Exam Overview

25 Oct: 14.00-18.00

LINKÖPING UNIVERSITY

Potential Exam Question Topics (1)

- Agents: Types of agents, comparison (diagrams useful)
- Physical Symbol System Hypothesis: Article (diagram useful)
- Turing Test: Article (diagram useful)
- Search: different algorithms/complexity and use, A* search , proving optimality of A* (both trees and graphs), admissible heuristics
- Local Search algorithms: Hill climbing. SAT solving, Davis Putnam algorithm. Genetic algorithms. Simulated annealing.
- Constraints: Arc Consistency (AC3), domain independent heuristics, modeling
- Resolution Theorem Proving: Propositional Case. Should be able to set-up and do a refutation proof. [appendix will be provided]
- Answer Sets, nonmonotonic reasoning: distinguishing NM reasoning from classical reasoning. generating reducts from answer set programs. Checking whether a model is an answer set of a program. [appendix will be provided]
- Reasoning about action and change: fundamental problems and solutions. How one might solve problems using NM techniques.

LINKÖPING UNIVERSITY

Potential Exam Question Topics(2)

- Bayesian Networks: Using to answer questions, background theory. Bayes Rule. [appendix will be provided]
- Planning: Modeling classical planning problems, Forward state space search, Plan space search, partial order planning, Planning heuristics: relaxation heuristics, pattern database heuristics.
- Machine Learning: Q-learning/reinforcement learning. Neural Networks/Back propagation. Supervised/Unsupervised learning techniques [appendix will be provided]
- **Not on the exam**: Contents of Fo12, 13, 14. But, one might refer to topics covered here in other questions.
- Responsible for reading material in course book specified in the reading list.
- Responsible for knowing material on the slides.
- Responsible for knowing lab material.
- Course book may not be taken to the exam.
- But, important to bring a calculator for the questions on Bayesian networks/machine learning.

Putting it all Together: An overview of a collaborative robotics framework

Seminar 14

Outline

- **Theory**
  - Delegation and collaboration framework
- **Implementation**
  - Distributed delegation framework
  - UAV Platforms
- **Application**
  - Alpine Rescue & 3D Map Construction
  - EU 7th Frame Project : SHERPA

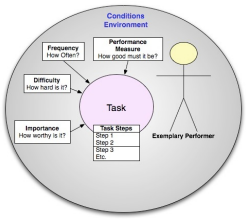
Collaborative Mission Processes

When humans have a difficult problem to solve . . .

Communicate and find appropriate people to help.

The get together physically or virtually

Discuss the problem and break it down (make a plan)



Delegate sub-tasks in the plan to those with the proper capabilities

Communication
Cooperation
Making Plans
Delegating Tasks
Executing Tasks
Following Up

LINKÖPING UNIVERSITY

Theory

LINKÖPING UNIVERSITY

Key Concept: Delegation of Contextualized/Constrained Tasks

Delegate(A,B,Task,Constraints)

Delegation

Cooperation

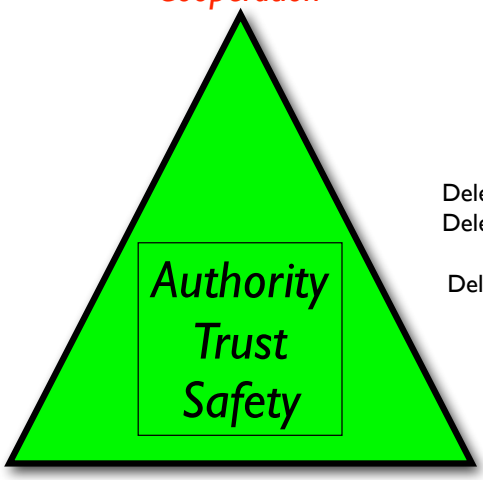
Key Idea I:
Correct Semantic Formulation of Delegation and Tasks

Key Idea II:
Pragmatic Instantiation of Delegation and Tasks in Robotic Architectures

Total *Loosen*

Delegate(A,B,Task,Constraints)

Partial *Tighten*



Delegate(Human,Robot,Task,Constraints)
Delegate(Robot,Human,Task,Constraints)

Delegate(A,B,Advice-Task,Constraints)

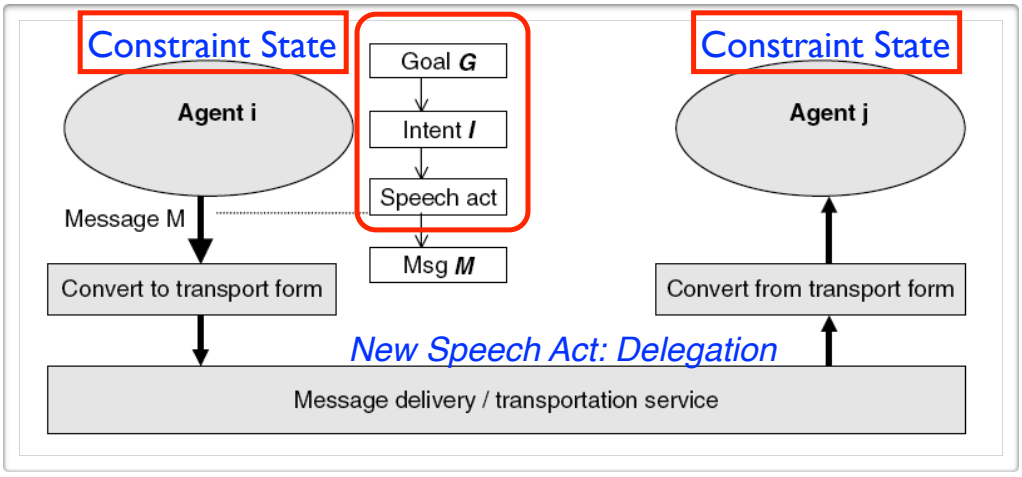
Adjustable Autonomy

Mixed-Initiative Problem Solving

LINKÖPING UNIVERSITY

Formal Theory of Delegation

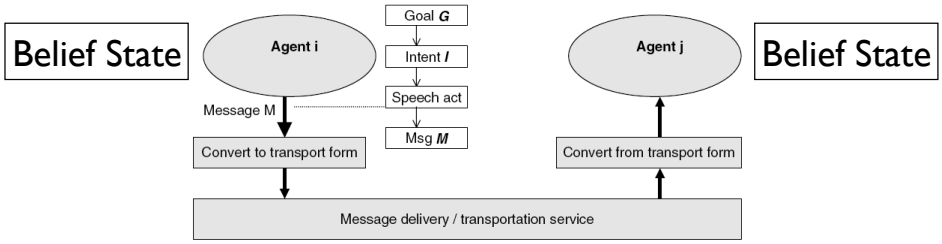
- Joint work with J.J. Meyers (Utrecht University)
- Based on intuitions from Castelfranchi
- Uses Speech Acts / Introduces a new speech act
- Based loosely on intuitions from BDI Architectures



LINKÖPING UNIVERSITY

Speech Acts

- Speech Acts have their origins in the area of pragmatics from linguistics
  - propositional content of an utterance U can be distinguished from its illocutionary force, the speakers intention in uttering U
  - From this perspective, utterances can be viewed as representing actions , or communicative acts.
    - Speech Acts or Performatives



LINKÖPING UNIVERSITY

Request

3.19 Request

Summary	The sender requests the receiver to perform some action. One important class of uses of the request act is to request the receiver to perform another communicative act.
Message Content	An action expression.
Description	The sender is requesting the receiver to perform some action. The content of the message is a description of the action to be performed, in some language the receiver understands. The action can be any action the receiver is capable of performing, for example, pick up a box, book a plane flight, change a password, etc. An important use of the request act is to build composite conversations between agents, where the actions that are the object of the request act are themselves communicative acts such as `inform`.
Formal Model	$\langle i$, request $(j, a)\rangle$ FP: FP (a) $[i\backslash j] \wedge B_i$ Agent $(j, a) \wedge \neg B_i\ I_j$ Done (a) RE: Done (a) FP(a) $[i\backslash j]$ denotes the part of the FPs of *a* which are mental attitudes of *i*.

Examples	Agent *i* requests *j* to open a file.
	
	`(request`
	`:sender (agent-identifier :name i)`
	`:receiver (set (agent-identifier :name j))`
	`:content`
	`"open \"db.txt\" for input"`
	`:language vb)`

Agent *i* requests *j* to open a file.

```
(request
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content
    "open \"db.txt\" for input"
  :language vb)
```

LINKÖPING UNIVERSITY

Inform

3.8 Inform

Summary	The sender informs the receiver that a given proposition is true.
Message Content	A proposition.
Description	`inform` indicates that the sending agent: • holds that some proposition is true, • intends that the receiving agent also comes to believe that the proposition is true, and, • does not already believe that the receiver has any knowledge of the truth of the proposition. The first two properties defined above are straightforward: the sending agent is sincere, and has (somehow) generated the intention that the receiver should know the proposition (perhaps it has been asked). The last property is concerned with the semantic soundness of the act. If an agent knows already that some state of the world holds (that the receiver knows proposition *p*), it cannot rationally adopt an intention to bring about that state of the world, that is, that the receiver comes to know *p* as a result of the `inform` act. Note that the property is not as strong as it perhaps appears. The sender *is not* required to establish whether the receiver knows *p*. It is only the case that, in the case that the sender already happens to know about the state of the receiver's beliefs; it should not adopt an intention to tell the receiver something it already knows. From the receiver's viewpoint, receiving an `inform` message entitles it to believe that: • the sender believes the proposition that is the content of the message, and, • the sender wishes the receiver to believe that proposition also. Whether or not the receiver does, indeed, adopt belief in the proposition will be a function of the receiver's trust in the sincerity and reliability of the sender.
Formal Model	$\langle i,\ \text{inform}\ (j,\ \phi)\rangle$ FP: $B_i\phi \wedge \neg B_i(\text{Bif}_j\phi \vee \text{Uif}_j\phi)$ RE: $B_j\phi$

Examples	Agent *i* informs agent *j* that (it is true that) it is raining today.
	`(inform`
	`:sender (agent-identifier :name i)`
	`:receiver (set (agent-identifier :name j))`
	`:content`
	`"weather (today, raining)"`
	`:language Prolog)`

Agent *i* informs agent *j* that (it is true that) it is raining today.

```
(inform
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content
    "weather (today, raining)"
  :language Prolog)
```

LINKÖPING UNIVERSITY

Call for Proposal

3.4 Call for Proposal

Summary	The action of calling for proposals to perform a given action.
Message Content	A tuple containing an action expression denoting the action to be done, and a referential expression defining a single-parameter proposition which gives the preconditions on the action.
Description	`cfp` is a general-purpose action to initiate a negotiation process by making a call for proposals to perform the given action. The actual protocol under which the negotiation process is established is known either by prior agreement or is explicitly stated in the `protocol` parameter of the message. In normal usage, the agent responding to a `cfp` should answer with a proposition giving the value of the parameter in the original precondition expression (see the statement of rational effect for `cfp`). For example, the `cfp` might seek proposals for a journey from Frankfurt to Munich, with a condition that the mode of travel is by train. A compatible proposal in reply would be for the 10.45 express train. An incompatible proposal would be to travel by airplane. Note that `cfp` can also be used to simply check the availability of an agent to perform some action. Also note that this formalization of `cfp` is restricted to the common case of proposals characterized by a single parameter (x) in the proposal expression. Other scenarios might involve multiple proposal parameters, demand curves, free-form responses, and so forth.
Formal Model	$\langle i,\ \text{cfp}\ (j, \langle j,\ act\rangle,\ Ref\ x\ \phi(x))\rangle \equiv$ $\langle i,\ \text{query-ref}\ (j,\ Ref\ x\ (I_i\ \text{Done}\ (\langle j,\ act\rangle,\ \phi(x)))$ $(I_j\ \text{Done}\ (\langle j,\ act\rangle,\ \phi(x))))\rangle$ FP: $\neg\text{Bref}_i(Ref\ x\ \alpha(x)) \wedge \neg\text{Uref}_i(Ref\ x\ \alpha(x)) \wedge$ $\neg B_i\ I_j\ \text{Done}\ (\langle j,\ \text{inform-ref}\ (i,\ Ref\ x\ \alpha(x))\rangle)$ RE: $\text{Done}\ (\langle j,\ \text{inform}\ (i,\ Ref\ x\ \alpha(x) = r_1)\rangle \mid \ldots \mid$ $\langle j,\ \text{inform}\ (i,\ Ref\ x\ \alpha(x) = r_x)\rangle)$ Where: $\alpha(x) = I_i\ \text{Done}\ (\langle j,\ act\rangle,\ \phi(x))\quad I_j\ \text{Done}\ (\langle j,\ act\rangle,\ \phi(x))$ Agent *i* asks agent *j*: "What is the '*x*' such that you will perform action 'act' when '$\phi(x)$' holds?" Note: $Ref\ x\ \delta(x)$ is one of the referential expressions: $\iota x\ \delta(x)$, $\text{any}\ x\ \delta(x)$ or $\text{all}\ x\ \delta(x)$. Note: The rational effect of this is not a proposal by the recipient. Rather, it is the value of the proposal parameter. See the example in the definition of the `propose` act.

Example	Agent *j* asks *i* to submit its proposal to sell 50 boxes of plums.
	`(cfp`
	`:sender (agent-identifier :name j)`
	`:receiver (set (agent-identifier :name i))`
	`:content`
	`"((action (agent-identifier :name i)`
	`(sell plum 50))`
	`(any ?x (and (= (price plum) ?x) (< ?x 10))))"`
	`:ontology fruit-market`
	`:language fipa-sl)`

Agent *j* asks *i* to submit its proposal to sell 50 boxes of plums.

```
(cfp
  :sender (agent-identifier :name j)
  :receiver (set (agent-identifier :name i))
  :content
    "((action (agent-identifier :name i)
      (sell plum 50))
     (any ?x (and (= (price plum) ?x) (< ?x 10))))"
  :ontology fruit-market
  :language fipa-sl)
```

LINKÖPING UNIVERSITY

Propose

Agent *j* proposes to *i* to sell 50 boxes of plums for $5 (this example continues the example of `cfp`).

```
(propose
  :sender (agent-identifier :name j)
  :receiver (set (agent-identifier :name i))
  :content
    "((action j (sell plum 50))
     (= (any ?x (and (= (price plum) ?x) (< ?x 10))) 5)"
  :ontology fruit-market
  :in-reply-to proposal2
  :language fipa-sl)
```

3.13 Propose

Summary	The action of submitting a proposal to perform a certain action, given certain preconditions.
Message Content	A tuple containing an action description, representing the action that the sender is proposing to perform, and a proposition representing the preconditions on the performance of the action.
Description	`propose` is a general-purpose act to make a proposal or respond to an existing proposal during a negotiation process by proposing to perform a given action subject to certain conditions being true. The actual protocol under which the negotiation process is being conducted is known either by prior agreement, or is explicitly stated in the `protocol` parameter of the message. The proposer (the sender of the `propose`) informs the receiver that the proposer will adopt the intention to perform the action once the given precondition is met, and the receiver notifies the proposer of the receiver's intention that the proposer performs the action. A typical use of the condition attached to the proposal is to specify the price of a bid in an auctioning or negotiation protocol.
Formal Model	$\langle i, \text{propose}\ (j, \langle i, act\rangle, \phi)\rangle \equiv$ $\langle i, \text{inform}\ (j, I_j\ \text{Done}\ (\langle i, act\rangle, \phi)\quad I_i\ \text{Done}\ (\langle i, act\rangle, \phi))\rangle$ FP: $B_i\,\alpha \wedge \neg B_i\,(\text{Bif}_j\,\alpha \vee \text{Uif}_j\,\alpha)$ RE: $B_j\,\alpha$ Where: $\alpha = I_j\ \text{Done}\ (\langle i, act\rangle, \phi)\quad I_i\ \text{Done}\ (\langle i, act\rangle, \phi)$ Agent *i* informs *j* that, once *j* informs *i* that *j* has adopted the intention for *i* to perform action *act*, and the preconditions for *i* performing *act* have been established, *i* will adopt the intention to perform the communicative act.

Example	Agent *j* proposes to *i* to sell 50 boxes of plums for $5 (this example continues the example of `cfp`).
	
	`(propose`
	`  :sender (agent-identifier :name j)`
	`  :receiver (set (agent-identifier :name i))`
	`  :content`
	`  "((action j (sell plum 50))`
	`  (= (any ?x (and (= (price plum) ?x) (< ?x 10))) 5)"`
	`  :ontology fruit-market`
	`  :in-reply-to proposal2`
	`  :language fipa-sl)`

LINKÖPING UNIVERSITY

Accept Proposal

Agent *i* informs *j* that it accepts an offer from *j* to stream a given multimedia title to channel 19 when the customer is ready. Agent *i* will inform *j* of this fact when appropriate.

```
(accept-proposal
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :in-reply-to bid089
  :content
    "((action (agent-identifier :name j)
       (stream-content movie1234 19))
     (B (agent-identifier :name j)
       (ready customer78)))"
  :language fipa-sl)
```

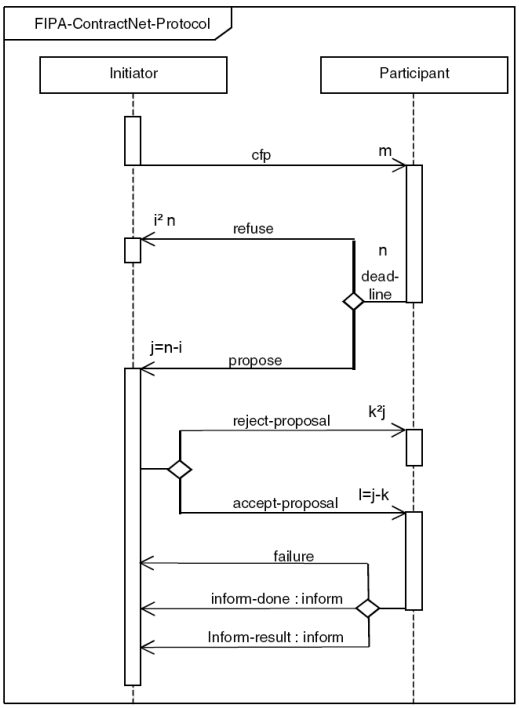
3.1 Accept Proposal

Summary	The action of accepting a previously submitted proposal to perform an action.
Message Content	A tuple consisting of an action expression denoting the action to be done, and a proposition giving the conditions of the agreement.
Description	`accept-proposal` is a general-purpose acceptance of a proposal that was previously submitted (typically through a `propose` act). The agent sending the acceptance informs the receiver that it intends that (at some point in the future) the receiving agent will perform the action, once the given precondition is, or becomes, true. The proposition given as part of the acceptance indicates the preconditions that the agent is attaching to the acceptance. A typical use of this is to finalize the details of a deal in some protocol. For example, a previous offer to "hold a meeting anytime on Tuesday" might be accepted with an additional condition that the time of the meeting is 11.00. Note for future extension: an agent may intend that an action become done without necessarily intending the precondition. For example, during negotiation about a given task, the negotiating parties may not unequivocally intend their opening bids: agent *a* may bid a price *p* as a precondition, but be prepared to accept price *p'*.
Formal Model	$\langle i, \text{accept-proposal}\ (j, \langle j, act\rangle, \phi)\rangle \equiv$ $\langle i, \text{inform}\ (j, I_i\ \text{Done}\ (\langle j, act\rangle, \phi))\rangle$ FP: $B_i\ \alpha \wedge \neg B_i\ (\text{Bif}_j\ \alpha \vee \text{Uif}_j\ \alpha)$ RE: $B_j\ \alpha$ Where: $\alpha = I_j\ \text{Done}\ (\langle j, act\rangle, \phi)$

Example	Agent *i* informs *j* that it accepts an offer from *j* to stream a given multimedia title to channel 19 when the customer is ready. Agent *i* will inform *j* of this fact when appropriate.
	
	`(accept-proposal`
	`  :sender (agent-identifier :name i)`
	`  :receiver (set (agent-identifier :name j))`
	`  :in-reply-to bid089`
	`  :content`
	`  "((action (agent-identifier :name j)`
	`  (stream-content movie1234 19))`
	`  (B (agent-identifier :name j)`
	`  (ready customer78)))"`
	`  :language fipa-sl)`

LINKÖPING UNIVERSITY

Interaction Protocols



Specification of principled interactions using structured combinations of speech acts

Contract Net Protocol

Delegation as a Speech Act

Used as a formal specification of a delegation process

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

Plan, Goal, constraints

Preconditions:

Conditions on mental states before Delegation

(1) $Goal_A(\phi)$

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

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

(4) $Bel_B Can_B(\tau)$

Postconditions:

Conditions on mental states after Delegation

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

(2) $Committed_B(\tau)$

(3) $Bel_B Goal_A(\phi)$

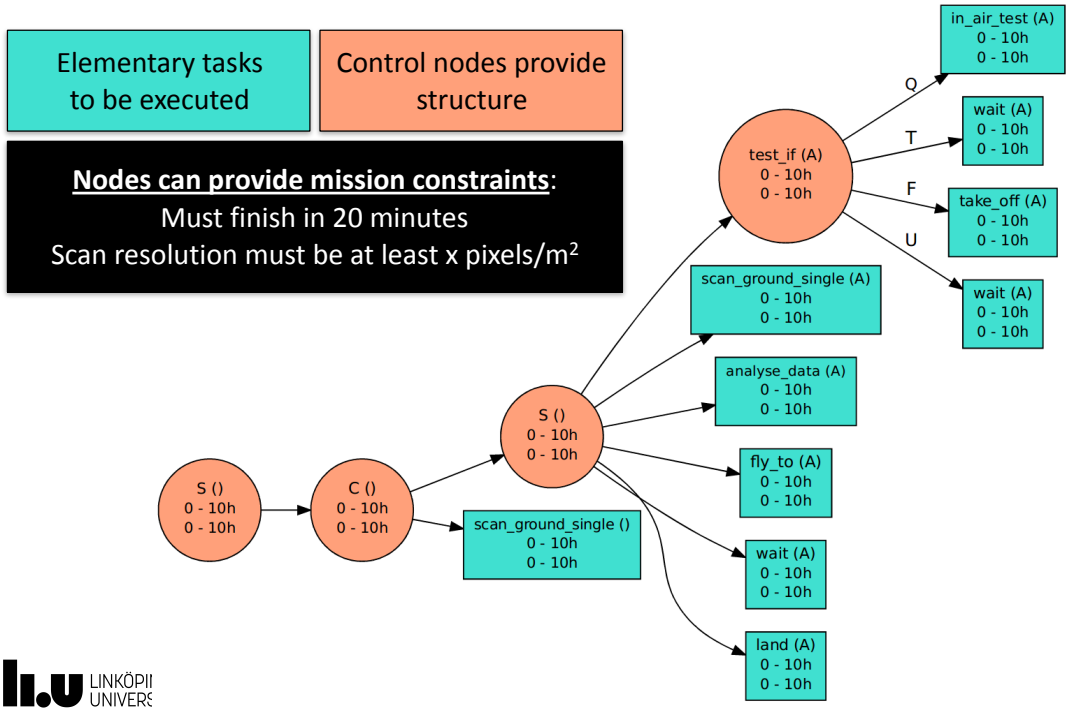
(4) $Can_B(\tau)$ (and hence $Bel_B Can_B(\tau)$, and by (1) also $Intend_B(\tau)$)

(5) $Intend_A(do_B(\langle \alpha, constraints \rangle))$

(6) $MutualBel_{AB}$("the statements above" $\wedge$ $SociallyCommitted(B, A, \tau)$)[4]

Task Specification Trees

- Missions are specified declaratively resulting in Task Specification Trees



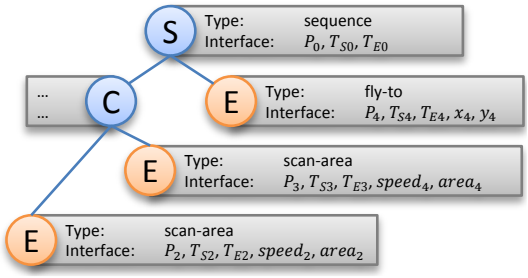
Task Specification Trees

Delegated Robotic Tasks are specified using Task Specification Trees (TSTs)

Temporal Constraints Built In!

TSTs are generic executable declarative specifications of Complex (Distributed) Tasks!

- Control Nodes
  - **S**: Sequence
  - **C**: Concurrent
  - **W**: While-Do
  - **IF**: Test-IF, IF
  - **FE**: Foreach-Do
  - **G**: Goal
- Task Specific Nodes
  - **E**: Elementary Action
    - scan-cell
    - fly-to
    - monitor-radiation
    - hover-at



- Each TST node represents a task (sub-task)
- Tasks (nodes) are delegated/allocated

TSTs can be generated Dynamically!

TST with Constraints = Contextual Task

Mission: Concurrent Scan Area
2 platforms scan sub-regions
1 platform flies to an observation position

Node Interface: {P_0,T_{S_0},T_{E_0}}
Task: coordinate-sequence

Node Parameters = External Constraint Variables

S — Type: sequence; Interface: P_0, T_{S0}, T_{E0}
C — ...
E — Type: fly-to; Interface: $P_4, T_{S4}, T_{E4}, x_4, y_4$
E — Type: scan-area; Interface: $P_3, T_{S3}, T_{E3}, speed_4, area_4$
E — Type: scan-area; Interface: $P_2, T_{S2}, T_{E2}, speed_2, area_2$

Node Interface: {P_2,T_{S_2},T_{E_2},speed$_2$, area$_2$}
Task: scan(Speed$_2$,Area$_2$)

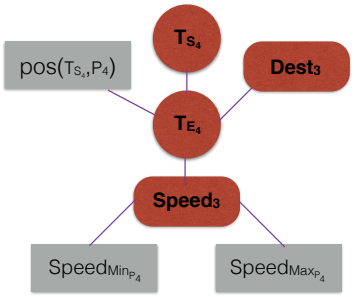
Node Interface: {P_3,T_{S_3},T_{E_3},speed4, area4}
Task: scan(Speed$_4$,Area$_4$)

Node Interface: {P_4,T_{S_4},T_{E_4},x_4,y_4,Speed$_3$,Dest$_3$}
Task: flyto(Speed$_3$,Dest$_3$)

Internal Constraints for the flyto Task

$$T_{E_4} = T_{S_4} + \frac{\text{distance}(\text{pos}(T_{S_4}, P_4), Dest_3)}{Speed_3}$$
$$\wedge \quad (Speed_{Min_{P_4}} \leq Speed_3 \leq Speed_{Max_{P_4}})$$

Constraints for the flyto Task



External Constraint Variables: Node Parameters (red)
Internal Constraint Variables: Specific to Task/Platform (gray)

Alternative TST Representations

BNF: TST Formal Language

The syntax of a TST specification has the following BNF:

SPEC ::= TST

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

TSTS ::= TST | TST ';' TSTS

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

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

VARS ::= VAR | VAR ',' VARS

CONSTRAINT ::= <constraint>

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

ARG ::= VAR | VALUE

ARGS ::= ARG | ARG ',' ARGS

VALUE ::= <value>

NAME ::= <node name>

COND ::= <ACL query>

GOAL ::= <goal statement>

ACTION ::= <elementary action>

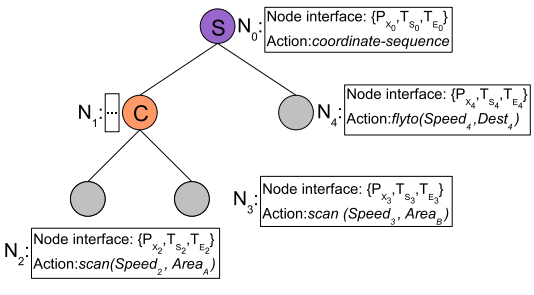
Where

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

TST's can be the Output of Automated Planners!

Sub-Trees can be Distributed, Delegated and Passed to Team Members!

Graphical Notation



TALFP : Logical Language and Semantics

$[t,t']$scan-for-people$(uav) \rightsquigarrow [t,t']$with u,u' do
$[u,u']$while $[t_c]\exists x,y[$owns$(uav,x,y) \wedge \neg$scanned$(x,y)]$ do
$[u,u']$with x,y do
$[u,u']$**scan-cell**(uav,x,y)
where $[t_c]$owns$(uav,x,y) \wedge \neg$scanned(x,y)

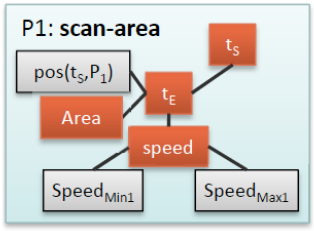
Text Manipulable Form

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

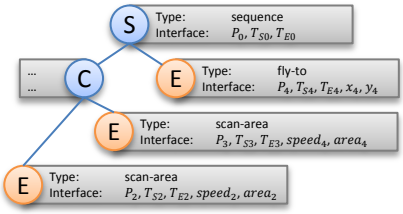
$cons_{\tau_0} = \{T_{S_0} \leq T_{S_1} \wedge T_{S_1} \leq T_{E_1} \wedge T_{E_1} \leq T_{S_4} \wedge T_{S_4} \leq T_{E_4} \wedge T_{E_4} \leq T_{E_0}\}$
$cons_{\tau_1} = \{T_{S_1} \leq T_{S_2} \wedge T_{S_2} \leq T_{E_2} \wedge T_{E_2} \leq T_{E_1} \wedge T_{S_1} \leq T_{S_3} \wedge T_{S_3} \leq T_{E_3}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

WITH <constraint vars> DO <process> WHERE <constraints>

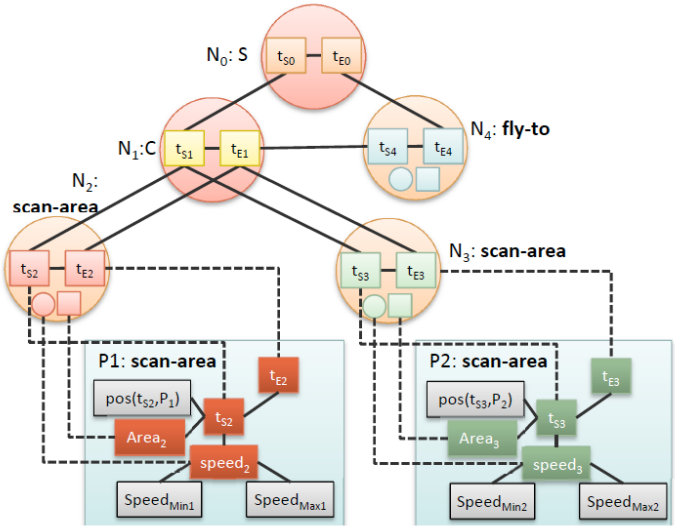
Constraints and TSTs



Agent-Specific Constraint Structure:
Combines external and internal parameters



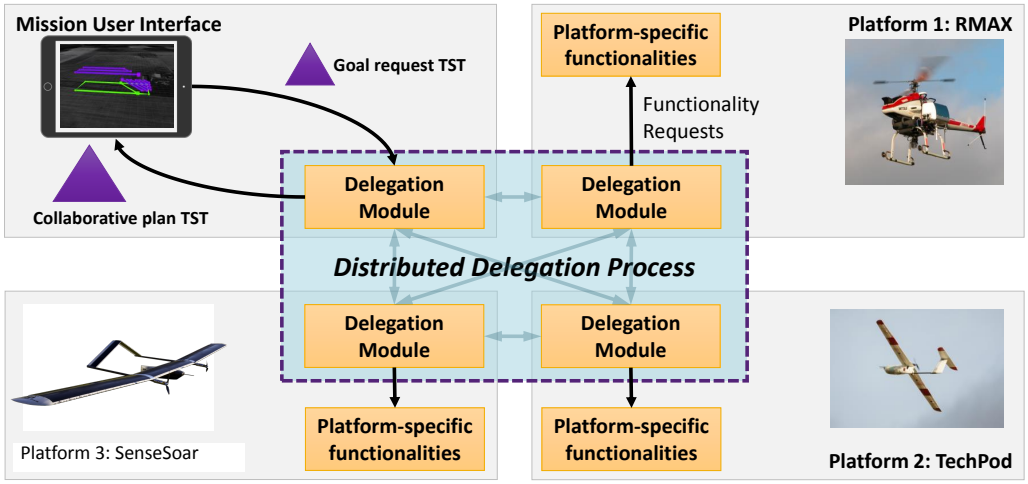
Delegation Process integrates Constraint Solving as a means for formally defining and answering the question CAN!



Mission Constraint Structure after allocation

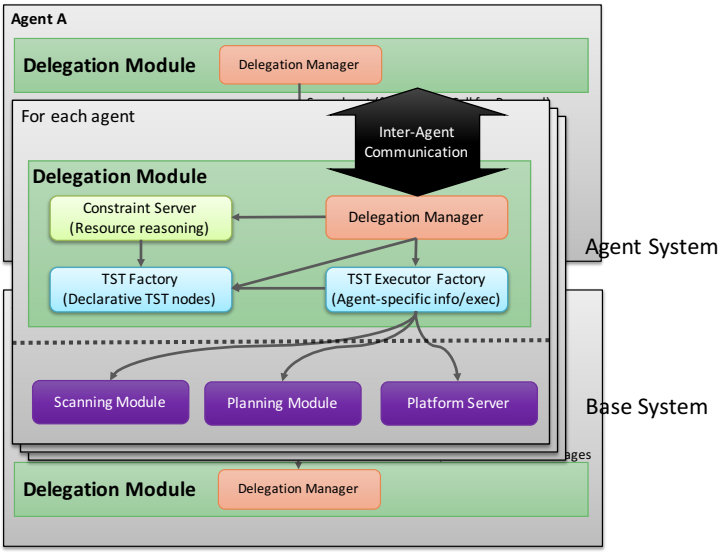
Implementation

Task Delegation & Planning Framework



Task Representation: **T**ask **S**pecification **T**rees (TSTs)

Delegation Module



JSON
(JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write.

Zyre
Open source framework for proximity-based peer-to-peer applications. Built on Zeromq

Abstract Delegation Process

1: **procedure** DELEGATE-FIRST-PHASE(task T, constraint set C)
2: **if** basic capabilities for root(T) are missing **then reply** REFUSE
3: Add constraints and parameters specified in root(T) to C
4: Add platform-specific constraints for root(T) to C
5: **if** C is inconsistent **then reply** REFUSE
6: **if** root(T) is a leaf and this platform wants to expand it **then**
7: Expand root(T), adding new children
8: **for every** child c_i of root(T) corresponding to a subtree T_i **do**
9: Broadcast a REQUEST to find P = potential contractors with capabilities for c_i
10: Perform auction for c_i among P, and sort P accordingly
11: **nondeterministically choose** $p \in P$:
12: $(T_i', C) \leftarrow p$.DELEGATE-FIRST-PHASE(T_i, C)
13: **replace** T_i with T_i' in T
14: Provisionally commit to the delegation
15: **reply** PROPOSE(T, C)

LINKÖPING UNIVERSITY

SHERPA

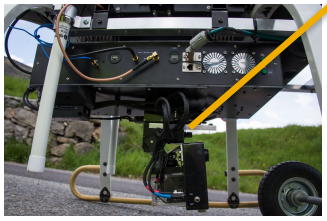
LiU/AIICS Robotic Systems

SEVENTH FRAMEWORK PROGRAMME

LiU/AIICS Robot Team

SHERPA

LKU RMAX (Scanner)

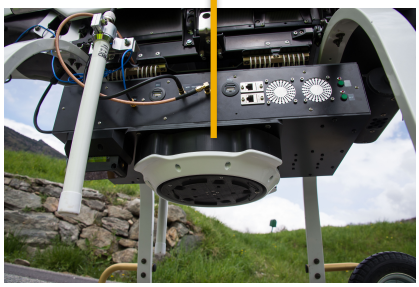


CCD Camera
Infrared Camera

SICK Laser Rangefinder

SHERPA

LKU RMAX (Deployer)



Prototype
Deployment Device
for Medical Supplies
& Wasps