# Speculative Planning With Delegation

Tolga Abacı
tolga.abaci@epfl.ch

Ján Cíger
jan.ciger@epfl.ch

Daniel Thalmann
daniel.thalmann@epfl.ch

VRlab
Swiss Federal Institute of Technology (EPFL)
IN-J Ecublens, 1015 Lausanne, Switzerland

## Abstract

*This paper presents a novel method of collaborative problem solving for intelligent agents in virtual environments. We describe a planner for STRIPS-like domains enhanced by techniques of "delegated computing". Instead of having one central planner with a complete information about the world, the concept of delegation is used. In this way, we achieve agent collaboration in a dynamic system, where each of the agents has its own partial plan, but the plans are implicitly coordinated towards a common goal. We demonstrate these techniques in a virtual reality simulation with virtual humans solving a logistical problem.*

## 1. Introduction

Many researchers investigated the fields of both multi-agent systems (mostly artificial intelligence) and collaborative VR (mostly multi-user systems). An interesting problem in these domains is the use of collaborative agents for VR applications and having them work together with the human operator to solve a problem. Our intention in this paper is to address the important issues of agent communication, collaboration and planning.

We present a new method of intelligent agent collaboration in a virtual environment. Our approach is based on combination of several techniques:

- STRIPS–like problem representation
- Graphplan planner (solver)
- Task delegation framework

Instead of using a centralized planning system or shared plans, we implement agent collaboration using "delegated computing." This allows us to achieve meaningful agent collaboration in a dynamic system, where none of the agents has full knowledge of the environment (e.g. does not know about the other agents or the complete state of the environment).

Potential applications include training (emergency response, medical training, air traffic control, even military), simulations and games.

## 2. Collaborative agents

There are several common approaches for agent control in the systems with collaborating agents:

- Scripting. Scripting allows very detailed level of control, but is very inflexible.
- Planning. Usually centralized, tends to be inflexible in handling unexpected situations. To mitigate this to some extent, hierarchical planning is often used (e.g. [3]).
- Reactive agents. Agents are not scripted, but react to the changing environment according to the sets of rules (e.g. described by Badler in [2]). There are few other popular techniques, e.g. BDI logic introduced by Bratman [6], cognitive modeling [10] or just simple finite state machines.

With the use of these three approaches, several difficulties arise – scripting is problematic in the case of contingencies (e.g. unexpected obstacles), centralized planning tends to be complex, because it has to produce detailed plans for every agent in the system and does not cope well with unexpected events. Reactive techniques perform usually well for single agent, but the lack of a global state awareness hinders meaningful coordination – the agents try satisfy their own desires, but those do not have to be compatible with the higher level goal.

Traditionally, specialized collaboration algorithms have been quite complex. For example, shared plans [11] or project COLLAGEN [20] require the agents to know about each other and communicate among themselves in the process of solving the problem. This is not always practical, especially when the environment itself is dynamic and the agents may appear and disappear arbitrarily at run-time – often the case in VR applications.

However, a simpler approach is possible – a task, not communication-oriented system. It is usually not important who performs the task but that the task is done. This idea appeared in blackboard systems, where the agents communicate among themselves by posting data to a shared data store – the blackboard. Classical example of a blackboard system is HEARSAY-II, described in [8].

More recent work uses the concept of "delegation" – asking another agent to perform a service or sub–task. This idea appeared in [7] and more recently in [17]. The technique is described as "delegated computing."

Delegated computing achieves agent coordination using a *facilitator*, which keeps the information about the global state of the system and brokers the information exchanges among the agents. The coordination process itself is transparent to the agents. This greatly simplifies the implementation of such system, because each agent has to be aware only about the facilitator(s), it does not have to communicate with other agents directly unless it wants to.

The basic heuristics often leading to the successful solution of the problem is to delegate everything that the agent is unable to solve itself to the facilitator. The facilitator arranges the completion of the sub-goals by delegating the task further to the agents capable of completing it if it is possible in the given situation (e.g. there has to be agent or set of agents capable of solving that particular sub-problem).

The collaboration does not have to be limited only to the autonomous agents. Human operator could also delegate the tasks to the facilitator, which will handle the work coordination. This scheme alone allows solution of simple problems, but more sophisticated solution is necessary for more complex objectives.

This paper proposes a new method of action planning. The combination of planning with delegated computing enables creation of a *speculative* planner. During the search for the plan satisfying the goals, it is possible to "cheat" by assuming that some action is possible (even though the agent does not know how to perform it itself) and continue further. While executing such plan, these actions will be replaced by delegation to the facilitator. It means, that the plan may be potentially not valid (not satisfying the goal, because no agent is capable of successfully solving the delegated sub-goal) in runtime, but in most cases the delegation will lead to a solution even in cases which are unsolvable by the agent alone.

```
(define (operator move)
    :parameters
        ((Agent ?who)
         (Place ?from)
         (Place ?to))

    :preconditions
        (:and (:neq ?from ?to)
              (at ?who ?from))
    :effect
        (:and (at ?who ?to)
              (:not (at ?who ?from))))
```

**Figure 1. The "move" operator**

## 3. Planning

Planning has two main interpretations/meanings (according to [15]):

- In artificial intelligence, planning is usually understood as a search for a sequence of logical operators/actions that transform an initial world state into a desired goal state.
- In robotics, planning is concerned mainly with motion, including problems such as the "piano–mover's problem."

Planning in artificial intelligence has a long-standing tradition. One of the first works was the STRIPS planner in 1971 [9]. STRIPS introduced the concept of operators, which have preconditions and effects. The state of the system is expressed by predicates. Many planners use this representation of the problem – such as UCPOP [18], Graphplan [5], PRODIGY [22], etc. An example of a STRIPS operator for moving an agent from one place to another in UCPOP notation is given in figure 1.

The plan can be typically produced in two forms. A total-order plan specifies exact sequence of actions leading from the initial state to the goal (steps have to be ordered). Partial-order plan can contain steps where the ordering between actions is not specified (e.g. Graphplan is a partial-order planner).

There are many extensions to the basic planners, e.g. Sensory Graphplan (adds sensing and uncertainty – [1, 24]), Probabilistic Graphplan (modification for the probabilistic planning – [4]), planners using durative actions (taking the estimated duration of an action into account) and many others.

On the other hand, robotics deals mostly with motion planning techniques, aiming to synthesize collision-free motions. Attempts to achieve practical planning resulted in

schemes that utilize the probabilistic roadmaps. Visibility-based roadmaps (see [21]) employ visibility criteria to reduce the number of the nodes in the roadmap. On the other hand, rapidly–exploring random trees (RRTs – [14]) generally have much higher number of nodes, but they explore the configuration space more uniformly. These techniques have also been applied to animation of virtual characters. For example, [13] describes a motion planning system for generating grasping and reaching motions.

## 4. Planning with delegation

We introduce the concept of *delegation* into a planner using STRIPS-like problem representation. The Graphplan planner described in [5] was adapted to use delegated actions.

Graphplan uses the notion of the planning graph which compactly encodes the planning problem. Described in a very simplified way, the planning graph is constructed layer by layer. Each layer consists of preconditions (predicates) and actions, for which the corresponding preconditions exists in it. The search is attempted whenever the current layer contains all goal predicates and they are not mutually exclusive. It is done in a backward-chaining way, going from the goals back to the actions, using their preconditions as subgoals for a recursive step.

The planner has four important properties:

- Graphplan is sound, if the plan is found, it is correct
- Graphplan is complete, if the plan exists, it will find it
- Graphplan always stops, even if the solution does not exist
- The generated plan is the shortest partial-order plan possible.

```
(define (operator open_door)
    :parameters
        ((Agent ?who)
         (Door ?door))

    :preconditions
        (:and (not_encumbered ?who)
              (at ?who ?door)
              (closed ?door))
    :effect
        (:and (open ?door)
              (:not (closed ?door))))
```

**Figure 2. Standard "open-door" operator**

We introduced delegation into Graphplan by means of *delegation operators*. From the planner's point of view, a delegation operator is the same as a standard operator (has parameters, preconditions and effects), except that during the creation of the planning graph the instantiation of delegation operators is tried only after all other options (no-ops and standard operators) are exhausted. This means that an agent delegates something only if it cannot do it itself. In this way, we are able to avoid generation of meaningless plans where every agent delegates all actions, even those that they are perfectly capable of performing themselves. Moreover, this ordering of operators ensures that if a plan using delegated operations is not valid (does not lead to a successful completion of the goal) then there is no valid plan without delegation either.

```
(define (operator delegate_open_door)
    :parameters
        ((Door ?door))

    :preconditions
        ((closed ?door))

    :effect
        (:and (open ?door)
              (:not (closed ?door))))
```

**Figure 3. Delegated "open-door" operator**

Figure 2 shows a standard "open_door" operator, figure 3 shows a delegated variant of the same. The difference for the planner is that the delegated operator is easier to apply (has usually less preconditions) and never specifies which agent has to perform it, because this information is determined only at runtime by the facilitator.

Plans containing delegated actions are executed by agents in a similar way as normal (non-delegated) plans. For each step, the preconditions are checked, action is performed and the state of the simulation is updated according to the effects of the action. The important difference between a delegated and non-delegated action lies in the way the action's effects are introduced into the simulation state.

For standard actions, the agent performing the operation is also responsible for updating the state of the simulation. However, for the delegated action, the facilitator arranges its execution and the update of the simulation state is the responsibility of the agent(s) finally performing it (to which it was delegated from the facilitator). The reason for this difference is very simple – if the delegating agent is waiting for the effects of the delegated action in order to be able to proceed with the next step of the plan (which has some of the effects as preconditions), it will need to synchronize with the agent really performing the action.

```
Initial conditions:
     (at, martin, anywhere)
     (closed, door)

Goal conditions:
     (behind, martin, door)
     (closed, door)

Resulting plan:
  1. (move, martin, anywhere, door)
     (delegate_open_door, door)
  2. (approach, martin, door)
  3. (walkthrough, martin, door)

  4. (delegate_close_door, door)
```

**Figure 4. Plan for traversing the door using delegation**

Figure 4 shows an example of a partial-order plan using delegation to solve a problem of an encumbered agent (carrying a large crate) having to negotiate a closed door. The actions and predicates are expressed using tuples, where the first element is the functor describing the action and the remaining elements are arguments. The plan was created using the operators in table 1. Columns "add" and "del" effects denote the predicates, which have to be added or removed from the simulation state, when the action is executed. In the UCPOP notation, the "del" effects are preceded by the `:not` keyword.

Compared to the standard plans generated by Graphplan, plans using delegation may not always lead to successful completion of the goal. The planning is *speculative*, the delegation operators assume, that it is possible to execute the corresponding actions in run-time. That may or may not be true, depending on the situation (e.g. if there are no agents capable of performing the action).

In our context, an agent delegating an action means that it asks the facilitator to take care of how and by whom this action should be executed. The facilitator maintains the capabilities of the agents and uses different strategies to solve this problem. This organization frees the delegating agent from keeping track of this information, greatly simplifying the process of collaboration.

The facilitator uses a process known as *unification* [16] to match the declared capabilities of agents with the task being delegated. By default, all solutions are used, that means, that all agents capable of solving the task are asked to do so and they execute the tasks in parallel. This is not always desired behavior (e.g. three agents rushing to open the door), so it is possible to restrict the amount of desired results (e.g. to one). In such case it is up to the facilitator to decide which agent will receive the delegated task.

Unification also allows easy tracking of the state of the simulation, because predicates expressing the current state can be stored by the facilitator and queried in a simple way. For example, agent Gino reports that the door is open by declaring a predicate `(open, door)`. Agent Martin can query whether the door is open either directly by delegating the expression `(open, door)` to the facilitator (returns empty substitution if the unification succeeds) or he can ask for the status of the door by delegating `(?status, door)`, which returns in our case substitution {`open/?status`}. This technique is similar to blackboards, where agents are posting information into a shared data repository. Agents executing their plans use it to ensure that preconditions of the actions are satisfied or to communicate changes in their environment.

## 5. Results

We would like to illustrate the effectiveness of our approach by a simple example. We have constructed a virtual environment, with two agents (virtual humans Martin and Gino), a heavy crate and two rooms separated by a sliding door, activated by a nearby button. The environment is 3D-rendered in real-time, and a grid representation is used for fast path finding and collision detection.

The architecture of our implementation is outlined in the figure 5. The agents framework and the planner were implemented using Python. Communication among the agents (including the facilitator) is implemented using CORBA (OmniORB). For the animation and visualization of our VR environment we have used the VHD++ framework [19].

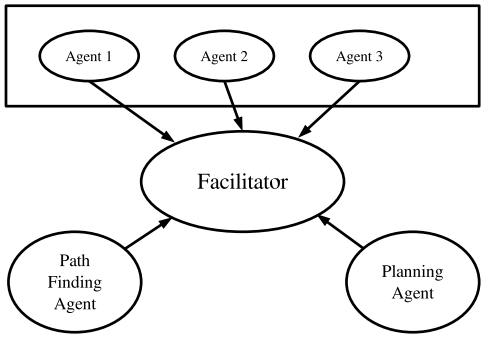


**Figure 5. Architecture using the facilitator**

In the scenario we have examined, one of the agents needs to carry the heavy object from one of the rooms to the next one. For this particular problem, our planning framework has come up with the sequence of actions depicted in figure 4, the plan was executed as shown in figure 6.

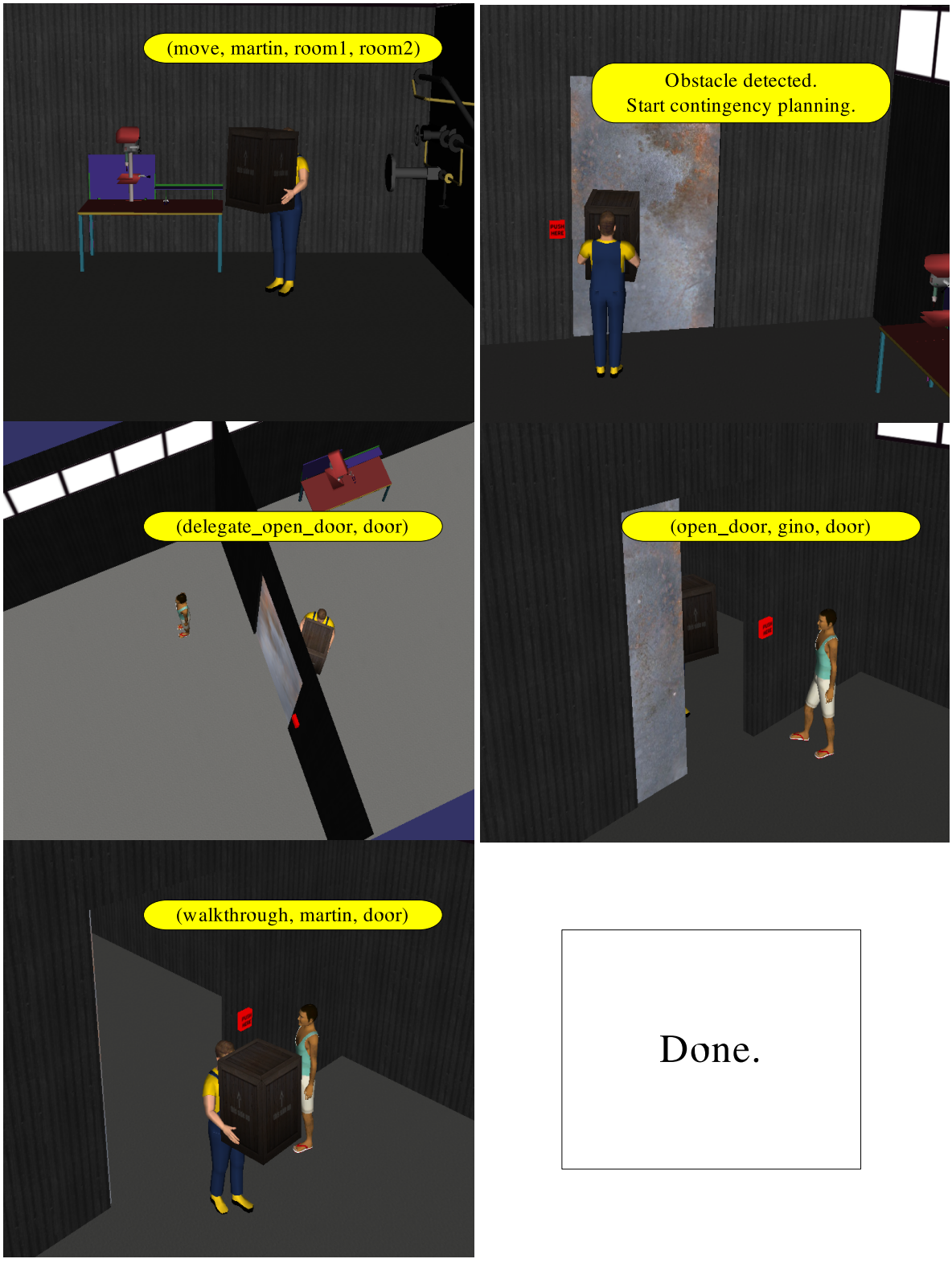


**Figure 6. Execution of the plan**

| Operator | Arguments | Preconditions | "Add" effects | "Del" effects |
|---|---|---|---|---|
| move | who, from, to | (at, who, from), (:neq, from, to) | (at, who, to) | (at, who, from) |
| walkthrough | who, door | (in_front, who, door), (open, door) | (behind, who, door) | (in_front, who, door) |
| approach | who, what | (at, who, what) | (in_front, who, what) | (at, who, what) |
| leave | who, what | (in_front, who, what) | (at, who, what) | (in_front, who, what) |
| open_door | who, door | (not_encumbered, who), (at, who, door), (closed, door) | (open, door) | (closed, door) |
| close_door | who, door | (not_encumbered, who), (at, who, door), (open, door) | (closed, door) | (open, door) |
| delegate_open_door | door | (closed, door) | (open, door) | (closed, door) |
| delegate_close_door | door | (open, door) | (closed, door) | (open, door) |

**Table 1. Operators for the door problem**

Upon receiving the request to move from the first room to the next one, Martin begins walking, carrying the crate. When he reaches the door, he senses that he has met an obstacle and the contingency planning is started. He delegates a planning job to the facilitator, which in turn delegates it to the available planning agent. It creates the plan, if one exists, and returns it.

For the plan described above, the door must be open for Martin to be able to pass through it and enter the other room. This can only be accomplished by pressing the button. Since Martin is already encumbered by the object, he cannot press the button, and he has to ask for assistance. He delegates the action and the facilitator arranges for Gino to execute it. Once Gino opens the door, the obstacle is removed, so Martin is able to fulfill the original request. The result is effectively the two agents collaborating to achieve a goal.

## 6. Conclusions and Future Work

We presented a novel method of collaborative problem solving with intelligent agents. The combination of the classical STRIPS planner with delegated computing has enabled us to create "smart" virtual humans that are capable of helping each other to solve problems. A simple scenario was used to illustrate the usefulness of the method.

The generated plans are usually simpler compared to a single plan including all agents and are also much faster to create. This is important in a real-time VR environment.

Additionally, the indirect communication of the agents via the facilitator makes the plans more flexible. They do not depend on an particular agent to be available for collaboration, thus reducing the need for run-time re-planning.

However, there are few disadvantages, which have to be taken into account. The facilitator could become a bottleneck in setups with large number of agents. This could be addressed by multi-facilitator systems, with the associated high complexity.

Another drawback is the fact that not every planning problem is easily expressed as a STRIPS-like domain (for example cases where new objects are created). However, this is a more general problem, which is not specific to our approach.

In the future, we plan to enhance the system with automatic generation of the problem descriptions for the planner. Currently, the problems are "hard-wired" into the agents – the agent has an a priori knowledge e.g. about how to open the door. There are several possible approaches for this, one is automated learning by observation [23], or by getting the necessary data from the environment – e.g. smart objects [12].

Another possible enhancement is to include aspect of time into the planner. Planners using *durative* actions are known already, however in case of delegated actions there are non-trivial challenges to be addressed.

## References


[1] C. R. Anderson, D. E. Smith, and D. S. Weld. Conditional effects in Graphplan. In *Proceedings of AIPS '98*, 1998.

[2] N. Badler. LiveActor: A virtual training environment with reactive embodied agents. In *Workshop on Intelligent Human Augmentation and Virtual Environments*, University of North Carolina at Chapel Hill, October 2002.

[3] J. Baxter and R. Hepplewhite. A hierarchical distributed planning framework for simulated battlefield entities. In *PLANSIG 2000*, 2000.

[4] A. Blum and J. Langford. Probabilistic planning in the graphplan framework. In *ECP*, pages 319–332, 1999.

[5] A. L. Blum and M. L. Furst. Fast plannig through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[6] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[7] P. R. Cohen, A. J. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In *AAAI Spring Symposium*, pages 1–8, Mar 1994. OAA.

[8] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty. In *ACM Computing Surveys*, volume 12 (2), pages 213–253. ACM Press, 1980.

[9] R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[10] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH 99*, Los Angeles, CA, August 11-13 1999.

[11] B. J. Grosz and S. Kraus. The evolution of shared plans. In A. Rao and M. Wooldridge, editors, *Foundations of Rational Agency*, pages 227–262. Kluwer, Dordrecht, 1999.

[12] M. Kallmann. *Object Interaction in Real-Time Virtual Environments*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2001.

[13] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. In *Proceedings of Eurographics 2003*, pages 313–322, Granada, Spain, 2003.

[14] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Dept. of Computer Science, Iowa State University, October 1998.

[15] S. M. LaValle. *Planning Algorithms*. [Online], 1999-2004. Available at http://msl.cs.uiuc.edu/planning/.

[16] G. F. Luger. *Artificial Intelligence*. Addison Wesley, 4. edition, 2002.

[17] D. L. Martin, A. J. Cheyer, and D. B. Moran. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[18] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial-order planner for ADL. In *Third International Conference on Knowledge Representation and Reasoning (KR-92)*, Cambridge, MA, October 1992.

[19] M. Ponder, T. Molet, G. Papagiannakis, N. Magnenat-Thalmann, and D. Thalmann. VHD++ development framework: Towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies. In *Computer Graphics International 2003*, pages 96–104, 2003.

[20] C. Rich and C. L. Sidner. COLLAGEN: When agents collaborate with people. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 284–291, New York, 5–8 1997. ACM Press.

[21] T. Simon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(2), 2000.

[22] M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.

[23] X. Wang. Learning planning operators by observation and practice. In *Artificial Intelligence Planning Systems*, pages 335–340, 1994.

[24] D. S. Weld, C. R. Anderson, and D. E. Smith. Extending Graphplan to handle uncertainty & sensing actions. In *Proceedings of AAAI '98*, 1998.