W911NF-08-1-0041 Project

I–Globe

Distributed Planning and Coordination of Team-oriented Activities in a Dynamic Environment

Final Report

December 2008

Agent Technology Center,
Gerstner Laboratory, Czech Technical University in Prague

Antonín Komenda, Jiří Vokřínek, Dušan Pavlíček
Principal Investigator: Michal Pěchouček

Artificial Intelligence Applications Institute,
School of Informatics, The University of Edinburgh

Gerhard Wickler, Jeff Dalton, Austin Tate

Contents

1 Integration and Software Development

The following sections summarize the key achievements of software integration and development during the I-Globe project.

1.1 Prototype Overview

The presented prototype is based on the concept of a multi-layer planning architecture (Section 1.5). The planning process for the overall plan is distributed among an arbitrary number of autonomous agents. The planning hierarchy of the entities is not predefined, and it emerges during planning. Each agent knows only its own planning domain, which describes the agent's capabilities in the terms of the environment domain. These private personal domains are described in the form of Hierarchical Task Networks. The planning process (Section 1.6) is initiated by externally tasked agent(s). The tasks are typically added by a human operator using the system's human-machine-interface (HMI). An I-X Process Panel, which is part of the I-X architecture, is used as the HMI. The agent tries to fulfill the task goals and may need to incorporate the sub-plans of other agents, in the case it is not able to fulfill the task on its own. Such agents recursively run the same planning process until the whole plan is formed and ready for execution. In the phase of incorporating sub-plans, the agents need to mark parts of the plans where the other agents continue plan execution. For that purpose, the designed concept of plan interconnection by synchronization-points (Section 3.3.3) can be used. The parameters of the spatio-temporal synchronization-points are negotiated during the process of forming the planning hierarchy. The synchronization-points are later used in the plan execution.

From the perspective of one agent, the planning process can be divided into three layers, which form the multi-layer planning architecture. In the strategic layer (the topmost layer), the I-X HTN Planner is used, creating an abstract plan for the long-term time horizon. The planner is part of the I-X architecture and originates in the O-Plan planner. The plan instantiating process uses distributed resource allocation (Section 3.3.4) based on the multi-agent contract-net-protocol. With the help of this protocol, the appropriate subordinate agents are found and the responsibilities of the plan actions are fixed. The tactical layer (the middle layer) optimizes the plan using the early-as-possible scheduling heuristic. The heuristic causes the earliest possible execution of the plan actions which affects the length of the whole plan in the non-deterministic environment. The effect is directly proportional to the amount of the non-determinism in the world. The personal layer (the bottommost layer) plans potential refinements of the tactical actions. One of these actions is the agent's movement, where the path is planned using the A* algorithm. The other responsibility of the personal layer is the execution of all low-level actions in the scenario simulator.

All plans are described in the form of social commitments (Section 3.1) (substituting plan actions). The commitment is a knowledge-base structure describing the agent's obligation to change the world-state and a set of rules for what the agent should do if the obligation is not satisfiable. The proposed structure is an extension of a widely used formalization of commitment. The proposed commitment recursivity enables more expressive description of the decommitment rules and thus the replanning process. The introduction of causal commitment inter-referencing enables real-time replanning. The mutual bindings and commitments form a commitment graph. The graph can be used for the process successively solving exceptional states (replanning). The process is based on the traversing through the commitment graph, starting with the first violated commitment. One of the decommitment rules is triggered. If the decommitment rule inter-references another commitment, the process crosses on the referenced commitment and starts one of the decommitment rules on the side of the referenced commitment. Provided that the decommitment rule terminates the commitment without a need for

crossing to other commitments, the process ends there, the violation is fixed, and the plan is successfully replanned. In other words, the replanning process by means of social commitments can be described as successive decommitting. For decommitment purposes, three basic decommitment rules were used: full decommitment, delegation, and relaxation. The most suitable decommitment rule set and ordering for non-deterministic domains was used, taking delegation, relaxation, and full decommitment in that order.

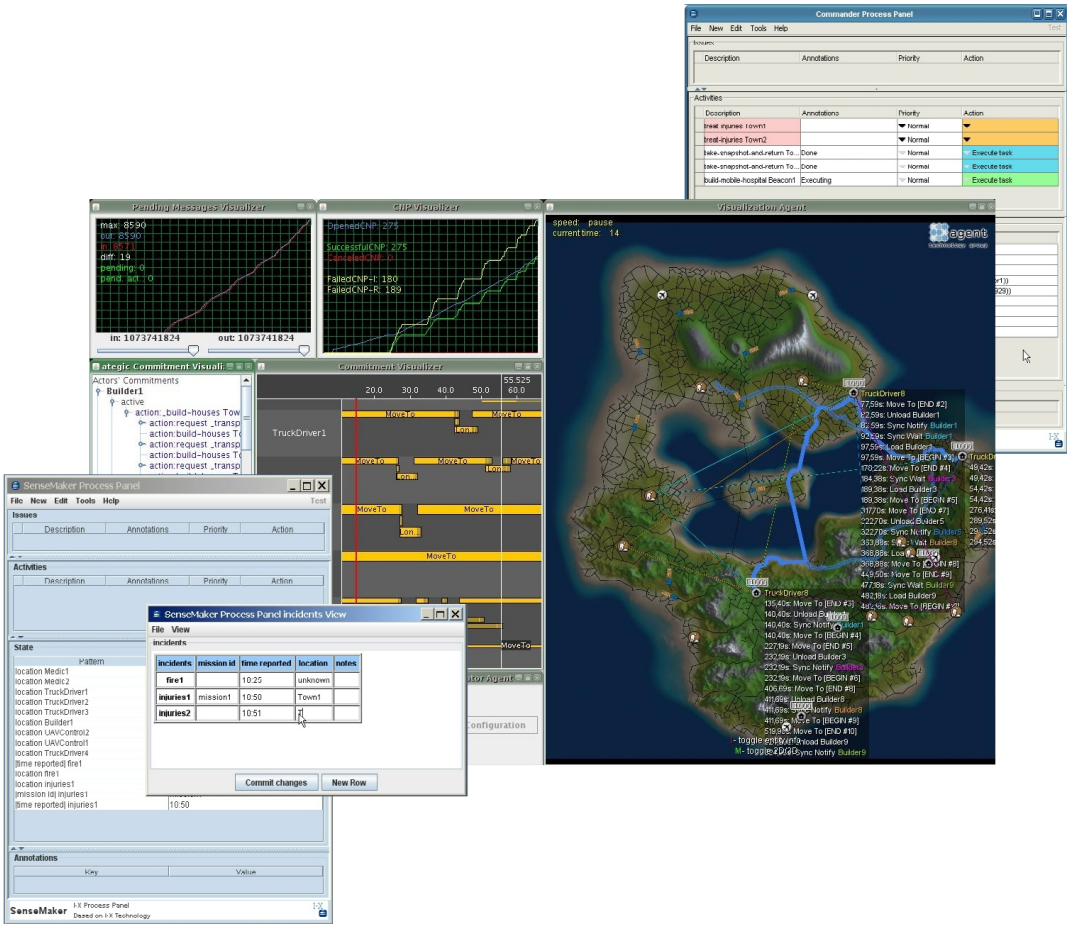

Figure 1: The I-Globe system

1.2 ACROSS2 Integration

To enable fast development, I-Globe is being built on top of the existing ACROSS2 simulation platform. The platform was adjusted and extended in many ways to fit the needs of the I-Globe project. One of the most significant extensions is the new support of turn-based simulation, allowing the entire (otherwise real-time) simulation to automatically pause itself and wait until all activities scheduled to be executed in a particular simulation clock tick are completed. This feature turned out to be crucial since the planning processes generally require significantly more time than other actions taking place in the virtual world, which is the exact opposite to the real-world situation we are trying to simulate.

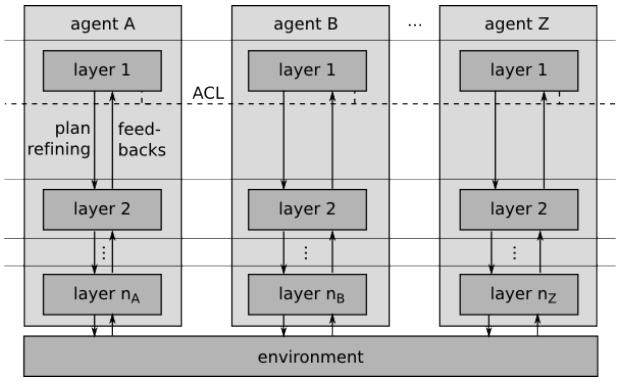

Figure 2: Multi-layer Planning Architecture

1.3 I-Plan Integration

For the purposes of the demonstrator, I-X was upgraded to I-X 4.5 which is the open source version of I-X that was released in March 2008.

All units/agents in the simulation use the I-X planner I-Plan to do their strategic planning. Each unit/agent has its own instance of I-Plan together with a corresponding I-Plan domain description that was designed for each type of unit present in the scenario: builder, truck, airplane and commander. It is important to point out that all demos use *the same* set of domains for two main reasons: (i) to keep the domain maintenance as simple as possible and (ii) to demonstrate the versatility of the domain definitions. I-Plan problem definitions for each planning session are automatically derived from the current state of the world as perceived by a particular unit/agent.

1.4 Agents' Control Modules

In order to maximize the modularity of the system, agents' decision-making algorithms are implemented in the form of pluggable control modules. These modules can be grouped into hierarchic trees through which time-update notifications, sensory data and action feedback flows transparently, depending on the particular control module type. This way the behavior of each unit/agent can be defined by a combination of different control modules. For instance, one control module can be responsible for path planning, another for strategic planning, and yet another can represent the agent's tactical layer etc.

1.5 Multi-layer Planning Architecture

One of the main accomplishments has been the design and implementation of the multi-layer planning architecture. The architecture is based on a hierarchy of planning layers using the extended form of social commitments as the integration part. Because of the modularity of the layers, the system is ready for implementation of various planning methods and can be easily used for their comparison. The layers are mutually interconnected, saying that each layer is directly connected only to its neighbour layers (Figure 2). As the layer forms a batch, each layer is connected to $\langle 0, 2 \rangle$ other layers.

The top-down direction means the successive refinement process of the plan, plan instantiation, concretization, and finally plan execution. On the other hand, the bottom-up direction is used for controlling

of the planning process, which means informing the higher levels of the planning results, success, and failure feedback.

The loose layer coupling and simple process flows enable volatility minimization of the plans, since there is no need to replan the plans on higher layers (typically more abstract and long-term plans) in the case of only minor violations (which can be solved by the bottom-level replanning). As the higher levels of planning typically invokes much more resource-consuming processes (as inter-agent negotiation, HTN planning, etc.), this separation helps to optimize the planning process as a whole.

Each agent uses all layers or their own subset. For example, the commanding unit does not use the bottom-most layer, because it is not typically executing the plan at all and instead uses its subordinates to implement the plan in the environment. The basic architecture design consists of three layers:

- Strategic layer: The layer provides an overall strategic plan for the middle- and long-term time horizon using I-Plan for high-level planning, and distributed allocation based on the multi-CNP protocol and social commitments.
- Tactical layer: On this layer, the strategic plan is optimized using a commitment condensation algorithm. The algorithm searches for blank time slots in the plan and replaces them with later commitments (taking in account the time constraints of the commitments).
- Individual layer: On this layer, the units perform reactive behavior and path-finding based on the tactical plan (i.e. an ordered set of commitments).

1.6 Distributed Plan Instantiation

Since in I-Globe there are several independent entities acting in the scenarios, the coordination and cooperation of such actors refers to a complex distributed planning and allocation problem. Although the planning is provided by a set of I-X planners, the distributed resource allocation has to be solved with respect to the individual actors' constraints and abilities.

We have designed and implemented mechanisms for distributed allocation based on allocation for recursively refined hierarchical plans. This approach enables efficient coordination of the distributed autonomous units. The basic idea is to create a raw plan to fulfill a task assigned to the agent. This plan contains activities that can be performed by the agent and some activities out of the scope of the agent's capabilities. So the agent has to find other agents to assign such activities to them. The allocation is achieved by the means of negotiation between agents in the contract-net-protocol manner. During the negotiation all the responding agents refine requested activities (i.e. create a more detailed plan) and potentially generate new requests in the lower level of the plan hierarchy. This process is recursively repeated until all the atomic activities are assigned to the particular actors.

For capturing the frame of negotiation we have introduced the commitment representation as a context of agents' agreements. The distributed allocation (and also planning) is then represented by the negotiation over commitments. The agent is able to commit itself to perform requested tasks within the scope of the agreed commitment. The plan to fulfill the task is prepared by the agent locally. For the parts of the plan that the agent is not able to perform by itself, the contract-net-protocol is initialized to find another agent that commits to it. This approach transforms the distributed allocation problem into negotiation over commitments. The whole distributed plan can be then viewed as a commitment graph.

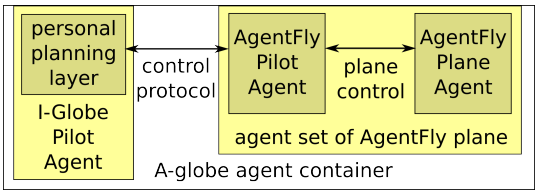

Figure 3: I-Globe – AGENTFLY integration schema

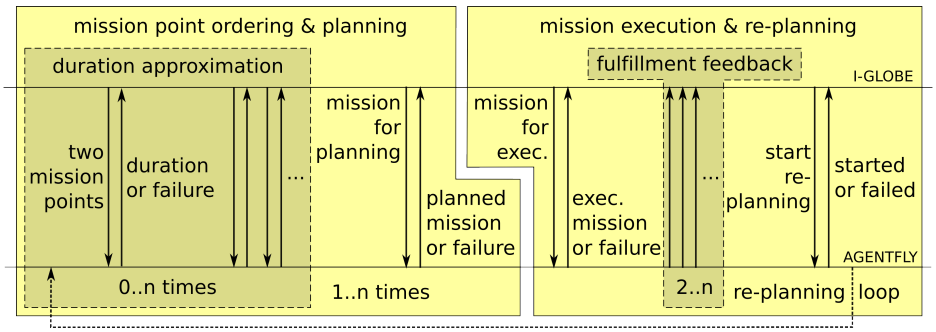

Figure 4: I-Globe – AGENTFLY control protocol

1.7 AGENTFLY Integration

The AGENTFLY system was integrated as one of the possible individual layers in the planning architecture. The AGENTFLY system is a simulation air-traffic-control system developed by ATG, focusing on a multi-agent control of unmanned aerial vehicles and adopting the free flight concept.

In order to seamlessly integrate the two existing systems, an AGENTFLY wrapper was implemented (Figure 3). On one hand, the wrapper fulfills the needs of the multi-layer planning architecture and on the other it acts as a control module of the AGENTFLY pilot agent. Using this wrapper, the I-Globe system can transparently control the missions of the simulated UAVs and does not need to implement the plane's behavior and simulation on its own. The planner wrapper is based on an agent-to-agent protocol (Figure 4).

The protocol can handle two main operations: mission definition with planning and plan execution with replanning. The initial definition of the mission is a repeatable process of plan duration approximations and planning attempts. The plan is executed after its agreement from both sides. During execution, AGENTFLY notifies I-Globe of successfully achieved mission points and/or of mission spatio-temporal deviations. I-Globe is able to change a part of the currently processed AGENTFLY plan. This process takes into account the time delays between the two planning systems. It is based on the concept of an unchangeable mission point. Such a point is designated in run-time and lies on the processed plan in the future. Only the plan parts after that point can be changed by replanning, which implies the communication between the two systems must be done in the same time interval as the unchangeable point lag. If the time interval is exceeded, the replanning starts again. The replanning process includes the whole planning process: mission definition, planning, and execution, but the new mission begins at the unchangeable point. The replanning process is always initiated by I-Globe, and it can be at any time the I-Globe system needs it. AGENTFLY can only notify I-Globe of deviated mission points, but the decision about whether replanning is necessary is down to the I-Globe system. The replanning necessity is resolved by the multi-layer planning architecture, concretely by the commitments modification process.

1.8 Commitments Implementation

We have implemented the commitment representation of the plans according to the in-project commitment research (Section 3.1 and 3.2). The commitment graph is implemented using the composition pattern and reference-linked graph. The operations on the graph are implemented as a mixture of several programming patterns: visitor, iterator, and strategy.

The implementation includes the local intra-agent monitoring of commitment execution, overall inter-agent commitment execution monitoring, feedback propagation, and the synchronization mechanisms based on synchro-points (Section 3.3.3) and violation links (Section 3.1.2). In the case of a non-deterministic scenario, the world is changing during commitment execution in a non-predictable way. Some events can lead to a need for commitment modification. We implemented the following cases: *Relaxation*, *Delegation*, *Decommitment*. All the above modifications of commitments can also affect the violation links and thus the entire commitment graph can be affected. Our goal of the design and implementation was to minimize the impact of commitment modifications and to minimize the changes in the commitment graph in general.

For the purposes of debugging and presentation we implemented two visualization tools for the commitments: a Temporal Commitment Visualizer and a Hierarchical Commitment Visualizer. These tools show the commitment graphs of the system in different ways.

1.9 I-P2 User Interface

The scenarios were extended so that the commander agent had an I-X Process Panel (I-P2) user interface[1, 2] to allow for mixed-initiative planning and to enable the commander to monitor the progress of subordinate agents.

A process panel acts as an intelligent, hierarchical "to-do" list. Since people are not good at remembering long lists of potentially unrelated tasks, a to-do list is part of virtually every tool that helps people organize their work. Listing tasks and ticking them off when they have been done is a simple means of ensuring that everything that needs to be done does get done, or at least that a quick overview of unaccomplished tasks is available. In responding to an emergency this is vital, and the larger the emergency, the more tasks need to be managed.

While the idea of using a to-do list as a basis for a distributed task manager is not new, I-X goes well beyond this metaphor and provides a number of useful extensions that facilitate the finding and adaptation of a complete and efficient course of action.

I-X is a framework that can be used to create an application in which multiple agents adopt a task-centric view of a situation, and which supports the necessary coordination of their activities to respond to that situation.

A process panel is the primary user interface to an I-X application. A panel reflects, and makes available

[1] Potter, S., Tate, A. and Wickler, G. (2006) *Using I-X Process Panels as Intelligent To-Do Lists for Agent Coordination in Emergency Response*, Proceedings of the Information Systems for Crisis Response and Management 2006 (ISCRAM2006), Special Session on "Multiagent Systems for Disaster Management and Response", Newark, New Jersey, USA, May 15-17, 2006.

[2] Tate, A., Dalton, J., and Stader, J. (2002) *I-P2 - Intelligent Process Panels to Support Coalition Operations*, Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations (KSCO-2002), 184-190, Toulouse, France, 23-24 April 2002.

to the user, the ontology that underlies all of I-X, <I-N-C-A>[3]. <I-N-C-A> is a generic description of a synthesis task, dividing it into four major components: Issues, Nodes, Constraints, and Annotations.

Issues may indicate outstanding questions to be addressed and can be about decisions yet to be taken, ways to satisfy objectives, items raised by analysis, and so on.

In task-centric applications, the nodes are the activities that need to be performed to carry out a course of action. They are thus the items in the to-do list. The activities can be broken down into subactivities and related by constraints of various kinds.

Constraints can impose a partial order on activities, indicate what resources they require, specify conditions that must be satisfied before an activity can take place, or the effects that an activity has, and so on. Constraints can also describe what is known about the state of the world in which an agent is acting, and it is possible for agents to exchange state constraints automatically so that they all have the same information.

Annotations may record the rationale behind decisions that were taken (for instance about which activities to include), progress reports, and other useful information.

The user can do more than just tick off activities as "done". An activity can be delegated to another agent, for example, or handled in a way that causes something to happen to the agent or in the world. New activity handlers can be plugged in to adapt the agent to particular tasks or situations.

In addition, there can be a library of standard operating procedures that break an activity down into sub-activities that, when all performed, accomplish the higher-level task. More than one way to break down, or "refine", an activity may be available, and the panel may be able to use world-state and other information to indicate which refinements can be used in the current situation. The knowledge contained in the library is important because it means that, from the user's perspective, I-X can suggest ways to perform activities and can allow the user to explore alternative courses of action.

A process panel also contains a planning component, I-Plan, that can find applicable procedures in the library and use them to automatically create a complete plan in which all activities are either broken down into subactivities or else cannot be broken down any further. I-Plan also ensures that all of the known constraints on the activities would be satisfied if the plan were carried out without interference.

1.10 State Subscription

Interfaces and protocols were developed and implemented to allow the commander to monitor relevant changes in the (simulated) world through a selective subscription mechanism.

I-X has a notion of an "I-Space" in which collaborating agents exist, and each I-X agent may have a model of its relationships to other agents and of their capabilities. Since hierarchical structures are common in organisations, the I-Space relationships include "superior" and "subordinate". An agent may also be another's "peer" or, when nothing more specific is known, a "contact".

I-Space mechanisms are used to specify the commander's subordinates; the commander then subscribes to the desired information from those agents, which in our demonstrations is their locations. The location information can indicate where an entity is or, if it is moving and its current position is not

[3] Tate, A. (2003) *<I-N-C-A>: a Shared Model for Mixed-initiative Synthesis Tasks*, Proceedings of the Workshop on Mixed-Initiative Intelligent Systems (MIIS) at the International Joint Conference on Artificial Intelligence (IJCAI-03), pp. 125-130, Acapulco, Mexico, August 2003.

precisely known, where it is travelling from and to, or what other entity is transporting it. To facilitate reasoning about locations, a particular location for an agent is represented by a list of the entities that are "co-present" with the agent at that place. For example, if agent Builder6 is in Airport3 in Town5, then it is in Town5 but also in Airport3 at the same time. Therefore, Airport3 and Town5 would both be included in the co-presence list.

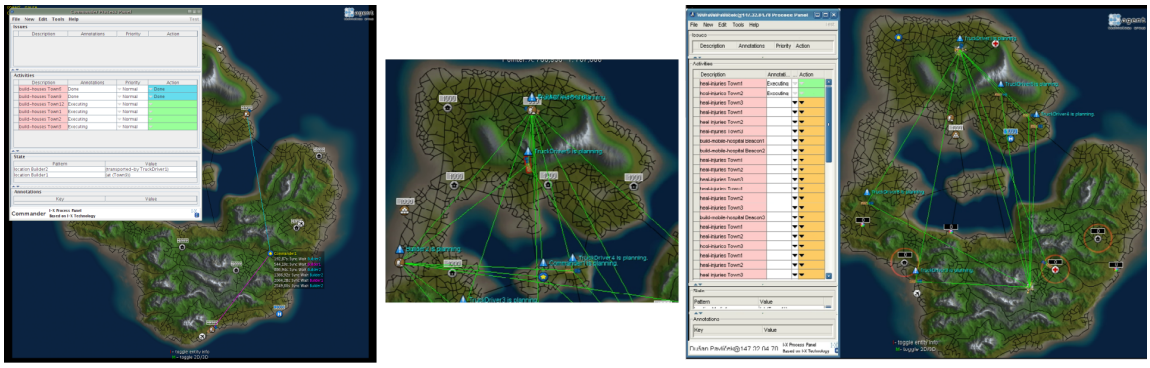

Figure 5: General scenarios situations - Commander demo, Truck breakdown demo, Complex demo

2 Scenario and Demos

We have created 8 major demonstration scenarios and a number of minor scenarios for testing purposes. All of the scenarios utilize the same environment, a hypothetical island with towns, harbors, airports, resource storages, roads etc. This virtual world is powered by the ACROSS2 simulation platform. Various parameters of all the demos can be adjusted via configuration files.

There are five general demo scenarios:

- **Commander demo (**`demo-commander`**)** – this demo presents a commander agent/unit responsible for assigning tasks to his subordinate builder units dynamically at runtime. Upon receiving a new task, the builders have to react appropriately and update their plans. Moreover, if construction material is not available at the construction site, the corresponding builder has to hire yet another transporter to deliver the materials there from a storehouse before the builder can start working.
- **Truck breakdown demo (**`demo-broken-trucks`**)** – this demo features dynamic decommitments and replanning as the trucks in the demo can and will break down. The builders in this scenario have their usual tasks to fulfill. Every time a truck transporting a builder breaks down on its way, the truck decommits all of its commitments, and the corresponding builder has to find a substitution, i.e. find a different unit to transport him to the destination.
- **Air-transport demo (**`demo-air-transport`**)** – this demo introduces transport airplanes to illustrate multi-level coordination. This time, builders hire helicopters (as they are faster than trucks) to handle the transportation, but since helicopters can only fly between heliports, the helicopters in turn have to hire trucks to take care of the transportation of units or resources between heliports and towns if necessary.
- **Complex demo (**`demo-complex`**)** – according to the scenario of this demo, the area has been struck by an unspecified disaster leaving many injured people in three cities in the south. Over the course of this demo, the commander issues orders to treat the injuries in each of the towns. The available medical units take care of these tasks. In order to get to the specified location, they have to hire either helicopters or trucks to transport them there. Before the medics can start working, medical material also has to be available at the location. The medics therefore hire trucks to handle the transportation of the material from the rather distant hospital. While the operation is running, the commander orders three mobile hospitals to be built closer to the affected area to speed up transportation of the medical material. Builders have to hire available

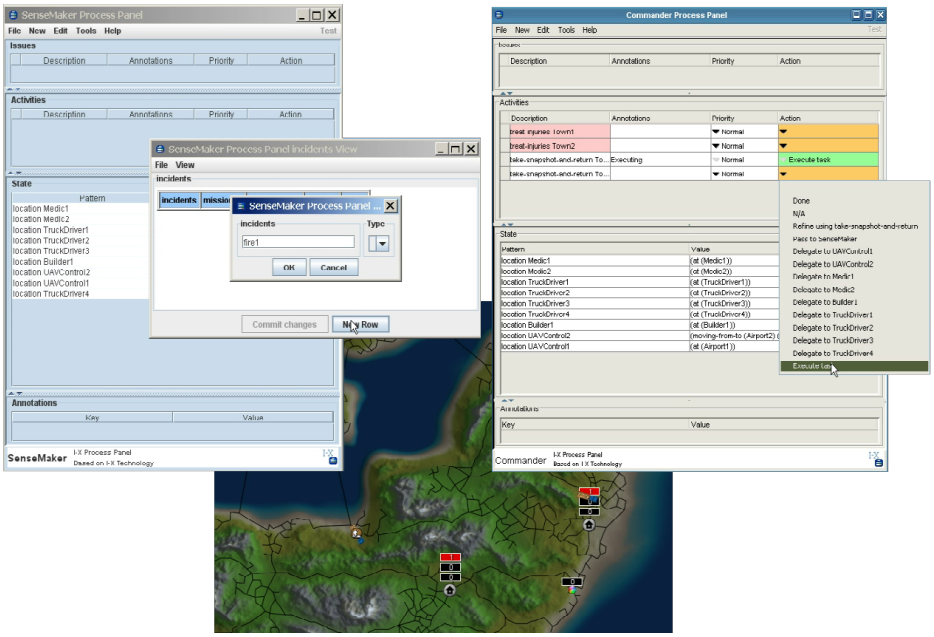

Figure 6: I-X integration demo - Sense maker GUI, Commander GUI

trucks to transport them and the construction material to the construction sites. After the mobile hospitals are built, the trucks no longer have to drive to the distant hospital for medical material and they take advantage of the nearby mobile hospitals instead.

- **I-X integration demo (`demo-sensemaker`)** – this demo features an external sense-maker operator who communicates with the commander via I-X panels. The sense-maker reports various events occurring in the area to the commander. Based on this information, the commander issues specific tasks to be performed by the subordinate field units. For each task, the field units generate and execute a corresponding distributed plan and subsequently report the tasks' completion back to the commander. The demo also presents multi-level realtime state sharing between the sense-maker, the commander, and the field units.

Three demos present I-Globe's successful integration with the AGENTFLY system.

- **Basic demo (`demo-agentfly`)** – builders hire airplanes to transport them to construction sites (where they build houses as usual). Unlike in the previous demos featuring helicopters, planning of airplanes' trajectories in this demo is carried out by the AGENTFLY system which essentially serves as an external library providing collision-free flight path planning functionality. Note that 'S', 'K' and 'L' keyboard shortcuts can be used in the visio to display realtime 3D data related to AGENTFLY path planning.

- **Delegation demo (`demo-delegation`)** – this demo illustrates task delegation among units. A helicopter in the north is tasked to perform a surveillance mission by taking snapshots in a number of towns all over the island. When the delegation process is invoked, the helicopter passes a portion of its mission (a subset of its original mission targets that are most difficult to reach in terms of distance and required time) to another helicopter in the south. Both helicopters then cooperatively fulfill the entire mission in a more efficient way.

Figure 7: AGENTFLY integration scenario situations - Basic demo, Delegation demo, Tracking and delegation demo

- **Tracking and delegation demo (`demo-surveillance`)** – there are two non-cooperating commanders present in this demo. The one in the north controls a helicopter while the one in the south controls a truck. Both the helicopter and the truck receive their initial mission from their commanders. The helicopter is tasked to perform surveillance of the island while the truck's task is to drive from its initial location in the north back to its commander located in the south. When the helicopter notices an enemy truck, it interrupts its original surveillance mission and starts tracking the truck instead. At the same time, the helicopter delegates the tracking task to an small unmanned helicopter so that it can carry on with its original surveillance mission.

There are also the following three testing demos:

- **Basic builders demo (`test-build-houses-in-city`)** – in this demo, a group of builders all have the same task to build houses in a given town. Since builders cannot move by themselves, they have to find (hire) transport units to transport them to the respective town in order to build houses there. Virtual synchro-points are inserted into all units' plans transparently in order to ensure proper spatio-temporal synchronization of collaborating units.
- **Resources demo (`test-resources`)** – this demo is derived from the previous one, only this time there's no construction material available at the construction sites by default; so apart from hiring trucks for their own transportation, all builders also have to hire trucks to transport the construction material which results in more complex negotiation and planning.
- **Multi-task demo (`test-multi-task`)** – this demo illustrates the concept of multitasking in I-Globe. Each builder in this demo is tasked to build houses in five different towns, and has to hire trucks to transport him to the particular towns. The units generate their plans covering the entire sequence of tasks (rather than generating the plan incrementally). This once again leads to a complex negotiation and planning process.

3 Research

3.1 Commitment Based Planning

We have analyzed the state of the art in the field of distributed planning and social commitments[4].

We have extended the well-known formal model of social commitment by Professor Wooldridge and proposed its recursive form, usable for nesting of commitments and a more expressive formal model of the commitment convention. This model allows us to describe the decommitment rules, which describe the mentioned commitment modifications within the commitment and thus within the plan. The modifications are:

- **Relaxation** – when the committed constraints cannot be kept, the constraints should be re-negotiated to preserve the commitment.
- **Delegation** – in case of a failed relaxation, the commitment may be delegated to another actor.
- **Decommitment** – when all the previous possibilities fail and the commitment cannot be executed with the agreed constraints, it can be dropped. In this case, reallocation and potentially replanning takes place.

All the above modifications of commitments can also affect the violation links and thus the whole commitment graph can be affected. Our goal is to minimize the impact of commitments modification and minimize the changes in overall commitment graph.

Using the extended form of the social commitment, we proposed a graph notation of the commitments. The mutual bindings and commitments form a commitment graph. The commitment graph describes the same properties of mutual decommitting as the logical notation. The graph notation can be used to describe the process of successive solving of the exceptional states. The process is based on traversing through the commitment graph.

3.1.1 Definition

A social commitment is a knowledge structure describing an agent's obligation to achieve a specific goal, if a specific condition is made valid, and how it can drop the commitment if it cannot be achieved. The commitment does not capture a description how the committed goal can be achieved. Individual planning for goal achievement, plan execution and monitoring is a subject of agents internal reasoning processes and is not represented in the commitment.

In the context of the planning problem defined in the Introduction, we understand the agent's specific goal (to which it commits) as an individual action, a component of the plan, which resulted from the given planning problem. While a typical action in a plan contains only a precondition and an effect, in this report we will describe how its representation can be extended so that the commitment-related information is included.

[4]More information can be found in Komenda, A., Pechoucek, M., Biba, J., Vokrinek, J.: *Planning and Re-planning in Multi-actors Scenarios by means of Social Commitments*, Proceedings of the International Multiconference on Computer Science and Information Technology, 2008, 39-45.

Michael Wooldridge defines the commitments formally as follows:

$$\begin{array}{l}(\text{Commit } A\ \psi\ \varphi\ \lambda),\\ \quad \lambda = \{(\rho_1, \gamma_1), (\rho_2, \gamma_2), \ldots, (\rho_k, \gamma_k)\},\end{array} \tag{1}$$

where A denotes a committing actor, ψ is an activation condition, φ is a commitment goal, and λ is a convention. The convention is a set of tuples (ρ, γ) where ρ is a decommitment condition and γ is an inevitable outcome. The convention describes all possible ways how the commitment can be dropped. Generally speaking, the actor A has to transform the world-state in such a way that the φ goal becomes true if ψ holds and any γ has not been made true yet. The actor is allowed to drop the commitment if and only if $\exists i : \rho_i$ which is valid. A decommitment is allowed provided that γ_i is made true. A formal definition in modal logic (working with the models of mental attitudes like Believes, Desires, Intentions, and temporal logic where the operator AG denotes an the inevitability and operator $\curvearrowleft$ denotes the temporal until) follows:

$$\begin{array}{l}(\text{Commit } A\ \psi\ \varphi\ \lambda) \equiv\\ \quad ((\text{Bel } A\ \psi) \Rightarrow \mathsf{AG}((\text{Int } A\ \varphi)\\ \qquad \wedge(((\text{Bel } A\ \rho_1) \Rightarrow \mathsf{AG}((\text{Int } A\ \gamma_1))) \curvearrowleft \gamma_1)\\ \qquad \ldots\\ \qquad \wedge(((\text{Bel } A\ \rho_k) \Rightarrow \mathsf{AG}((\text{Int } A\ \gamma_k))) \curvearrowleft \gamma_k)\\ \quad) \curvearrowleft \bigvee_i \gamma_i).\end{array} \tag{2}$$

This definition is used in a declarative way. Provided that whatever the agent does during a specific behavior run complies with the above defined commitment, the expression 2 is valid throughout the whole duration of the run.

One of the goals of the research described in this report was to provide a formalism for networked commitments to be used for replanning. The commitment conditions can represent variable bindings among preconditions and effects of the individual commitments achieved either by monitoring the environment status or by inter-agent communication (e.g. reception of a specific trigger message). Such representation would be very inflexible in practical applications as it would either need the agents to do nothing and wait for an inhibiting event to happen or risk that once an inhibiting event happens the agent will be busy performing other commitments. Therefore the agents may want to engage in booking and the commitment's precondition would contain a fixed time when the commitment is supposed to be adopted. The most flexible approach would be a combination of both - inhibition event and preliminary booked time window, specifying when the inhibiting event is likely to happen. Let us assume that this is the case in what follows.

In the distributed plan execution, a failure may occur. The indirect impact of this failure may be e.g. a situation where the arranged inhibition event will not happen in the preliminary booked time window. Such occurrence may invoke replanning and allow some agents to e.g. drop unnecessary commitments. This is the reason why the commitments shall not be linked one with other not only via preconditions but also by means of variable bindings among individual agent's decommitment rules. Using these bindings, we can describe the causal sequentiality of the commitments and requests for particular decommitments – Figure 8.

While we will be generalizing on the process of decommitment later in the report, let us work for now with the specific particular decommitment case suggested in the previous paragraph. Let us assume one agent A forcing decommitment of the other agent's B commitment by means of setting a value of a variable contained in the other agent's commitment. The agent A contains a commitment with a decommitment rule in the form $\langle \rho, v \rangle$ and the agent B contains a commitment with a decommitment

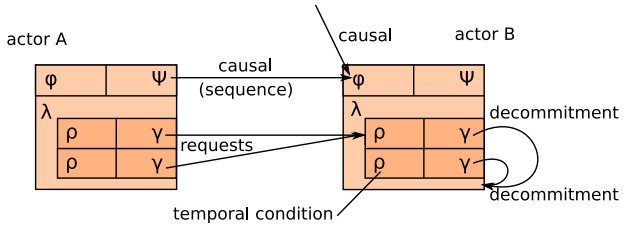

Figure 8: Commitments and bindings - the actor A's commitment influences the actor B's commitment using the causal (sequential) link, the link is described using the ψ and φ clauses (e.g. $\psi =$ `building-is-ready(B)` and $\varphi =$ `ready(B)`). The actor B's commitment is influenced by external causality too. The actor B's commitment can be decommitted in two cases: either the *temporal condition* ρ becomes true or one of the actor A's rules *requests* the decommitting. The decommitment request is triggered by one of the actor A's ρ conditions.

rule in the form $\langle v, \texttt{decommit}(B)\rangle \in \lambda_A$.The request is started by ρ precondition of the actor A (e.g. decommitting the A's commitment). Thus the actor A intends to make the variable v valid. This causes the agent B to intend to decommit by intending the variable `decommit`(B) to be valid (see Figure 8).

This clear example uncovers two needed extensions of the classical social commitment model: (*i*) recurrence of the commitment form – enabling a possibility to disable (decommit) a decommitment request and (*ii*) explicit termination condition – describing termination without any intentional part.

3.1.2 Commitment Recurrence

The original Wooldridge definition of a commitment makes a clear distinction between the commitment subject (φ) and the mini-goals set in the commitment convention (γ). While there is a mechanism for the agent to drop φ, a once adopted mini-goal γ cannot be decommitted. Due to high dynamism and uncertainty of the target scenario, we assume the replanning and plan repair mechanisms to be substantially more complex. We require that the mechanism would allow the agent to try out several different decommitment alternatives, based on the current properties of the environment. The set λ, allows listing various different decommitment rules, while no mechanism have specified how different decommitment alternatives are tried out.

That is why we propose generalization of the commitment so that each goal in the commitment structure can be treated equally. Let us introduce the recursive form of a commitment, which enables the nesting of the commitments – Figure 9:

$$\begin{aligned}&(\text{Commit } A\ \psi\ \varphi\ \lambda^*), \lambda^* = \\ &\quad\{(\text{Commit } x_1\ \rho_1\ \gamma_1\ \lambda_1^*),\\ &\quad(\text{Commit } x_2\ \rho_2\ \gamma_2\ \lambda_2^*), \ldots,\\ &\quad(\text{Commit } x_k\ \rho_k\ \gamma_k\ \lambda_k^*)\}.\end{aligned} \tag{3}$$

The formula 3 extends the definition in 2 not only by inclusion of a set of decommitment rules in each of the individual decommitment rules; it also allows the newly adopted commitments to be assigned to different actors. The delegation kind of decommitment between two agents A and B would have the following form:

$$(\text{Commit } A\ \psi\ \varphi\ \{(\text{Commit } B\ \rho\ \varphi\ \emptyset)\}), \tag{4}$$

representing that agent A can drop the commitment towards φ provided that ρ is valid and provided that B accepts a commitment towards φ on A's behalf.

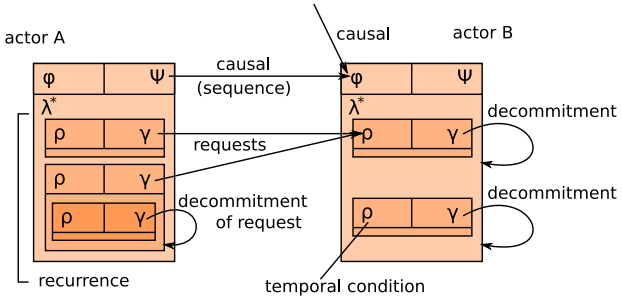

Figure 9: Commitment and its λ^* commitments – Figure 8 – is extended by one *decommitment of request* which can be decommitted if the most inner ρ condition becomes true. Decommitting of the request means the actor B's commitment cannot be decommited by the actor A's convention goal any more. Here the recursive form enables the nesting of the inner commitment.

This form is very expressive in the sense of the description of exceptional states. It allows us to have a branched chain of individual nested commitments for each individual situation. The recursive nature allows us to describe an arbitrarily complex protocol using only one knowledge base structure — a recursive form of the commitment. The recursive form of the commitment is thus defined as:

$$\begin{aligned}&(\text{Commit } A\ \psi\ \varphi\ \lambda^*) \equiv \\ &\quad ((\text{Bel } A\ \psi) \Rightarrow \mathrm{A}((\text{Int } A\ \varphi) \wedge \bigwedge_j \lambda_j^*) \curvearrowleft \bigvee_i \gamma_i).\end{aligned} \tag{5}$$

3.1.3 Decommitment Rules

We require the agents that perform intelligent planning and replanning by means of social commitments to be able to perform at least basic reasoning about the decommitment rules attached to the particular commitments. This is needed at the time of replanning, when an agent needs to decide which decommitment rule (i.e. a new commitment) to adopt, provided that conditions for more than one are satisfied. Similarly, agents, when they negotiate about who will accept which commitment, shall be able to analyze not only properties of the goal and costs associated with the goal completion process but also the various decommitment rules when considering likelihood of the particular failure to happen. Ideally, the agent shall be able to estimate costs of each decommitment rule. However, with the lack of information about the dynamics of the environment, we will only be able to partially order the decommitment rules by assigning them to different types. Let us introduce three different types of decommitment rules:

- *Termination conditions* (TC) - the most preferred decommitment rules as no further action is required for dropping the particular commitment.
- *Delegation* (D) - by using this type of commitments the agent shall be able to find some other agent who will be able to complete its commitment on the original agent's behalf. It is possible

that such a commitment will contain unbound variables representing the need to search for an agent suitable for delegation.

- *Relaxation* (R) - is a special decommitment, where the original commitment is replaced with a new commitment with relaxed condition and/or goal. The new commitment must be consistent with all other bound commitments. If the bound commitment is of another agent, the relaxation must be negotiated. The asked agent tries to fit the requested relaxed commitment into its knowledge base and eventually use some other decommitment rules of other commitments to change it and fulfill the request.

The main commitment properties can be summarized into three types:

- *Individual commitments* (IC) - commitments that do not involve an agent other than the agent itself. These commitments shall be used if the impact of a failure within the multi-agent community shall be minimized. Individual commitments shall represent several other ways how an agent can accomplish a given task.
- *Minimal social commitment* (MSC) - is the classical type of decommitment, where the agent is required to notify the members of the team about its inability to achieve the commitment.
- *Joint commitments* (JC) - these commitments provides mutually linked commitments (of several agents) via decommitment rules. In a replanning situation the joint commitments proactively assure that the cost of the failure is minimized. An example of the use of a joint commitment is decommitting another agent's linked commitment as explained in Section 3.1.1

During the replanning process, the preference relation over the commitments defines how the particular commitments can be fixed and thus the way the whole replanning of the plan can be done, and all this with consideration of current circumstances in the environment.

3.1.4 Commitment Graph

Using the extended form of the social commitment we can propose a graph notation of the commitments. The mutual bindings and commitments form a commitment graph – Figure 10. The commitment graph describes the same properties of mutual decommitting as the logical notation.

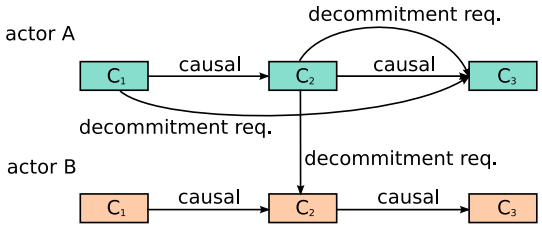

Figure 10: Commitment graph – the *causal* links define the sequentiality of the commitments of each actor. The commitment C_3 of the actor A can be decommitted by both C_1 and C_2 commitments. The C_2 commitment of the actor B can be decommitted by actor A using the decommitment request.

The graph notation can be used to describe the process of the successive solving of the exceptional states. The process is based on traversing through the commitment graph. The traversing starts with the first violated commitment. One of the decommitment rules is triggered (according to the violation

type). As the decommitment rule is a commitment, it starts an intention of the agent to terminate the commitment. In the case that the intention is a decommitment request, the process crosses on the requested commitment (decommitment rule respectively) and starts one of the decommitment rules on the side of the requested commitment. Provided that the decommitment rule terminates the commitment without a need to request other decommitments, the process ends here and the violation is fixed.

3.2 Decommitment Rules

As stated in Section 3.1.3, the decommitment rules are a crucial part of the replanning process. Since the decommitment rules can be used in several configurations (including ordering and selection), we needed to study how particular decommitment rules can affect the execution and thus replanning in a highly stressed non-deterministic environment.

We require the agents that perform intelligent planning and replanning by means of social commitments to be able to perform at least basic reasoning about the decommitment rules attached to the particular commitments. This is needed at the time of replanning, when an agent needs to decide which decommitment rule (i.e. a new commitment) to adopt, provided that conditions for more than one of them are satisfied. Similarly, when negotiating about who will accept which commitment, the agents shall be able to analyze not only properties of the goal and costs associated with the goal completion process but also the various decommitment rules when considering likelihood of the particular failure to happen. Ideally, the agent shall be able to estimate costs of each decommitment rule. In the scope of this report we are not addressing the agents' decision making, but we focus on the performance and usability of several decommitment strategies settings during execution of the commitments in a dynamic environment.

We have recognized three main types of decommitment usually used in commitments: (i) *Full Decommitment* for dropping the commitment, (ii) *Delegation* of the commitment to another agent, and (iii) *Relaxation* of the time frame of the commitment. The decommitment condition for each decommitment strategy is defined to enable flexibility of the commitment under various circumstances. During the planning process, the preference relation over the decommitments is defined as a part of the decommitment rule set. The decommitment rules are unordered according to the definition (2) and thus we must slightly change the definition of the commitment:

$$\begin{aligned}&(\text{Commit } A\ \psi\ \varphi\ \lambda),\\ &\quad \lambda = \big((\rho_1,\gamma_1),(\rho_2,\gamma_2),\ldots,(\rho_k,\gamma_k)\big),\end{aligned} \tag{6}$$

where the ordering is fixed and the rules are processed in a specific order. The processing of the rule means that dropping a part of the commitment definition in (2)

$$\bigvee_i \gamma_i \tag{7}$$

simplifies to

$$\gamma^*, \tag{8}$$

where γ^* is the inevitable outcome of an active decommitment rule. There is only one active rule for each commitment and the rules are switching from the first rule to the last one. The switch is performed only if γ^* is not realizable and ρ of the next active rule holds. The switching process uses the defined fixed ordering of the rules to determine the correct succeeding rule.

According to our understanding, each decommitment rule set (corresponding to Wooldridge's commitment convention) must contain two basic rules, which ensure the rationality of the agent's decision

making process. These rules are based on the definition of the *open-minded commitment* defined in [**?**]:

$$\begin{array}{l}(\text{Commit } A\ \varphi) \equiv \\ \quad \text{AG}((\text{Int } A\ \varphi) \curvearrowleft ((\text{Bel } A\ \varphi) \vee \neg(\text{Bel } A\ \text{EF}\varphi))),\end{array} \tag{9}$$

where the operator EF denotes future possibility. Thus in each and every commitment the initial rule should be the success rule

$$(false, (\text{Bel } A\ \varphi)) \tag{10}$$

and the decommitment rule set should be always closed with a fail-safe rule turning a violated commitment eventually off

$$(false, \neg(\text{Bel } A\ \text{EF}\varphi)), \tag{11}$$

which is used provided that no other rule can be used and the commitment became unrealizable.

The decommitment condition ρ in rules (10) and (11) cannot become true (as $\rho = false$), which means the agent will never intend to the decommitment rule outcome γ according to the definition (2)

$$\begin{array}{lcl}((\text{Bel } A\ \rho) & \Rightarrow & \text{AG}((\text{Int } A\ \gamma))) \curvearrowleft \gamma \\ (false & \Rightarrow & \text{AG}((\text{Int } A\ \gamma))) \curvearrowleft \gamma,\end{array} \tag{12}$$

nevertheless the rule can drop-out the commitment using the drop-out part

$$\bigvee_i \gamma_i \text{ or } \gamma^* \tag{13}$$

respectively. This principle can be used for any drop-out rule with no need for explicit intention.

Definition 3.1 *Each commitment can be decommitted if the commitment goal φ is achieved (the commitment succeeded) or if the commitment goal φ can not be achieved any more (the commitment is violated).*

The formal definition of the decommitment rule set in each commitment follows

$$\begin{array}{l}(\text{Commit } A\ \psi\ \varphi\ \lambda), \\ \quad \lambda = \big((false, (\text{Bel } A\ \varphi)), \\ \qquad \ldots \text{investigated decommitment rules} \ldots, \\ \qquad (false, \neg(\text{Bel } A\ \text{EF}\varphi))\big),\end{array} \tag{14}$$

where the three investigated rules are injected between two basic rules and thereby the last violation rule can be avoided. The rate of avoidance is one of the experimental metrics and is discussed in Section 4.1.

Three proposed decommitment rules can be defined using the adopted formalism as follows:

Definition 3.2 *Full decommitment decommits the original commitment if and only if the commitment goal φ is unrealizable.*

Definition 3.3 *Delegation decommits the original commitment if and only if the commitment goal φ is unrealizable and the new commitment on the other agent's side is formed.*

Definition 3.4 *Relaxation decommits the original commitment if and only if the commitment goal φ is unrealizable, the negotiated relaxation conditions hold and the relaxed commitment is formed.*

In the three following subsections, we describe the rules in more details and we formalize them using a temporal model, based on the duration time interval of the commitment being the only constraint of the commitment goal φ. This model is suitable for commitment-based planning, since the plan is a (partially) ordered list of temporally successive commitments.

3.2.1 Full Decommitment

The basic decommitment strategy is dropping the commitment. Under defined circumstances the agent is completely released from the commitment.

Let the commitment time interval $T_\varphi = \langle t_s, t_e \rangle$, where t_s is the starting time and t_e is the ending time of the commitment time interval. Let the commitment goal condition φ contain only defined temporal properties, then the decommitment rules can be described as:

$$\begin{array}{l} (\text{Commit } A\ \psi\ \varphi\ \lambda), \\ \quad (t_s^{est} > t_s \Rightarrow update(t_s, t_s^{est}), false) \in \lambda, \\ \quad (t_e^{est} > t_e, t_e^{est} > t_e) \in \lambda, \\ \quad t_e \in \varphi, \end{array} \tag{15}$$

where t_s^{est} and t_e^{est} are estimations of the real start and end of the activity. The first part of the rule describes continuous adjustment of the commitment's start time in the case the agent is forced to postpone its execution (which may not affect the end time condition and can not affect other commitments). The second part reflects Definition 3.2.

The t_s, t_e are the parameters of the commitment negotiated and fixed at the planning (contracting) time. The t_s^{est} and t_e^{est} are the agent's estimates of the real start and end of the activity and can vary over time.

3.2.2 Delegation

By using this type of the decommitment rule the agent shall be able to find some other agent who will be able to complete its commitment on the original agent's behalf. It is possible that such a commitment will contain unbound variables representing the need to search for an agent suitable for delegation. The basic idea is to find an agent that is able to undertake the commitment under circumstances when the decommitment condition (which is true in case of the original agent) became false, so the new agent is able to fulfill the commitment. The delegated commitment can contain a new set of decommitment rules.

Formally we can use the same variables as in full decommitment, but we are using

$$\begin{array}{l} (\text{Commit } A\ \psi\ \varphi\ \lambda), \\ \quad (t_s^{est} > t_s \Rightarrow update(t_s, t_s^{est}), false) \in \lambda, \\ \quad (t_e^{est} > t_e, (\text{Commit } B\ \psi\ \varphi\ \lambda)) \in \lambda, \\ \quad t_e, B \in \varphi, \end{array} \tag{16}$$

where B is the other agent undertaking the commitment.

3.2.3 Relaxation

Relaxation is a special decommitment, where the original commitment is replaced with a new commitment with relaxed condition and/or goal. In the scope of this text we are focusing on the relaxation of the commitment time interval for the sake of simplicity. The commitment time interval is usually captured by the commitment subject φ and specifies the time frame booked for the commitment execution. The temporal uncertainty can be a part of the commitment subject definition (and thus the whole commitment has to be renegotiated in case of any change) or, more preferable, it can be included in the commitment as an instance of a decommitment rule.

According to Definition 3.4, the decommitment rule can be then described as:

$$\begin{array}{l}(\text{Commit } A\ \psi\ \varphi\ \lambda),\\ \quad (t_s^{est} > t_s \Rightarrow update(t_s, t_s^{est}), false) \in \lambda,\\ \quad ((t_s^{est} < t_s) \wedge (t_s^{est} \in T_s^{rlx}),\\ \qquad (\text{Commit } A\ \psi\ \varphi\ \lambda) \wedge update(t_s, t_s^{est}) \in \lambda,\\ \quad ((t_e^{est} > t_e) \wedge (t_e^{est} \in T_e^{rlx}),\\ \qquad (\text{Commit } A\ \psi\ \varphi\ \lambda) \wedge update(t_e, t_e^{est}) \in \lambda,\\ \quad t_s, t_e \in \varphi,\end{array} \tag{17}$$

where T_s^{rlx} and T_e^{rlx} are the agreed relaxation intervals (negotiated relaxation conditions) for the start and end time. The T_s^{rlx} and T_e^{rlx} is an extended set of parameters negotiated and fixed at the planning time and the $update(t_s, t_s^{est}$ part changes the temporal parameters of the newly forming commitment to relaxed values.

3.2.4 Impact of Decommitment Rules

The impact of the particular rules is discussed in Section 4.1. We assume the complex combination of the decommitment rules provides non-trivial behavior and should improve the performance of the commitments' execution in non-deterministic environments under stress conditions (the system is overloaded).

Each of the presented rules provides different impact on the agent's current state. For example, relaxation helps to maintain the commitment execution, delegation effectively unblocks the agent's resources, and full decommitment releases the agent's resources by dropping the commitment. We expect a combination of the decommitment rules to emerge in a self-adaptation pattern that should lead to some sort of a real-time commitment execution optimization.

The decommitment rules introduced in this chapter have been implemented in the commitment-based planning system and experimentally evaluated. The next section describes the experimental scenario and discusses the influence of the rules on stability of the commitments execution and decommitment flexibility.

3.3 Distributed Planning

Distributed planning is decentralized process of the plan constitution. According to the proposed multi-layer planning architecture, the planning process take place on several levels (of abstraction and granularity) in several different agents. Those planning processes collaborate to form one global plan considering desired goals and particular agent constraints.

3.3.1 Distributed Plan Representation

In planning, the main role is acted by the appropriate distributed plan representation. The representation should be suitable for flexible planning and plan revision purposes and should provide execution robustness and effective apparatus for handling various types of plan execution exceptional effects. All these requirements can be handled using the concept of social commitments for planning (Section 3.1).

A social commitment is a knowledge structure describing an agent's obligation to achieve a specific goal, if a specific condition is made valid, and how it can drop the commitment if it cannot be achieved. For the planning purposes, the recursive form of the commitment (Equation 3) can be used.

3.3.2 Distributed Plan Forming

By the means of the commitments, the planning process can be described as committing to appropriate actions. The sequence of these actions can be found by any planning approach. We use HTN I-Plan as the main planner of the system. The planner is supported by distributed resource allocation based on extension of the CNPs (Section 3.3.4).

The instantiated plan is converted into commitments – Figure 11. The conversion process creates a commitment according to the particular plan action and according to forward causality links of the plan.

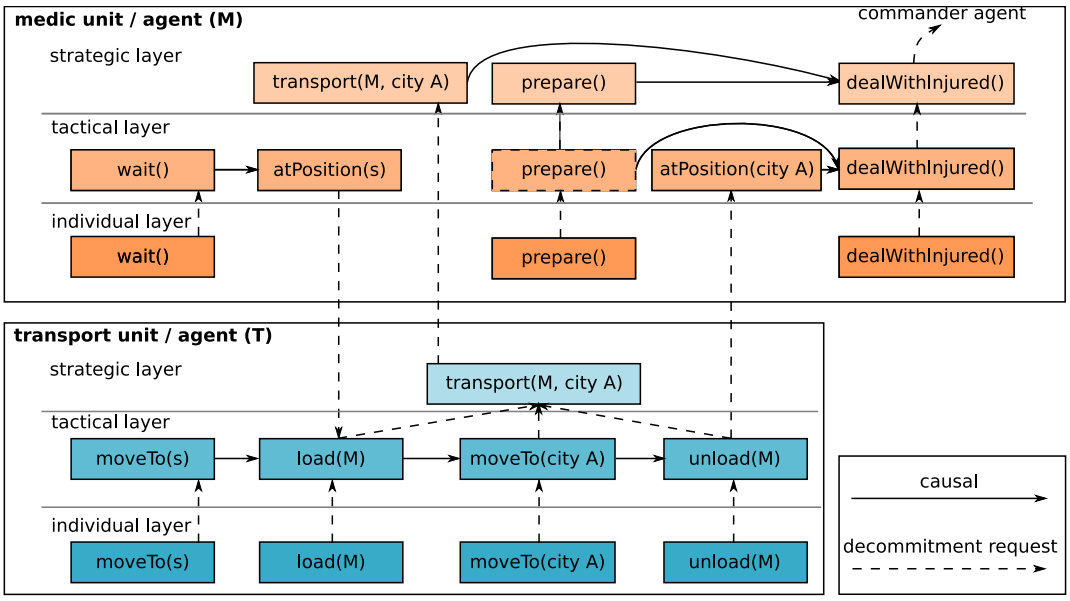

Figure 11: Example of commitment plan

The commitments of the tactical layer are based on strategic commitments. The layer uses negotiation to form the most suitable mutual commitments. The constraints for the negotiation respect the particular needs of the agents. The tactical commitments also define decommitments to the strategic layer, and they can additionally refine some strategic commitment too. They are much more refined than the strategic commitment in the sense of spatio-temporal constraints, and particular world-states. The individual layer plans commitments for later execution. These commitments copy the tactical commitments, but some of these can be omitted. Each individual commitment contains a decommitment request only to its parent commitment (from the tactical layer).

During the execution of the plan the commitments are processed. The commitments can be decommitted according to the plan or due to unexpected environment interactions.

The monitoring of the commitments is triggered by a change of the world: a tick of the world timer, movement of a unit, a change of a world entity state, etc. The process evaluates all commitments in the actor's knowledge base. The value of the commitment defines the commitment state and can start the decommitting process.

3.3.3 Distributed Plan Coordination

Since the plan is executed by autonomous actors in parallel, we had to define the efficient method for distributed plan execution coordination. For that purpose, we designed a concept of plan synchronization-points (or synchro-points in short). The synchro-point is a pair of commitments (or an actions in the planning terminology). One member of the pair is a wait commitment and the other is a notify commitment. Each side of the synchro-point is on different agent. The wait commitment does not succeed until the paired notify commitment is executed (or intended in the BDI terminology), which means that the execution of one agent's plan can be temporarily delayed until other agent finish some actions in its plan. The parameters of the spatio-temporal synchronization-points are negotiated during the process of forming of the plan.

In some cases there is need for synchronization of plans of agents, which are not neighbours in the planning hierarchy. Since the planning hierarchy can emerge into arbitrary number of sub-ordinate agent levels, the synchronization-points have to be able to handle synchronization of distant levels too. We chose an approach, in which the trans-level synchro-points are composed from primitive synchronization-points connecting only neighbour levels of the planning hierarchy and forming a chain across several plans and agents. More formally, we can define one line of the hierarchy as:

$$A \rightarrow B_1 \rightarrow B_2 \rightarrow \cdots \rightarrow B_n \rightarrow C, \tag{18}$$

where A, B_i, and C are agents. Now the trans-level synchro-point can be described using the primitive synchro-point as follows:

$$A \Longrightarrow C \equiv \{A \rightarrow B_1, B_1 \rightarrow B_2, \ldots, B_{n-1} \rightarrow B_n, B_n \rightarrow C\}. \tag{19}$$

The main advantage of this approach is no need for search of all causally affected agents in case of replanning.

3.3.4 Distributed Resource Allocation

We have defined three dimensions of resource allocation problem in our domain. The first two dimensions are two points of view on the similar type of problem. The third one is inverse problem.

The first resource allocation problem is to allocate a set of tasks, defined by the total ordered plan, which means that tasks are dependent and ordered. The tasks of the plan correspond to the linear horizontal commitment graph (see Section 3.1.4). Each agent creates a plan to achieve its commitment. According to the scenario and the domain of the agent, some parts of the plan cannot be executed by this agent and thus have to be delegated to other agents. The allocation problem is then to find the best candidates that are able to take over such parts of the plan (tasks). Every successive task allocation constraint depends on previous task allocations. The criteria function for allocation optimization is function of all tasks' allocations. This means (i) no task can be allocated optimally without knowledge

of other tasks' allocations and (ii) no task can be allocated without fixation of allocation of all preceding tasks.

The second resource allocation problem is hierarchical task allocation. This problem represents vertical commitment graph creation. When an agent commits (or even prepare the commitment) to some task, it has to decompose this task and make a plan for it (see Section 1.6). So each task that is being allocated implies the need of consequent allocation of other tasks on other actors.

The third problem occur when multiple independent tasks are requested simultaneously. This potentially produces (i) overbooking of best resource provider and (ii) unbalanced recourse utilization.

In the complex scenario, there should be a complicated hierarchical structure of the agent domains. All three allocation problems occur simultaneously. The agents' hierarchy leads to the both horizontal and vertical commitment dependencies. The task concurrency is caused by the multiple goals inserted in the system and by the substitute agents' capabilities. To get over the distributed allocation problem, we have implemented a progressive task allocation mechanism with an "as soon as possible" execution heuristic and backtracking. We have applied the recursive task allocation mechanism using iterative CNP with backtracking when allocation fails because of overbooking caused by concurrency. The efficiency and speed of convergence of the allocation mechanism is illustrated in Section 4.

4 Experimental Evaluation

This section shows the experimentally evaluated behavior of the system. We focus on the planning performance, resources utilization, and execution stability in various settings.

In the first experiment the influence of the decommitment rules on the stability of plan execution in the stressed environment is investigated. We focus on the individual decommitment rules and their combination[5]. The second experiment focuses on relaxation decommitment rule in various distributions of environment uncertainty and relaxation rule settings[6]. In the third experiment the planning and allocation algorithms are empirically investigated. The plan execution duration (lengths of the plans and their deviation across all actors in the system) is examined. In the fourth experiment we focus on the communication complexity of the distributed task allocation process. The amount of contract-net-protocols needed for planning and the speed of convergence is examined.

4.1 Decommitment Rules

The experiments show the influence of the selected decommitment rules and their order on flexibility, robustness and execution stability in the non-deterministic stressed environment.

In the experiments, the environment dynamics is simulated by non-deterministic prolonging of the activities. This dynamics is not taken into account by agents during the planning process. The prolonging events are generated for each agent individually using a uniform distribution with mean value $\bar{e} = 15000$ time units and variance $\sigma^2 = 8333$ time units. Each experiment has been performed on a sequence of 10 randomly generated runs. The duration of the prolonging event varies from 0 to 14000 to evaluate the system behavior under different stress conditions and it is referred to as *repair time* t_r[7]. The system is critically overloaded when $t_r = 15000$, where the repair time is equal to prolonging events' meantime and the execution of commitments fails.

To enable the possibility of delegation rule execution, we introduce vacant resources - the agents with no plans that are joining the system during the execution phase and are able to undertake delegated commitments. The number of *vacant agents* is set to 5 which produces 10% of overall free resources.

Each agent has a plan containing 100 commitments. The duration of the commitment execution t_d (ideally with no prolonging events) is randomly generated with uniform distribution from 5000 to 15000 time units. The start time of the commitment t_s is set to the earliest possible time of the winning resource agent and the end time is set to

$$t_e = t_s + t_d, \tag{20}$$

which makes 100% load of the agents in ideal conditions (with no prolonging events taken into account). The relaxation intervals are set to

$$T_s^{rlx} = \langle 0.7 \times t_s, 1.3 \times t_s \rangle, T_e^{rlx} = \langle 0.7 \times t_e, 1.3 \times t_e \rangle \tag{21}$$

[5]More information about this experiment can be found in Vokrinek, J., Komenda, A., Pechoucek, M.: *Decommitting in Multi-agent Execution in Non-deterministic Environment: Experimental Approach*, submitted to Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS09), to be held May 10-15, 2009.

[6]More information about this experiment can be found in Vokrinek, J., Komenda, A., Pechoucek, M.: *Relaxation of Social Commitments in Multi-agent Dynamic Environment*, accepted for publication on International Conference on Agents and Artificial Intelligence (ICAART09), to be held January 19-21, 2009.

[7]The individual agent load can be computed as $1 + \frac{t_r}{\bar{e} - t_r}$ and is varying from 0 to 1500%.

that makes 30% relaxation intervals. The commitment execution is non-interruptible, so if the decommitment rule applies after the commitment execution is started the resource is blocked for the whole t_d.

When we enable the prolonging events the overall system performance is very stressed. In the experiments we focus on the qualitative results of presented decommitment strategies rather than fine tuning commitment parameters according to current experimental settings. The evaluation and discussion is presented in the next chapter.

In the following we summarized the results of experiments based on the experimental setting described above. First, we will discuss the influence of individual decommitment rules used separately. Next, we will show the influence of the mixed strategies. We measure the number of executed decommitment rules and the number of successfully achieved commitments. Due to the over-stressed system, the utilization of agents is 100%, thus this parameter is not evaluated.

Single Rule

The first experiment provides the results of influence to the commitment execution for single rule usage. The delegation (D), relaxation (R) and full decommitment (Fd) rules have been used separately. For comparison we also measured the empty decommitment set noted as basic. For $t_r = 0$ there are zero rule executions and 100 successful commitments. The individual agent stress experiment results are the following (see Figure 12):

Basic – the number of successful commitments varies from 0 to 2 in the whole range. No decommitment rules are executed.

Full decommitment – the number of rule executions grows with the increasing t_r. The curve converges to the maximum number of commitments for a critically overloaded system. The number of successful commitments decreases with increasing t_r from 18 and converges to 0 for the critically overloaded system.

Delegation – the number of rule executions corresponds to possibility of delegation to the vacant agents. When vacant agents are saturated the delegation uses agents freed by the delegation of longer commitments. The number of successful commitments goes from 19 to 10. The variance between agents starts to be significant when $t_r > 10000$ so the robustness of this rule is decreasing.

Relaxation – the relaxation rule provides the best stability. It is executed for every commitment and provides no violations when the system is overloaded below the relaxation interval limitations. When the relaxation interval fails, the number of executed rules goes to 0 very fast and so does the number of successful commitments.

The delegation rule execution is also affected by the number of vacant agents. The relaxation and full decommitment rules are obviously not affected by the number of vacant agents.

Combined Rules

This set of experiments inspects the influence of decommitment rule combinations and their ordering. The main focus is on the two ordering scenarios – R-D-Fd for

$$relaxation \succ delegation \succ fulldecommitment$$

and D-R-Fd for

$$delegation \succ relaxation \succ fulldecommitment$$

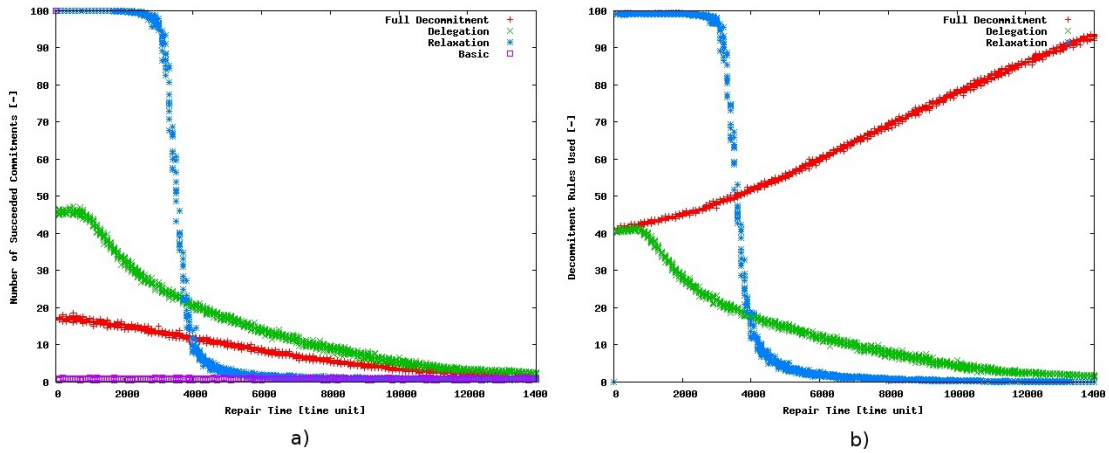

Figure 12: Number of decommitment rules execution (a) and succeed commitments (b) for different t_r in single rule setting.

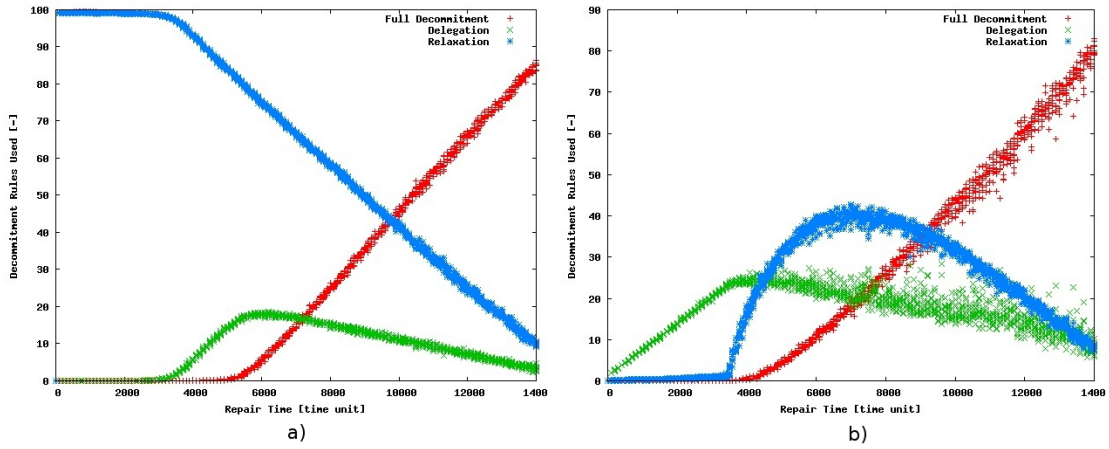

Figure 13: Number of rules execution for varying t_r in combined rule R-D-Fd setting (a) and in combined rule D-R-Fd setting (b).

that provide the most significant results. The full decommitment rule is ordered as the last one, because of its nature – no decommitment rule can be applied after the application of full decommitment.

The combination of rules provides complex results. The number of individual rule executions can be seen in Figure 13. The number of the full decommitment rule executions is similar in both cases but with different impact on the number of successful commitments. In the first part of the chart, when the system is *slightly overloaded* ($t_r < 3000$), only the first rule applies. In the range of an *overloaded* system ($t_r \in \langle 3000, 12000 \rangle$) the second and the third rules start to apply. When the system is close to the *critical overload* ($t_r > 12000$) the number of rules executed starts to provide increasing deviation across the experiments runs and the robustness of the execution is reduced.

The number of successful commitments is compared for R-D-Fd and D-R-Fd with D-Fd, R-Fd, R-D and D-R sets and presented in Figure 14.

The experiments show that for an overloaded system there is an increasing number of dropped commit-

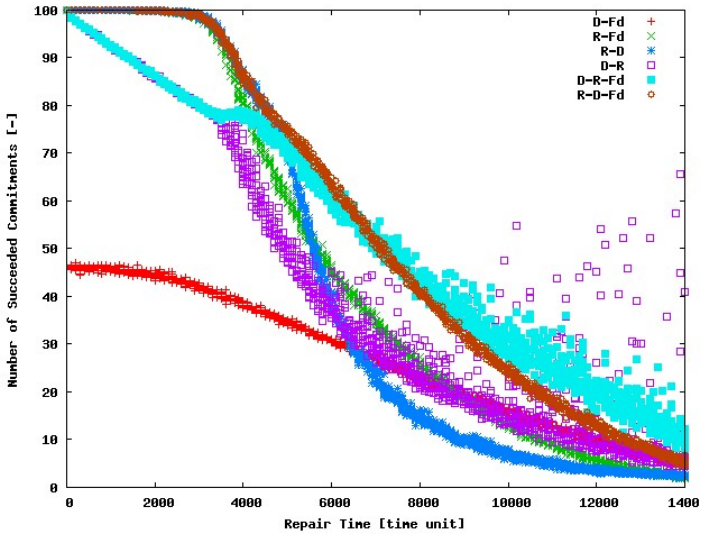

Figure 14: Number of violated commitments for varying t_r for different rules settings.

ments using full decommitment rule. The full decommitment rule applies when all preceding rules fail. This rule effectively release resources originally booked for dropped commitments. This causes a bigger chance for delegation of commitments and space for relaxation. The delegation rule provides the ability of real-time re-allocation of commitments according to current agents performance. The experimental results show the ability of the system to adapt to the overload and thus to increase the number of succeeded commitments with increasing size of the decommitment rule set and to keep high utilization of available resources (execution time of the commitments compared to free time of resources excluding prolonging events).

As shown in Figure 14, the number of successful commitments is reaching 50% for R-D-Fd (D-R-Fd) for $t_r = 7500$ (7000) that corresponds to system load of 200% (187%). At this point, the R-D-Fd (D-R-Fd) method is able to utilize 100% (94%) of the overall system resources available. For $t_r = 7500$, the single rule settings (including basic rule set with no decommitment rules) reach maximum of 15% of succeed commitments for delegation rule (Figure 12b), which is 30% of utilization of available resources. Combined rule sets composed from decommitment rule pairs reach maximum of 30% of succeed commitments, which is 60% of utilization of available resources.

The best performance provides the biggest sets of decommitment rules. The R-D-Fd set has the biggest success rate of commitments execution until $t_r = 7000$. For bigger t_r the higher success rate can be observed for D-R-Fd, but with lower stability (higher variation of experiment runs). The best success rate for near critical load ($t_r > 10000$) can be reached with D-R set, but with minimal stability (most of the experiment runs provide worse results then both R-D-Fd and D-R-Fd).

The sets containing Fd provide generally better results, but may not be suitable in all application domains because of commitment drop-out by this rule.

The combinations of particular rules provide complex decommitment behavior and significantly improve commitment execution performance and stability. The success rate of commitment execution and available resources utilization significantly increases with the size of the decommitment rule set. Different rule combinations have to be chosen for different application scenarios. We have identified, evaluated and discussed the strong and weak points of the presented combinations of decommitment rules.

4.2 Relaxation in Commitments

The experiments evaluate three methods of incorporating uncertainty into the relaxation decommitment rule. The first method is based on statistical error evaluation and uses a constant safety margin without use of the commitment relaxation rule. The other two are based on relaxation with different relaxation intervals estimation.

The experiments were performed on a scenario with one requester agent (RA) and one provider agent (PA). The PA maintains a single resource that is used for task execution. The task execution is interruptible and only one task can be executed at any given time. The RA requests a set of 1000 tasks from the PA. The PA makes allocation for the tasks and proposes a commitment for each individual task. The commitment includes the start time, end time and relaxation intervals. The duration of the task is $t_d = 10$ seconds.

Environment uncertainty is modeled as a resource breakdown with a variable breakdown mean time t_b and reparation time of $t_r = 5$ seconds. An event simulation has been performed for 1000 randomly generated sets of breakdown events for each experiment setting. The commitment parameters were computed as follows:

(M_1): Constant – this method extends the duration of the task by the relative reparation time computed by the probability of the breakdown for each task. The start time of the commitment $t_s(i) = t_e(i-1)$ and the $t_e(i) = t_s(i) + t_d(i) + t_r * p_b(i)$, where $p_b(i)$ is the probability of the breakdown during the commitment's execution.

(M_2): Linear – this method doesn't change the duration of the activity, so the $t_s(i) = t_e(i-1)$ of the previous commitment and the $t_e(i) = t_s(i) + t_d(i)$.

Commitment parameters computations are based on the estimation of breakdown mean time $\bar{e}_{est} = 15$ and known reparation time $t_r = 5$ seconds. The p_b^{worst} estimation is set to $\bar{e}_{est}$. We measure the robustness of the commitments and the resource utilization (total execution time) under various conditions generated by several environment uncertainty models.

The uncertainty models have been generated using three methods. Each method produces event sets with various properties of mean value $\bar{e}$ and standard deviation σ. The environment uncertainty models are the following:

(U_1): Deterministic – breakdown events are generated evenly with t_b period. This method produces constant $\bar{e} = t_b$ and $\sigma = 0$.

(U_2): Gauss – generates a set with normal distribution with $\bar{e} = t_b$, $\sigma = t_b/10$ with a delay between two subsequent events restricted to $\langle 0, 2 * t_b \rangle$.

(U_3): Uniform – generates a set of uniformly distributed events with $\bar{e} = t_b$ with a delay between two subsequent events restricted to $\langle 0, 2 * t_b \rangle$.

The experiments were run with $\bar{e}$ value varied according to Table 1. The random sets were generated uniformly in this interval to evaluate the robustness of the commitment relaxation setting methods. Because of the relatively small event sets, the generated pseudo-random values don't fit exactly to the desired parameters (especially mean time value). The real $\bar{e}$ of each set has been computed within the simulation run and corresponds to the x-axis in the provided evaluation figures.

Table 1: Properties of the breakdown distributions.

distribution	$\bar{e}(s)$	σ (s)
U_1	10 $\sim$ 20	0
U_2	10 $\sim$ 20	1.5
U_3	10 $\sim$ 20	8.66

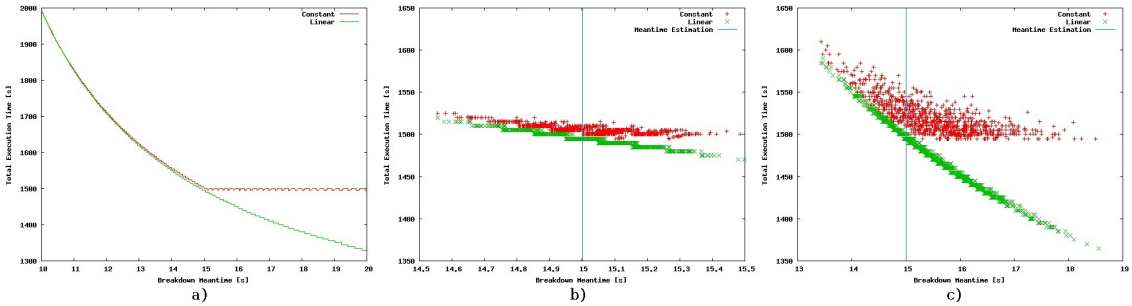

Figure 15: Total execution time with (a) deterministic mean time of breakdowns, (b) normally distributed mean time of breakdowns ($\bar{e} = t_b$, $\sigma = t_b/10$), (c) uniformly distributed mean time of breakdowns.

Figures below show the number of violated commitments, tardiness of the commitments, and total execution time for all methods under various uncertainty settings. The total execution time of all commitments is $\sum t_d(i) = 1500$ seconds and represents the ideal execution duration in a breakdown-free environment. The length of the plan represents the end time of the last commitment for the M_1 method and the latest time of the relaxation intervals for the M_2 method. The plan length is influenced by the experiment settings and in our case it lengthens the plan for both methods by 50% (caused by the parameters $\bar{e}$, t_d and t_r).

During the simulation, the agents kept all the commitments as agreed at the beginning. The experiments show the impact of non-accurate estimation on the error mean time. As expected, both methods provide good results when $\bar{e} > \bar{e}_{est}$. When the breakdown mean time is shorter, both methods start to generate commitment violations.

Figure 16 shows the mean time influence on the number of violations. The commitment is violated when it cannot be finished within the agreed limits (t_e for M_1 and relaxation interval T_e^{rlx} for M_2). The robustness of the method M_1 is very limited. It provides good results only for deterministic uncertainty U_1 with $\bar{e} \geq \bar{e}_{est}$. When $\bar{e} < \bar{e}_{est}$, all the commitments are violated (Figure 16a). The M_2 method provides better results in the left-hand part of the graph ($\bar{e} < \bar{e}_{est}$) because of greater safety margin (caused by p_b^{worst}) but still converges quickly to the 100% violated commitments. For normal uncertainty U_2 the situation changes. The M_1 method fails in the entire range of $\bar{e}$ and there is a low amount of non-violated commitments even in the range $\bar{e} > \bar{e}_{est}$ (see Figure 16b). The M_2 method provides a minimal amount of violated commitments in the region $\bar{e} > \bar{e}_{est}$ and $\bar{e} \sim \bar{e}_{est}$ and the number of violated commitments slowly grows in the range $\bar{e} < \bar{e}_{est}$. In the case of uniform uncertainty U_3 both methods fail (Figure 16c). The lowest number of violated commitments is in the right-hand region and it goes from approximately 50% to more than 80% for $\bar{e} = \bar{e}_{est}$. The method M_2 provides good results for $\bar{e} \geq \bar{e}_{est}$. The number of violations grows with descending $\bar{e}$. For $\bar{e} = \bar{e}_{est}$ the average number of violated commitments is about 50%.

The total execution time is presented in Figure 15. Commitment execution is not started before the agreed time (t_s for M_1 and the relaxation interval T_s^{rlx} for M_2), so the execution time mainly corresponds to the number of violated commitments. In case of the M_1 method, the minimal execution

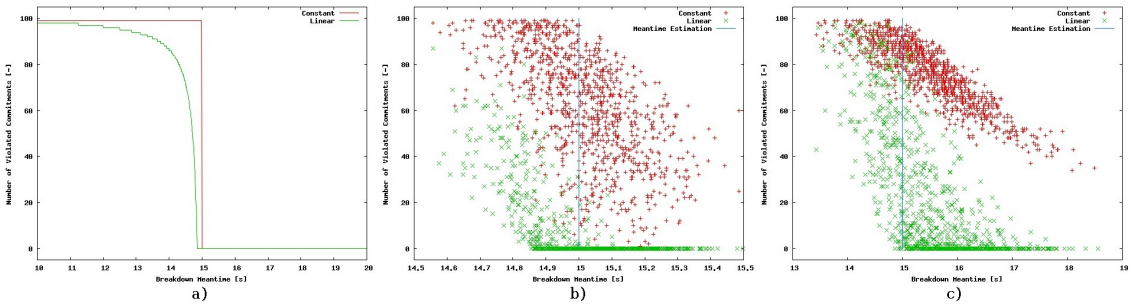

Figure 16: Number of violations with (a) deterministic mean time of breakdowns, (b) normally distributed mean time of breakdowns ($\bar{e} = t_b$, $\sigma = t_b/10$), (c) uniformly distributed mean time of breakdowns.

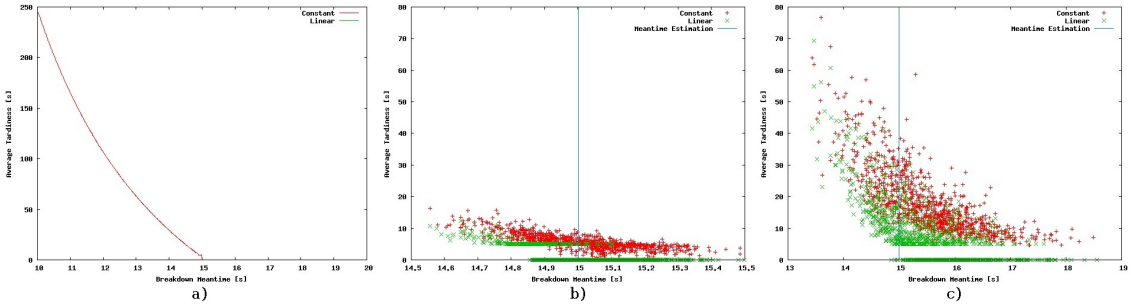

Figure 17: Average tardiness with (a) deterministic mean time of breakdowns, (b) normally distributed mean time of breakdowns ($\bar{e} = t_b$, $\sigma = t_b/10$), (c) uniformly distributed mean time of breakdowns.

time is equal to the plan length (1500). The M_2 method execution time converges to the $\sum t_d(i)$ for the increasing $\bar{e}$.

For deterministic uncertainty U_1 both methods provide hyperbolic growth of the total execution time with decreasing $\bar{e}$ (see Figure 15a). The relatively small difference between the methods is caused by the fast growth of the violated commitments of method M_2 and the low tardiness of the M_1 commitments in the range where M_2 keeps the number of violated commitments low. For normal uncertainty U_2 the total execution time grows almost linearly with decreasing $\bar{e}$. The difference between the methods is given by the difference in the number of violated commitments, which is considerably higher for M_1. For $\bar{e}$ smaller than the depicted value range, the execution time converges to the execution time curve of the U_1 (as the number of violations grows). The same situation occurs for the case of uniform uncertainty U_3. The disruption of the execution time curve of method M_1 is given by high variation of the number of violated commitments. The total execution time of the M_2 is similar to the total execution time for the other two environment settings. The only difference is given by small variation ($\pm 2\%$ or less) of the execution time for the particular $\bar{e}$. This variance depends on the variation of number of violated commitments for this $\bar{e}$ across the simulation runs.

The average tardiness of the commitment completion presented in Figure 17 is computed for all violated commitments. The non-violated commitments are not taken into account. If there is no violated commitment, the average tardiness is set to zero. For deterministic uncertainty U_1 the results of both methods are very similar. The tardiness grows with decreasing $\bar{e}$ in a similar way as the total execution time. For normal uncertainty U_2 both methods provide low tardiness that again converges to the curve for the U_1 environment setting for small values of $\bar{e}$ (the convergence is not captured in Figure 17b). Similarly to the total execution length, the disruption of the curves is given by the variation of the number of violated commitments. For uniform uncertainty U_3 the average tardiness grows faster with

the decreasing $\bar{e}$ (see Figure 17c). The method M_1 provides relatively high average tardiness of the commitments even in the region $\bar{e} > \bar{e}_{est}$ and $\bar{e} \sim \bar{e}_{est}$. The method M_2 provides better results and the tardiness grows mainly in the range $\bar{e} < \bar{e}_{est}$.

The basic method M_1 is suitable mainly for a deterministic environment U_1 where the relaxation decommitment strategy method M_2 brings no significant improvement. Extending the safety margins in both methods can scale the results towards lower $\bar{e}$ but lengthens the plans (and also the total execution time for M_1). For environments U_2 and U_3, increasing the safety margin brings no significant advantage because of higher distortion of the breakdown distribution.

From the point of view of the number of violated commitments, which is extremely important in the multi-actors scenarios, the method M_1 fails for U_2 and provides even worse results for U_3. In this case, the relaxation decommitment method M_2 is beneficial for U_2 and keeps certain advantages even in U_3, where the average number of violated commitments is about 50%.

Another advantage of the M_2 method is its robustness. We have experimentally proved that the total execution duration and commitment tardiness does not depend very much on the breakdown distribution function (the results of experiments don't differ by more than 2%). With the increasing $\bar{e}$ the total execution time converges to $\sum t_d(i)$, which is the minimal possible execution time. Due to the start time and end time relaxation ability, the method enables both optimistic and pessimistic execution without breaking the commitments. The relaxation decommitment strategy greatly increases the flexibility, stability and robustness of the agents' social commitments in the dynamic uncertain environment.

4.3 Execution Length

This section shows the behavior of the system in the complex interaction scenarios. We focus to the planning performance and resources utilization in various settings. The measured parameter is length of plans (measured in seconds of simulation) and its deviation across all actors in the system. The test has been performed on the following settings:

- **Scalability test** – this test has been performed on *build-houses* scenario (see Section 2 – Basic builders demo). There are 20 builders in the scenario, each with one *build houses* task. The number of trucks varies from 1 to 20.
- **Over-booking test** – this test has been performed on *resources* scenario (see Section 2 – Resources demo). There are 10 builders in the scenario, each with one 1 *build houses* tasks. The decomposition of *build houses* generates two requests: *request transport* and *request resources*. Those two requests are allocated to the trucks sequentially. The number of trucks varies from 1 to 20.
- **Cross-booking test** – this test has been performed on *multi-task* scenario (see Section 2 – Multi-task demo). There are 10 builders in the scenario, each with sequence of 5 *build houses* tasks. The number of trucks varies from 1 to 10.

Figure 18 shows the total execution time (a.k.a. length of plan) in the described settings. All settings provide similar results. The execution time goes down fast with increasing number of trucks. Due to overheads needed for truck sharing (waiting times, empty kilometers, etc.), the improvements slows down for approx. 7 trucks in the scalability test, 5 trucks in the over-booking test and 4 trucks in the cross-booking test. The general tendency of the execution time provides logarithmic dependency on the

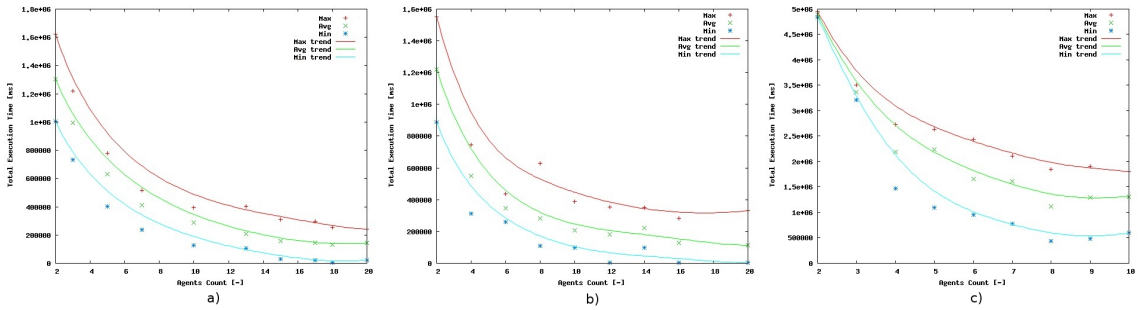

Figure 18: Length of the plans for different agents counts and scenarios: (a) scalability test (build-houses scenario), 20 builders, 1 task, n trucks; (b) over-booking test (resources), 10 builders, 1 task refined to 2 requests, n trucks; (c) cross-booking test (multi-task scenario)), 10 builders, each with sequence of 5 tasks, n trucks.

number of trucks in all three scenarios. The saturation of trucks is the lowest (the highest scalability) in the scalability test because of low dependency of the tasks. In the overbooking scenario the system is saturated for approx. 10 trucks and then doesn't provide execution time reduction (average total execution time converges to 200s). The cross-booking scenario provides the worst scalability. The saturation can be observed for approx. 6 trucks and the average total execution time converge to 1500s.

4.4 Negotiation Complexity

This section shows the behavior of the system in the complex interaction scenarios. We focus to the negotiation complexity of the distributed task allocation (as a part of the commitment planning – see Section 3.3.4). We investigate the number of open contract-net-protocols, successfully closed protocols and failed protocols (corresponds to backtracking in planning process) and it's tendency over the time. The number of failed CNPs is the aggregated number of failures on the protocol initiator and responder's sides. The CNP is failed when one party drops the agreed commitment, the initiator is no longer interested in the commitment (because of another conflicts), or the responder is not able to fulfill the commitment (because of overbooking). The values are measured in time intervals $u = 500ms$ of processor time and provide the overview of the complexity of task allocation during the planning phase.

The first set of experiments have been performed on the same scenarios as in Section 4.3. The number of trucks has been fixed to 17 for the scalability test, 14 for the over-booking test, and 10 for the cross-booking test. The results are presented in Figure 19.

In the scalability test, there are 20 open CNPs at the beginning (corresponds to 20 builders requesting single transport). The number of successful CNPs grows linearly. In the time $u = 40$ the conflicts are detected and number of canceled CNPs starts to grow. The agents replan and thus number of open CNPs also starts to grow. The number of open and canceled CNPs grows almost linearly until the number of successful CNPs converges to the number of open CNSs in the time $u = 100$. At this moment, the planning phase is finished and all transport tasks are allocated to the trucks. The allocation of 20 transport tasks has been accomplished using 49 contract-net-protocols.

In the over-booking test, there are 10 open CNPs at the beginning (corresponds to 10 builders requesting transport to the construction site). Sequentially, the builders start to request transport of construction

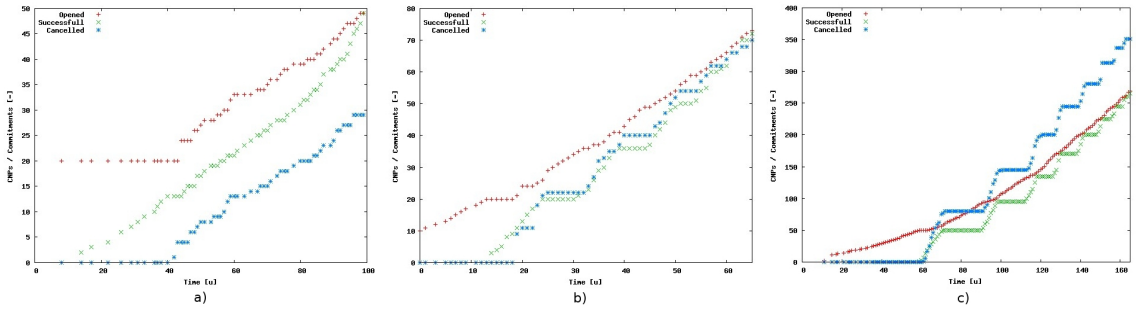

Figure 19: Course of one execution run showing the dynamic values of the CNPs for different scenarios: (a) scalability test (build-houses scenario), 20 builders, 1 task, n trucks; (b) over-booking test (resources), 10 builders, 1 task refined to 2 requests, n trucks; (c) cross-booking test (multi-task scenario)), 10 builders, each with sequence of 5 tasks, n trucks.

materials (resources). The tendency of the CNPs is similar to the scalability test, but the number of canceled CNPs is higher because of trucks overbooking. The conflicts are started to be detected in the time $u = 12$. The planning procedure converges in time $u = 65$. The allocation of 20 transport tasks (transporting builders and materials) has been accomplished using 74 contract-net-protocols.

The cross-booking test provides the highest number of CNPs. There are 10 open CNPs at the beginning (corresponds to first transport of each builder). Sequentially, the builders start to request transport to a second construction site and following transports. The tendency of the CNPs is also similar to the scalability test. The curve of the open CNPs clearly shows the polynomial complexity of the negotiation. because of high cross-booking, the number of failed CNPs is higher then number of open CNPs (the commitment is dropped by both party simultaneously). The conflicts are started to be detected in the time $u = 60$ (the number of open CNPs is 50 – all the transportation are allocated and builders start to close the CNPs. At this moment, number of successful CNPs grows, but also the number of failed CNPs. The horizontal steps of the curves correspond to the planning activity and opening new CNPs for previously failed allocations. The planning procedure converges in time $u = 165$. The allocation of 50 transport tasks has been accomplished using 275 contract-net-protocols. For comparison, the setting of this scenario with 4 builders and 4 trucks (20 transport tasks total) the total number of contract-net-protocols is 50.

The second set of experiments shows the behavior of multi-level task allocation. We have 1 commander, 4 builders and n trucks in the scenario. The commander requests a sequence of m *build-houses* tasks from builders. The builders request transport to the construction site concurrently. Figure 20 shows the dependency of number of CNPs on number of agents (trucks varying from 1 to 15) and on number of tasks (varying from 2 to 20). For a fixed number of tasks, the number of needed CNPs grows logarithmically with increasing number of truck agents. With increasing number of simultaneous tasks in the system there is exponential grow of the needed CNPs.

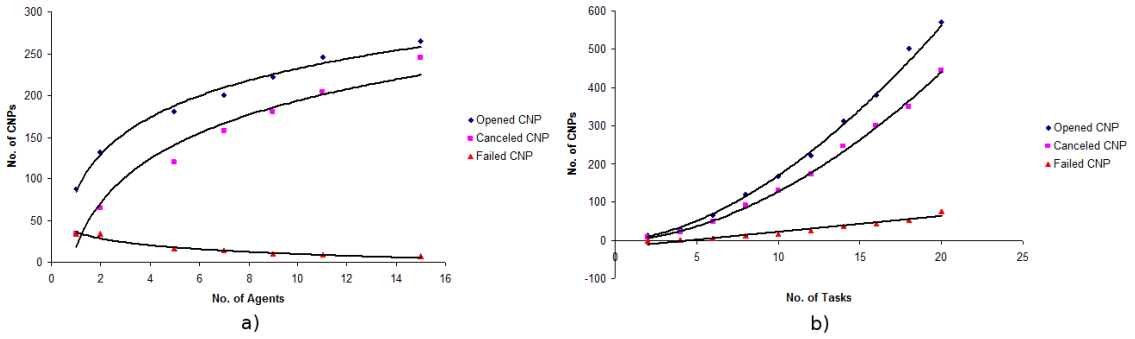

Figure 20: The number of CNPs in multi-level scenario for (a) n truck agents (1 commander and 4 builders) and (b) m tasks.

5 Technical Issues Encountered

We realized the importance of planning with quantitative time for fully automated planning (e.g. within UAVs) but also in mixed-initiative planning (e.g. with the commander).

We had to extend the multi-agent simulation system A-Globe with time-scalability of the simulation: now the simulation can be arbitrary set to work in any value of the interval: turn-based simulation – real-time simulation. The problem is how the load of the agent can be measured. We chose measurement based on counting sent and received messages of each agent. The measuring algorithm is in one of the simulation agents and controls the whole flow of simulation time.

ACROSS2 had to be extended with new action processing, handling real durations of the actions, and the ability to stop currently processed actions at any time. The linkage system (linkage of the entities, e.g. if the truck transports the builder) had to be rewritten to be able to work in the turn-based simulation.

The integration of the AGENTFLY system required a complete new ad-hoc protocol designed to fulfill the needs of both projects. We had to implement a new control module of the AGENTFLY pilot agent to be able to receive control commands from I-Globe. On the I-Globe side, we had to implement a new personal layer, which can handle specifics of controlling the airplanes (plan must always end on an airfield, there must not be any stops, etc).

To integrate with the I-X architecture, we implemented bidirectional interfaces between the two architectures. For the I-X planner, I-Plan, this took the form of a wrapper that, from the I-Globe perspective, allowed arbitrarily many instances of the planner to be used and, from the I-X perspective, allowed I-Globe agents to be treated as I-X agents. A similar approach was taken with I-X process panels. In addition, we had to take steps to allow multiple instances of the process panel GUI to exist in the same Java virtual machine.

We had to implement the extended versions of the CNP protocol (which classical version was already in A-Globe platform). The distributed resource allocation in the I-Globe system is based on that enhanced CNP. The implementation had to handle the three dimensions of the resource allocation problem (depending tasks allocation, hierarchical task allocation, and concurrent tasks allocation) and had to be effective enough for the simulation.

6 Possible Future Improvements

There are several areas for possible future improvement. The summary follows:

- **Extending the distributed multi-agent planning algorithm** - The current approach to the collaborative multi-peer planning problem does not take into account soft constraints. Further research could explore this option. Another assumption is that all agents have the same knowledge. Dealing with agents that only have partial knowledge or even inconsistent knowledge would present interesting research challenges. This could lead to new tools for high-level, mixed initiative planning.
- **Temporal planning at the strategic level**: The current planning domain for the strategic layer does not make use of temporal constraints. Although temporal planning is possible in I-Plan, it was not clear how some of the problems could be encoded. Further research could analyse this problem and result in either a refined methodology for encoding temporal information, or a more expressive formalism with a new constraint manager might be required.
- **Distributed resource allocation** - We want to focus on improving the allocation mechanism's effectiveness, mainly in over-booking and cross-booking scenarios. In the prototype, we have implemented a method with guaranteed convergence and good utilization of resources. The future improvement can be towards the proof of optimality, reduction of complexity (back-tracking), etc. A promising approach seems to be the concept of semi-agree/semi-fail, which can be described as an extension of the implemented CNPs utilizing only partial agreements among the agents.
- **Decommitment rules enhancement** - Another research topic (and thus a system improvement) can be introduction of extended sets of decommitment rules and design of enhanced control mechanisms for rule generating and execution. The improved rules should provide more flexibility of the replanning process and tailored plan representation.
- **Critical state information propagation** - In the case of long-term plans with complex inter-effects of plan actions, finding the distributed plan collisions is a key issue. The solution of the problem would provide us with sooner detection of such a conflicting event and its avoidance. The mechanism should be based on the identification of the critical effects and their propagation through the agent community. The distributed planning process should be enriched by a process of informing the actor community about planned action effects, and forming interest groups according to the actor domains.
- **Planning to a limited time horizon** - Currently agents try to plan as far into the future as possible. That is, for an incoming task the refinements in the domain model are used to expand the current plan until all sub-tasks are accomplished. If there is uncertainty in the environment, planning far into the future is often not very useful. The idea here would be to research new planning algorithms the plan only to a limited time horizon, extending plans as time moves on. This might be of particular relevance to fully automated planning as is required for UAVs, for example.
- **Synchronization-points** - The synchronization of the distributed plans currently uses simple and compound synchro-points. This plan element should be described using one of known formalisms (aka PETRI-NETs). There is wide potential for improvements using model checking methods, deadlock search algorithms, etc.
- **Individual layer algorithms** - There is a wide area for improvements on the individual layer. The most promising algorithms are: spatio-temporal search algorithms of rendezvous points, unit covering, convoys, formations planning, etc.