, making sure the variable have unique names. A DisCSP Node has an agent view, describing the variables of other platforms' DisCSP Nodes it is interested in. Those are variables involved in the platform's global constraints, i.e. the constraints from the TST. A DisCSP Node is checked for consistency during task allocation.

Resource constraints: The resource constraints are created when a platform is going to be assigned to a task. Typically, the resource constraints are different on different platforms for the same task, and depend on the current context and how the platform implements the capability. The resulting resource constraints only involve the parameters of the resource predicate and internal variables of the resource constraints. For some resource predicates the resource constraints are parameterized and must be instantiated with *node parameters* before they can be used. Meaning that node parameters determines the values of the parameters and resource constraint variables, whereas some resource constraint variable are internal and can not be modified by the delegator.

Abstract constraints: *Abstract constraints* are constraints containing functions where the node parameters and internal resource variables are parameters. The functions cannot be replaced by values through constraint solving. Instead, the parameters should be given values according to the platforms state and the functions should be evaluated and replaced with their function values. This operation makes the constraints concrete.

An example is shown on page 68.

Concrete constraints: *Concrete constraints* are constraints without functions. Such constraints are formed when preparing resource constraints for a platform's current state.

Booking constraints: A *booking constraint* is a type of resource constraint that represents the unavailability of a resource. Booking constraints are added to the resource constraints when a capability with previously booked resources is loaded. Booking constraints represent intervals when a resource is booked and assure resources are not overbooked.

The Delegation Service

The Delegation Service, implemented by a Delegation Agent, coordinates delegation requests to and from the platform using the Execution, Resource and Interface Services. It does this by implementing the Delegation Process described in Section 4.2. The Delegation Service implements the Delegation Protocol described in Section 5.3.2.

The Execution Service

The Execution Service, implemented by an Execution Agent, is responsible for executing tasks using the legacy system (with the help of the Gateway Agent) on the platform. In the simplest case, this corresponds to calling a single function in the legacy system, while in more complicated cases the

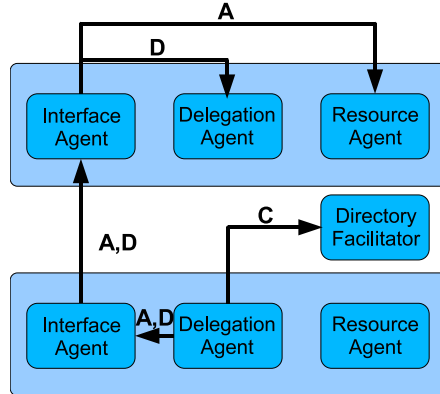


Figure 5.6: An overview of the agents involved in the Auction (A), Capability Lookup (C), and Delegation (D) protocols.

Execution Service might have to call local planners to generate a local plan to achieve a task with a particular set of constraints.

5.3.2 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 5.6.

The Capability Lookup Protocol

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

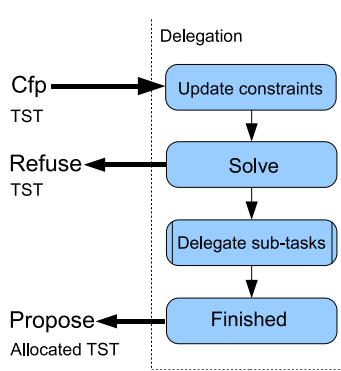


Figure 5.7: An overview of the Delegation Protocol.

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 [51, 88], implements the Delegation Process described in Section 4.2. 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 pre-conditions of the S-Delegate speech act and the second phase executes the task satisfying the post-conditions 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 pre-conditions 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 (possibly later in time). If the execution is successful, then the contractor returns an *inform* message otherwise a *failure* message. Such failure messages will invoke repair processes and are left for future work.

An overview of the steps in the Delegate Protocol is shown in Figure 5.7. 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 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 reserves resources and schedules activities relative to the temporal constraints that are part of the TST solution.

For each sub-task of the TST, the Delegation Protocol goes through the steps shown in Figure 5.8. First, it uses 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 is allocated the task. Allocating a task to a platform involves sending a *call-for-proposal* message with the task to the platform. This triggers the Delegation Protocol on that platform. If

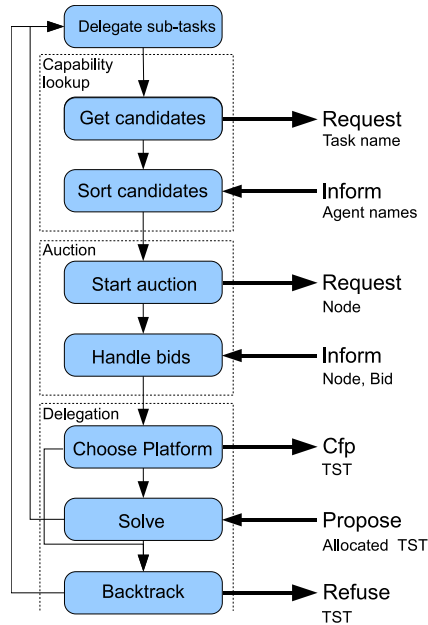


Figure 5.8: An overview of the recursive delegation of sub-tasks part of the Delegation Protocol.

an allocation fails, then backtracking starts. If backtracking has exhausted 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.

5.4 Task Allocation Algorithm Implementation

This section describes an implementation of the task allocation algorithm. The AllocateTST algorithm and its sub-modules are illustrated in Figure 5.9. The algorithm is used by the Delegation Agent to delegate task specification trees.

The work of AllocateTST can be seen as the interplay between two algorithms, the asynchronous weak commitment search (AWCS) algorithm and a TST allocation algorithm. The AWCS algorithm is used for checking the consistency of the constraint network formed when the constraints of the TST are connected with the resource constraints of allocated tasks. The TST allocation algorithm has an overarching role in that it makes the actual allocation and uses the AWCS algorithm in the process. The TST allocation algorithm performs a number of sub-tasks during the allocation, including: finding platforms with a certain capabilities, handling node auctions and determining the order of platforms for nodes when there are choices, selecting the appropriate node constraints for capability names and loading the platform's DisCSP Node with the constraints. The TST allocation algorithm uses the AWCS algorithm to determine consistency in the current constraint network. The algorithm also handles the booking of resources using the platform's resource database interface. The algorithm also performs backtracking when all possible candidates have been tested for a node without success. During backtracking the allocation algorithm removes constraints from the constraint network and removes bookings for resources in the resource database.

The TST allocator is the main module in the implementation of AllocateTST. The module parses the TST and extracts the next node to allocate. Using the capability lookup and auction handler, the platform finds candidates for the node. The delegation handler delegates the node to the selected platform. If the delegation fails, then a new platform is selected for the current node, and if no platforms are available, or all tried without success, the backtracking module is used to revert to the backtrack point.

When delegated a task, the TST Allocator derives the resource constraints using the capability loader. The constraints are added to the platform's DisCSP Node using the DisCSP Node interface. When consistency has been determined, a booking is added to the platform's resource database

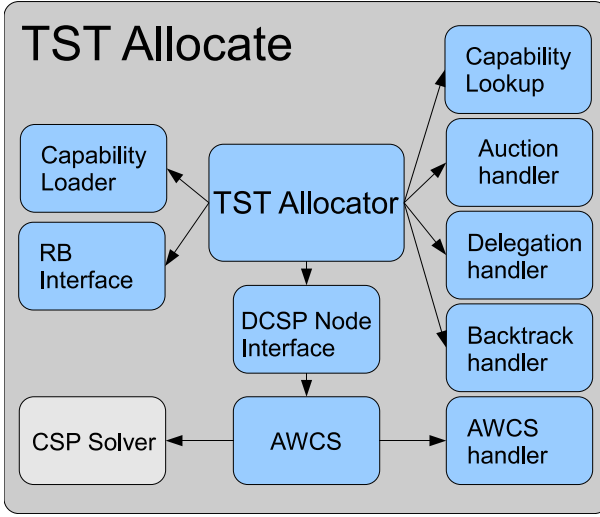


Figure 5.9: The TST Allocator and its components. Arrows show uses–relationships with submodules.

if the allocated node uses resources on the platform. In case of inconsistency the delegation is refused.

5.4.1 Capability Lookup

For a given task name the capability lookup handler searches for platforms that have the capability to achieve the task in the Directory Facilitator and returns the matching platforms (using the capability lookup protocol). Queries can be combined to return platforms that have the capabilities for a set of tasks.

Example

Before platform P_0 can delegate the task represented by node N_2 in Figure 5.10, it must find candidates for the node. Following the capability lookup protocol, P_0 sends a *request* message containing the task name **scan**. A list of platform names is returned in an *inform* message, containing platforms P_0 and P_1 .

5.4.2 Auction Handler

The auction handler provides an auction mechanism that auctions out a node to a set of candidates and collects the bids using the auction protocol. The handler is used both to hold and participate in auctions. As an auctioneer, the platform uses the handler to ask for bids on a node from a number of

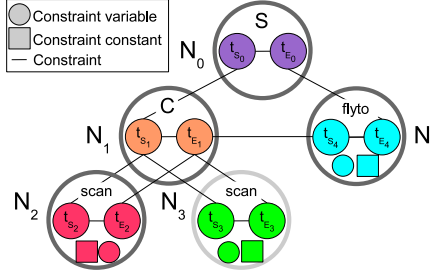


Figure 5.10: A TST with tree constraints and unallocated nodes.

candidates, to gather the bids and to determine an ordering of the candidates according to the bids. As a bidder, a platform uses an auction strategy to derive a bid. An auction strategy could for instance be the marginal cost bidding strategy where the bid is the marginal cost (in time) for a platform to take on the task that is up for auction.

With the marginal cost bidding strategy, each platform bids the time it would take for it to carry out the task, given its previous commitments. A platform always records the position of the platform for the last allocated task and uses it to calculate the distance to the next task. The time to carry out a task is based on the distance between the two positions, and with the assumption that the platform moves at its highest speed.

Example

Platform P_0 has two possible candidates for node N_4 (in Figure 5.10), itself and platform P_1 . Following the auction protocol, P_0 sends a *request* message, containing the task for node N_4 to itself and to P_1 . Platform P_1 bids 18 for the task, whereas platform P_0 bids 45. The difference in bids depends on where the platforms are in the world in relation to the position to fly to (actually where the platforms are expected to be after carrying out their last previously allocated task). Platform P_1 has previously allocated itself to node N_3 , the elementary action scan ($Area_B$ in Figure 5.11). The position of the elementary action scan is in the middle of the field for the area to be scanned. P_0 has previously been allocated the other elementary scan action N_2 ($Area_A$ in Figure 5.11). P_0 is closer to the position of the auctioned elementary action N_4 ($Dest_4$ in Figure 5.11) and therefore bids lower than P_1 . Both platforms return their bids together with the task in an *inform* message. P_0 can now order the candidates for node N_4 according to the bids, P_0 followed by P_1 .

5.4.3 Delegation Handler

The delegation handler is used to delegate a node (and the sub-TST rooted in that node) to a platform, using the delegation protocol. The handler is

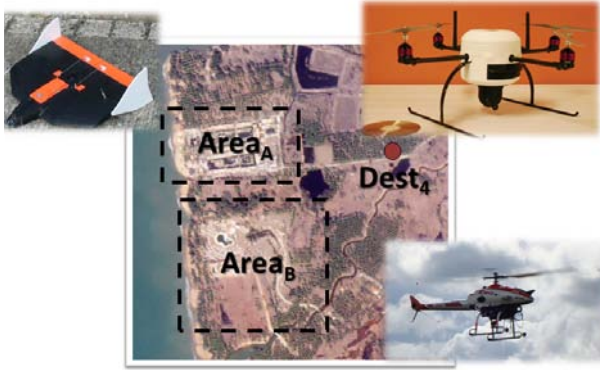


Figure 5.11: P_0 is allocated to the scan task for area $Area_A$ and P_1 is allocated for the scan task $Area_B$. In the auction of N_4 , P_0 returns the lowest bid because it is closer than P_1 .

in charge of the sending and receiving of delegation messages, and reporting whether the delegation succeeded or not.

The second part of the protocol describes the choice and activation of a solution. In the second part of the protocol, an *accept-proposal* or *reject-proposal* is sent from the delegator, and an *inform* or *failure* message is returned by the contractor. The protocol ends when τ has been executed.

Example

Platform P_0 has two possible candidates for node N_2 in Figure 5.10, the bid order after auctioning is P_1 , P_0 . Following the delegation protocol, P_0 sends a *call-for-proposal* message containing the task for node N_2 to P_1 . Platform P_1 receives the task and allocates itself to the node. The constraint network is extended (using the Capability Loader and the DisCSP Node Interface) and checked for consistency (using the DisCSP Node Interface). The network is consistent, and platform P_1 sends a *propose* message back.

When the entire TST is allocated and consistent, platform P_0 sends an *accept-proposal* message to P_1 , making the platform ready for executing the task represented by N_2 . When the task is executed and finished, platform P_1 returns an *inform* message to P_0 or a *failure* message in case the execution failed.

5.4.4 Capability Loader

The capability loader loads the resource constraints for the task to be allocated. It also instantiates the resource constraints with the platform's

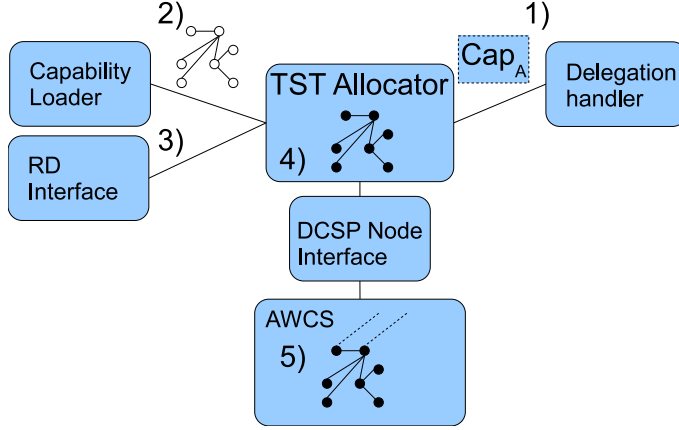


Figure 5.12: 1) A task which requires capability Cap_A is delegated to the platform containing the TST allocator. 2) The corresponding resource constraints are loaded, 3) and instantiated with the platform's current resources, state, etc. 4) forming a concrete instantiation of the resource constraints. 5) The constraints are connected to the tree constraints.

current resource status, state, etc., following the steps described in Section 5.3.1. The end result is a set of concrete resource constraints. The constraints are connected to the tree constraints in the current constraint network, which is a preparation stage for running the AWCS algorithm (see Figure 5.12).

The capability loader is also used by the backtrack handler to determine which constraints to remove from the constraint network during backtracking. The module is used in combination with the DisCSP Node Interface to add and remove constraints to and from the platforms DisCSP Node.

Example

Platform P_1 has allocated itself to node N_2 in Figure 5.10. The capability for the task in N_2 is *scan*. The capability model for *scan* is loaded by the capability loader. The capability model contains the following abstract constraints (scan_coverage is the area size the platform can scan at any given time point):

$$t_e = t_s + \frac{area}{scan_coverage \cdot speed}$$

$$Speed_{Min} \leq speed \leq Speed_{Max}$$

Instantiating the constraints creates the concrete constraints:

$$t_e = t_s + \frac{1000}{speed}$$

$$1 \leq speed \leq 10$$

The constraint set is connected to the tree constraints in N_2 , by connecting the node interface of N_2 with the capability interface of the constraint network for *scan* as displayed in Figure 4.2 on page 32.

5.4.5 DisCSP Node Interface

Every platform participating in the allocation of a TST has a DisCSP Node. The AllocateTST algorithm accesses the platform's DisCSP Node configuration through the DisCSP Node Interface. Operations provided by the interface are adding and removing constraints retrieved from the capability loader or from the TST in the form of tree constraints. Other operations include testing for consistency with the current constraint network and retrieving a solution for the constraint variables.

5.4.6 AWCS

AllocateTST uses a DisCSP algorithm for checking the consistency of a constraint network, in this case the AWCS algorithm. The AWCS module uses the local CSP solver and the AWCS handler for sending and receiving messages.

In the AWCS algorithm, each *agent* (node in the distributed problem) has a number of variables and constraints. Constraints can be *local* or *global*. Local constraints only involves the agent's own variables. Global constraints involve more than one agent's variables. An agent has an *agent view*, which is a list of variables belonging to other platforms that are involved in the agents global constraints.

When an agent solves its local constraint problem, it sends suggestions of values for its variables that are involved in the global constraints to the affected agents. Such values are sent in *Ok* messages. If an agent receives an *Ok* message and it is not possible to find a solution given the value of its agent view, it will return a *Nogood* message, containing the combination of values that made the problem unsolvable. The combinations, or Nogoods as they are called, are saved to make sure that they are only sent once. The algorithm will terminate when no more *Ok* messages are sent or when the agent can not satisfy its constraints and at the same time can not produce a new *Nogood*. The AWCS algorithm is described in [101, 103].

A termination detection algorithm is used in conjunction with AWCS to detect when no more *Ok* messages are sent [11].

5.4.7 AWCS Handler

The AWCS Handler handles the message passing between platforms for AWCS, such as the *Ok* and *Nogood* messages.

5.4.8 Resource Database Interface

The Resource Database Interface provides operations on the platform's resource data-base. Operations relevant for AllocateTST include reading the current status of resources and adding or removing bookings of resources in the resource database. Bookings are needed to describe the current resource usage on the platform.

When a resource is booked and a booking interval is added to the resource database, a booking constraint will be derived the next time a node with a capability requiring the same resource is allocated to the platform.

Resource Database Query Protocols

Two protocols, both based on the FIPA-request protocol, are available when using the resource database interface. The first protocol is an add or remove protocol for bookings, where the content of the *request* message is an add or remove operation, a time interval and a resource. An *inform* message is returned, describing the result of the request.

The second protocol is used to retrieve booking constraints for a particular resource. The *request* message in the protocol contains a resource name, the returned *inform* message contains a booking constraint, or is empty when no bookings exists.

Example

Platform P_1 has been allocated to scan an area during 200s, starting at 13:00. A booking constraint for the platform is registered in the resource database describing that the platform is occupied during this time. The next task to be allocated to P_1 is an elementary action of type *flyto*. The platform has been previously booked for another elementary action, *scan*. The scan action takes place during the interval 0 to 200 (where time 0 denotes when the mission starts). The booking constraint on the platform forbids the node interface variables t_s and t_e of the elementary action to be assigned values less than 200.

5.4.9 Backtrack Handler

The backtrack handler keeps track of the platform – node combinations that have already been tried for the part of the TST the platform is responsible for (i.e. where the platform has worked as a delegator).

Backtracking can be carried out using different strategies, such as chronological backtracking or backjumping. The backtracking strategy determines to which point in the allocation the algorithm should revert to and how to find that point.

Backjumping Protocol

The chronological backtracking strategy does not need any special protocol, since the delegation protocol can be used to determine when to backtrack, e.g. when all platforms have been tried for delegation of some task without success.

The backjumping strategy, on the other hand, needs a special protocol. The protocol is based on the FIPA-request protocol. The *request* message contains an activation message, sent to the *failure point allocator*, an *inform* message containing an *allocation failed* or *allocation succeed* is returned.

Example

Consider the TST in Figure 5.10 again. Let us assume that platform P_0 has been allocated to node N_0 , N_1 and N_3 , and P_1 has been allocated to node N_2 . The last node, N_4 is to be allocated, but none of the candidates P_0 or P_1 can do the task. Platform P_0 is now the *failure point allocator*. The backtracking starts by removing all previous allocations to the left of the failure point. P_0 achieves this by removing the connection points to the left branch from its agent view. The network is now consistent. P_0 can try out a candidate for node N_4 . It selects P_0 . Following the search downwards pattern (described in Section 4.5.1), P_0 reconnects the left branch, and then disconnect the sub-branches in reverse order as they were allocated. Disconnecting N_3 has no effect. Disconnecting N_2 makes the network consistent. The failure point is found.

P_0 continues from N_2 testing the next best candidate, P_0 , itself. The network is consistent. Node N_3 is allocated to platform P_1 and node N_4 to P_0 .

5.5 Related Work

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 coordinate 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 [79], 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+ [8] integrates mission planning, task refinement and cooperative task allocation. It uses a task allocation protocol based on the Contract Net protocol with explicit, pre-defined capabilities and task

costs. 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 [80], 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 [71], MURDOCH system [58], Hoplites [67] and TAEMS [22].

In this chapter we described how the collaborative robotic shell could be realized. The collaborative UAS is an *open* multi-agent system. Multiagent systems of this type are characterized by the lack of knowledge about what agents are present and what their capabilities are. Such features lead to interoperability issues, such as how can the agents communicate, what common language should they use, what are the communication protocols of interest, etc. Much of the research behind the FIPA specifications [48, 49, 50, 51] is relevant since it deals with interoperability issues in open multi-agent systems. The FIPA communication protocols are general concepts and are meant to be extended for each multi-agent system using them, as we extended them to create the capability lookup, auction and delegation protocols. the protocols are not enough on their own. Whereas FIPA has mainly concentrated on agent communication, they have also done some work on the representation of services. The director facilitator is the outcome of this research, and the idea is that the services an agent can provide to another are described in an ontology. The agents that communicate regarding service usage must then have access to the relevant ontology, which can be accessed, for instance, from a specific ontology agent (and each agent understands a basic ontology to communicate with the ontology agent).

The InfoSleuth [78] project argues that the agents in an open multi-agent

system need three supporting functionalities to use capabilities. The first requirement is that it must be possible to exchange information about capability between agents. The second requirement is it must also be possible to formulate requirements on the capabilities, and the third is that there must be some means of semantic matchmaking or other ways of determining if capabilities meet the task at hand. In the InfoSleuth project, the matchmaking is handled with a logical deduction language called LDL++ [4]. InfoSleuth uses KQML as an agent communication language. If we try to relate our solution to the three requirements, we see that the first requirement is met by the registration of the platforms static part of their capabilities in the Directory Facilitator and the ontology used to retrieve this information. The second requirement is described in the task specification trees (described Chapter 3) and the third requirement is achieved with distributed constraint solving (described in Chapter 4).

Representing capabilities with constraints or logical formulations are two approaches. Another way is to pre-calculate the scope (all the possible values) of the capabilities, which can be done if the capabilities are few and/or the scope is small. In [104] the reachability of a robot's arm is represented as a 3D map. The representation is pre-calculated and can be used to determine which movements are possible in a given situation.

Interoperability issues is one aspect related to the realization of the collaborative robotic shell, another related issue is the need of *situation dependent* capability models. This problem has been researched in projects that combine the idea of service-oriented-architectures and multi-agent systems. A merge between multi-agent systems and semantic web services to form an architecture for context aware services has been proposed [18]. Another system that combines service oriented architectures (SOA) and agent systems is the Flexible User and ServIces Oriented multiageNt Architecture (FUSION) [17].

5.6 Summary

In this chapter we have described how the collaborative robotic shell may be realized. We demonstrated how the FIPA Abstract Architecture can be extended to support delegation among collaborative agents. An implementation of the task allocation algorithm described in Chapter 4 and the modules needed for its operation was described. In the next chapter we will exemplify the use of the system in a number of case studies of realistic multi-UAV missions.

Chapter 6

UAS Case Studies: Assisting Emergency Services

In this chapter we describe the operations of the collaborative UAS in a number of different scenarios. The purpose of this chapter is to give the reader a feel for how the task allocation algorithm is used in the delegation of tasks in practice. We also show how the operator can adjust the autonomy of the contractor by adding user constraints on a TST, and how goal nodes are intended to be handled in a future extension of the system that incorporates a planner that can generate TSTs.

The case studies consist of three scenarios, each can be seen as an integrated part of a larger search and rescue mission.

6.1 Introduction

On December 26th, 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.

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 (UAV platforms, ground operators, etc.) 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 people 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 UAV platforms should cooperatively scan large regions in an attempt to identify injured people. 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 UAVs 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 UAVs to deliver relief packages to the injured. This should also be done in a cooperative manner.

One approach to solving 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 UAV. We have previously shown how to generate pre-allocated plans and monitor their execution [35, 68]. In this thesis we show how a plan without explicit allocations expressed as a complex task tree can be allocated to a set of UAV platforms which were not known at the time of planning.

6.2 The Victim Search Scenario

We will now consider a particular instance of the emergency services assistance scenario. In this instance there is a UAS consisting of two platforms

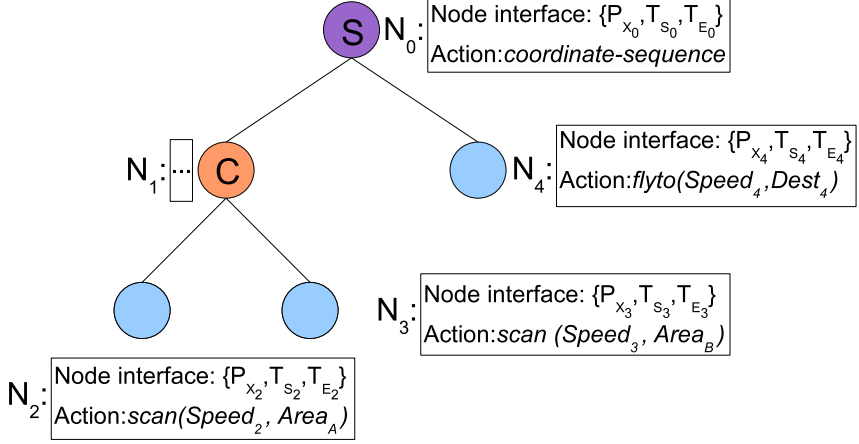


Figure 6.1: The TST for the victim search scenario.

(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 [83]. 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.

6.2.1 The TST for Victim Search Scenario

The scenario starts with the scanning of the region in the search for survivors. The region assigned to operator OP_1 is divided into two sections, each can be scanned by a platform. When the scan is finished, one platform should return to the operator and be ready to load emergency supplies on a carrier next to the operator's control center. We use the TST for the mission, shown again in Figure 6.1.

The goal of the mission is to create a saliency map with the victims as saliency points. The map is needed as input for the next part of the mission, where emergency supplies should be delivered to the victims.

6.2.2 Allocating the Victim Search TST

To allocate the TST in Figure 6.1 the operator OP_1 invokes **AllocateTST** on the top node N_0 . After an auction between P_1 and P_2 for N_0 , OP_1 sends a *call-for-proposals* message with the TST to the winner P_1 . This invokes **TryAllocateTST** for N_0 on P_1 .

P_1 is now responsible for N_0 and for allocating the remaining nodes in the TST (see Figure 6.2 for the schedule). The task allocation algorithm traverses the TST in depth-first order. P_1 should first find a platform for node N_1 , and when the entire sub-TST rooted in N_1 is allocated, find an allocation for node N_4 . Node N_0 and N_1 are two composite action nodes which have the same marginal cost for all platforms. P_1 therefore recursively allocates N_1 to itself (see Figure 6.3 for the extended schedule). 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.

Node	11:00	11:30	Platform
0:	N0s	N0e	1:

Figure 6.2: The schedule after assigning node N_0 .

Node	11:00	11:30	Platform
0:	N0s	N0e	1:
1:	N1s	N1e	1:

Figure 6.3: The schedule after assigning node N_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-proposals* message containing N_2 is sent to platform P_2 .

P_2 received the *call-for-proposals* message, loads and instantiates the resource constraints for the platform's *scan* task. The constraint network retrieved is connected to the constraint network from the TST. The network is checked for consistency. The network is consistent and node N_2 is now allocated to platform P_2 . The constraint network now involves both platforms. Figure 6.4 displays the schedule. P_2 returns a *propose* message to P_1 .

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

Node	11:00	11:30	Platform
0:	N0s ————— N0e		1:
1:	N1s ————— N1e		1:
2:	N2s ——— N2e		2:

 Figure 6.4: The schedule after assigning node N_2 .

itself. The extended constraint network is consistent. Figure 6.5 shows the extended schedule.

Node	11:00	11:30	Platform
0:	N0s ————— N0e		1:
1:	N1s ————— N1e		1:
2,3:	N2s ——— N2e N3s ——— N3e		2,1:

 Figure 6.5: The schedule after assigning node N_3 .

The remaining node, N_4 is delegated to platform P_2 . The entire TST is now allocated. The complete schedule is in Figure 6.6.

Node	11:00	11:30	Platform
0:	N0s ————— N0e		1:
1,4:	N1s ————— N1e N4s — N4e		1,2:
2,3:	N2s ——— N2e N3s ——— N3e		2,1:

 Figure 6.6: The complete schedule after assigning node N_4 .

The operator is satisfied with the allocation and starts the mission. An *accept-proposal* message is sent to P_1 . P_1 traverses the TST, marking the nodes as ready for execution in depth-first order. Nodes not allocated to the platform, are marked by sending an *accept-proposal* to the platform owning the node. P_1 sends *accept-proposal* to P_2 for node N_2 and N_4 . The execution starts, and the platforms scan the area creating the saliency map in Figure 6.7.

6.3 The Supply Delivery Scenario

The supply delivery scenario is the second part of the search and rescue mission described in Section 6.1. The goal of the mission is to deliver emergency supplies to the injured people found in the first part of the mission.

The supply delivery scenario involves victims, boxes containing emergency supplies, carriers, an operator and platforms, some with the capability of delivering a supply box, some with the additional capability of delivering a carrier. A carrier can hold boxes. Sometimes it is beneficial to load a carrier and deliver it to a key position in the region and distribute the boxes from there, instead of distributing each box from its original site to its destination. The goal of the scenario is to deliver an emergency supply box to each victim.

6.3.1 The TST for the Supply Delivery Scenario

In this instance of the supply delivery scenario, shown in Figure 6.7, five survivors (S_1 – S_5) are found in Leg I, and there are three platforms (P_1 – P_3) and one carrier available. 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 [68]. The resulting TST is shown in Figure 6.8. 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 region, far away from where most of the survivors were found (N_{13}). The delivery of packages can be done concurrently to save time, but the loading, moving, and unloading of the carrier is a sequential operation. UAVs and equipment should be allocated carefully to assure that all relief packages reach their destinations in time.

Another operator OP_2 is performing a scan mission, with the platforms P_3 and P_4 north of the area in Figure 6.7. P_3 is currently idle and OP_1 is therefore allowed to borrow it.

6.3.2 Allocating the Supply Delivery TST

To allocate the TST in Figure 3.1 the operator OP_1 invokes `AllocateTST` on the top node N_0 . After an auction between P_1 and P_2 for N_0 , OP_1 sends a *call-for-proposals* message with the TST to the winner P_1 . This invokes `TryAllocateTST` for N_0 on P_1 .

P_1 is now responsible for N_0 and for allocating the remaining nodes in the TST. The task allocation algorithm traverses the TST in depth-first order, so P_1 should first find a platform for node N_1 , and when the entire sub-TST rooted in N_1 is allocated, find an allocation for node N_{13} . Node N_1 and N_2 are two composite action nodes which have the same marginal cost for all platforms. P_1 therefore recursively allocates N_1 and N_2 to itself.



Figure 6.7: The disaster area with platforms P_1 – P_3 , survivors S_1 – S_5 , and operators OP_1 and OP_2 .

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 platform 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 and N_4 , P_2 is now closest to the depot, resulting in the lowest bid and being allocated to N_5 and N_6 . The schedule defined by nodes N_0 – N_2 is now constrained further by how long it takes for P_1 and P_2 to carry out action nodes N_3 – N_6 . The constraint network is now shared between platforms P_1 and P_2 .

The next node to allocate for P_1 is node N_7 , the carrier delivery node.

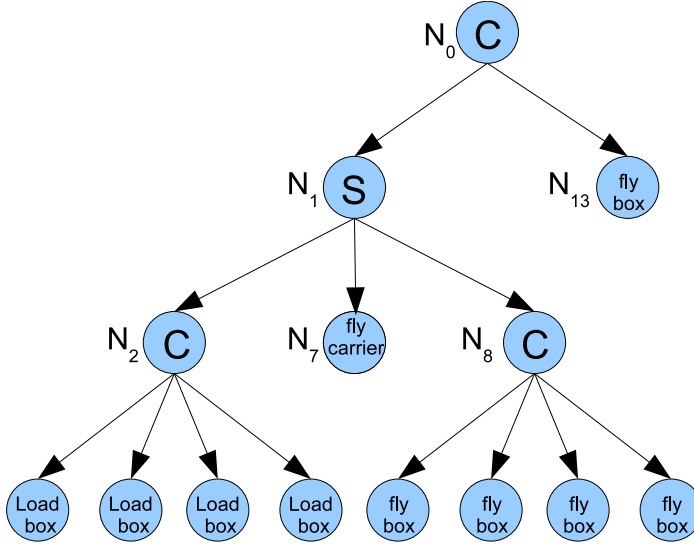


Figure 6.8: The TST for the supply delivery scenario.

P_1 is the only platform that has the capability required for this task. The node is allocated to P_1 . Continuing with the 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 to nodes N_8 – N_{12} .

The final node, N_{13} , is allocated to platform P_2 and the allocation is completed. The only non-local information used by P_1 were 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, much longer than the operator expected. Since the constraint problem defined by the allocation to the TST is shared between the platforms, it is possible for the operator to modify the constraint problem by adding more constraints, and in this way modify the task allocation. The operator puts a time constraint on the mission, restricting the total time to 30 minutes.

To re-allocate the modified TST, operator OP_1 sends a reject-proposal to platform P_1 . The added time constraint to the mission 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 newly added 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 the node N_{13} , P_1 discovers 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 , to start an Upward Search. 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 show 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. Removing the allocations for 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 .

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 made 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, does not have enough time to deliver the final package. Instead, a new backjump point search starts, finding node N_5 , and continuing from there. This time around, 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 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 starting the execution of the mission.

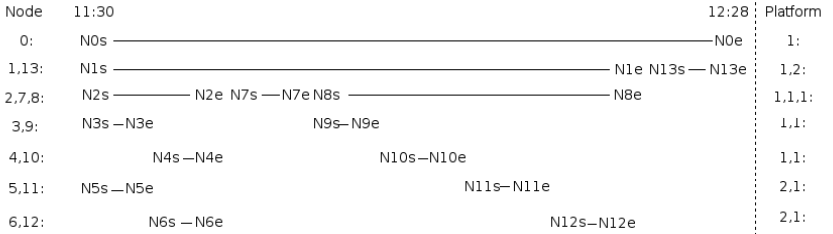


Figure 6.9: The complete schedule when using two platforms and no deadline.

6.4 The Communication Relay Scenario

Another impact of a disaster such as an earthquake or tsunami is the destruction of the infrastructure in the affected area. A related problem to the search and rescue is then how to establish a communication relay chain so

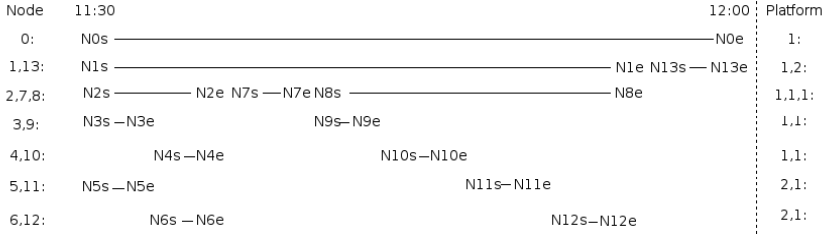


Figure 6.10: The complete schedule when adhering the new deadline.

that the gathered information can be relayed back to the operator’s control center or between different operators’ control centers.

In the communication relay scenario data should be relayed from one platform, the observer, to a ground station. The operation is carried out in a city, where building and other obstacles can obstruct the communication range of the platforms. Each platform also has a limited communication range even without any obstacles around it. The communication relay problem and algorithms for generating relay chains are described in [9].

6.4.1 The TST for the Communication Relay Scenario

The TST for communication relay scenario describes how a number of platforms should take positions and form a relay chain so that the data from the survey platform can be relayed back to the operator’s control station. The TST contains goal nodes that must be expanded before the TST can be allocated. For example, see Figure 6.11(a), where a goal node contains the goal of generating a relay chain. Goal nodes must be expanded by some form of planning. In the relay scenario, the planning is done using an algorithm that places platforms in a relay chain [9]. A sub-TST describing the relay chain, is added to the right of the expanded node, under the goal node’s parent. The TST in Figure 6.11 displays the TST before (Figure 6.11(a)) and after (Figure 6.11(b)) the expansion of the first goal node. The relay chain consists of four relay points, containing sub-TSTs describing how a platform flies to a position and relays data from that position. The last sub-TST describes the work of the surveyor platform – the platform in the end of the relay chain that is producing the data that is relayed back to the operator through the relay chain. Each sub-TST must be executed in parallel, which is enforced by the constraints.

6.4.2 Allocating the Communication Relay TST

The operator specifies a relay mission by pointing out a starting point and an end point in the main user interface, see Figure 6.12. The starting point

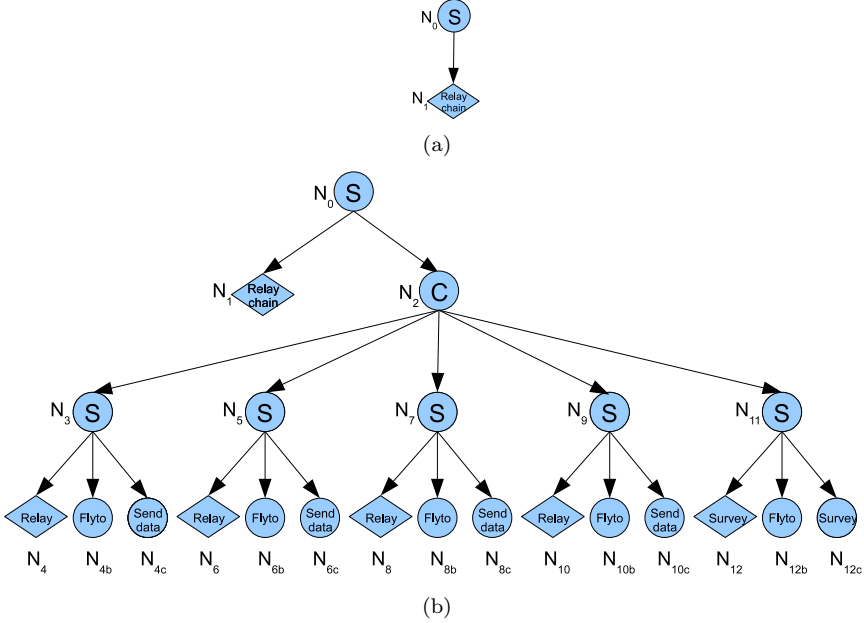


Figure 6.11: The TST for the communication relay scenario.

marks the operator's position, the end point marks the area to survey. The operator may also adjust other mission parameters, such as the communication distance of platforms, the maximum number of platforms that can be allocated for the mission, and the algorithm used to calculate the relay chain.

When the operator presses the “Create Mission” button in the main user interface, the mission is compiled into a TST. The TST is added to the mission list in the mission user interface, (see Figure 6.13). Selecting the mission displays the TST, (see Figure 6.15). For now the TST only has two nodes: A sequential node with a child goal node. The operator can click on the nodes in the TST and the node's data is displayed in a new window, see Figure 6.14. The operator can also change parameters in the nodes. By specifying P_0 as the platform for the sequence node, the operator assures that the TST must be delegated to that platform. Unassigned nodes, such as the goal node, can be delegated to any platform that has the required capabilities. By selecting the sequence node and pressing “FIPA Delegate”, a delegation from the operator to P_0 is initiated.

P_0 receives a *call-for-proposal* message, containing the TST with top-node N_0 . The constraint network is extended for the node and is checked for consistency. P_0 allocates itself to N_0 . The schedule after allocating node N_0 is depicted in Figure 6.16. N_0 has one child, the goal node N_1 . Since

both node N_0 and node N_1 belongs to the same compound task, the only platform candidate is P_0 .

P_0 is allocated to N_1 . The node is expanded. A new sub-tree is added under N_0 , after N_1 . The extended schedule is depicted in Figure 6.17.

P_0 continues with the new child node N_2 . The node is unassigned and needs the capability for the task **coordinate-concurrent**. A capability lookup returns a list with candidates P_0 – P_5 . The node is auctioned out. P_0 sends a *call-for-proposal* with N_2 to the winner, which is itself.

P_0 tries to allocate N_2 to itself. The constraint network is extended to include node N_2 . The constraint network is consistent (see Figure 6.18). Node N_2 has five child nodes, each is a sequential node with a single goal node as child. Each sub-tree is grouped together in a compound task. The capability lookup returns a list of platforms P_1 – P_5 as possible candidates for the child N_3 . A node auction for N_3 determines the candidate order. P_0 sends a *call-for-proposal* message to the first candidate P_1 , containing the sub-tree with top-node N_3 .

P_1 extends the constraint network to include node N_3 , the network is consistent and P_1 assigns itself to node N_3 . N_3 and N_4 belongs to the same compound task. P_1 is also assigned to N_4 . The constraint network is extended to include node N_3 . The network now spans platforms P_0 and P_1 .

P_1 is allocated to node N_4 . N_4 is expanded. Two more nodes (N_{4b} , N_{4c}) are added under N_3 , after N_4 .

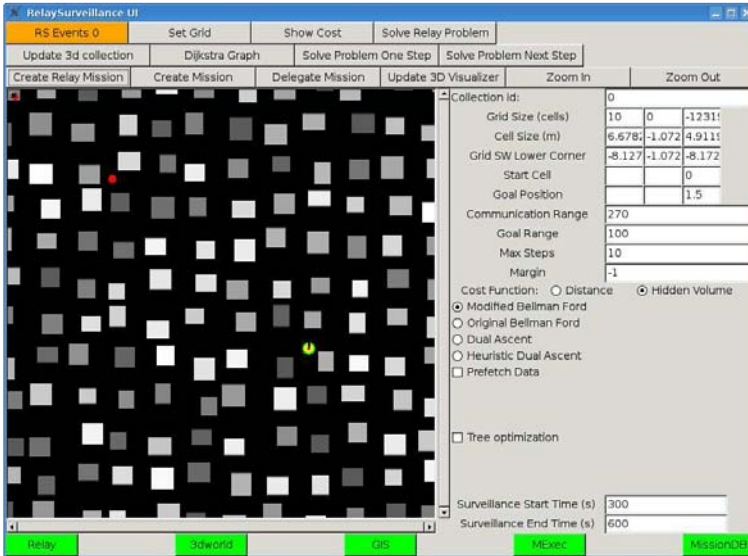


Figure 6.12: The operator's user interface for the relay scenario.



Figure 6.13: The Mission User Interface containing the list of TSTs.

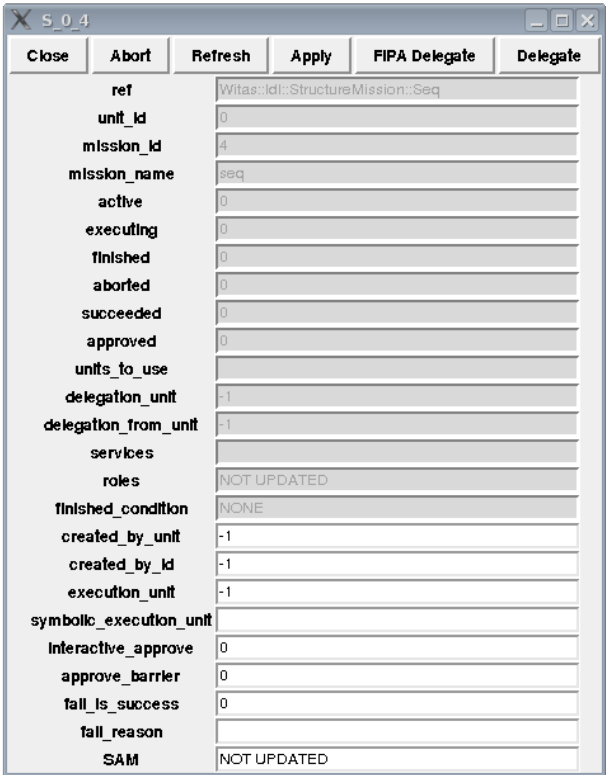


Figure 6.14: The content of a node in the relay TST.

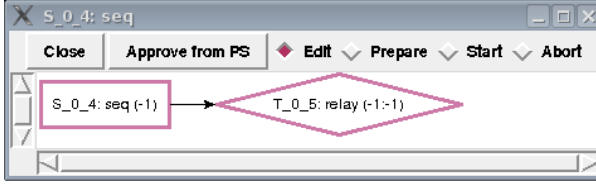
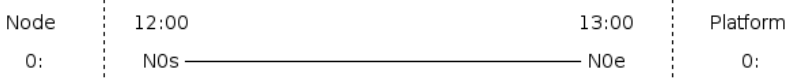


Figure 6.15: The relay TST before it is expanded.

Figure 6.16: The schedule after assigning node N_0 .

P_1 continues with the new child node N_{4b} . The node is not assigned to any platform, since it belongs to the same compound task as N_3 , making P_1 the only candidate. P_1 allocates itself to node N_{4b} . In contrast to previous encountered tasks, the *flyto* is an exclusive task because the platform cannot make more than one flight at the same time. The resource database on P_1 is queried, and yields an empty constraint set. The resource constraints for the elementary action node is loaded, instantiated and the resulting constraint network is connected to the constraint network from the TST. The network is checked for consistency. The platform is booked during this time interval, and the interval is added to the platform's resource database.

P_1 continues with child node N_{4c} . The platform has the capability for the task *send data*. N_{4c} is assigned to P_1 . The resource constraints for the elementary action node is retrieved, instantiated and the resulting constraint network is connected to the constraint network from the TST. The task *send data* is also an exclusive task because the platform must hover and transfer data. Querying the resource database on P_1 yields a constraint set describing the N_{4bs} – N_{4be} interval. The interval is added to the resource constraints. The network is checked for consistency. The solution determines the interval for N_{4c} . The platform is booked during this time interval, and the interval is added to the platform's resource database.

P_0 has now managed to delegate the sub-tree with top-node N_3 to P_1 . P_0 continues with the second child, N_5 . P_0 examines if N_5 is assigned. The node is unassigned and the capability lookup returns the list of platforms P_1 – P_5 . P_1 wins the auction for node N_5 .

P_1 receives a *call-for-proposal* message containing the TST N_5 . P_1 assign itself to N_5 . N_5 and N_6 belongs to the same compound task. P_1 is also assigned to N_6 . The extended schedule is depicted in Figure 6.20.

Node N_6 is expanded. Two more nodes (N_{6b} , N_{6c}) are added under N_5 , after N_6 . N_6 is already assigned to P_1 . The extended schedule is depicted

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e		0:

Figure 6.17: The schedule after assigning node N_1 .

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:

Figure 6.18: The schedule after assigning node N_2 .

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:

Figure 6.19: The schedule after assigning node N_{4c} .

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:
5:		N5s ————— N5e	1:

Figure 6.20: The schedule after assigning node N_5 .

in Figure 6.21.

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:
5:		N5s ————— N5e	1:
6:		N6s — N6e	1:

Figure 6.21: The schedule after assigning node N_6 .

P_1 tries to allocate N_{6b} to itself. The platform has the required capability for the task *flyto*. N_{6b} is assigned to P_1 . Querying the resource database on P_1 yields a constraint set describing the N_{4bs} – N_{4be} and N_{4cs} – N_{4ce} intervals. Those constraints are combined with the constraint network derived from the resource constraints of node N_{6b} . This time, the constraint network is inconsistent because the intervals overlap, see Figure 6.22(a). A *refuse* message is sent back. Backtracking starts, but no solution can be found, meaning that P_1 cannot be used for the sub-tree N_5 . N_6 is unexpanded (the sub-TST created by the expansion of the goal node is removed). The children N_{6b} and N_{6c} are removed. The constraints added from this delegation are removed from the constraint network. The schedule is reverted, see Figure 6.22(b).

P_0 receives a *refuse* message from P_1 . P_0 tries the next best candidate for N_5 . It can be assigned to the nodes in the sub-TST with top-node N_5 . P_2 is the next candidate for the sub-TST.

P_0 continues with the remaining children N_7 , N_9 , N_{11} . The procedure is similar, except for the sub-tree N_{11} with the task *survey*, which requires another capability than the previous sub-TSTs. When all children of N_2 are assigned, and all *propose* messages are returned, we have the schedule in Figure 6.23. The solution is presented to the operator.

The operator approves it by sending an *accept-proposal* to P_0 containing the TST N_0 . Further *accept-proposal* messages are sent to the child nodes of N_0 .

The process continues until all nodes in the TST N_0 are approved. In the nodes that have variables to update, the values from the solution are chosen and the TST updated. The execution starts from the top of the approved TST, executing the TST, following the ordering of the composite action nodes. When a leaf node is successfully executed an *inform* message is returned to the delegator. If there is a failure, a *failure* message is returned instead. When the TST is completely executed, a final *inform* message is

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:
5:		N5s ————— N5e	1:
6:		N6s — N6e N6bs — N6be	1:

(a)

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:

(b)

 Figure 6.22: The schedule after trying to assign node N_6 and after removing the conflicting assignment.

sent back to the operator indicating that the mission is completed.

Node	12:00	13:00	Platform
0:	N0s ————— N0e		0:
1,2:	N1s — N1e N2s ————— N2e		0:
3:		N3s ————— N3e	1:
4,4b,4c:		N4s — N4e N4bs — N4be N4cs — N4ce	1:
5:		N5s ————— N5e	2:
6,6b,6c:		N6s — N6e N6bs — N6be N6cs — N6ce	2:
7:		N7s ————— N7e	4:
8,8b,8c:		N8s — N8e N8bs — N8be N8cs — N8ce	4:
9:		N9s ————— N9e	3:
10,10b,10c:		N10s N10e N10bs N10be N10cs N10ce	3:
11:		N11s ————— N11e	5:
12,12b,12c:		N12s N12e N12bs N12be N12cs N12ce N12ds N12de	5:

Figure 6.23: The complete schedule when the problem is solved.

6.5 Summary

With this chapter we have shown the use of the collaborative UAS. The case studies show how different types of TSTs are allocated by AllocateTST. The delegation process relieves the operator of micro-managing all the aspects of the task allocation, but the operator can also, when necessary, give input in the form of user constraints that express the operators requirements and constraints on the mission. By adding user constraints and pre-allocating platforms to nodes, the autonomy of the platforms is restricted during task allocation and execution.

The TST of the communication relay scenario is different from previous TSTs in this thesis by its use of goal nodes. Goal nodes must be expanded so that their expansion can be allocated. In this scenario we used a special-purpose algorithm for creating a TST for a goal node, in this case for calculating a relay chain. Future work includes building a planner that creates TSTs for goal nodes in general.

Chapter 7

Performance Evaluation

In this chapter we evaluate the scalability of the task allocation algorithm AllocateTST, by applying it to different instances of the logistics task described in the UAS case studies chapter (Chapter 6). To evaluate the scalability we varied the size of the TST (the number of boxes to be delivered), the number of available platforms and the maximal allowed expected execution time for the tasks. The test cases are based on the supply delivery scenario (the TST in Figure 6.8 on page 81, without the last fly box action N_{13}). The load-deliver-unload-carrier pattern described by the sub-TST rooted in N_1 was repeated, creating the TSTs shown in Figure 7.1. Each occurrence of the pattern consists of 12 nodes and moves 4 boxes. The number of platforms available were also varied from one to four. A test case consisting of x carrier patterns and y platforms is denoted by $Cx-Py$.

AllocateTST is evaluated using both chronological backtracking and backjumping.

An important aspect of the performance evaluation is the *expected execution time*. With this term we mean the time it is expected to take to execute an allocated TST. As an allocated TST describes a schedule of all its sub-tasks, the expected execution time is the end time of the last sub-task in the schedule. For instance, the expected execution time is 1 hour for the allocation of the TST in Figure 6.11. The allocation of the TST is shown in Figure 6.23. Another important term is the *bound* on the expected execution time. With this we mean an upper limit on the permitted expected execution time for a TST which is specified before task allocation. Since constraining the total expected execution time makes it harder to find an allocation this corresponds to varying the difficulty of the allocation problem. The bound is important because in many cases a mission is only allowed to take a certain amount of time. The operator has a task that should be done and the question is, can the available platforms carry out the task within a specified time frame? The allocation with the shortest expected execution time is called the *tightest allocation*.

With *task allocation time* we mean the time needed to find an allocation.

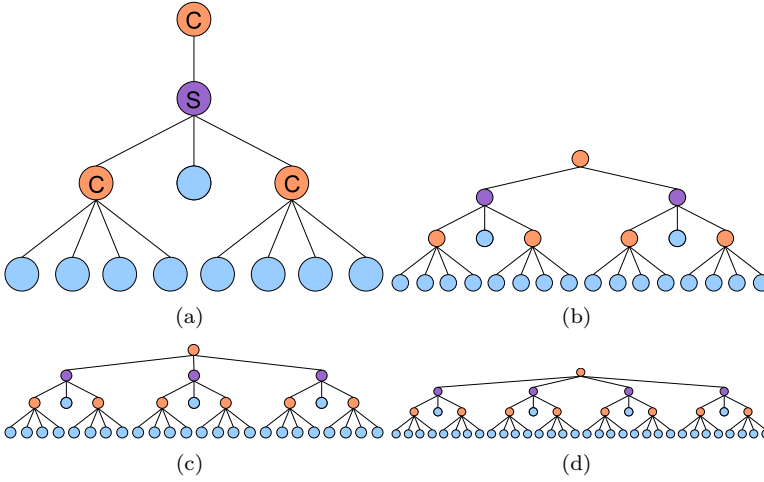


Figure 7.1: The C1, C2, C3 and C4 TSTs used in the evaluation.

7.1 Comparison Metrics

Task allocation algorithms can be compared qualitatively on how complex problems they can solve, like we compared AllocateTST to other task allocation algorithms in Chapter 4. In this chapter, we make a quantitative evaluation of the AllocateTST algorithm. Before we can evaluate the algorithm, we must decide what comparison metrics to use, i.e. which aspects of task allocation should be used to rank the allocations.

One possible metric is the task allocation time, i.e. how much time is required to find a solution. Using task allocation time has the drawback that it is dependent on the hardware and the quality and optimization of the implementation. All experiments are run on two identical laptops. When an experiment is run, the computer that hosts the experiment runs the legacy system, agent layer and associated process for all the platforms involved in the experiment. Therefore, the measured task allocation time is not an accurate measurement in an absolute sense. The task allocation time measured on the UASTech platforms would most likely be shorter since the work required would be distributed and each UASTech platform would only have to run a single legacy system. On the other hand, the computers on the UASTech platforms are less powerful, and there is an overhead in sending and receiving messages over the air. Therefore, we try to find better comparison metrics than task allocation time.

Comparison metrics that do not depend on the hardware include the

type and number of messages sent during the task allocation, the number of times the AWCS algorithm is restarted, and how many times a platform's local CSP solver is used by the AWCS algorithm. The message types are auction, delegation and constraint messages. Each message, not protocol, is counted, meaning a failed or successful delegation produces two messages, and an auction produces two messages per bidding platform. The constraint messages are produced by single-message protocols sending *Ok* and *Nogood* messages, as described by Yokoo in [101]. Only messages sent between different platforms are counted. Nodes with only one possible candidate are not auctioned out, but delegated to the candidate directly. Such cases occur when the operator has specified a platform for the task or only one platform has the required capabilities for the task. Distributed constraint satisfaction algorithms are also usually evaluated on the number and sizes of messages sent, and / or operations needed. Choosing this as a comparison metric for task allocation follows the same convention which makes it easier to compare the results with other work.

Another comparison metric candidate for task allocation problems such as ours, that does not depend on the hardware, is the expected execution time. If we use the expected execution time as a comparison metric it would be natural to try to minimize it during the task allocation. However, in our case, this is not always desirable. If the expected execution time is minimized then the expected execution time of each sub-task in the TST will also be minimized, resulting in a very tight schedule. However, for a multi-robot system, a tight schedule may result in cascading problems when an action is not executed exactly as planned, since the model was slightly inaccurate and breaking the rest of the schedule. We will therefore not try to optimize the expected execution time. For our type of missions it is common that the operator only accepts solutions within a bound, which is represented by a user constraint on the TST. That is why we want to determine if a solution can be found with an expected execution time lower than a (possibly unlimited) bound.

In summary, we will use the following as comparison metrics:

- The number of messages sent (delegation messages, constraint messages and auction messages) by the AllocateTST and AWCS algorithms.
- The number of times the AWCS algorithm is invoked and the number of times the local CSP solver is invoked, (AWCS and Local CSP invocations).

To be more exact, one could count the non concurrent constraint checks [75], but for showing the scalability of AllocateTST it is enough to count the number of times the local CSP solvers are called.

7.2 The Purpose of the Experiments

In this section we describe what we want to evaluate and why. The experiments can be divided into two parts. The first part contains scalability evaluations of AllocateTST in different circumstances. The evaluation is needed to show what size of TSTs that can be used in the collaborative UAS and how bounds affect the task allocation. In the second part we estimate the efficiency of the AllocateTST approach to the task allocation problem by comparing it to the results of an alternative DisCSP formulation and a centralized CSP formulation for the same problem. Different formulations result in slightly different versions of the problem, so the comparison will not be exact, but it will give a coarse ordering of the efficiency of the different approaches.

Each experiment class is described below.

Scalability of AllocateTST – Unbounded Allocation. The purpose of this experiment is to show the scalability of AllocateTST when increasing the TST size and number of platforms. An important feature of the experiment is that there is no bound on the expected execution time, and the results describe how AllocateTST behaves in the most beneficial circumstances. The experiment can be seen as a way to determine the baseline for AllocateTST (the performance for the most beneficial cases). We will carry out this experiment by increasing the TST size from C1 to C8 and varying the platforms from 2 to 4. Chronological backtracking and the marginal cost auction strategy is used.

Scalability of AllocateTST – Bounded Allocations. The purpose of this experiment is to show the scalability of AllocateTST when applying bounds on the expected execution time and increasing the TST size and number of platforms. The rationale behind this experiment is that in typical collaborative UAS scenarios, the operator wants a mission to be done within a certain time. The question is, can it be done in time with the available platforms? We expect that for each test case, there is a region where AllocateTST will have a hard time finding a solution, and that will be around the bound on expected execution time for the tightest allocation. We want to determine how AllocateTST behaves for such bounds for different TSTs.

Note that the experiment is not about finding the tightest allocation, but to find *a* solution, given a bound. It is an evaluation of how the hardness of the problem depends on the bound. Chronological backtracking is compared to backjumping, and the auction strategy used is the marginal cost auction strategy.

Compare AllocateTST to a Centralized Approach. The purpose of this experiment is to compare AllocateTST to solving a centralized CSP formulation of the task allocation problem. With this experiment we want to determine an upper bound for the performance of

AllocateTST. We also want to show that it is the features of the problem itself rather than the distributed solving method that makes the problem hard. Like the unbounded and bounded experiments with AllocateTST, the centralized CSP formulation is expressed in the Tailor constraint format.

Compare AllocateTST to a Different DisCSP Approach. A straightforward DisCSP approach to the task allocation problem is to take the constraints and variables of the centralized formulation and distribute them. Each variable would then be owned by a virtual agent and the constraints would span between the agents. The purpose of this experiment is to show that our approach with AllocateTST is vastly better. With this experiment we also want to determine a baseline for AllocateTST.

The constraint code for this experiment is expressed in the FRODO format [70]. The FRODO constraint code is created by translating the Tailor constraint code for the centralized formulation to FRODO code. Thus, a different constraint solver is used in this experiment.

7.3 Scalability of AllocateTST – Unbounded Allocation

In the first experiment, the size of the TST and the number of available platforms is varied. The number of carriers is varied between 1 and 11 (C1-C11), corresponding to between 4 and 44 boxes and between 13 and 133 nodes. A carrier pattern consist of 43 Tailor variables, meaning the test cases contain between 45 and 475 Tailor variables (the concurrent top-node contributes 2 variables to each test case). The Tailor code in turn is translated to Minion code that contains additional auxiliary variables. The number of platforms is varied between 2 and 4. For each combination, the total number of exchanged messages is counted when the algorithm allocated the TST to the available platforms using chronological backtracking. The invocations (of AWCS and local CSP) are also counted.

There is no bound on the expected execution time of the task. Since there is no bound, there is no backtracking and no retries in the AllocateTST algorithm (but the underlying DisCSP algorithm can of course backtrack). The number of messages sent are shown in Figure 7.2. The experiment shows that it is possible to allocate TSTs of size C10 consisting of 121 nodes. The allocations fails for C11-P2, C12-P3 and C12-P4 due to exhausted memory.

The expected execution time of the solution found by AllocateTST divided by the expected execution time of the tightest allocation is shown in Figure 7.3. For the simple cases when there is only one platform, AllocateTST finds the tightest allocation, which is expected. For the remaining test cases, AllocateTST finds solutions whose expected execution time is at

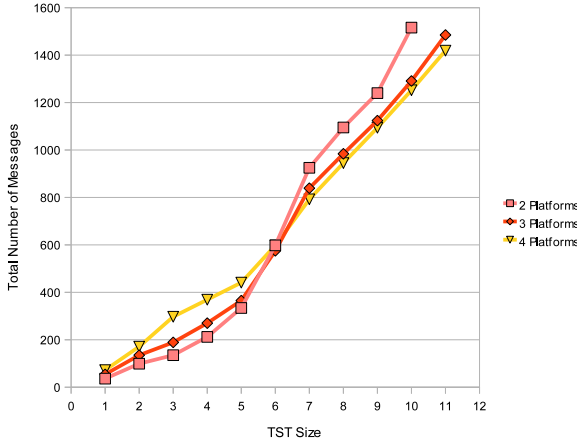


Figure 7.2: Total number of messages sent for each group of platforms, when increasing the size of the TST. Chronological backtracking is applied, but not used since no bound.

most 2.16 times the shortest possible execution time and most cases are not more than 1.5 times as long. The results show that the fewer platforms, the closer is AllocateTST to the tightest allocation, because fewer platforms means less possibilities for making suboptimal choices. For some cases, the results are somewhat better when the number of carrier patterns and platforms are easily divisible, such as for test case C2-P2, C4-P2 and C3-P3. For such cases it is often beneficial to allocate a whole sub-TST to one platform.

7.4 Scalability of AllocateTST – Bounded Allocation

To evaluate how AllocateTST scales when the problem becomes harder, we apply different bounds to a TST and measure the constraint solving activity and messages sent for each bound. As mentioned before, an allocated TST describes a schedule of all the tasks in a TST. The expected execution time for the tightest allocation for each test case is shown in Table 7.1. Adding a bound on permitted expected execution time for the task makes the problem much harder to solve. It is not possible to find a shorter solution than the shortest expected execution time, and in many cases it is very hard to find a solution yielding exactly the tightest bound. The intuition is that the more constrained the schedule is, the fewer solutions there are and the more important the choices of allocations are. With a very tight bound, even a slightly suboptimal choice may result in extensive backtracking.

Each test case is evaluated with a range of bounds covering cases where

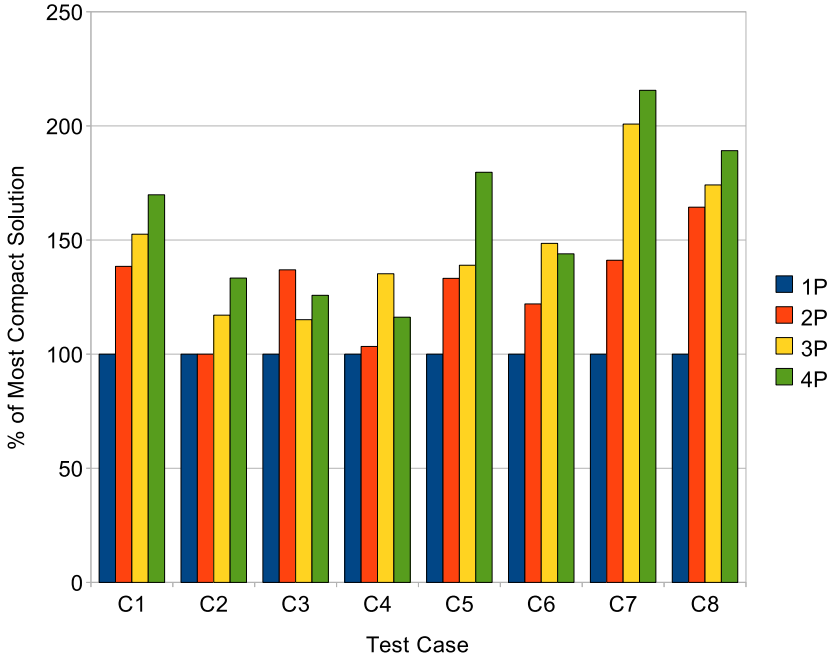


Figure 7.3: The ratio between the expected execution time of the solution found by AllocateTST and the expected execution time of the tightest allocation. Chronological backtracking is applied, but not used since no bound.

there is no solution until increasing the bound further makes no impact on the time required for finding a solution. There are 10 time steps between each measuring point for the bound on expected execution time. Both chronological backtracking and backjumping are used and the result is compared for each test case.

The results are shown in three types of graphs, each comparing chronological backtracking and backjumping. The first graph shows the number of messages, the second shows the number of times the AWCS algorithm is invoked and the third the number of times the local CSP solvers are invoked. The expected results for both messages and constraint solving activity, is hill-formed graphs, both increasing until the point where the first solution is found and then rapidly decreasing afterwards. The initial part of this shape is expected because the number of messages and the constraint solving activity will increase as larger and larger (but still too low) bounds allow the task allocation algorithm (which uses a depth-first search method) to process more of the TST before failing. From the point where the first solution is found, the number of messages and constraint solving activity should decrease for each increase in the bound, as more and more assignments re-

Scenario	P1	P2	P3	P4
C1	90	65	59	53
C2	189	96	82	72
C3	317	157	106	97
C4	427	208	142	105

Table 7.1: The shortest expected execution time (time steps) for the tightest allocation for the 16 test cases.

sult in valid allocations. The results for bound values around the bound of the tightest allocation corresponds to a phase transition in constraint satisfaction problems, the transitions from a problem without solutions to a problem where one solution can be found.

In the experiments either chronological backtracking or backjumping are used. The auction strategy used is marginal cost. The results are presented in several sections, starting with a section for the test-cases containing only one platform, followed by a section for each of the test cases C1-P2, C1-P3 and C1-P4, and finally a section for the remaining test cases. The results are presented in this way because the C1-P2 – C1-P4 test cases are the smallest but still interesting test cases, where more than one platform causes choices in the task allocation problem, and are therefore studied in detail. The remaining test cases show similar results and are therefore grouped into one section. The results of the Cx-P1 test cases are of less importance but are still included to show how the algorithm works in such special cases.

Only one platform can be coordinator in this experiment. The reason for this is that having multiple platforms with the coordinator capability, will only lead to additional backtracking when no solution exists. This is because the coordinator task is platform-independent. Trying different platforms for a platform independent task will not have any affect on the end result but for TSTs that can not be allocated this will lead to more costly backtracking, because more configurations must be tested. A solution to this problem could be to extend AllocateTST with the ability to identify symmetries and incorporate this information in the candidate selection method. As mentioned before, our priorities here are first to evaluate the basic version of AllocateTST before making any optimizations for handling special cases. Improving the algorithm by breaking symmetries is left to future work.

7.4.1 Performance for Bounded Allocation of Cx-P1

The four test cases C1-P1 to C4-P1 are special cases in that they only use one platform. No messages will be sent and there are no choices between candidates. The problem is purely centralized, since all constraints will end up on one platform. Following the AllocateTST algorithm, the constraints for each (task-to-platform) allocation are added to the DisCSP Node on the

platform, node by node. If the bound is too tight, the allocation fails and not all nodes can be allocated. In such cases backtracking takes additional time, until all previously allocated nodes are unallocated. Since there is only one platform, each step in the backtracking will lead to another backtracking step, until all allocations are removed. This is the reason the allocation time decreases somewhat for the case when the solution is found (at 90 for C1-P1, at 189 for C2-P1, at 317 for C3-P1 and at 427 for C4-P1), and no deallocation is needed (see Figure 7.4). The time lines fall into each other because the larger test cases contains the smaller test cases, i.e., C1 is part of C2, C2 is a part of C3 etc. In this and in the following figures, the bound on expected execution time yielding the tightest allocation are marked out with black vertical lines.

AllocateTST could be improved so that it returns directly when there are no more platforms to try. This is a minor optimization that is left for future work. Since there are no messages sent between different platforms in this experiment we measure the task allocation time instead.

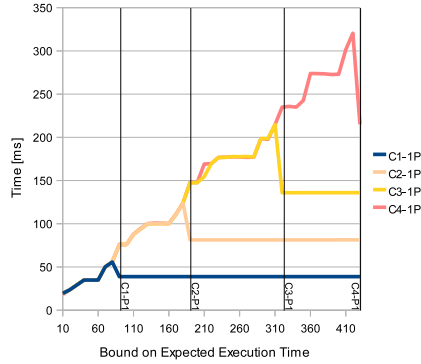


Figure 7.4: The task allocation time for C1–C4 with one platform and varied bounds. Chronological backtracking is used in the experiments.

7.4.2 Performance for Bounded Allocation of C1-P2

In this experiment we study how AllocateTST behaves when the bound on the expected execution time is varied while the size of the TST is fixed to C1 and the number of available platforms is fixed to 2. The bound is varied between 10 and 100. When the bound is lowered the constraint problem becomes harder and there are fewer solutions. The tightest allocation, with respect to the total expected execution time used to complete the task, is 65. This means that for bounds less than 65 there is no solution. The algorithm has its hardest time at 60, where despite extensive backtracking no solution can be found. As shown in Figure 7.5, the messages needed are less at the

bound of the tightest allocation (65), than for 70, and 80. The reason for this is that with a higher bound, more allocations can be tested in the sub-TST rooted in the second concurrent node (the unloading of the carrier), before AllocateTST backtracks to the sub-TST rooted in the first concurrent node (the loading of the carrier) and makes the necessary change needed for the solution. With a bound of 65, AllocateTST does not get the chance to test as many allocations in the sub-TST rooted in the second concurrent node, which speeds up the process of finding the solution. With a bound of 90, the bound has no effect since the marginal cost strategy directly finds such an allocation (in this case giving all the nodes to platform P_0). In the tightest allocation both platforms, P_0 and P_1 , load the carrier with two boxes each and also unload the carrier together. AllocateTST will be guided by the marginal cost strategy to allocate all the nodes to the platform that is allocated the first load-box-to-carrier node (since this platform is then the closest). The tightness of the bound will then determine how much the other platform must be used. As the bound increases the algorithm ends up using relatively few messages to find an acceptable solution. The number of messages sent and the constraint solving activity needed are shown in Figure 7.5. For the sake of clarity, only the total number of messages are shown in Figure 7.5(a). The different message types adding up to the total number of messages for C1-P2 using backjumping are shown in Figure 7.5(d). The proportions of the message types are similar in the remaining test cases.

Chronological backtracking uses more messages than backjumping as shown in Figure 7.5. The difference is mainly for smaller bounds. For such cases, backjumping is able to determine that no allocation is possible, because no platform can be allocated to the unallocated task that initiated the backtracking, even if all previous tasks are considered unallocated. The graphs for the constraint solving activity follows the same pattern as for the messages.

7.4.3 Performance for Bounded Allocation of C1-P3

In this experiment we study how AllocateTST behaves when the bound on the task is varied while the size of the TST is fixed to C1 and the number of available platforms is fixed to 3. The bound is varied between 10 and 100. For the test case C1-P3, the first solution is at 59. The solution is obtained when platforms P_1 and P_2 load and unload one box each from the carrier and platform P_0 handles the remaining two boxes.

The expected result is a hill-shaped graph. The results are in Figure 7.6(a), Figure 7.6(b) and Figure 7.6(c). As expected, the results show that the greatest effort is needed when the bound is less than the lowest expected execution time (at 59). The graphs are similar to those in Figure 7.6, but higher, which is expected because there is one more platform to try.

Similar to C1-P2, chronological backtracking uses more messages than backjumping, mostly in the beginning where the effect is enhanced by the in-

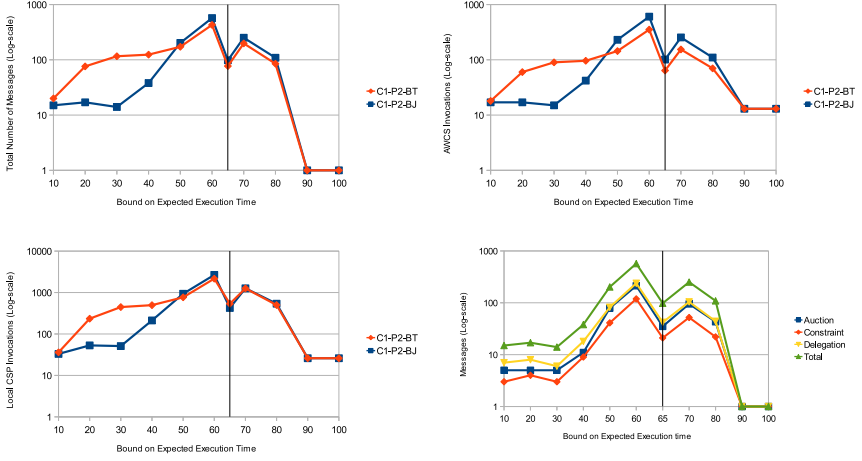


Figure 7.5: For chronological backtracking and backjumping, the number and types of messages sent (top-left) and the constraint solver activity (top-right, bottom-left) for allocating the C1 TST to 2 platforms when the bound on the expected execution time to complete the task varies. Note the log-scale for invocations and messages. The last graph (bottom-right) shows the composition of messages for the backjumping case.

crease in available platforms. After the bound of the tightest allocation (59), backjumping uses slightly more messages than back-tracking, because only minor consecutive changes are needed in the later part of the configuration and for such cases backjumping does more operations than backtracking, basically backjumping follows the same path as backtracking.

7.4.4 Performance for Bounded Allocation of C1-P4

In this experiment we study how AllocateTST behaves when the bound on the task is varied while the size of the TST is fixed to C1 and the number of available platforms is fixed to 4. The bound is varied between 10 and 100. The first solution is at 53 for the C1-P4 test case. Similar to test case C1-P3, the tightest allocation is found when each platform loads one box onto the carrier, and unloads one box of the carrier. The tightest allocation can only be found after backtracking because an effect of using the marginal cost as the auction strategy, in this case, is that all tasks are allocated to platform P_0 . When platform P_0 is allocated to the first task, it is also the best candidate for the next task because when platform P_0 has loaded the first box onto the carrier it is also closer to the next box to be moved. Therefore, P_0 will win the next auction too, and so on. This problem occurs for other test cases too, but is illustrated best with the C1-P4 test case, whose tightest allocation uses all four platforms concurrently. Since there are more platforms available to test, the constraint solving activity and the

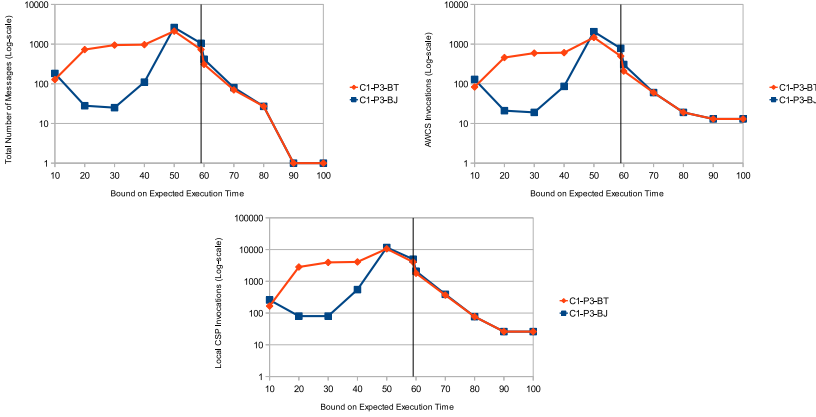


Figure 7.6: For chronological backtracking and backjumping, the number and types of messages sent (top-left) and the constraint solver activity (top-right, bottom) for allocating the C1 TST to 3 platforms when the bound on the expected execution time varies. Note the log-scale for invocations and messages.

messages sent increases compared to the C1-P2 and C1-P3 test cases. The increase is tenfold in the constraint solving activity, showing that the number of platforms is an important factor when no allocation can be found. The results are in Figure 7.7(a), Figure 7.7(b) and Figure 7.7(c).

Chronological backtracking uses more messages than backjumping for smaller bounds because of the previous mentioned reason. The results are similar to C1-P3, but the effects are enhanced both before and after the bound of the tightest allocation. The reasons are the same as for C1-P3.

7.4.5 Performance for Bounded Allocation of C2-P2 – C4-P4

The results for the remaining detail studies are similar, but more time- and resource-consuming, using more messages and more constraint solver and AWCS invocations. It is important to note that a solution can be found fast if the bound is not too tight. As the bound on expected execution time increase and surpasses the bound for the tightest allocation, less messages and constraint solving activity is needed to find an allocation.

The results of the remaining detail studies are in Figure 7.8, Figure 7.9 and Figure 7.10. The gaps in the graphs represent that it takes more than 18000s to find a solution due to the fact that those bounds are requiring too much time for backtracking. The graphs have similar characteristics as the previous detail studies, but magnified, which was also expected.

Similar to test cases C1-P2, C1-P3 and C1-P4, chronological backtracking uses more messages than back-jumping in the beginning, where back-

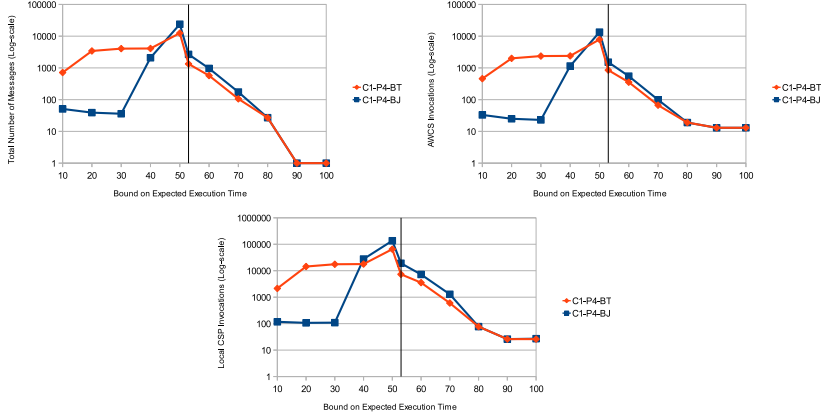


Figure 7.7: For chronological backtracking and backjumping, the number and types of messages sent (top-left) and the constraint solver activity (top-right, bottom) for allocating the C1 TST to 4 platforms when the bound on the expected execution time varies. Note the log-scale for invocations and messages.

jumping can cancel the allocation when the failed task can not be allocated even on its own. For larger bounds, backjumping can not cancel a failed allocation attempt, because with larger bounds even the later tasks can be allocated on their own. Chronological backtracking and backjumping uses similar amount of messages for such cases.

For bounds where the backjumping algorithm can not determine if the entire allocation is impossible, the number of messages needed are similar or more than for backtracking.

7.4.6 Discussion – Bounded Allocation Results

The results for the bounded allocations experiments show that the messages sent and constraint solving activity needed for finding an allocation or determining that none exists, increases very fast towards the time point where the allocation with the tightest schedule is found. After this time point, the number of messages sent and constraint solving activity levels out as more and more solutions can be found. As shown in figures 7.4–7.10, the computations are most expensive around the bound. The experiments also show that a more efficient backtracking method or auction strategy is needed for larger TSTs, when no solutions can be found. Improvements in the auction strategy is left as future work.

How many platforms and how large TSTs the task allocation algorithm can handle depends very much on how tight the bound on the expected execution time for the problem is. Figure 7.11, Figure 7.12 and Figure 7.13 (on page 108) show how the number of messages and constraint solving

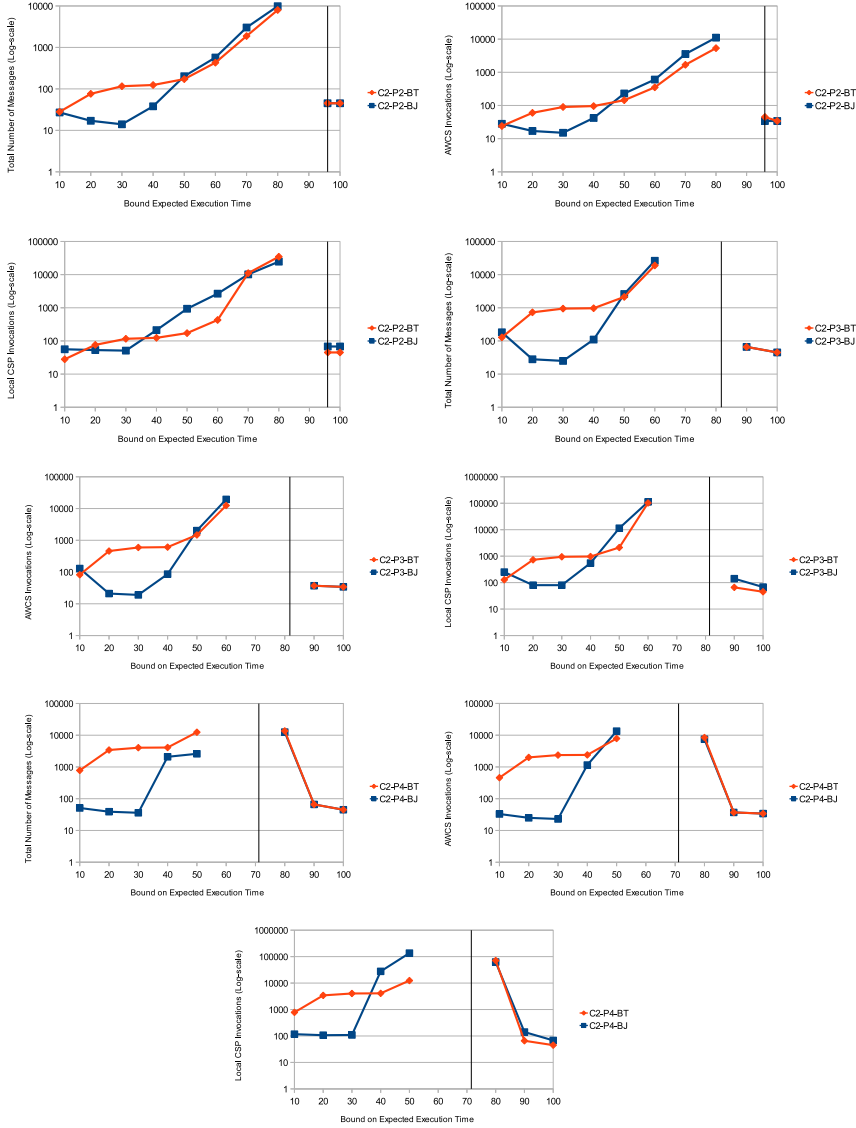


Figure 7.8: For chronological backtracking and backjumping, the constraint solver activity and the number of messages sent when allocating the C2 TST with 2–4 platforms and varied bounds. Note the log-scale for invocations and messages.

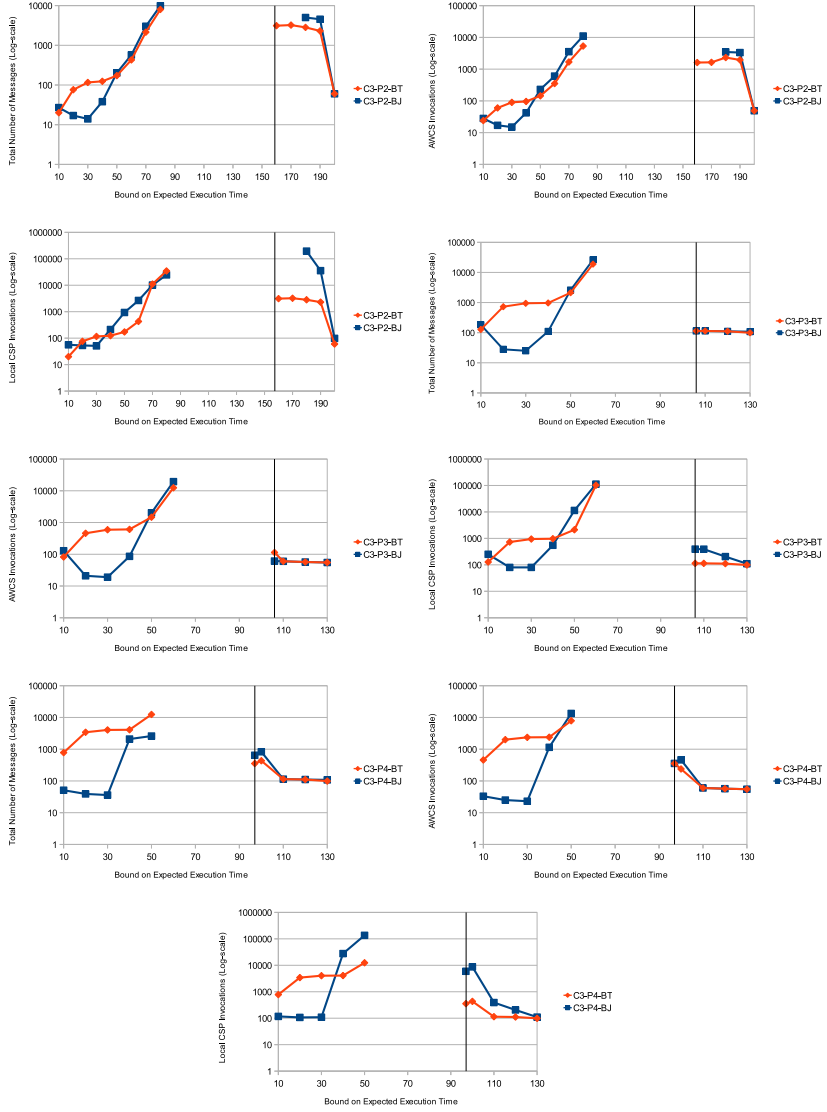


Figure 7.9: For chronological backtracking and backjumping, the constraint solver activity and the number of messages sent when allocating the C3 TST with 2–4 platforms and varied bounds. Note the log-scale for invocations and messages.

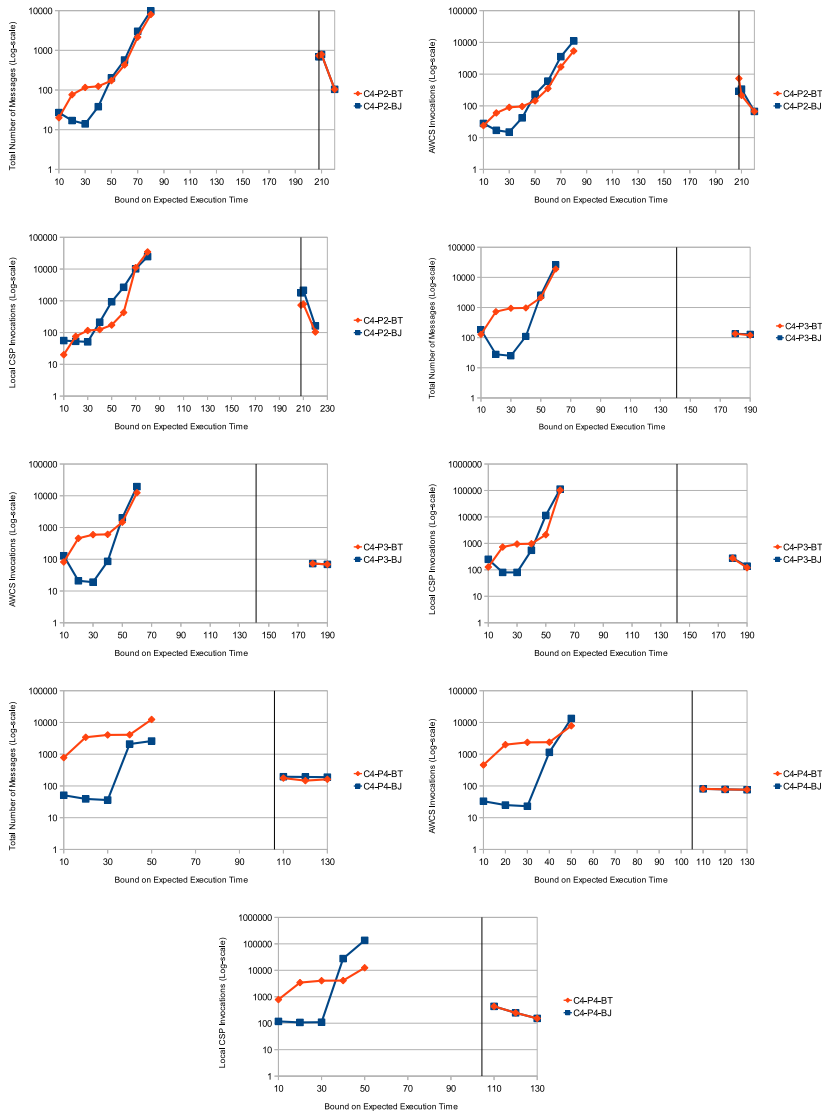


Figure 7.10: For chronological backtracking and backjumping, the constraint solver activity and the number of messages sent when allocating the C4 TST with 2–4 platforms and varied bounds. Note the log-scale for invocations and messages.

activity increases, when the bound is tightened (when using backjumping). The x-axis shows the percentage ratio between the bound on the expected execution time and the expected execution time of the tightest allocation. We call this ratio the *compactness*. The messages sent and the constraint solver activity increases sharply as the bound on the expected execution time gets closer to the bound of the tightest allocation. The results for backtracking is similar.

The marginal cost strategy can produce suboptimal allocation suggestions for the child nodes of a concurrent node in some cases. In some cases it is beneficial to strive for maximal concurrent execution for the child nodes of a concurrent node (allocate to different platforms), sometimes it is not. For example compare test cases C1-P4 and C4-P4. In C1-P4, the tightest allocation is found when each platform loads and unload one box on the carrier. For the C4-P4 case, the tightest allocation is found when each platform is responsible for one carrier pattern each. Improving the auction strategy to take into account the platforms' already committed tasks and the time left to the bound (if any) would likely also reduce the search for cases when there is no solution.

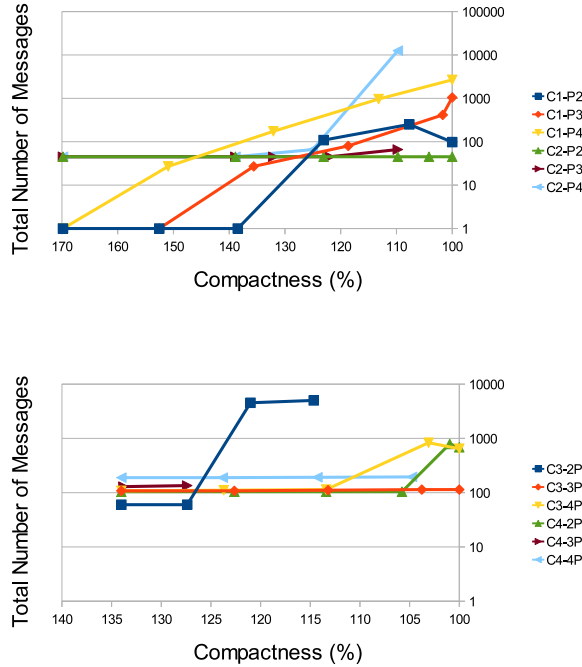


Figure 7.11: The number of messages sent when tightening the bound for the test cases.

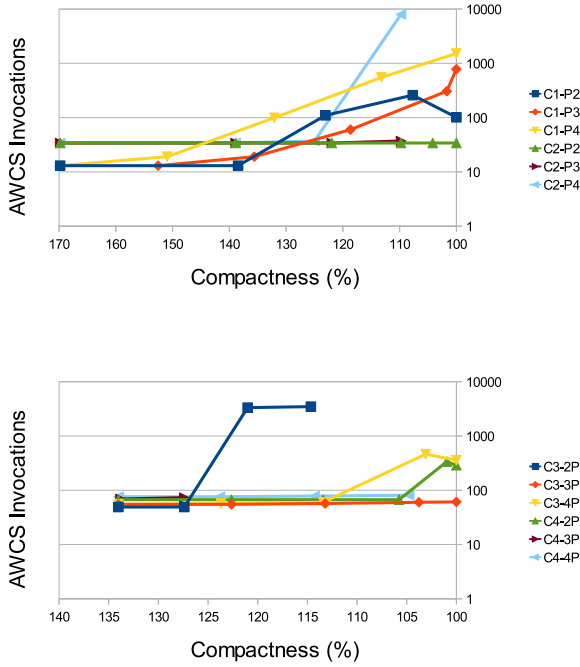


Figure 7.12: The number of AWCS invocations when tightening the bound for the test cases.

The resource model of a TST node consists of many variables and constraints. Each node is a complete task and can be seen as a constraint problem by itself. Usually, a TSTs has also interrelated utilities, where previous individual allocations affect further allocations, which occurs when there are more elementary action nodes than platforms. With this as background, it is good result that AllocateTST can handle TSTs with up to 10 nodes without a bound and about 25 nodes with bounds of at most 110% compactness. For TSTs with up to 25-50 nodes, the algorithm can find solutions with at most 130% compactness (for some cases, 100% compactness). It should also be noted that DisCSP solving algorithms, such as the AWCS used by AllocateTST, can usually handle fewer variables than their centralized counter parts. In Section 7.6 we will also show that existing DisCSP approaches can not even solve the $C1 - Py$ test cases.

One could think that having many platforms would be a limiting factor. Whether this is the case or not depends on the situation. Having many platforms to choose from will often make the problem easier to solve, if there is no bound on the TST and a solution exists. If there is no solution, the number of platforms will lead to more choices that must be tried during

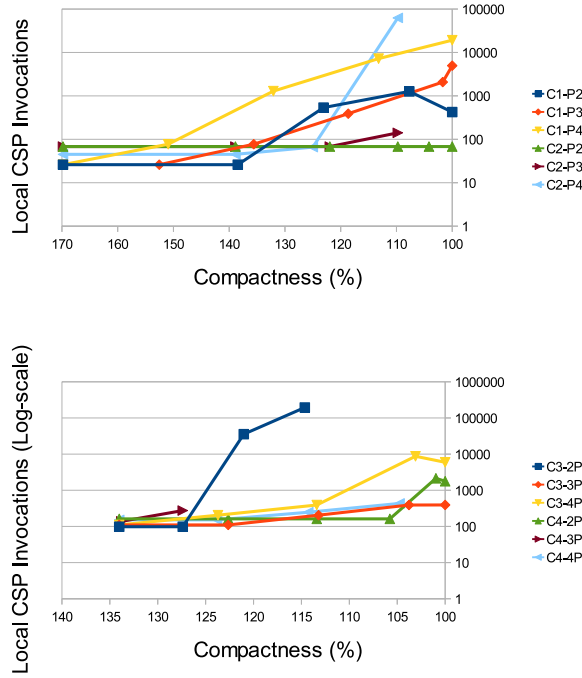


Figure 7.13: The number of local CSP invocations when tightening the bound for the test cases.

backtracking. The same is true for TSTs with bounds when backtracking is needed to find a solution. How the number of platforms influences the performance should rather be seen as a relation between the number of nodes and the bound on the expected execution time. With too few platforms, the problem can quickly be detected as unsolvable. With too many platforms, the problem can be solved without much backtracking. The same is true for the bound. A very low bound leads to less backtracking before it is discovered that no solution exists, compared to the case when the bound is only slightly lower than the expected execution time of the tightest allocation. The auction strategy has the most impact on the cases where the bound is slightly above the shortest expected execution time for the available platforms to allocate the TST. In such cases there is a solution and a good auction strategy will select the candidates in such a way that a solution is found without extensive backtracking. The platforms must be allocated in the right order so that the allocated TST fits into the time frame expressed by the bound.

In summary, the AllocateTST algorithm can handle task specification trees for TSTs containing up to 50 nodes and finds solutions with at least

130% compactness, when applying bounds. For large TSTs, and when having many platforms, it is important to note that a tight bound can lead to very expensive computations. Improving the auction strategy might improve the results. A problem with the marginal cost strategy is that it only considers the extra cost for the platform and not for the team. In cases with child nodes of a concurrency node, it is sometimes better to assign the nodes to different platforms, regardless of the marginal cost.

7.5 A CSP Formulation

An alternative formulation of the task allocation problem is to represent all possible allocations as a single centralized CSP. The formulation has variables representing which platform a task is allocated to together with constraints describing that a task can only be allocated to one platform. The formulation also includes constraints describing that distances depend on what other tasks a platform is committed to. AllocateTST does not have such constraints because it incrementally adds the constraints for each task and platform combination. In AllocateTST, the platform to task assignments are completely outside the DisCSP, with no variables representing task assignments, whereas in the centralized CSP formulation, those variables are in the formulation and are determined by the CSP solver. In AllocateTST, the adding / removing mechanism assures that only one platform is allocated to a task. A side effect of this is that the distances can be calculated when the constraints of a task are added because the platforms previous commitments are determined. The centralized CSP formulation thus requires more variables and constraints than AllocateTST does, but AllocateTST needs to run the constraint solver more than once, at least as many times as there are nodes in the TST.

The CSP formulations uses the following types of variables:

X_S, X_E : Variables for start and end time of an interval. X is the name of some action, for instance: $C_LoadC_1_S$, means the start time for the concurrent node containing the sub-task of loading carrier 1. The corresponding end time is $C_LoadC_1_E$.

$X_dist_unloaded, X_dist_loaded$: Variables for distances a platform must move in some action described by X . For example $flyC_1_dist_unloaded$, meaning the distance to the position of the carrier 1. Unloaded means before the carrier is loaded. The opposite is $flyC_1_dist_loaded$, meaning the distance to fly with the carrier when loaded with boxes.

$X_speed_nobox, X_speed_wbox$: Variables for the speed of a platform when flying and holding nothing, and when flying and holding a box.

$flyC_1, loadC_1B_1$: Task selector variables. The value of a variable represents which platform (by platform number) that is assigned to the task.

The example variables means the platform that will deliver carrier 1, respectively the platform to load carrier 1 with box 1.

There are also a few constants in the CSP formulations, for example *LOADBOX_TIME* meaning the time needed to pick up a box and *MAX_SPEED* meaning the maximum speed of a platform.

The constraints in the CSP formulation are of the following types: *structural constraints*, *action constraints*, *booking constraints* and *distance constraints*. Examples are shown below:

structural constraints Constraints of this type describe the relations between a task's start and end times as specified by a TST. This group of constraints represents the constraints of composite actions, such as the constraints of the sequence node (described on page 25).

Example: $C_LoadC_1_S \leq C_LoadC_1_E$

action constraints Constraints of this type describe how the end time of an elementary action depends on the start time and the time needed for the activity of the elementary action. The example below shows the action constraints of moving a box. The end time depends on the start time, the distance to fly to the box that should be picked up, the time required to pick up the box, the distance to fly to the destination from the place where the box is picked up, and the speed of the platform. This group of constraints represents the constraints for elementary action nodes, and the following example is similar to the constraints in the example on page 28.

Example:

$$\begin{aligned} loadC_1B_1_E = & loadC_1B_1_S + \\ & (loadC_1B_1_dist_unloaded/loadC_1B_1_speed_nobox) + \\ & LOADBOX_TIME + (loadC_1B_1_dist_loaded/loadC_1B_1_speed_wbox) \end{aligned}$$

booking constraints Constraints of this type describe the fact that actions allocated to the same platform may not overlap if they use the same resources. A booking constraint thus assures that a mutual exclusive resource is not used concurrently. In the cases when a booking constraint forces child nodes of a concurrent node to be executed in sequence (because they belong to the same platform and uses a mutual exclusive resource), the nodes have the same order as the child nodes of a sequence node, i.e. a child node *A* that is to the left of another child node *B*, means that *A* comes before *B* in time. The resource itself is not explicitly represented by a variable but is implicitly represented by task selector variables in a booking constraint.

Example:

$$(loadC_1B_1 = loadC_1B_2) \rightarrow (loadC_1B_1_E \leq loadC_1B_2_S)$$

distance constraints Constraints of this type describe the distance needed to move a platform before or during a task for a particular platform and task, given the other tasks the platform is committed to. In the CSP formulation, all possible routes are specified in this manner.

Example:

$$\begin{aligned} & ((flyC_1 = 1) \wedge (flyC_1 \neq loadC_1B_1) \wedge \\ & (flyC_1 \neq loadC_1B_2) \wedge (flyC_1 \neq loadC_1B_3) \wedge \\ & (flyC_1 \neq loadC_1B_4)) \rightarrow (flyC_1_dist_unloaded = 36) \end{aligned}$$

The constraints describe that if platform P_1 (the value 1) is assigned to the task of flying with the carrier $flyC_1$ and not to any of the other actions mentioned in the constraint, the distance for flying to the position of the carrier will be 36 meters.

An instance of the centralized CSP formulation can be illustrated with the following example (see Figure 7.14). For example, a structural constraint describes that t_{s_1} must begin before t_{e_1} and that t_{e_3} is before or equal to t_{e_1} . The structural constraints build up the schedule represented by the TST. Action constraints involve the start and end time variables t_{s_2} , t_{e_2} , t_{s_3} and t_{e_3} . There are two action constraints per elementary action node (one for each platform) and they are similar to the action constraint in the example above. In the example, a booking constraint describes that if the same platform is allocated to both of the elementary action nodes (N_2 and N_3), the elementary actions must be in sequence, i.e. $t_{e_2} \leq t_{s_3}$. There are two such booking constraints for the example, since both platforms can take on both N_2 and N_3 . Distance constraints describe how the distance a platform must move when allocated to a task depends on the platform's previous commitments. Allocating a platform to node N_3 creates different distances, depending on what other tasks the platform has committed to. For this particular case, two distances are possible for a platform, depending on if it moves from the area where the first box was loaded (the platform is also allocated to node N_2) or if it moves directly from its start position. In Figure 7.15, the distances are marked for platform P_0 .

The number of constants, variables and constraints needed for the centralized formulation are shown in Table 7.2. We solved the task allocation problem as a centralized problem, using the same CSP solver as used on each platform by AllocateTST. We compare how much time is needed to find the tightest allocation for various TST sizes and number of platforms.

Figure 7.16 and Figure 7.17 show the time needed to find the tightest allocation (and all other allocations) for the test cases C1-P1 – C8-P4. Each experiment is given 18000 seconds (5 hours) to complete. A bar of 18000 seconds (from bottom to top in the graph) means that it could not be proven that the most tight allocation found is the tightest allocation. For the smaller TSTs C1–C2 it is possible to determine the tightest allocation in a very short time. Among the test cases using the TST sizes C1–C4, only C4-P4 was not completed within 18000s. The number of platforms is a

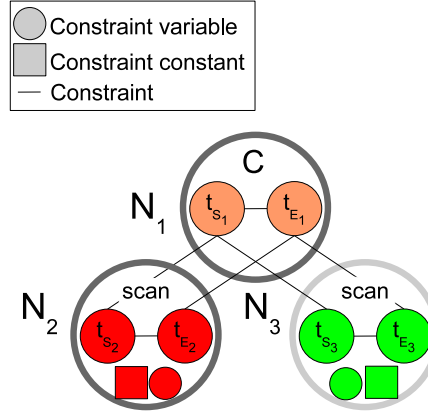


Figure 7.14: A TST describing a concurrent scan operation.



Figure 7.15: Provided that P_0 is assigned to N_3 , the distance P_0 has to move to reach N_3 depends on P_0 's previous commitments. If P_0 is allocated to N_2 , there is one distance (d_1) for P_0 when the platform is also allocated to N_3 . If N_3 is platform P_0 's first allocation, the distance is counted from the platform's start position instead (d_2). The distance constraints describe such distance rules for all possible platforms and task combinations.

	constants	variables	constraints
C1-P1	48	22	129
C1-P2	35	35	138
C1-P3	35	35	147
C1-P4	35	35	156
C2-P1	87	43	417
C2-P2	62	68	435
C2-P3	62	68	453
C2-P4	62	68	471
C3-P1	126	64	867
C3-P2	89	101	894
C3-P3	89	101	921
C3-P4	89	101	948
C4-P1	165	85	1479
C4-P2	116	134	1515
C4-P3	116	134	1551
C4-P4	116	134	1587

Table 7.2: The number of constants, variables and constraints for each test case in the CSP formulation.

significant factor because it determines the number of choice constraints (of platforms for tasks) and the number of variables in those constraints that have to be evaluated. The special case with only one platform is simple because the problem contains no constraints describing choices. For larger TSTs (C5-P1–C8-P4), it is not possible to find and determine the tightest allocation in 18000s, when using more than one platform.

Figure 7.16 and Figure 7.17 only shows the time needed to determine the tightest allocation. We also evaluated the time needed to find each allocation for twelve of the sixteen test cases. The test cases with only one platform were excluded, because there is only one allocation for each of them, which makes them less interesting in this experiment. By increasing the allowed task allocation time more allocations can be found. The experiments show that finding an allocation is not hard, but finding the tightest allocation is, which can be compared to the tests with AllocateTST where a decreasing bound permits fewer and fewer allocations which becomes increasingly costly to find.

Each allocation is plotted as a relation of a percentage value and the time it takes to find the allocation. The percentage value is the expected execution time of the tightest allocation divided by the expected execution time of the allocation. The experiments show for each allocation and test case the time it takes to find an allocation and how close the found allocation is to the tightest allocation.

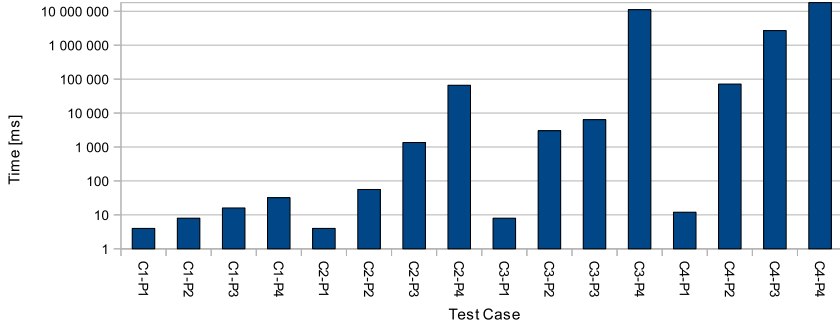


Figure 7.16: The time needed to find the tightest allocations for test cases C1-P1 – C4-P4. Note the log-scale for time.

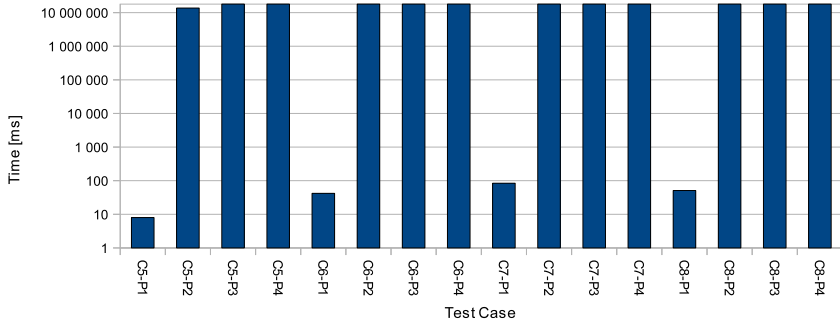


Figure 7.17: The time needed to find the tightest allocations for test cases C5-P1 – C8-P4. Note the log-scale for time.

For instance, see Figure 7.18. The graph shows that the more platforms there are, the longer time it takes to determine the tightest allocation and the longer time it takes to find an allocation of a certain expected execution time. This value is expressed as compactness, which is the ratio between the bound on the expected execution time and the expected execution time of the tightest allocation. A solution with 200% compactness for test case C3-P2 can be found in 5 milliseconds, for the C3-P3 and C3-P4 it takes about 100 milliseconds respectively 200 milliseconds to get the same compactness.

The results for the rest of the test cases are displayed in Figure 7.19, one graph per TST size. The time needed for finding all solutions for a particular number of platforms when varying the size of the TST is shown in Figure 7.20, which is another way of organizing the same results shown in the previous figure.

The centralized approach shows that the task allocation problem is a

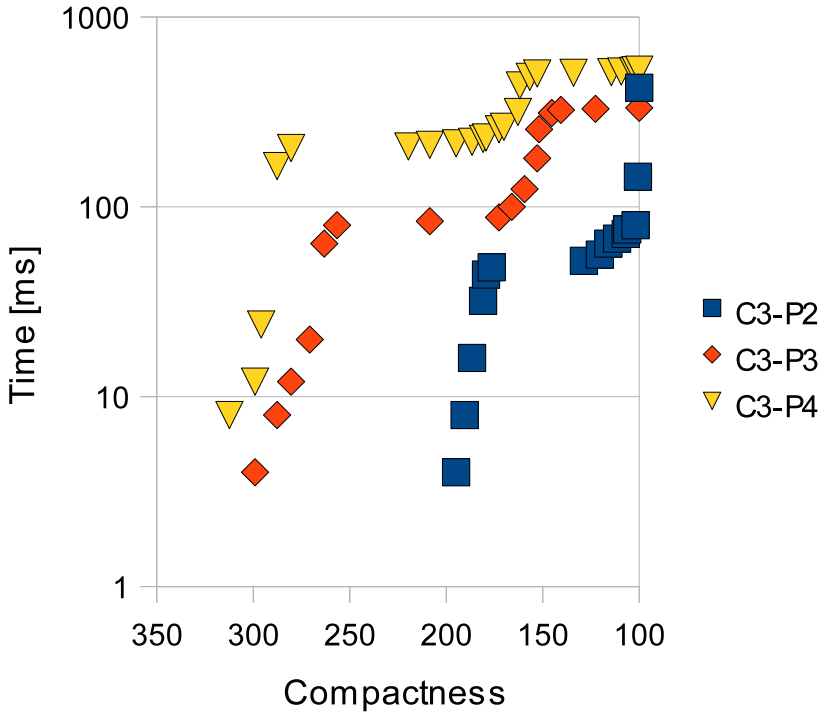


Figure 7.18: The time needed for finding all solutions for TST size C3, when varying the number of platforms P2–P4. Note the log-scale for time.

hard and costly problem to solve when searching for the tightest allocation, meaning that the decentralization of the problem (in `AllocateTST`) is not the only factor making the problem hard. Finding the tightest allocation for TSTs in the test case range C1–P1–C2–P4 was possible in a relatively short time for the centralized approach. Most test cases were solved around 1 second (see Figure 7.16). For the C3–P1–C4–P4 test cases, it takes longer time, up to hours to determine if the tightest allocation is found (see Figure 7.16). The number of platforms is a significant factor because it determines the number of choice constraints (of platforms for tasks) and the number of variables in those constraints that has to be evaluated. The special case with only one platform is simple because the problem contains no constraints describing choices. It was not possible to find the tightest allocation for many of the larger test cases (C5–P1–C8–P4) in 5 hours time. Figures 7.16 and Figures 7.17 shows the time needed to determine if the most tight allocation found is the tightest allocation, just finding an allocation takes very little time, as shown in Figures 7.19–7.20.

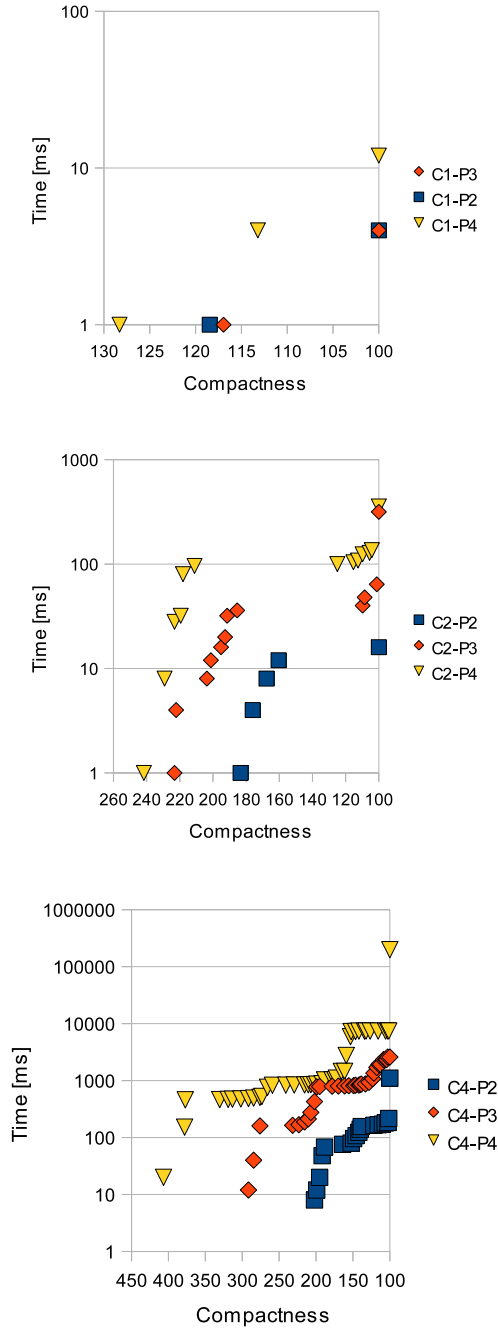


Figure 7.19: The time needed for finding all solutions for TST size C1–C4, when varying the number of platforms P2–P4. Note the log-scale for time.

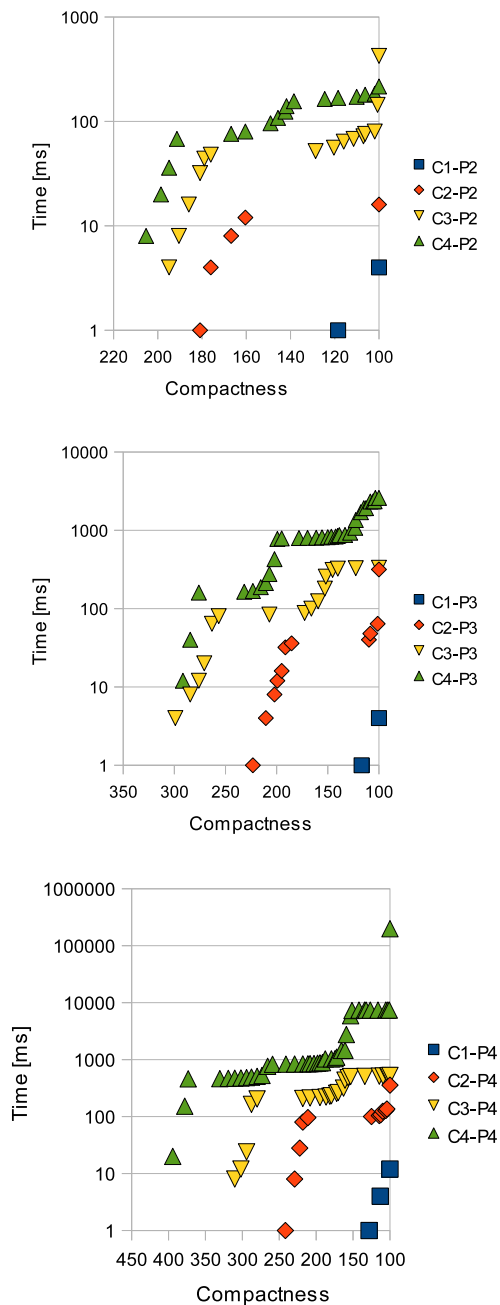


Figure 7.20: The time needed for finding all solutions for a particular number of platforms (two, three or four platforms) when varying the size of the TST (C1–C4). Note the log-scale for time.

7.6 An Alternative DisCSP Formulation

Another alternative problem formulation is created when taking the centralized formulation and reformulating it as a DisCSP. In this case, each variable in the DisCSP will be owned by a virtual agent. The constraints from the centralized case will then become global constraints, i.e. constraints between different agents. As in the centralized formulation, the DisCSP formulation has constraints describing that a task can only be allocated to one platform. The difference compared to the DisCSP approach taken with AllocateTST is that AllocateTST adds the constraints for each task and platform combination that is tested to the DisCSP and only keeps the constraints if the DisCSP is consistent. If it is inconsistent, the constraints of the combination are removed and another allocation is tried. In the alternative problem formulation, we will never add or remove constraints: everything is in the DisCSP from the beginning. We expect this approach to perform worse and it can be seen as a baseline for our approach.

The characteristics of the alternative DisCSP formulation are shown in Table 7.6. To get any results, we had to simplify the test case. A test case is denoted by $Bx-Py$ meaning x boxes are moved by y platforms, in a simplified version of the C1 TST.

It should be noted that in these experiments, each variable in the constraint problem is held by one agent, which makes the problem more distributed than AllocateTST, even if the problem is of the type $Bx-P1$. It is usually the case that one agent owns one variable in DisCSP and DisCOP algorithms. For the alternative DisCSP formulation, there is no clear mapping between platforms and virtual agents. The virtual agents are simulated on a single computer and they have nothing to do with the distribution among platforms. It is actually not obvious how the problem can be distributed without reformulating the DisCSP. How should the global selector constraints be distributed over the platforms when they are not known in advance? In AllocateTST on the other hand, the platform – agent mapping is straightforward because we use a DisCSP algorithm where one agent can own many local variables. In AllocateTST the content of a conditional branch, containing the constraints of a task and platform combination, belongs to the platform in question.

We evaluated the alternative DisCSP approach using the ADOPT (Asynchronous Distributed OPTimization) [77], DPOP (Distributed Pseudotree Optimization Procedure) [82] and Synch-BB (Synchronous Branch and Bound) [64] algorithms, in the FRODO framework [70]. When these algorithms are applied to the full problem they do not find a solution within reasonable time. Therefore, we had to decrease the size of the test TST to get any results. We used a TST similar to the C1 TST in the logistics scenario, but with only 1, 2 or 3 boxes. The distances were scaled down with a factor of 5 and the maximum speed was also scaled down with a factor of 2. We had to decrease those values because the size of the domains influences the perfor-

mance and the problem could not be solved without decreasing the domains. The task allocation time is shown in the following tables, for ADOPT see Table 7.3, for DPOP see Table 7.4 and for SynchBB see Table 7.5. We use task allocation time as a measurement instead of counting messages since the latter causes problems with exhausted memory. The '-' means that the algorithm could not find a solution.

With this approach to the task allocation problem only very small TSTs can be handled (1-2 boxes and 2-4 platforms). This is fewer boxes than in the C1-Px test cases, which shows that the approach taken with AllocateTST is better. The differences in performance can probably be traced to the following factors. In the alternative DisCSP formulation, the DisCSP contains more constraints than what AllocateTST must handle. The selector constraints describing the different choices of platforms and tasks must be explicit in this formulation, compared to AllocateTST, where this is handled by the algorithm by adding or removing the constraints of individual task allocations to the problem. With this mechanism, AllocateTST solves a conditional DisCSP, where the conditional variables are platform / task selectors, and determines which platform and task combinations that should be included in the problem. AllocateTST then solves a sequence of simpler problems. In AllocateTST, the allocation time is further decreased because each platform solves its local problem in a centralized manner, and does not use one virtual agent per variable as in the alternative DisCSP formulation. In AllocateTST, a new individual allocation (a platform to a task) is done with the accumulated knowledge of previous allocated platforms, and there is no need for complex distance constraints as there is in the alternative formulations.

Another reason is that the solver for the alternative DisCSP formulation has no problem-specific heuristic such as our marginal cost heuristic to guide which constraints to evaluate next. The solver also lacks efficient constraint propagation and a domain dependent variable ordering heuristic.

	1 Platform	2 Platforms	3 Platforms	4 Platforms
1 Box	2.2s	6.4s	7s	8s
2 Boxes	492s	-	-	-
3 Boxes	-	-	-	-

Table 7.3: Task allocation time with ADOPT.

7.7 Discussion

In this chapter we have evaluated the scalability of the task allocation algorithm AllocateTST on TSTs with and without bounds. We have also evaluated two alternative approaches to the task allocation problem, a centralized

	1 Platform	2 Platforms	3 Platforms	4 Platforms
1 Box	15s	479s	740s	-
2 Boxes	-	-	-	-
3 Boxes	-	-	-	-

Table 7.4: Task allocation time with DPOP.

	1 Platform	2 Platforms	3 Platforms	4 Platforms
1 Box	0.6s	1.3s	1.7s	1.6s
2 Boxes	825s	148s	2608s	240s
3 Boxes	-	-	-	-

Table 7.5: Task allocation time with SynchBB.

	Domains	Values	Variables	Rel.	Tup.	Cons.
1B-P1	3	42	20	6	564	29
1B-P2	5	46	50	18	1818	62
1B-P3	5	47	56	20	1864	71
1B-P4	5	48	61	21	1887	79
2B-P1	3	46	28	9	1163	48
2B-P2	5	49	93	21	2689	123
2B-P3	5	50	104	23	2741	139
2B-P4	5	51	113	24	2767	153
3B-P1	3	48	36	9	1359	71
3B-P2	5	49	148	23	2881	204
3B-P3	5	50	164	25	2933	227
3B-P4	5	51	177	26	2959	247

Table 7.6: The number of domains, values, variables, relations, tuples in the relation and constraints in the test cases for the DisCSP formulation.

problem formulation and an alternative DisCSP problem formulation. Since AllocateTST and the alternative approaches use different comparison metrics, task allocation time for the alternative approaches and sent messages and constraint solving activity for AllocateTST, it is not possible to make an exact comparison between the different approaches. However, we can make a coarse comparison. A coarse ordering of the different approaches, from best to worst, is the centralized CSP formulation, followed by the AllocateTST and last the straightforward DisCSP formulation. We are not interested in the centralized formulation as an approach to the task allocation problem for the following reasons. Usually, the reasons for using a DisCSP approach over a CSP approach is that the time to gather the data for the CSP is great, because each part of the problems is large, or because

the gathering will have to be redone for each change in the problem. For our case, changes to a platform's state, resources, etc., affect the platform's local constraints for the task allocation problem. All such changes will not necessarily lead to changes that must be conveyed to other platforms. However, with a centralized solution, all such changes would lead to a resolving of the entire problem. Another common reason for the DisCSP approach is privacy and security concerns [102]. The agents do not want to send their internal constraints for privacy reasons. One of the ideas with the TST approach is to keep the resource constraints hidden. The only connection point is the node interface variables in the TST. With this setup, the platforms can even have different constraint representations of their resource constraints and different local solvers as long as they can participate in the DisCSP algorithm.

The centralized problem is useful though as a baseline comparison. The results show that the approach taken for the AllocateTST algorithm is better than the alternative DisCSP formulation. It was also shown that it is the features of the problem itself rather than the distributed solving method that makes the problem hard (comparing the results of AllocateTST and the centralized CSP formulation with tight bounds).

Without bounds, AllocateTST can allocate TSTs with up to 130 nodes, containing up to 450 tailor variables and using 2 to 4 platforms. With help of the marginal cost strategy, AllocateTST finds allocations that are at most 2.16 times the shortest possible execution time and in most cases not more than 1.5 times as long. The better cases occur when the number of carrier patterns and platforms are easily divisible.

With bounds, the results show that the effort to find an allocation increases towards the bound of the tightest allocation. As the bound on expected execution time increases and surpasses the bound for the tightest allocation, fewer messages and less constraint solving activity is needed to find an allocation. The result also shows that the messages sent and the constraint solver activity increases sharply as the bound on the expected execution time gets closer to the bound of the tightest allocation. The AllocateTST algorithm can handle task specification trees for TSTs containing up to 50 nodes, and find solutions with at most 130% compactness, when applying bounds.

Chapter 8

Conclusions

8.1 Summary

The problem of determining who should do what given a set of tasks and a set of agents is called the task allocation problem. The problem occurs in many multi-agent system applications where tasks somehow should be distributed to a number of agents. In our case, the task allocation problem occurs as an integral part of a larger problem of determining if a task can be delegated between two agents. In order to solve this problem, three related concepts must be determined. The task specification format used to express the tasks, the multi-agent system infrastructure needed to enable delegations, and finally the delegation problem itself (and its inherit task allocation problem). The delegation concept is used in a multi-robot system of UAVs, interacting with one or more operators. Since a platform only can carry a limited payload, efficient resource usage becomes an issue when delegating tasks.

In this thesis we have developed a task specification formalism (*task specification trees*), a complex task allocation algorithm for allocating the tasks in a task specification tree to UAV platforms (*AllocateTST*), and a generic collaborative system shell for robotic systems (*an extension of the FIPA Abstract Architecture*). Each part is the answer to a research question involved in the ambition to make an earlier proposed collaborative robotic framework concrete [36, 40]. The framework builds on three fundamental, interdependent conceptual issues: delegation, mixed-initiative interaction and adjustable autonomy. The concept of delegation is particularly important and in some sense provides a bridge between mixed-initiative interaction and adjustable autonomy. A speech act for delegation has previously been formalized with pre- and post-conditions specified in the KARO formalism, which 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 formalization is inspired by the work of

Falcone & Castelfranchi on delegation [10, 46].

Delegation is the basis for the collaborative framework. In a realization of the framework, the agents work as delegators and contractors of complex tasks. There is no fixed agent hierarchy, instead the agent forms collaboration structures depending on how they can contribute with their capabilities to the solution of the delegated task.

The delegation theory and the proposed collaborative robotic framework only provide an abstract definition of a task, and do not describe exactly how the pre- and post-conditions of delegation can be fulfilled. The first step in our work towards a concretization of the collaborative robotic framework was to connect the *Can* predicate of the speech act to a representation of the tasks and the agents capabilities for such tasks. In this process we identified three central problems that must be solved before the collaborative robotic framework can be realized.

The first problem is stated as the research question R1 – “R1 - How can a task τ in the delegation theory be specified to make the realization of the delegation speech act possible?” In addressing the problem of representing tasks, we introduced a task specification formalism, called the TST formalism. With this formalism it is possible to specify distributed task-structures, called task specification trees that can be delegated. The formalism can be used to express complex tasks suitable for different types of collaborative UAS missions. The constraint-based task trees make it possible to realize a form of adjustable autonomy – where the contractor’s autonomy can be restricted by the delegator but not changed by the contractor.

We also described a model, a platform specification, for how platforms use their static and dynamic resources when carrying out a task.

The second problem is stated as the research question R2 – “How can the preconditions of the delegation speech act be assured, so that a task τ specified according to R1 can be delegated?” This is the core of the delegation process, how do we find contractors that actually have the capabilities and resources for the task that should be delegated? The problem of finding and allocating platforms to tasks is an instance of a complex task allocation problem, a problem which turns out to be very hard to solve for realistic missions given constraints on time and resources. In Section 4.3, we extended the multi-robot task allocation classification introduced by Gerkey and Mataric [59] with four new dimensions and argued that allocating task specification trees is more challenging than most allocation problems currently considered.

The problem of allocating TSTs to robot platforms was defined and a distributed heuristic algorithm for finding a consistent allocation was presented. The algorithm recursively searches among the potential allocations in a distributed manner and uses distributed constraint satisfaction techniques to check if an allocation satisfies the constraints.

The third problem is stated as the research question R3 – “How can R1 and R2 be realized on a collaborative UAS?” A collaborative robotic

shell was developed as an extension of the FIPA Abstract Architecture. The extension includes functionality to support the delegation of tasks and the concept of a platform specification for describing how a platform carries out tasks. A platform specification describes how a platform fulfils the *Can* predicate in the delegation speech act for a given task. The specification offers a more sophisticated capability and resource model than the one provided by the FIPA Abstract Architecture, which only describes what services a platform provides. With the extended model we can describe how resources are used and the time needed for carrying out a task. An implementation in JADE [92] of the collaborative robotic shell was used to evaluate the system.

A number of case studies was presented, describing how the task allocation algorithm works in the collaborative UAS. We show how the operator can add user constraints to a TST to restrict the task allocation and thereby the autonomy of platforms during the allocation of a task specification tree.

The task allocation algorithm was evaluated on task specification trees from the case studies. The experiments show that the algorithm is capable of allocating relatively large TSTs, and that the number of platforms does not significantly influence the number of messages sent when there is no bound on the available time for the execution of the allocated task. They also show that user constraints, such as bounds on the time to complete a task, is a highly significant factor.

We also formulated the task allocation problem as a centralized problem and as an alternative DisCSP, where each constraint variable is handled by a virtual agent. The centralized problem formulation shows that the task allocation problem is a complex problem, even without the extra complexity brought by the distributed approach to the problem. As we expected the alternative DisCSP formulation performs worse than the approach taken with AllocateTST. The DisCSP approach only managed to solve a sub-set of the test cases solved with AllocateTST. One reason for the discrepancy in performance between AllocateTST and the alternative DisCSP formulation can be attributed to the lack of constraint propagation rules and variable order heuristics in the DisCSP algorithms used in this experiment. But it is probably also because AllocateTST solves a conditional CSP, where different instances of the problem are solved depending on the choices of platforms and tasks. The problem solved by AllocateTST is thus smaller than in the alternative approaches. AllocateTST can also be helped by its marginal cost heuristics.

8.2 Future Work

The collaborative UAS that is the outcome of the work in this thesis is a prototype of the conceptual robot framework described in Chapter 1–2. The prototype should be seen as an intermediate step towards a complete realization of the conceptual framework.

Most aspects of the collaborative UAS can be improved. In this section we focus on the parts that were developed in the thesis and tied to the research questions R1–R3. For research question R2, the task allocation algorithm AllocateTST can be improved in its *auction strategy*, in its *pre-processing of the TST*, and in its *constraint solving algorithm*.

The task allocation algorithm can be improved by developing other candidate orderings methods. One improvement would be to take into account both how many other tasks the platform is committed to and how much time that is left before the bound and integrate the information with the marginal cost bidding strategy. For example, the platforms could report the end time of all the tasks they are committed to and this could be weighed together with the bids to avoid serialization of concurrent tasks. Such a heuristic might be better than only using marginal cost.

Another improvement would be to extend AllocateTST with the ability to identify symmetries and incorporate this information in the candidate selection method. Identifying and avoiding symmetrical configurations would save much effort, especially during backtracking, when fewer configurations need to be tested.

Another related question is whether it is possible to decompose a task specification tree and allocate the parts in parallel. A TST can easily be decomposed if there are no interrelated utilities between the decomposed parts. A TST with interrelated utilities might be decomposed by introducing more constraints on the decomposed parts. It would then be interesting to see if it is possible to design a candidate selection method that creates decomposable TSTs during the task allocation. One way is to minimize the width of the resulting DisCSP tree, which is achieved by allocating adjacent nodes to the same platform so that the number of regions in the TST that the platform is allocated to is minimized. In general, algorithms and tools that can be used to pre-process the TST before the task allocation starts and give additional information to the candidate selection method are interesting issues. Examples are analysis of how the area the tasks in the TST cover and how the sub-areas can be grouped together to be allocated to a single platform.

Another way of improving AllocateTST is in the choice of DisCSP algorithm. An important question is how the DisCSP algorithm can be specialized and made more efficient for our type of task allocation problem? Since the allocated TST creates a schedule it might be beneficial to integrate functionality for constraint-based scheduling, for instance a local solver with special scheduling constraints. Another related issue is the type of constraint problem solving that can be used for the TST allocation problem. Can the problem be solved more efficiently if represented and solved as an integer programming problem? This question was to a lesser degree studied in Chapter 4, but needs more work before we can rule out this approach to the problem.

For research question R1, the task specification format can be improved

by adding more composite action nodes, such as the selector and loop nodes and their corresponding *Can()* predicates. But it makes most sense to focus on R2 and improve the task allocation algorithm before adding more TST nodes that our algorithm must handle. For research question R3, future work depends on what extensions are made to R1 and R2.

8.3 Conclusions

In the beginning of this thesis three research questions were stated:

- R1 - How can a task τ in the delegation theory be specified to make the realization of the delegation speech act possible?
- R2 - How can the preconditions of the delegation speech act be assured, so that a task τ specified according to R1 can be delegated?
- R3 - How can R1 and R2 be realized on a collaborative UAS?

A task specification format (TST) was constructed as an answer to research question R1. With its help it is possible to specify complex multi-agent tasks that can be delegated. The task specification tree is a form of compromise between an explicit plan in a plan library and a plan generated through an automated planner [68]. We need this flexibility to express adjustable autonomy and open and close delegation. The task allocation format serves its purpose and is useful for expressing real-world collaborative UAS missions, which was demonstrated with a number of case studies.

A delegation process and a task allocation algorithm *AllocateTST* was developed as an answer to the second research question R2. Task allocation is a part of delegation because delegating a task includes the problem of finding who can carry out the task. For complex tasks, this problem may include relations between sub-tasks, and between the agents that are to be assigned the sub-task, turning the task allocation problem into a complex task allocation problem. *AllocateTST* was evaluated on a number of realistic collaborative UAS tasks. It was shown that *AllocateTST* can allocate task specification trees with up to 130 nodes, which cover the TST sizes of typical collaborative UAS scenarios. For TSTs restricted by user constraints, the task allocation algorithm can be costly in the cases when the bound is set to a value lower than expected execution time of the tightest allocation because of excessive and fruitless backtracking. Still, the *AllocateTST* gave better results than the alternative approach.

A collaborative robotic shell was developed as an answer to research question R3. The shell extends the FIPA Abstract Architecture and contains delegation functionality and a representation of the platforms capabilities and resources. The extension has a more sophisticated service model than the original FIPA service model, which is necessary for our type of resource-bounded scenarios. The collaborative robotic shell was implemented in

JADE [92] forming an agent wrapper around the UASTech platforms' legacy systems. The viability of the system was shown with a number of case studies and a performance evaluation of the implemented task allocation algorithm AllocateTST.

With the task specification format, the task allocation algorithm AllocateTST, and the extension of the FIPA Abstract Architecture we have realized a prototype for the proposed collaborative UAS framework.

Bibliography

- [1] R. Alami and S. C. Botelho. Plan-Based Multi-robot Cooperation. In *Revised Papers from the International Seminar on Advances in Plan-Based Control of Robotic Agents*, pages 1–20, 2002.
- [2] R. Alami, F. Ingrand, and S. Qutub. A Scheme for Coordinating Multi-robot Planning Activities and Plans Execution. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, pages 617–621, 1998.
- [3] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [4] F. Arni, K. Ong, S. Tsur, H. Wang, and C. Zaniolo. The Deductive Database System LDL++. *Theory and Practice of Logic Programming (TPLP)*, 3(1):61–94, 2003.
- [5] Y. Bartal, J. W. Byers, and D. Raz. Fast, Distributed Approximation Algorithms for Positive Linear Programming with Applications to Flow Control. *SIAM J. Comput.*, 33:1261–1279, 2004.
- [6] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE – a Java agent DEvelopment framework. In J. Dix R. H. Bordini, M. Dastani and A. Seghrouchni, editors, *Multi-Agent Programming - Languages, Platforms and Applications*, pages 125–147. Springer, 2005.
- [7] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley and Sons, Ltd, 2007.
- [8] S. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1234–1239, 1999.
- [9] O. Burdakov, P. Doherty, K. Holmberg, and P-M. Olsson. Optimal placement of UAV-based communications relay nodes. *Journal of Global Optimization*, 48:511–531, 2010.

- [10] C. Castelfranchi and R. Falcone. Towards a Theory of Delegation for Agent-Based Systems. *Robotics and Autonomous Systems*, 24:141–157, 1998.
- [11] K. M. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3:63–75, 1985.
- [12] P. Cohen and H. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42(3):213–261, 1990.
- [13] P. Cohen and H. Levesque. Teamwork. *Nous, Special Issue on Cognitive Science and AI*, 25(4):487–512, 1991.
- [14] Zeev Collin, Rina Dechter, and Shmuel Katz. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*, 1999(115), 1999.
- [15] G. Conte and P. Doherty. Vision-based Unmanned Aerial Vehicle Navigation Using Geo-referenced Information. *Journal of Advances in Signal Processing (EURASIP)*, pages 10.1–10.18, 2009.
- [16] G. Conte, M. Hempel, P. Rudol, D. Lundström, S. Duranti, M. Wzorek, and P. Doherty. High Accuracy Ground Target Geolocation Using Autonomous Micro Aerial Vehicle Platforms. In *Proceedings of the AIAA-08 Guidance, Navigation, and Control Conference*, 2008.
- [17] J. M. Corchado, D. I. Tapia, and J. Bajo. A Multi-Agent Architecture for Distributed Services and Applications. *Computational Intelligence*, 2009.
- [18] L. O. Bonino da Silva Santos, F. Ramparany, P. Dockhorn Costa, P. Vink, R. Etter, and T. H. F. Broens. A Service Architecture for Context Awareness and Reaction Provisioning. *2007 IEEE Congress on Services*, pages 25–32, 2009.
- [19] M. Dastani and J-J. C. Meyer. A Practical Agent Programming Language. In M. P. Papazoglou M. Dastani, K. V. Hindriks and L. Sterling, editors, *Proceedings of the AAMAS07 Workshop on Programming Multi-Agent Systems (ProMAS2007)*, pages 72–87, 2007.
- [20] E. Davis and L. Morgenstern. A First-Order Theory of Communication and Multi-Agent Plans. *Journal of Logic and Computation*, 15(5):701–749, 2005.
- [21] S. de Vries and R. Vohra. Combinatorial Auctions: A Survey. *Journal on Computing*, 15(3):284–309, 2003.

- [22] K. Decker. TAEMS: A Framework for Environment Centered Analysis and Design of Coordination Mechanisms. *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448, 1996.
- [23] M. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-Based Multirobot Coordination: a Survey and Analysis. *Proceedings of IEEE, Special Issue on Multirobot Coordination*, 94(1):1257–1270, 2006.
- [24] P. Doherty. Artificial Intelligence & Integrated Computer Systems Division (aiics). <http://www.ida.liu.se/divisions/aiics>.
- [25] P. Doherty. UAS Technologies Sweden AB. www.uastech.com.
- [26] P. Doherty. Advanced Research with Autonomous Unmanned Aerial Vehicles. In *Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning*, pages 731–732, 2004. Extended abstract for plenary talk.
- [27] P. Doherty. Knowledge Representation and Unmanned Aerial Vehicles. In *Proceedings of the IEEE Conference on Intelligent Agent Technology (IAT 2005)*, pages 9–16, 2005.
- [28] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS Unmanned Aerial Vehicle Project. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 747–755, 2000.
- [29] P. Doherty, P. Haslum, F. Heintz, T. Merz, T. Persson, and B. Wingman. A Distributed Architecture for Intelligent Unmanned Aerial Vehicle Experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems*, pages 221–230, 2004.
- [30] P. Doherty, F. Heintz, and D. Landén. A Delegation-Based Collaborative Robotic Framework. In *Proceedings of Collaborative Agents – REsearch and development (CARE)*, 2011.
- [31] P. Doherty, F. Heintz, and D. Landén. Delegation-Based Architecture for Collaborative Robotics. In *D. Weyns and M.-P. Gleizes (Eds.): Agent Oriented Software Engineering (AOSE) 2010, LNCS 6788*, 2011.
- [32] P. Doherty and J. Kvarnström. TALplanner: A Temporal Logic Based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence*, 30:119–169, 2001.
- [33] P. Doherty and J. Kvarnström. TALplanner: A Temporal Logic Based Planner. *Artificial Intelligence Magazine*, pages 95–102, Fall Issue 2001.

- [34] P. Doherty and J. Kvarnström. Temporal Action Logics. In *The Handbook of Knowledge Representation*, chapter 18, pages 709–757. Elsevier, 2008.
- [35] P. Doherty, J. Kvarnström, and F. Heintz. A Temporal Logic-based Planning and Execution Monitoring Framework for Unmanned Aircraft Systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.
- [36] P. Doherty, J. Kvarnström, F. Heintz, D. Landén, and P-M. Olsson. Research with Collaborative Unmanned Aircraft Systems. In *Proceedings of the Dagstuhl Workshop on Cognitive Robotics*, pages 1–14, 2010.
- [37] P. Doherty, D. Landén, and F. Heintz. 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.
- [38] P. Doherty, W. Lukaszewicz, and A. Szałas. Approximative Query Techniques for Agents with Heterogenous Ontologies and Perceptual Capabilities. In *Proceedings on the 7th International Conference on Information Fusion*, pages 459–468, 2004.
- [39] P. Doherty, W. Lukaszewicz, and A. Szałas. Communication between Agents with Heterogeneous Perceptual Capabilities. *Journal of Information Fusion*, 8(1):56–69, January 2007.
- [40] P. Doherty and J-J. C. Meyer. Towards a Delegation Framework for Aerial Robotic Mission Scenarios. In *Proceedings of the 11th International Workshop on Cooperative Information Agents*, pages 5–26, 2007.
- [41] P. Doherty and P. Rudol. A UAV Search and Rescue Scenario with Human Body Detection and Geolocalization. In *Proceedings of the 20th Australian joint conference on Advances in artificial intelligence*, AI’07, pages 1–13, 2007.
- [42] B. Dunin-Keplicz and R. Verbrugge. *Teamwork in Multi-Agent Systems*. Wiley, 2010.
- [43] S. Duranti, G. Conte, D. Lundström, P. Rudol, M. Wzorek, and P. Doherty. LinkMAV, A Prototype Rotary Wing Micro Aerial Vehicle. In *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace*, 2007.
- [44] E. H. Durfee and V. R. Lesser. In Les Gasser and Michael N. Huhns, editors, *Distributed artificial intelligence: vol. 2*, chapter Negotiating task decomposition and allocation using partial global planning, pages 229–243. Morgan Kaufmann Publishers Inc., 1990.

-
- [45] E. H. Durfee and V. R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:1167–1183, 1991.
 - [46] R. Falcone and C. Castelfranchi. The Human in the Loop of a Delegated Agent: The Theory of Adjustable Social Autonomy. *IEEE Transactions on Systems, Man and Cybernetics—Part A: Systems and Humans*, 31(5):406–418, 2001.
 - [47] B. Faltings and M-G. Santiago. Open Constraint Satisfaction. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 356–370, 2002.
 - [48] Foundation for Intelligent Physical Agents. FIPA Abstract Architecture Specification. <http://www.fipa.org>.
 - [49] Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. <http://www.fipa.org>.
 - [50] Foundation for Intelligent Physical Agents. FIPA Communicative Act Library Specification. <http://www.fipa.org>.
 - [51] Foundation for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification. <http://www.fipa.org>.
 - [52] J. Frank, A. K. Jónsson, and P. H. Morris. On Reformulating Planning as Dynamic Constraint Satisfaction. In *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation, SARA '02*, pages 271–280, 2000.
 - [53] A. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel. The Design of ESSENCE: A Constraint Language for Specifying Combinatorial Problems. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 80–87, 2007.
 - [54] D. Gale. *The Theory of Linear Economic Models*. McGraw-Hill Book Company, Inc., 1960.
 - [55] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A Fast Scalable Constraint Solver. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI) 2006*, pages 98–102, 2006.
 - [56] I. P. Gent, I. Miguel, and A. Rendl. Tailoring Solver-Independent Constraint Models: A Case Study with Essence’ and Minion. In *Proceedings of the 7th International conference on Abstraction, reformulation, and approximation (SARA2007)*, pages 184–199, 2007.
 - [57] B. Gerkey. *On Multi-Robot Task Allocation*. Phd thesis, University of Southern California, 2003.


- [58] B. Gerkey and M. J. Matarić. Sold!: Auction Methods for Multirobot Coordination. *IEEE Transactions on Robotics and Automation*, pages 758–768, 2001.
- [59] B. Gerkey and M. J. Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.
- [60] F. Heintz and P. Doherty. DyKnow: A Knowledge Processing Middleware Framework and its Relation to the JDL Fusion Model. *Journal of Intelligent and Fuzzy Systems*, 17(4):335–351, 2006.
- [61] F. Heintz and P. Doherty. DyKnow Federations: Distributing and Merging Information Among UAVs. In *Proceedings of Eleventh International Conference on Information Fusion (FUSION-08)*, pages 1199–1205, 2008.
- [62] F. Heintz, J. Kvarnström, and P. Doherty. A Stream-Based Hierarchical Anchoring Framework. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 5254–5260, 2009.
- [63] F. Heintz, J. Kvarnström, and P. Doherty. Bridging the Sense-Reasoning Gap: DyKnow - Stream-Based Middleware for Knowledge Processing. *Journal of Advanced Engineering Informatics*, 24(1):14–25, 2010.
- [64] K. Hirayama and M. Yokoo. Distributed Partial Constraint Satisfaction Problem. In *Proceedings of Third International Conference on Principles and Practice of Constraint Programming (CP-97)*, pages 222–236, 1997.
- [65] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161:89–115, 2005.
- [66] A. K. Jónsson and J. Frank. A Framework for Dynamic Constraint Reasoning using Procedural Constraints. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pages 93–97, 2000.
- [67] N. Kaldra, D. Ferguson, and A. Stentz. Hoplitēs: A Market-Based Framework for Planned Tight Coordination in Multirobot Teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1170–1177, 2005.
- [68] J. Kvarnström and P. Doherty. Automated Planning for Collaborative Systems. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1078–1085, 2010.

-
- [69] D. Landén, F. Heintz, and P. Doherty. 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.
- [70] T. Léauté, B. Ottens, and R. Szymanek. FRODO 2.0: An Open-Source Framework for Distributed Constraint Optimization. In *Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09)*, pages 160–164, 2009. <http://liawww.epfl.ch/frodo/>.
- [71] T. Lemaire, R. Alami, and S. Lacroix. A Distributed Tasks Allocation Scheme in Multi-UAV Context. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3622–3627, 2004.
- [72] V. R. Lesser and D. D Corkill. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine*, 4(3):15–33, 1983.
- [73] D. C. MacKenzie, R. Arkin, and J. M. Cameron. Multiagent Mission Specification and Execution. *Autonomous Robots*, 4(1):29–52, 1997.
- [74] M. Magnusson, D. Landén, and P. Doherty. Planning, Executing, and Monitoring Communication in a Logic-based Multi-agent System. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 933–934, 2008.
- [75] A. Meisels, I. Razgon, E. Kaplansky, and R. Zivan. Comparing performance of Distributed Constraints Processing Algorithms. In *Proceedings of AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, 2002.
- [76] T. Merz, P. Rudol, and M. Wzorek. Control System Framework for Autonomous Robots Based on Extended State Machines. In *Proceedings of the International Conference on Autonomic and Autonomous Systems*, 2006.
- [77] P. Modi, W-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161:149–180, 2006.
- [78] M. H. Nodine, J. Fowler, T. Ksiezzyk, B. Perry, M. C. Taylor, and A. Unruh. Active Information Gathering in InfosleuthTM. *International Journal of Cooperative Information Systems (IJCIS)*, 9(1-2):3–28, 2000.
- [79] L. E. Parker. Alliance: An Architecture for Fault Tolerant Multi-robot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

- [80] L. E. Parker and F. Tang. Building Multirobot Coalitions Through Automated Task Solution Synthesis. *Proceedings of the IEEE, Special Issue on Multi-Robot Systems*, pages 1289–1305, 2006.
- [81] J. P. Pearce, M. Tambe, and R. T. Maheswaran. Solving multiagent networks using distributed constraint optimization. *AI Magazine*, 29(3):47–62, 2008.
- [82] A. Petcu and B. Faltings. A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 266–271, 2005.
- [83] P. Rudol and P. Doherty. Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery. In *Proceedings of the IEEE Aerospace Conference*, pages 1–8, 2008.
- [84] P. Rudol, M. Wzorek, G. Conte, and P. Doherty. Micro Unmanned Aerial Vehicle Visual Servoing for Cooperative Indoor Exploration. In *Proceedings of the IEEE Aerospace Conference*, pages 1–10, 2008.
- [85] K. Saenchai, L. Benedicenti, and R. Paranjape. Solving Dynamic Distributed Constraint Satisfaction Problems with a Modified Weak-Commitment Search Algorithm. In *Engineering Self-Organising Systems, Third International Workshop, ESOA 2005*, pages 130–137, 2005.
- [86] W-M. Shen and B. Salemi. Towards Distributed and Dynamic Task Reallocation. *Intelligent Autonomous Systems*, 7:570–575, 2002.
- [87] R. Simmons and D. Apfelbaum. A Task Description Language for Robot Control. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, pages 1931–1937, 1998.
- [88] R. Smith. The Contract Net Protocol. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.
- [89] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS Infrastructure. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.
- [90] G. Tack. *Constraint Propagation - Models, Techniques, Implementation*. Phd thesis, Saarland University, Germany, 2009.
- [91] M. Tambe. Agent Architectures for Flexible, Practical Teamwork. In *Proceedings of the National Conference on Artificial Intelligence*, pages 22–28, 1997.
- [92] Telecom Italia Lab. The Java Agent Development Framework (JADE). <http://jade.tilab.com>.

-
- [93] P. Ulam, Y. Endo, A. Wagner, and R. C. Arkin. Integrated Mission Specification and Task Allocation for Robot Teams - Design and Implementation. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4428–4435, 2007.
 - [94] A. Viguria, I. Maza, and A. Ollero. Distributed Service-Based Cooperation in Aerial/Ground Robot Teams Applied to Fire Detection and Extinguishing Missions. *Advanced Robotics*, 24:1–23, 2010.
 - [95] B. van Linder W. van der Hoek and J-J. C. Meyer. An Integrated Modal Approach to Rational Agents. In *Foundations of Rational Agency*, volume 14 of *Applied Logic Series*, pages 133–168. Kluwer, 1998.
 - [96] M. G. Wallace, J. Schimpf, and S. Novello. A Platform for Constraint Logic Programming. *ICL System Journal*, 12(1):159–200, 1997.
 - [97] M. Wzorek, G. Conte, P. Rudol, T. Merz, S. Duranti, and P. Doherty. From Motion Planning to Control – A Navigation Framework for an Unmanned Aerial Vehicle. In *Proceedings of the 21st Bristol International Conference on UAV Systems*, pages 17.1–17.15, 2006.
 - [98] M. Wzorek and P. Doherty. Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling*, pages 438–441, 2006.
 - [99] M. Wzorek, J. Kvarnström, and P. Doherty. Choosing Path Replanning Strategies for Unmanned Aircraft Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 193–200, 2010.
 - [100] M. Wzorek, D. Landén, and P. Doherty. GSM Technology as a Communication Media for an Autonomous Unmanned Aerial Vehicle. In *Proceedings of the 21st Bristol International Conference on UAV Systems*, pages 24.1–24.15, 2006.
 - [101] M. Yokoo. Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 88–102, 1995.
 - [102] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Trans. on Knowl. and Data Eng.*, 10(5):673–685, 1998.
 - [103] M. Yokoo and K. Hirayama. Distributed Constraint Satisfaction Algorithm for Complex Local Problems. In *Proceedings of the 3rd International Conference on Multi Agent Systems, ICMAS '98*, pages 372–379, 1998.

- [104] F. Zacharias, C. Borst, and G. Hirzinger. Capturing Robot Workspace Structure: Representing Robot Capabilities. In *Proceedings of International Conference on Intelligent Robots and Systems (IROS)*, pages 3229–3236, 2007.
- [105] R. Zlot. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams*. Phd thesis, Carnegie Mellon University, 2006.
- [106] R. Zlot and A. Stentz. Complex Task Allocation For Multiple Robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1515–1522, 2005.

 Avdelning, Institution Division, Department AIICS, Dept. of Computer and Information Science 581 83 Linköping		Datum Date 2011-10-25
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input checked="" type="checkbox"/> Licentiatavhandling <input type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN 978-91-7393-048-2 <hr/> ISRN LiU-Tek-Lic-2011:45 <hr/> Serietitel och serienummer ISSN Title of series, numbering <u>0280-7971</u>
URL för elektronisk version URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-70536		Linköping Studies in Science and Technology Thesis No. 1506
Titel Title Complex Task Allocation for Delegation: From Theory to Practice		
Författare Author David Landén		
Sammanfattning Abstract <p>The problem of determining who should do what given a set of tasks and a set of agents is called the task allocation problem. The problem occurs in many multi-agent system applications where a workload of tasks should be shared by a number of agents. In our case, the task allocation problem occurs as an integral part of a larger problem of determining if a task can be delegated from one agent to another.</p> <p>Delegation is the act of handing over the responsibility for something to someone. Previously, a theory for delegation including a delegation speech act has been specified. The speech act specifies the preconditions that must be fulfilled before the delegation can be carried out, and the postconditions that will be true afterward. To actually use the speech act in a multi-agent system, there must be a practical way of determining if the preconditions are true. This can be done by a process that includes solving a complex task allocation problem by the agents involved in the delegation.</p> <p>In this thesis a constraint-based task specification formalism, a complex task allocation algorithm for allocating tasks to unmanned aerial vehicles and a generic collaborative system shell for robotic systems are developed. The three components are used as the basis for a collaborative unmanned aircraft system that uses delegation for distributing and coordinating the agents' execution of complex tasks.</p>		
Nyckelord Keywords Multi-agent systems, task allocation, distributed constraint satisfaction, delegation, UAV		

Department of Computer and Information Science
Linköpings universitet

Licentiate Theses

Linköpings Studies in Science and Technology
Faculty of Arts and Sciences

- No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)
- No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.
- No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.
- No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.
- No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.
- No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.
- No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.
- No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.
- No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.
- No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.
- No 104 **Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.
- No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.
- No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.
- No 113 **Ralph Rönquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.
- No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.
- No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.
- No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.
- No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.
- No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.
- No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.
- No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.
- No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.
- No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.
- No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.
- No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.
- No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.
- No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.
- No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.
- No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.
- No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.
- No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.
- No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.
- No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.
- No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.
- No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.
- No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.
- No 253 **Torbjörn Näslund:** SL DFA-Resolution - Computing Answers for Negative Queries, 1990.
- No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.
- No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge- Bases, 1991.
- No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.
- No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.
- No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.
- No 326 **Andreas Kägedal:** Logic Programming with External Procedures: an Implementation, 1992.
- No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.
- No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.
- No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.
- No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.
- No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.
- No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.
- No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

- No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
- No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
- No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
- No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
- No 398 **Johan Boye:** Dependency-based Groudnness Analysis of Functional Logic Programs, 1993.
- No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
- No 406 **Anna Moborg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
- No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
- No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
- No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
- No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
- No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
- FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
- FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
- No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
- No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
- No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
- No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
- No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
- No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
- FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
- No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
- No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
- No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
- No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
- No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
- No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
- No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
- No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
- FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylprogram, 1995.
- No 482 **Eva L. Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
- No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
- No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
- No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
- No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
- No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
- FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
- FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
- No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
- No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
- No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
- No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
- No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
- No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
- No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
- FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
- No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
- No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
- No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
- No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
- No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
- No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.

- No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
- No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
- No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
- No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
- No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
- No 591 **Niclas Wahlöf:** A Default Extension to Description Logics and its Applications, 1996.
- No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
- No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.
- No 598 **Rego Granlund:** C³Fire - A Microworld Supporting Emergency Management Training, 1997.
- No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
- No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
- No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
- FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
- FiF-a 6 **Tommy Wedlund:** Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
- No 615 **Silvia Coradeschi:** A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
- No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
- No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
- No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
- No 629 **Gunilla Ilevfors:** Krigsspel och Informationsteknik inför en oförutsägbart framtid, 1997.
- No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
- No 639 **Jukka Mäki-Turja:** Smalltalk - a suitable Real-Time Language, 1997.
- No 640 **Juha Takkinen:** CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
- No 643 **Man Lin:** Formal Analysis of Reactive Rule-based Programs, 1997.
- No 653 **Mats Gustafsson:** Bringing Role-Based Access Control to Distributed Systems, 1997.
- FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
- No 674 **Marcus Bjärelund:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
- No 676 **Jan Håkegård:** Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
- No 668 **Per-Ove Zetterlund:** Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
- No 675 **Jimmy Tjäder:** Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
- FiF-a 14 **Ulf Melin:** Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
- No 695 **Tim Heyer:** COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
- No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
- FiF-a 16 **Marie-Therese Christiansson:** Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
- No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
- No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
- No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
- No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
- No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
- No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
- No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
- No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
- FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
- FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
- No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.
- No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
- FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
- No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
- No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
- No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999.
- No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
- No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.

- No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
- No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
- No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
- No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
- FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
- No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
- No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
- No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
- No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
- No 800 **Anders Subotic:** Software Quality Inspection, 1999.
- No 807 **Svein Bergum:** Managerial communication in telework, 2000.
- No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
- FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
- No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
- No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
- No 823 **Lars Hult:** Publika Gränssytor - ett designexempel, 2000.
- No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
- FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
- No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
- No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
- FiF-a 37 **Ewa Braf:** Organisationers kunskapsverksamheter - en kritisk studie av "knowledge management", 2000.
- FiF-a 40 **Henrik Lindberg:** Webaserade affärsprocesser - Möjligheter och begränsningar, 2000.
- FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
- No 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
- No 863 **Dan Lawesson:** Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
- No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
- No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
- No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue systems, 2001.
- FiF-a 47 **Per-Arne Segerkvist:** Webaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.
- No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
- No 906 **Lin Han:** Secure and Scalable E-Service Software Delivery, 2001.
- No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
- No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
- FiF-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
- FiF-a-51 **Per Oscarsson:** Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
- No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
- No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
- No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
- No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
- No 938 **Bourhane Kadmiry:** Fuzzy Control of Unmanned Helicopter, 2002.
- No 942 **Patrik Haslum:** Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
- No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
- FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
- No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
- No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
- No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
- FiF-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
- No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
- No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
- No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
- No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
- No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.

- No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002.
- No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
- No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
- No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
- FiF-a 62 **Lennart Ljung:** Utveckling av en produktivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
- No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
- No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
- No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
- No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
- No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
- No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
- No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
- FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
- No 1024 **Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
- No 1034 **Arja Vainio-Larsson:** Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
- No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befärade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
- FiF-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
- No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
- No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
- No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
- FiF-a 71 **Emma Eliason:** Effekttanalys av IT-systems handlingsutrymme, 2003.
- No 1055 **Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
- No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
- FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
- No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
- No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
- FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
- No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
- No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensive Data Mining Models, 2004.
- No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
- No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
- No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
- FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
- No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
- No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
- No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
- No 1130 **Ioan Chisalitã:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
- No 1138 **Thomas Gustafsson:** Maintaining Data Consistency in Embedded Databases for Vehicular Systems, 2004.
- No 1149 **Vaida Jakonienė:** A Study in Integrating Multiple Biological Data Sources, 2005.
- No 1156 **Abdil Rashid Mohamed:** High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
- No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
- No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
- FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
- No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
- No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
- No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
- FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
- No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
- FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
- No 1172 **Petter Ahlström:** Affärsstrategier för seniorbostadsmarknaden, 2005.
- No 1183 **Mathias Cöster:** Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
- No 1184 **Åsa Horzella:** Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
- No 1185 **Maria Kollberg:** Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
- No 1190 **David Dinka:** Role and Identity - Experience of technology in professional settings, 2005.

- No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
- No 1192 **Nicklas Bergfeldt**: Towards Detached Communication for Robot Cooperation, 2005.
- No 1194 **Dennis Maciuszek**: Towards Dependable Virtual Companions for Later Life, 2005.
- No 1204 **Beatrice Alenljung**: Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.
- No 1206 **Anders Larsson**: System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
- No 1207 **John Wilander**: Policy and Implementation Assurance for Software Security, 2005.
- No 1209 **Andreas Käll**: Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
- No 1225 **He Tan**: Aligning and Merging Biomedical Ontologies, 2006.
- No 1228 **Artur Wilk**: Descriptive Types for XML Query Language Xcerpt, 2006.
- No 1229 **Per Olof Pettersson**: Sampling-based Path Planning for an Autonomous Helicopter, 2006.
- No 1231 **Kalle Burbeck**: Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.
- No 1233 **Daniela Mihailescu**: Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.
- No 1244 **Jörgen Skågeby**: Public and Non-public gifting on the Internet, 2006.
- No 1248 **Karolina Eliasson**: The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.
- No 1263 **Misook Park-Westman**: Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.
- FiF-a 90 **Amra Halilovic**: Ett praktikerspektiv på hantering av mjukvarukomponenter, 2006.
- No 1272 **Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.
- No 1277 **Viacheslav Izosimov**: Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.
- No 1283 **Håkan Hasewinkel**: A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.
- FiF-a 91 **Hanna Broberg**: Verksamhetsanpassade IT-stöd - Design teori och metod, 2006.
- No 1286 **Robert Kaminski**: Towards an XML Document Restructuring Framework, 2006.
- No 1293 **Jiri Trnka**: Prerequisites for data sharing in emergency management, 2007.
- No 1302 **Björn Häggglund**: A Framework for Designing Constraint Stores, 2007.
- No 1303 **Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.
- No 1305 **Magnus Ingmarsson**: Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.
- No 1306 **Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.
- No 1307 **Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.
- No 1309 **Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.
- No 1312 **Henrik Svensson**: Embodied simulation as off-line representation, 2007.
- No 1313 **Zhiyuan He**: System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.
- No 1317 **Jonas Elmqvist**: Components, Safety Interfaces and Compositional Analysis, 2007.
- No 1320 **Håkan Sundblad**: Question Classification in Question Answering Systems, 2007.
- No 1323 **Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.
- No 1329 **Martin Magnusson**: Deductive Planning and Composite Actions in Temporal Action Logic, 2007.
- No 1331 **Mikael Asplund**: Restoring Consistency after Network Partitions, 2007.
- No 1332 **Martin Fransson**: Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.
- No 1333 **Karin Camara**: A Visual Query Language Served by a Multi-sensor Environment, 2007.
- No 1337 **David Broman**: Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.
- No 1339 **Mikhail Chalabine**: Invasive Interactive Parallelization, 2007.
- No 1351 **Susanna Nilsson**: A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.
- No 1353 **Shanai Ardi**: A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.
- No 1356 **Erik Kuiper**: Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.
- No 1359 **Jana Rambusch**: Situated Play, 2008.
- No 1361 **Martin Karresand**: Completing the Picture - Fragments and Back Again, 2008.
- No 1363 **Per Nyblom**: Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.
- No 1371 **Fredrik Lantz**: Terrain Object Recognition and Context Fusion for Decision Support, 2008.
- No 1373 **Martin Östlund**: Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.
- No 1381 **Håkan Lundvall**: Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.
- No 1386 **Mirko Thorstensson**: Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.
- No 1387 **Bahlol Rahimi**: Implementation of Health Information Systems, 2008.
- No 1392 **Maria Holmqvist**: Word Alignment by Re-using Parallel Phrases, 2008.
- No 1393 **Mattias Eriksson**: Integrated Software Pipelining, 2009.
- No 1401 **Annika Öhgren**: Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.
- No 1410 **Rickard Holmsmark**: Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.
- No 1421 **Sara Stymne**: Compound Processing for Phrase-Based Statistical Machine Translation, 2009.
- No 1427 **Tommy Ellqvist**: Supporting Scientific Collaboration through Workflows and Provenance, 2009.
- No 1450 **Fabian Segelström**: Visualisations in Service Design, 2010.
- No 1459 **Min Bao**: System Level Techniques for Temperature-Aware Energy Optimization, 2010.
- No 1466 **Mohammad Saifullah**: Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011

No 1468	Qiang Liu: Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.
No 1469	Ruxandra Pop: Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011.
No 1476	Per-Magnus Olsson: Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011.
No 1481	Anna Vapen: Contributions to Web Authentication for Untrusted Computers, 2011.
No 1485	Loove Broms: Sustainable Interactions: Studies in the Design of Energy Awareness Artefacts, 2011.
FiF-a 101	Johan Blomkvist: Conceptualising Prototypes in Service Design, 2011.
No 1490	Håkan Warnquist: Computer-Assisted Troubleshooting for Efficient Off-board Diagnosis, 2011.
No 1503	Jakob Rosén: Predictable Real-Time Applications on Multiprocessor Systems-on-Chip, 2011.
No 1504	Usman Dastgeer: Skeleton Programming for Heterogeneous GPU-based Systems, 2011.
No 1506	David Landén: Complex Task Allocation for Delegation: From Theory to Practice, 2011.