Planning Agents in James

Article *in* Proceedings of the IEEE · February 2001

DOI: 10.1109/5.910852 · Source: CiteSeer

CITATIONS
50

READS
131

2 authors:

Bernd Schattenberg
3B Intelligent Solutions
33 PUBLICATIONS **413** CITATIONS
SEE PROFILE

Adelinde M. Uhrmacher
University of Rostock
275 PUBLICATIONS **3,286** CITATIONS
SEE PROFILE

Planning Agents in James

Bernd Schattenberg
Faculty Computer Science
University of Ulm
D-89081 Ulm
schatten@informatik.uni-ulm.de

Adelinde M. Uhrmacher
Computer Science Department
University of Rostock
D-18059 Rostock
lin@informatik.uni-rostock.de

Abstract— **Testing is an obligatory step in developing multi-agent systems. For testing multi-agent systems in virtual, dynamic environments, simulation systems are required that support a modular, declarative construction of experimental frames, that facilitate the embeddence of a variety of agent architectures, and that allow an efficient parallel, distributed execution. We introduce the system James (A Java-Based Agent Modeling Environment for Simulation). In James agents and their dynamic environment are modeled as reflective, time triggered state automata. Its possibilities to compose experimental frames based on predefined components, to express temporal interdependencies, to capture the phenomenon of pro-activeness and reflectivity of agents are illuminated by experiments with planning agents. The underlying planning system is a general purpose system, about which no empirical results exist besides traditional static benchmark tests. We analyze the interplay between heuristics for selecting goals, viewing range, commitment strategies, explorativeness, and trust in the persistence of the world and uncover properties of the agent, the planning engine and the chosen test scenario: Tileworld.**

I. Introduction

Testing is an obligatory step of each software engineering process and becomes even more important if the development of a software system must be considered as experimental itself. "At the time of writing, the development of any agent system - however trivial - is essentially a process of experimentation. There are no tried and trusted techniques available to assist the developer" [61]. The development of agents faces problems associated with traditional distributed concurrent systems. Additional difficulties arise from complex interactions between autonomous problem solving components [29, p.298]. The construction of agents reincarnates troubles of embedded, real-time knowledge based systems due to the environment of agents being typically dynamic, inaccessible, and nondeterministic.

The complexity of agents, their environment, and the interaction between agents and environment suggest that experimental testing represents a major research effort in the area of multi-agent systems. However, systematic experiments with agents have not found the expected attention in designing agent architectures. Jennings and his colleagues observe at the end of the 90ties that testing agents is the least developed research area in multi-agent systems [29]. Some time has elapsed since Paul Cohen, Steve Hanks, and Martha Pollack wrote their paper on controlled experimentation, agent design, and associated problems [26]. Their controversy about the role of concrete test beds in designing agent systems has neither lost its topicality nor its virtue, though. Test beds, e.g. Dvmt [17], Phoenix [24], Tileworld [43], the soccer game [34], and the proposed large scale disasters [33] represent a complement to conventional benchmark tests, offering test scenarios which are aimed at revealing prototypical problems in dynamic environments. Within this testing in the small, it is not the purpose to confront the agent with a valid model of the concrete environment the agent shall dwell in. Instead, it aims at presenting a simplistic world in which properties of a concrete agent or a simplified version of an agent can be analyzed in isolation, e.g. properties of the built-in planning components [35] – of course, finding such "correct" simplifications is a very crucial part in the experimentation process. Test beds help to explain and understand why agents behave as they do by illuminating behavior facets of a given agent architecture.

Planning agents are a research area where the need for experimental testing becomes especially apparent as their behavior is often difficult to predict. Adding planning capabilities to an agent architecture offers the advantage of an explicit knowledge representation combined with a stable and robust decision procedure for the strategic behavior of the agent (for an introduction to planning techniques see [60], [59]). Planning showed to be very useful in many agent-based applications, e.g. the *New Millennium Remote Agent* controlling the Deep Space 1 probe [40] or for collecting data on the Internet [22]. The techniques itself are well developed and in most cases provably correct. But the difficulty in prediction arises from the fact that we still do not have enough theoretical knowledge about the structure of complexity in planning problems in general, as well as the specific needs of a given domain and their influence on planning performance. Most symbolic planners have fully deterministic algorithms and are well understood by the researchers, but e.g. the planning competition at the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)[1] still leads to big surprises. Furthermore, these competitions are based on (compared to real-world problems) relatively simple benchmark problems – definitely not the kinds of environments agent-applications would be used for. So the question still is to determine the behavior of an agent using a planning system for decision making in a dynamic world: Given the planning system, will it be fast enough to produce a valid plan for the

[1] See http://www.cs.toronto.edu/aips2000/ for the results of the AIPS 2000 competition.

changing settings, and is the environment not too complex in terms of memory consumption?

Therefore, planning agents shall illustrate the approach we took with the simulation system JAMES to meet the need for flexible, efficient, and sound methods that support experiments with and thus the development of multi-agent system. The experiments will be executed within the TILEWORLD scenario. Therefore, both, planning agents and the scenario are modeled in JAMES. In contrast to other projects in which agents utilize specialized planning systems to maximize their score in the TILEWORLD, in our experiments agents are equipped with general planners, based on which the interplay between time pressure, social awareness, opportunism, trust and viewing range will be explored – in terms of the planning system we vary the time the initial state will persist, the used heuristic functions and the size of the domain model.

Whereas in the paper we only describe experiments we executed with planning agents in the TILEWORLD, JAMES has been used to test other types of agents in other scenarios as well, e.g. to test mobile agents and their strategies in virtual networks [55]. Its broad applicability is one of the properties which distinguishes JAMES from other approaches to test multi-agent systems and among which its firm roots in a formal approach to discrete event simulation might be the most dominant.

II. JAMES - A TOOL FOR DESIGNING MULTI-AGENT TEST BEDS

The objective in developing JAMES, a Java-based Agent Modeling Environment for Simulation, has been to support a modular and flexible construction of experimental frames for multi-agent systems. A declarative and modular model design is central in designing test beds which might range from simple benchmarks to application-oriented test scenarios. As Hanks points out, testing should ultimately be aimed at analyzing how agents will perform in the environment they are constructed for. "The ultimate interest being not simplified systems and environments but rather real world systems deployed in complex environments."[26, p.18]. This is interestingly the very reason why Steve Hanks doubts the effectiveness of small world experimentation and Martha Pollack interprets them as the only possible approach to begin with. If we assume that testing in the small and testing in the large complement each other in constructing agents, compositional experimental designs are asked for which help to re-use and refine components according to the user's and the objective's needs.

JAMES is based on DEVS [62] which belongs to the formal and general approaches to discrete event simulation. The model design is coined by a modular compositional construction of models. It clearly separates the model from the experimental frame, i.e. the environment and context, a model shall be tested in. Unlike other formal approaches to discrete event simulation, it emphasizes a state based view rather than focusing the modeler's attention on events and processes: This allows the modeler to capture agents and their environment as situated automata which are widely used as a unifying abstract perception throughout the literature on agents and multi-agent systems. E.g. Rosenschein and Kaelbling introduce a situated automaton whose interaction with its environment is characterized as the transformation of a stream of inputs into a stream of outputs [47]. Thus, control systems and agents are similar, in the sense that both maintain an ongoing, non-terminating relationship with their environment.

Within situated automata, knowledge is compiled into transition rules. Therefore, they are typically associated with reactive agent architectures, or "reflex agents with state", which do not totally depend on perfect perceptions. Their internal state helps them to distinguish between environmental states which produce the same input but, according to the given *context* (encoded within the agent's state), are different and should be treated as such. An agent updates its state according to the percepts that are interpreted with respect to its current state, chooses its action, updates its state to report its selection, and finally executes the action. Whereas strategies to select the best action vary and the state is structured differently, the sketched scheme resurfaces in different architectures.

State automata are not only used as a frame to construct agents, they also form an abstraction to specify the behavior of agents. This type of abstraction can also be employed by agents themselves. Agents might perceive other agents as state automata. In this case, learning the behavior pattern of another agent means deriving a state automaton which mimics the "opponent's" behavior and allows the agent to choose the most rational action [13]. Thus, state automata play a central role in bringing together "the different strands of research and development in multi-agent systems" as d'Inverno and his colleagues observe and demonstrate "by taking a commercially developed system (AGENTIS) and providing it with a firm, formal foundation (i.e. Z) that lends itself to further analysis and investigation" [16].

Another of DEVS's benefits is making the temporal dimension of an agent's behavior explicit by adding time triggered events. Often, test beds represent time only implicitly, and proceed stepwise [37]. In contrast DEVS defines for each state the time it will persist "per se" and thus allows to simulate a variety of "pro-active" and "reactive" behaviors.

However, as do other formal approaches to discrete event simulation, the DEVS formalism presupposes static model structures and does not provide means for reflection; both are crucial in modeling and simulating agents. To support variable structure models which are able to assess and access their composition, interaction, and behavior structure from an agent's perspective, a new formalism was introduced which expresses agents as reflective concurrent dynamic systems [53]. JAMES represents an implementation of this formalism.

As does DEVS, JAMES distinguishes between atomic and coupled models. An atomic model is described by a set of input events, a state set, and a set of output events. Transitions dictate state and model transitions due to internal

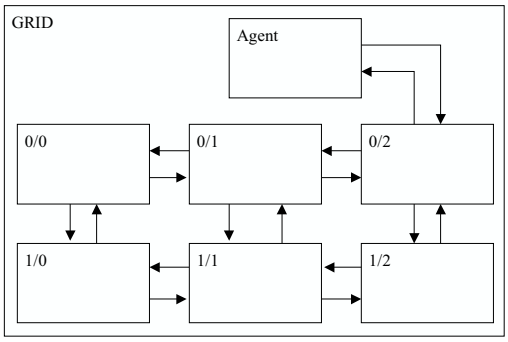

Fig. 1. Composition and coupling of the models

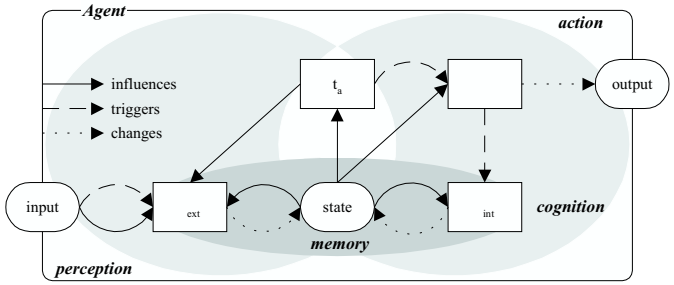

Fig. 2. The internal structure of atomic models and their role in modeling agents in JAMES

events, respectively external input. An output function generates events as outputs based on the current state, and the time advance function determines the time of the next internal event, i.e. how long the state will persist "per se".

A coupled model does not have a state nor a behavior of its own. Its interface to its environment is given by a set of external input and output events. It is made up of a set of (sub-)models, i.e. its components, which may be atomic or coupled, and the couplings which exist among these components. The structure of a coupled model and the behavior pattern of each single atomic model are frequently apt to change depending on the activities of the atomic models.

Coupled models support the modular, compositional construction of experimental frames. It is a matter of indifference for the coupled model whether its components are atomic or coupled. Thus, compositional hierarchies can be constructed. The composite structure of models supports the re-use of components as it supports the refinement of components.

Figure 1 illustrates the structure of a test scenario as a JAMES model. It comprises seven atomic models, the agent and six grid fields, named after the positions they represent. These models are components of the coupled model `grid`, but could equally be re-used to specify refined components for other grid worlds, as has been done in another scenario where the neighborhood of each grid field embraces adjacent, diagonal grids as well and fire spread through the grid world. Another advantage of modeling the environment in this way is the uniform representation of possibly very different *locations*. We could plug-in physically more realistic atomic models, e.g. to simulate the spreading of fires in national parks [14], or by using coupled models instead, we could e.g. simulate the effects of disasters in urban regions [33] or scenarios like loading docks as in [39]. The coupled model can easily represent a refined structure of a location – logical or "real". This means, that on the one hand the modeler can use the compositional structure of a location as an abstraction, a support tool for developing the environment. On the other hand the location can have sub-locations and other entities, like e.g. a building with machines in it. Outside the buildings, the agents use the streets, but inside the granularity of motion increases, and the agents ascend stairs, enter rooms, etc.

At this stage, we do not have such elaborated, complex models, but our approach supports them already in two ways: First, the formalisms provides flexible composition, and second, the simulation can be run distributed. The last point is especially important for the use of "sub-locations", where defined parts of the simulation with potentially many interactions, namely the location itself and the interacting agents, can be re-located to reduce load of the local simulation engine.

The state space of an atomic model can easily be structured to reflect aspects of deliberative agents, e.g goals (desires), intentions, and beliefs. Figure 2 shows an agent's representation in terms of an atomic DEVS model: the cognitive component embraces the transition functions which are responsible for invoking a planning system.

The activities of an agent are modeled by charging its output ports with messages, or by accessing the structure of its environment. Output ports are charged the moment an internal transition function is executed. Thus, they are triggered by the flow of simulation time.

Accordingly, an agent's "output" activities are decoupled from receiving external events. Both, the perception of events and the reaction directed to the environment interact via state and the time advance function (Figure 2). An agent's first reaction to external perturbations will be a change of its internal state. With each state, the time advance function associates the time this state will persist "per se". It determines the time of the next internal event and output, e.g., the time an agent needs for reacting. In the meanwhile, external events might shorten or lengthen the time period until the next output. E.g. some external events might require an agent's immediate reaction which is achieved by producing a state with which a time-advance close to zero is associated. Besides modeling the temporal aspect of an agent's reaction to external events, the time-advance function allows also for a pro-active behavior of an agent, since outputs depend on an agent's current state and are triggered by the flow of time. Thus, an agent does not require any external event to become and stay active.

The "reaction" time refers in some cases to an agent's "deliberation" time. A valid model of the deliberation time is central in testing agents, whose performance typically depends on a timely response. Most approaches model the

deliberation time as a function of the actual computation time [14], other estimate it according to the used deliberation component and the size of the problem [17], again others weight and count instructions which are executed during "deliberation" [3]. JAMES supports the former two time models, since the latter presupposes that the deliberation component can be executed in a "timed" programming language, e.g. "Timed Lisp" which overloads the standard lisp functions [3]. The deliberation components that are tested in JAMES are typically implemented in different languages, e.g. JAVA and C, and started as external threads. Neither JAVA nor C nor other non-interpretative languages lend themselves to an easy implementation of a timed variant.

If internal information about the deliberation component can be accessed, other measures than the real time consumption might be introduced as well: the size of the visited search space by counting expanded nodes, the number of rules triggered, or the number of resolution steps. The size of the beliefs, their structure in terms of literal length or expression depth, or the number of query operations within the beliefs databases can be used as an estimator for deliberation complexity.

Although the above measure support reproducible test runs, their applicability is restricted. E.g. if different planning systems shall be tested and compared, the number of expanded nodes is no valid measure. SAT-based planners generate every possible state and operator instance to start with whereas GRAPHPLAN style planners expand nodes when required. Similar arguments might apply when different knowledge revision algorithms or rule-based systems shall be compared.

Since measures based on easily available information, e.g. the size of the knowledge base, can often only be used as a first very coarse approximation, the consumption of realtime, although polluted by noise introduced due to the runtime environment, seems often the best applicable, most valid time-model to employ. This showed also in experiments we executed in JAMES.

Beliefs about itself and the world around it, not only affect a model's behavior which is articulated by charging its output ports, but determine its possibility to manipulate its surrounding, as well, e.g. to create other models or to change its interaction structure. Structural changes are intrinsically connected with agents, as agents are constructed to work in open environments whose composition and interaction might vary due to the environment's inherent dynamics or due to an agent's deliberative manipulation.

In JAMES, all atomic models are able to create new models and to add existing ones within the boundary of the embedding coupled model. They can delete themselves and their couplings from their interaction context, and determine their own interaction with their environment [56]. Models can initiate their movement from one interaction context (in)to another. Please note, that the *interaction context* is determined by the surrounding coupled model *and* the coupling structure. E.g. in the above model (Figure 1) the agent might move "from field `0/2` to field `0/1`" by establishing a coupling with the destination grid element and removing the connection to the original one. This kind of structural changes can be handled rather simply – in other scenarios, e.g. a softbot within clusters of database servers, the agent might move from one cluster (*coupled* model) *into* another. It can initiate such a move however for its completion, i.e. for being embedded within the new context, it needs the cooperation of an on-site model [56].

To initiate structural changes outside their boundary, agents have to turn to communication and negotiation in JAMES. Thus, a movement from one coupled model to another implies that another atomic model complies with the request to add the moving model into the new interaction context. To facilitate modeling, all atomic models are equipped with default methods that allow them to react to requests, e.g. to add models, to create new ones, or to delete themselves. However, these default reactions can be suppressed to decide deliberately what requests shall be executed. The freedom to decide whether to follow a certain request, and its knowledge, i.e. beliefs, about itself and its environment, distinguish active agents from more "reactive" entities [29]. Thus, typically only some of the models which make up the overall model will form agents.

As do many simulation systems, JAMES distinguishes between the model, where the dynamic system is described declaratively, and the execution of the model, which interprets the dynamics as it is specified within the modeling formalism. The functions of a model, i.e. transition, output, and time advance function are implemented in Java. However, the modeling formalism describes "what" causal and temporal dependencies exist. To exploit this knowledge and to transform the description into the production of trajectories are the responsibility of the simulation layer.

The simulation layer adopts the structure of the abstract simulator proposed by Zeigler [62]. With each atomic model a simulator is associated and with each coupled model a coordinator. They form a tree of processors whose leaves are the simulators. The simulation is realized as an exchange of messages between simulators and coordinators up and down the hierarchy. Activated by messages, coordinators direct the inputs or outputs to the addressees and the simulators invoke the corresponding transition and output functions to update state and to produce outputs. They determine the time of next event of the associated model by applying the time advance function. This information they make known to the superior coordinator. If the models are not affected by incoming events they will be activated the next time when their own internal event is scheduled.

To efficiently test more than a single deliberative agent, which consumes significant space and computation resources, often a concurrent, distributed simulation proves beneficial [36]. Therefore, the processor tree realizes different execution strategies, whereas one adopts a conservative strategy where only processes which occur at exactly the same simulation time are processed concurrently [56], two other strategies split simulation and agent processes into different threads and allow simulation and de-

liberation to proceed concurrently by utilizing simulation events as synchronization points. One of the latter two implements a moderately optimistic simulation strategy and has been developed to support the simulation of planning agents where the agents are modeled within JAMES but invoke external planning and deliberation processes. As experiments demonstrated, the algorithm simulates several planning agents close to the cost of a single agent and increases the performance significantly, given that a sufficient number of machines are available and the agents plan sufficiently long [54].

To test agents of the mobile agent system MOLE [6] within virtual networks a library to describe networks with some simple routing protocols and an interface for plugging existing MOLE agents into the virtual world have been implemented in JAMES [55]. This approach renders an additional modeling of agents superfluous, as they can be tested as they are. This "plug and test" has also been raised as a requirement in testing multi-agent systems, e.g. in the contributions by Logan and Theodoropoulos [36] and Sarjoughian and his colleagues [49]. In JAMES, the entire MOLE agents with their states can be "plugged" into the virtual world. Therefore, a conservative variant of the above optimistic strategy has been developed, which allows simulation and deliberation to proceed concurrently however which avoids rollbacks from the outset. Performance differences of the conservative and optimistic variant have still to be explored.

```
class SimplePlanningAgent extends AtomicModel {

  State deltaExt(State state, double elapsedTime) {
    state.beliefs.update(input, elapsedTime);
    if (intentions.plan == noOp) {
      state.goals.update();
      state.intentions.update(beliefs,goals,operators);
      state.setTimeAdvance(intentions.deliberationTime);
    } else {
      state.setTimeAdvance(REACTION);
    }
    return state;
  }

  void lambda(State state) {
    Action action = state.intentions.getAction();
    if (state.beliefs.entail(action.pre))
       outPortPut("out",action.outputEffect(state));
    else outPortPut("out", noOp);
  }

  State deltaInt(State state) {
    Action action = state.intentions.getAction();
    if (state.beliefs.entail(action.pre)) {
       action.transitionEffect(state);
       state.intentions.popAction();
       state.setTimeAdvance(INFINITY);  }
    else { // re-planning  ...
      state.intentions.update(beliefs,goals,operators);
      state.setTimeAdvance(intentions.deliberationTime);
    }
    return state;
  }
}
```

Fig. 3. Extract of the "simple planning agent" in JAMES

III. PLANNING AGENTS IN JAMES

The "simple planning agent" (Figure 3) shall illustrate how an agent can be modeled in JAMES. The agent generates a plan and then executes the plan stepwise and blindly. Its activity is triggered by perceptions only, the agent is called from the environment with new percepts. Time is considered in a rudimentary manner by increasing an internal count by one. The simple planning agent is obviously inspired by the beliefs, desires, and intention (BDI) model of agent architectures [9].

The attitudes, beliefs, desires, and intentions, comprise an agent's internal state. The external, internal transition, and output function are responsible for updating attitudes and determining suitable (re-)actions:

- The external transition function encodes the reaction of the agent to incoming events in terms of state changes. First, its beliefs are updated. Updating the intentions means to develop a plan to achieve the newly selected goals. The transition function relates the different attitudes of an agent, i.e. its beliefs, desires, and intentions. The time advance function determines the reaction time, the time an agent needs to produce its output.
- Part of the agent's activities is communicating actions to other models. The output function takes the "first" of the intentions and charges its output ports with effects directed to the environment. These are messages which have to be interpreted by the receiving model.
- The internal transition completes the activity by updating the agent's state. E.g. the expected effects of the action are added to its beliefs, and the time to the next internal event is determined. Sometimes an agent's activity might also refer to structural changes. In this case, the internal transition function will execute one of a set of predefined methods which are responsible for e.g. creating, adding a model, removing itself, or changing an interaction with another model.

The approach assumes actions with two layers. The symbolic layer is comprised by beliefs, goals and intentions, which are interpreted by the planning system. On the lower layer, the actual execution of the intended symbolic activity in JAMES has to be specified, i.e. `action.outputEffect` and `action.transitionEffect`. It has to be determined with what kind of information which output port is charged, and what effect the action has on the agent's state, particularly beliefs, and its environment in terms of structural changes. In the latter case the transition precipitates a change of the overall structure of the model.

The "simple planning agent" in Figure 3 takes its deliberation time into account, and considers whether the current action is executable as planned. If the current planning step is not executable according to its beliefs, it starts re-planning. The time for deliberation determines when the agent will be able to start with plan execution, it models the time needed for planning. The longer the agent needs for plan generation, the more likely re-planning is required in a dynamic environment.

The agent works well if the completion or failure of a plan step is the only thing to be observed in the environment.

```
class PatientPlanningAgent extends AtomicModel {

  State deltaExt(State state, double elapsedTime) {
    state.beliefs.update(input, elapsedTime);
    if (intentions.plan == noOp) ||
       (state.beliefs.planFailed()) ||
       (elapsedTime > state.timeForAction) {
      state.goals.update();
      state.intentions.update(beliefs,goals,operators);
      state.setTimeAdvance(intentions.deliberationTime);
    } else {
      if (state.beliefs.entail(action.pre)) {
        state.setTimeAdvance(REACTION);
      } else {
        state.setTimeAdvance(INFINITY);
      }
    }
    return state;
  }
```

Fig. 4. External transition of the "patient simple planning agent" in JAMES (δ_{int} and λ are identical with the "simple planning agent", Figure 3)

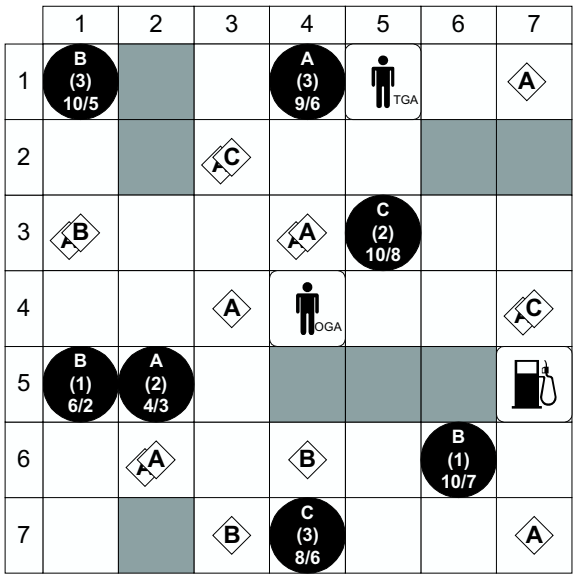

Fig. 5. The TILEWORLD scenario in JAMES. Diamonds denote tiles, black circles holes which are attributed with a type, a depth, and with a score to be gained by filling it with the correct and incorrect type of tiles (6/2). The grey grids are obstacles.

However, it does not refer to the problem that some plan step might take some time to be completed, and in the meantime other things might happen in the environment. Interrupted while executing the agent will notice that the preconditions for the next plan steps are not achieved and will start re-planning. The moment the agent is triggered by an incoming event, it is not able to distinguish between a failure in execution and a still ongoing plan execution. This requires differentiating between inputs that refer to the current plan step and inputs that do not. A "fail" requires re-planning. If the desired effect of the current plan step is confirmed, the agent continues executing the plan after the reaction time. If the environment does not inform the agent about a failure, our agent needs some other means to proceed. The time an action will likely take to complete helps in designing a less irritable agent (Figure 4). Whereas the first step of a plan is executed after generating the plan, to proceed executing the plan steps both agents wait for inputs.

IV. THE "TILEWORLD" SCENARIO

TILEWORLD [43] has been initially created to test different control, particularly commitment strategies of IRMA agents [8].

TILEWORLD is a two dimensional grid world with tiles which can be moved, and holes which should be filled with tiles. There are obstacles which impede the movement of agents, and gas stations which allow the refilling of consumed energy. Tiles, holes, and obstacles appear and disappear at a certain rate, according to global parameter settings. Thus, the environment displays probabilistic (we will come back to this point later) dynamic behavior.

The effectiveness of an agent is measured in terms of scores that summarize the number and kind of holes filled, and the type of tiles used for filling. An agent's activity consumes more resources, the more tiles it carries and the longer it walks. Therefore, it has to re-charge its energy frequently by visiting a "gas station" (Figure 5). TILEWORLD combines a counting problem, i.e. how many more tiles of what type does the agent need to fill a particular hole, with route planning in a grid world. This setting puts only few constraints on the search space and implies a costly deliberation with respect to computing time and memory.

It is in fact very similar to the logistics benchmarks, e.g. in the AIPS competitions, but has some unique characteristics. First of all the navigation task is more difficult than that of typical benchmarks, as they deliver parcels via a smaller set of possible paths. The goal to have not only "a" solution but an "optimal" solution by choosing the appropriate tiles for the holes, once again increases the difficulty. Last but not least, the changing environment has never been captured by classical planning systems' benchmarks, as they are primarily designed for static environments only – and for exactly one acting entity.

Most test beds for multi-agent systems represent the environment by a single entity the agents interact with. Describing the grid world as a composition of multiple grids enables the experimentator to plug in new physical features or types of grid elements, and to adapt the given test bed easily to more complex spatial dynamics, e.g. the spreading of fires or other disasters which call for the agents' decisions and actions. Each grid is modeled as an atomic model which changes its internal behavior with a certain probability after a certain time. The behavior pattern and state space of a grid change depending whether it plays the role of a "normal" grid, a hole, an obstacle, or a gas station. Also the interaction structure between grids and agents exhibits some dynamics. Agents are located on the grid, they are attached to by couplings. They move by removing and adding those couplings to adjacent grids.

The agents, whose primary goal is to increase their scores, are based on the refined version of the "simple plan-

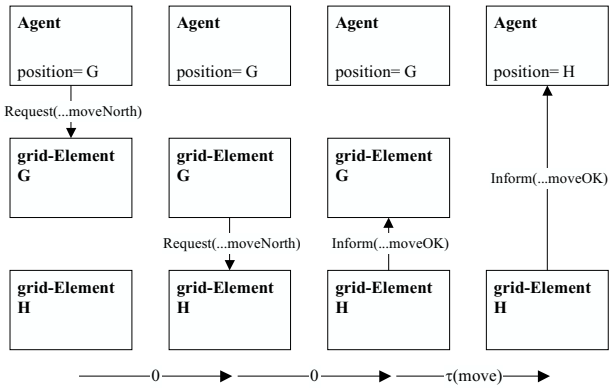

Fig. 7. The agent is moving from grid position G to H

ning agent" with distinguished phases of plan generation and execution. They are bold agents: as long as their plan is executable, they will stick to that plan and reconsider new options only if a plan explicitly fails. More reactive agents would tend to adapt their behavior strategy each time an action is required, see also [20]. In addition and in contrast to the "simple planning agent", the activity of our TILEWORLD-agent is not triggered by external events only, but can easily exhibit pro-activity as well, e.g. if no plan can be generated, an agent can start to explore its surroundings, with the hope that new options might present themselves. The interaction with its environment is based on the agent's request and the environment which does or does not grant what has been requested. Each interaction with the environment is structured by protocols [50].

scanning the environment: The agent sends a request to the grid it is attached to asking for information (Figure 6). This request is propagated according to the sensor range of an agent among adjacent grids. The result is collected by the central grid and sent back to the agent. The agent interprets the results and updates its egocentric map which is part of its beliefs. This modeling facilitates an easy extension of the scenario with physically more accurate sensor models, e.g. directed or non-perfect sensors, like decreasing accuracy or noise. Thus, we do not take omniscience for granted, as Gelenbe and his colleagues do [20], but explore how incomplete information might influence the performance of the agents.

picking up: The agent sends a request for picking up tiles to the grid it is attached to. The grid answers the request by forwarding the tiles if available, otherwise it declines the request.

putting down: The agent puts down tiles, either to gain the score connected with a hole or to lighten its burden if it is running short on gas. All grids except obstacles will welcome tiles. A hole answers to the put request by sending the scores gained to the agent.

moving: If the agent wishes to move, it does so by requesting a move from the grid it is attached to (Figure 7). The grid asks the desired location, and if the location is free, it will grant the moving request and the agent is moved, i.e. its couplings are moved to the new grid.

refilling: If the agent is running out of gas, it will send a request to the grid to refill the gas. The request is positively answered only if the grid represents a gas station.

For an example of these protocols from the agent's point of view see Figure 8.

The agent gets the requested facts about its environment at time point t_1, saying among other things that there is a hole on the field to the east. After some deliberation (planning, etc.) the agent sends its current intention, i.e. to move east, to the associated grid element. The (simulation) time the agent needs for its deliberation, $d - time$ is calculated according to the real-time consumed during deliberation multiplied with the "real-time knob". The processing of the move takes a pre-defined amount of simulation time, and the agent, which has set its time of next internal event to infinity, is waken up at t_3 by the field it moved to. The agent then decides after some time (t_4) to lay down one of its tiles, which is acknowledged by the field and rewarded with ten points at t_5.

In our agent architecture the agent plans its movement between locations and which tile to pick up, whereas the goal selection, i.e. which hole shall be filled, is done by heuristic functions.

V. EXPERIMENTS IN JAMES

The mediation between reactivity and deliberation and its effect on the performance of agents is often at the core of evaluating agents, e.g. [14], [32].

In designing rational agents, commitment strategies decide whether agents are likely to drop goals and seize new options. The filtering of options, as done in the IRMA architecture, means that according to the current commitment, an agent will bypass options that conflict with its goals and the goals of its group. With less committed and more opportunistic agents, the coordination of agents becomes more difficult. But an absolute commitment is not realistic in a dynamic environment. An "appropriate" response to environmental change depends on valuing the options an agent has with respect to the given context [27]. To become aware about relevant changes in the environment, monitoring strategies have to be realized. E.g., Pollack introduces a rationale-based monitoring which focuses the attention of an agent to that part of the environment which is included in the plan rationale and thus likely related to the current commitments [57]. The trade off between reactive and deliberative capabilities implies agent architectures that comprise different layers. The design of these agent architectures requires answering the question of what aspects should be included within a plan, and what should be deferred to lower, "more reactive" layers. Within multi-agent scenarios questions of inter-agent interaction arise, e.g. how shall work load and tasks be distributed among agents [15], what benefits are to be gained by a cooperative community of agents [23], and what strategies shall agents use to negotiate [48]. All of these design decisions are subject to testing.

Our objectives in testing our agent appear comparatively modest, which is also caused by the agents' simple struc-

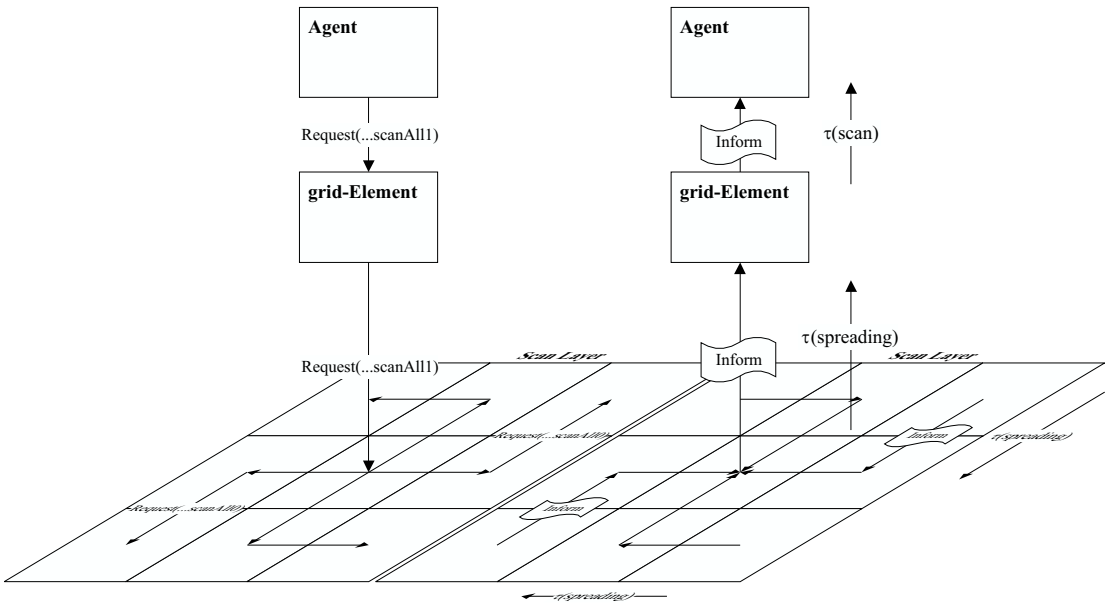

Fig. 6. The agent scans its environment

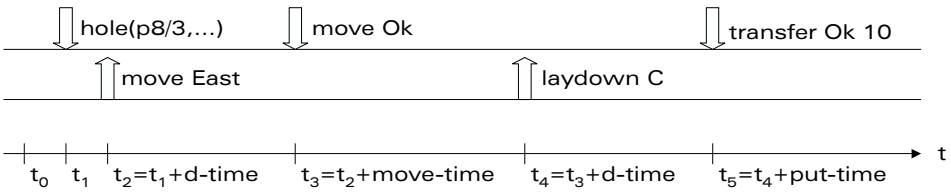

Fig. 8. An example for the interaction between the agent and its environment.

ture. To speed up the design process of an agent application, we wanted to use a *standard general planning system* for decision making instead of building highly specialized TILEWORLD-planning components. We found it more feasible to look for existing tools, in this case GRAPHPLAN [7], a recent planning system that induced a large number of successor systems, due to its extraordinary good performance on several benchmark problems. We considered it being capable of doing the job, because of two reasons: Although GRAPHPLAN's strength lies in producing plans with many steps being executed in parallel – which is not applicable here – it is searching for many plan step sequences in parallel. By filtering out mutual exclusive plan steps it is relatively easy for it to find the executable sequences, which is a big advantage in a domain, where you can do many things at each point in time (moving somewhere or picking up). The second thing is the memoisation technique GRAPHPLAN uses, it helps especially in the navigation task. Please note that these have been (and still are) conjectures, as all experimental results were not applicable to the given situation.

The question still remained, how useful such a general purpose tool would be, after it seemed obvious that specialized components would do better, anyway. Therefore we wished to explore the principles in the underlying domain that affect the planner's performance. What possibilities does the architecture offer to meet deficiencies of the general planner, and what principles of the underlying domain could be used to support the planning system, e.g. by modifying the domain model? We started with the most fundamental question of any deliberative architecture, namely what difference planning for one or two goals makes for a resource-bounded agent in a TILEWORLD scenario: in which situations it pays to plan more far ahead. On the other hand we were interested in the conditions under which the architecture reaches a balance between reactive and deliberative behavior. Therefore we conducted two series of experiments.

To initialize the scenarios, holes up to a certain depth, obstacles, tiles, gas stations, and agents are distributed stochastically. As in original TILEWORLD, the scenario is controlled with a set of parameters which define the size of the grid, the maximum number of holes, tiles and hole depth, the frequency of environmental changes, etc.

A. The First Experimental Series

In our first scenario two agents are concurrently active on a 7×7 grid. Their relationship is characterized by competition rather than cooperation.

The "real time factor" parameter is set to relate the actual computing time in *seconds* to the agent's simulated deliberation time in *time units*. It determines the time

Agents	rtf: 1.000	rtf: 0.100	rtf: 0.010	rtf: 0.001
OGA	71 %	63 %	65 %	53 %
TGA	29 %	37 %	35 %	47 %

Fig. 9. Percentage of scores gained by the agents OGA and TGA with different real time factor values

Agents	rtf: 1.000	rtf: 0.100	rtf: 0.010	rtf:0.001
OGA	67 %	54 %	51 %	46 %
TGA*	33 %	46 %	49 %	54 %

Fig. 10. Percentage of scores gained by the agents OGA and a socially aware TGA*

pressure agents are exposed to, e.g with a real time factor of 0.1, ten seconds of CPU-time are mapped to one time unit. Each agent is able to scan the area 3 grids around it. Thus, it is equipped with an ego centric map with size 7×7: In the center of the grid an agent sees its entire world. Each experiment consists of 10 simulation runs each of which comprises 200 time units. Please note, that the simulator processes are event based, and deliberation time and execution time over a continuous time base are taken into account.

In our first experiment we let one agent ponder two goals simultaneously, whereas the other agent considers only one goal at each time point. Both agents use GRAPHPLAN as the underlying planning system.

At first glance surprisingly, the "one goal agent" (OGA) proves in most cases to be more efficient than the "two goal agent" (TGA) (Figure 9). This seems non-intuitive, because the environment's dynamics are comparatively low, i.e. in most scenarios things change slower in the environment than it takes the agents to plan and execute. Thus, both agents should have a decent chance to generate and execute their plans in a still valid environment. Furthermore, the running time of GRAPHPLAN-style planners is mostly influenced by the size of the domain and not by the length of the produced plan. The time to generate a plan should variate only slightly between both agents.

The time consumption for the agents' action-execution is relatively high: with a real time factor of 0.1 an agent can e.g. do only one or two `move` actions while the other one is occupied by planning. Therefore, one would assume, that the agents' scores should converge much faster. Moreover, an agent satisfying two goals at once should do its job more "efficiently" than one with a more local view on its task. Actually, the number of actions required to fill two holes is typically less than filling two holes subsequently and independently. This effect is also shown in the energy consumption.

Concerning *energy consumption* per achieved point, planning for two goals outperforms planning for a single goal by about 10-25%. But when it comes to the overall performance in points, the smarter TGA apparently cannot compensate for the (slight) time advantage of OGA for generating a plan. Due to the used time model, which multiplies the real time used by 1/1000, OGA has only a small time advance left, even if the TGA needs hours to complete its plan whereas OGA is usually finished after a few seconds.

A remark on the used heuristics: The agents in Figure 9 use the same utility function for selecting their options (holes). The utility is defined by the distance of the agent to the target, its value and complexity (depth). In the experiments of Figure 10 TGA uses a modified function, that has different weights on the factors, and that also takes into account the distance of an option to other agents. When looking at the raw data, we can find that TGA's absolute score is increased by 33%, while on the other hand its variance is reduced by 35% and its fuel consumption by 10-25%. This still cannot compensate OGA's success. We can isolate several reasons for this behavior.

Since both agents are living in a "small" world, it is likely that both will head for the same tiles to fill the same holes with, independently of the used heuristics; in fact, variance in the scores decreases significantly when increasing the board size. Whereas one agent is still thinking, the other agent is already putting its plan into action. Typically, OGA changes the preconditions TGA bases its plan upon and not vice versa. The moment TGA notices that the plan is no longer executable as intended (an action fails), it starts to reconsider its options, i.e. to re-plan. Thus, TGA seems always slightly behind the development of its environment – as long as the experiment's duration is short enough for both competitors to stay on fuel. TGA's situation slowly improves when we decrease the impact of thinking time (Figure 9) or when we equip it with the capability to roughly assess the intentions of OGA and consider those in selecting goals (new heuristic in TGA*, Figure 10). Otherwise the dynamics in TILEWORLD are unpredictable. A grid might turn each moment into a hole, a gas station, or an obstacle, it might start growing tiles, totally unimpressed by the current situation. There are no causal structures about the behaviors of grids to be learned or to be taken into consideration. Deliberation about how the world evolves might seem not worth the effort. This is one reason why Hanks doubts the value of the TILEWORLD scenario in testing single deliberative agents [26]. However, since the environment evolves sufficiently slow, most of the deliberation efforts of an agent are not in vain. Also this argument of Hanks does not take into account that the causal structures underlying the behavior of the real environment are often neither known in advance nor deducible by the agent. Thus, the randomness in TILEWORLD might reflect even aspects of the agent's later application domain.

In a third experiment we test GRAPHPLAN against a planner which has demonstrated its supremacy in a series of benchmark tests [46]. In the chosen scenario however, it unexpectedly fails to meet the high expectations. The agent that uses GRAPHPLAN for deliberation outperforms the other agent in the above problem domain by far.

One of the reasons might be that GRAPHPLAN records all search paths (even the non successful), a control strategy which also proved beneficial in the "Tower of Hanoi" benchmark test. However, further experiments are neces-

sary to explain the poor performance of the second planner. GRAPHPLAN runs regularly into problems, as well. At a certain constellation of the scenario, e.g. planning for two holes with a certain depth, GRAPHPLAN is no longer able to generate a plan within a reasonable amount of time. We see that the variance of the score can be decreased by more than 30% when beginning with structured environment settings, i.e. "corridors" and "rooms". This phenomenon is puzzling, since the problem structure of this situations exhibits no obviously different quality.

We think, the explanation lies in the separation of accessible spaces: To get into a room, the agent has to follow some corridor. Every other path to the final destination is (in most cases) significant longer or even leading not at all to the goal position and therefore it is easier to identify shorter paths. In a totally randomized setting, many comparably good paths exist in parallel, they share intermediate steps and then split again. In addition, randomly distributed obstacles happen to be placed more often on the bord of the grid compared to the structured scenario. In the peripheral position they reduce the freedom of decision, i.e. planning complexity, less than obstacles in other positions. These are the properties which make search much more expensive.

The sensitivity with which the agents reacted to the slight changes in their deliberation strategies, motivated us to explore more systematically the behavior of a couple of different agents by variating a few easily scalable parameters which characterize the agent's interaction with its environment.

B. The Second Experimental Series

One problem agents are faced with is their uncertain and incomplete knowledge about their environment. One possibility to handle this problem is to frequently scan the environment. Since such an update of its knowledge is not a cost-free operation, the agent has to balance costs and benefits. Another central design decision refers to the agent being more reactive or rather goal-driven, or, putting it differently, being opportunistic or strongly committed. Whereas the first parameter determines how fast the agent becomes aware of changes in its environment, the second one determines whether it takes those changes into consideration. Thus, the second parameter determines whether those changes will have an effect on the agent. The first parameter, we call *trust*, the second its *boldness*.

We define the *trust* of an agent by the number of plan steps, the agent executes blindly. If it does not get a failure notification by the environment, e.g. when trying to move onto an obstacle, the agents checks its options after as many steps as indicated by its *trust*. New target holes are only taken into consideration if they are evaluated at least *boldness*% better than the ones the agent already committed to. For example, an agent with *trust* 5 and *boldness* 10, committed to an option with an estimated value of 20, will check every 5 steps if its plan is still executable. If this is not the case it looks for a hole, rated at least 22 – if none is found it tries to find a new plan for the committed goals. In our experiments, *trust* ranges from 2 to 10 and *boldness* from 0 to 50.

The strategy responsible for selecting the goals had a significant influence in the last setup, so we tried to improve the heuristics of our agents by making more sophisticated estimations of the number of steps necessary to fulfill a commitment. The first agent uses the heuristic of the old TGA*, the second one (TGA**) introduces adapted weights and takes into account the average distance of the nearest usable tiles. Whereas the first two agents wait for new options to arise when no plan could be found, a more explorative behavior would doubtlessly improve the performance, we guessed. Thus, we designed a new "explorative" TGA$_e$** that walks randomly around if GRAPHPLAN fails. It happens quite often that the planner does not have enough space to build up the planning graph. In this case the agent starts to stroll around, "looking" for new options – it has no commitment. On a non-empty grid, a random movement might resemble a wall-following behavior as not all directions are equally probable or possible to pursue.

Each of these three different agents we equipped with a different visibility. Each series is started with a sensor range of 5, letting the agent always see the entire world. We reduce this range to 4 and 3, giving the agent a global view only in the center.

Strategy	Scan range 5	Range 4	Range 3
TGA*	*trust* 2 – 10 *boldness* 0% – 50% 10 runs / combination	...	...
TGA**	...	...	...
TGA$_e$**	...	...	...

Fig. 11. Overview over the second experimental setup.

In the second experimental series we use basically the same kind of environment as in the first one. This time we experiment with a single agent on a 6×6 plane. Different agents' strategies and sensor ranges are tested for each combination of which we vary the two parameters – *trust* and *boldness* – and conduct with each setting 10 experimental runs. The abscissa denotes the boldness and the ordinate the trust. The darker the shades the higher the scores which the agent has received on average in this particular setting. Please note that the experiments where executed with settings that represent subsets of the overall range, i.e. we choose boldness out of $[10, 20, \ldots 50]$ and trust out of $[1, 2, \ldots 10]$. The regions in the figures represent interpolated versions of our results.

Figures 12 and 13 show some results of the experiments with the TGA* agent. The five colors represent the five value classes, darker shaded areas are associated with higher values. Depending on the experiment the values differ between 0-45 (Figure 12), 40-210 (Figure 13), 45-150 (Figure 14), and 120-235 (Figure 15).

The overall average score was 30 points for the TGA* agent with scan range of five, rising up to 104 for the agent with a scan range of three. The results of average score depending on boldness and trust are depicted in Figure 12

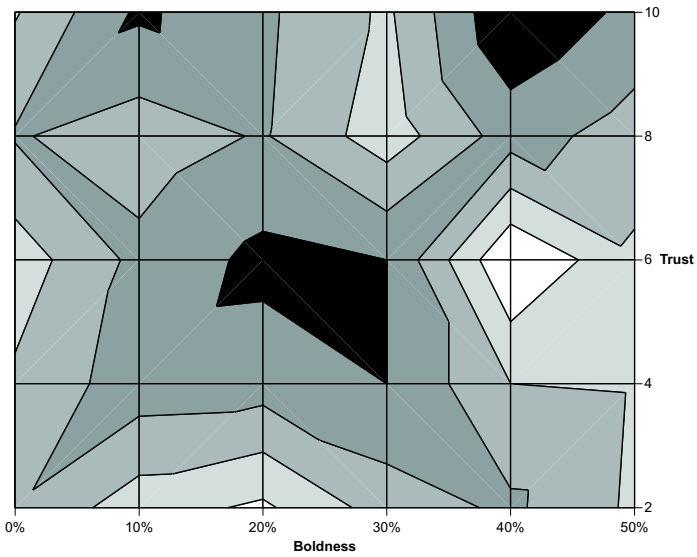

Fig. 12. Average scores for TGA* with scan ranges 5.

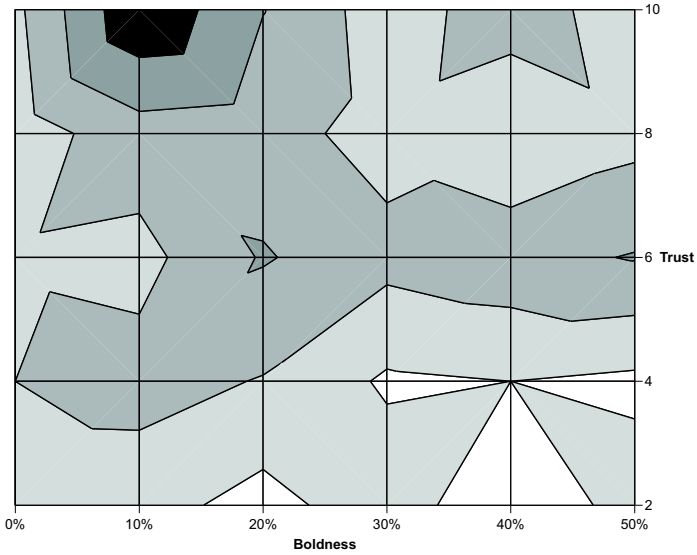

Fig. 13. Average scores for TGA* with scan ranges 3.

and Figure 13 respectively. The minimum in the diagram (Fig. 12) at 40%/6 demonstrates the difficulties for the TGA* agent with high boldness and medium trust. The smaller scan range (Fig. 13) stabilizes the agent's performance by reducing the standard deviation by 12%. Within the TGA* experiments the standard deviation and scores received seemed hardly correlated. Some areas of darker shade (e.g. boldness 40% and trust 10 in Fig. 12) are characterized by a low variance.

What we find is that the agent seems to perform best with a small scan range and that large trust and small boldness values do pay off. But the variance is still unacceptable high.

In the next series we tested the agent with the advanced heuristic function, i.e. TGA**. For the wide scan range this lead to an average score of 96. But what looked like

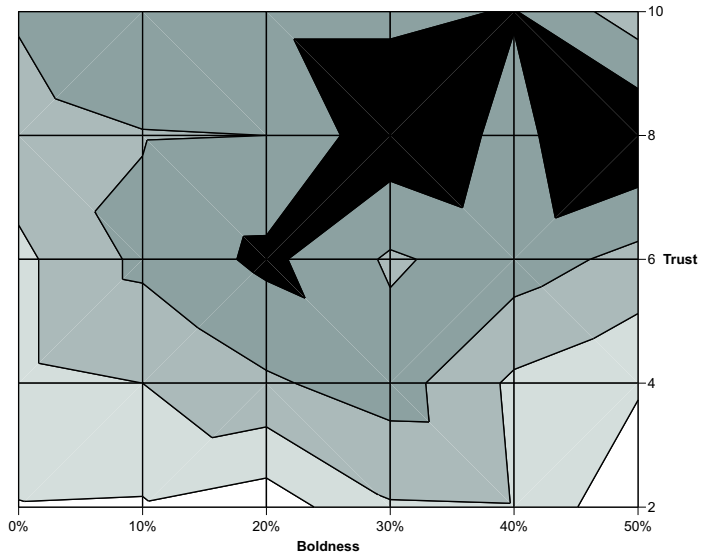

Fig. 14. Average scores for exploring TGA$_e$** with scan ranges 5.

an improvement in the first runs turned out to get somewhat contra-productive for the reduced scan ranges: The heuristic is easier mislead if its calculation involves less objects of a smaller area. If the agent sees just a 3×3 field in the corners, the estimation for a committed hole typically changes rapidly after nearly every movement and the heuristic puts emphasize on the options towards the center where the agent can see more tiles. Thus, as a side effect the agent will automatically move from wherever it initially started to the central region of TILEWORLD where its vision improves – and its performance degrades. In the midth of TILEWORLD the agent overlooks the entire field and it is more likely that the agent will not find a plan due to the domain size. Without plans which can be generated the agent is again waiting for new option to arise. This time of inactivity reduces the scores that could potentially be gained. But at least the overall stability improves, mostly for the larger scan ranges.

Figure 14 and Figure 15 show the last experiments with the explorative version of the TGA** architecture.

For the completely visible field we find a distribution of scores for TGA$_e$** which equals those of TGA**, and the overall average score is nearly the same (102). That result was expected, since the exploration starts when no plan can be found. That in turn is in the "omniscience" setting only happening if GRAPHPLAN fails due to memory overflow – but this can only be avoided if the exploration leads to choosing simpler goals (shallower holes). For few runs, the agent found those simpler goals by chance, but not in the majority, which explains the slightly better performance. In the cases with limited visibility however, the agent's performance significantly improves up to an average score of 188 points and with an overall standard deviation reduced by about 10% (relative to score).

If an agent sees its complete environment the introduced heuristic signs responsible for the good performance of the TGA$_e$** agent since exploration in the face of unrestricted

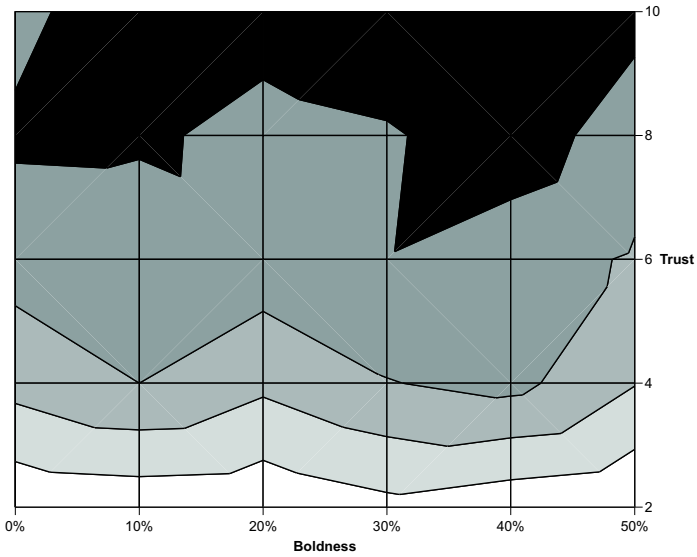

Fig. 15. Average scores for exploring TGA$_e$** with scan ranges 3.

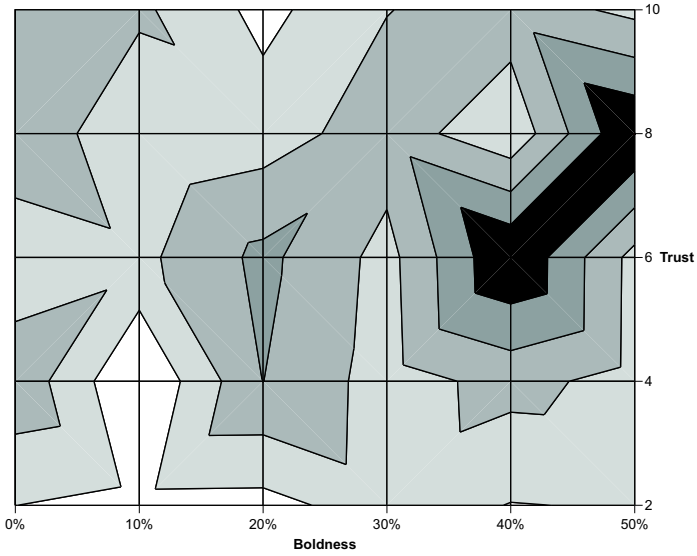

Fig. 16. Score variance of the exploring TGA$_e$** with scan range 3.

visibility does not offer new opportunities. As it can be seen in Figure 14, it does not pay to take new options too willingly into account – the heuristic already selects the "best options" for the complete world at a given time point and considering new options is expensive. In the case of restricted visibility, the introduced heuristics and the capability of exploration together increase the scores.

If the sensory input is restricted the effect of the first series of experiments with the TGA* agent reappears, i.e. planning gets more tractable and new opportunities can be more easily adopted, planned for.

If we compare the variances in Figure 16 with the scores in Figure 15, again the local maxima of the variances and the scores do not overlap. Thus, in our experiments a better score coincides with a more stable behavior.

To summarize our results, it seems the most promising strategy to combine limited sensory capabilities with explorative strategies, or to speak in terms of the planning system renunciation of a better logistics in favor of finding a solution in more problem instances. The exploration phases replace the phases where the agent typically waits for new options to arise. Since no memory capabilities are implemented, exploration changes the view of the agent rather than extending it and the domain size is directly correlated to the range of vision.

In the case of unrestricted sensory capabilities the agent has an extended view of its environment. In contrast to a restricted view with memory, this view contains the current state of the world. However, since planning based on a 6×6 world for two holes proves unexpectedly expensive in some cases, an opportunistic behavior is no longer recommendable. The costs outrun the benefits. The agent will often be unable to generate a plan. If the agent had been equipped with a memory and means for belief revision it had been reasonable to equip it with some means to focus its knowledge for plan generation, as well. Thus, our experiments confirm other experiments conducted with agents in complex environments [27]. Obviously, TILEWORLD offers a complexity where planning systems react very sensitive to an increase of domain size.

Trust in the environment pays in most experimental settings. One should note, that with a restricted view the length of most plans will be less than 10. With a trust of 10 an agent is likely to execute its entire plan. The effect of trust on the scores might be an indicator that the costs of scanning (in terms of time used) are chosen comparatively high and a punishment of failure should be introduced for a more balanced view on the effects of trust. According to the current experimental results, it seems overall better to willfully follow a plan than to scan and check every once and again the validity of its intentions and the availability of better options which seems, in general not a sensible procedure.

Throughout the experiments some properties of JAMES proved to be of particular importance, among those were the modular, compositional design of models, the reflective time triggered automata perspective in modeling agents and environment likewise. In this way, we could plug the different kinds of grid elements together step by step. Debugging the model was facilitated by the automata perspective, because every entity could be described separately without having to take execution control into consideration. The ability to include different explicit time models helped us to guide the experiments by having means to emphasize actions of interest, and the execution within a distributed environment enabled us to perform test runs even on small machines. In the following, we will conclude with comparing JAMES to some other approaches that support testing agents in dynamic environments.

VI. UNDE VENIS ET QUO VADIS?

As the number of agent architectures increases so does the desire to assess and contrast the results achieved by quantitative experiments. "Although garnering many plau-

dits and continually increasing in popularity, the technology and research field remain relatively immature. ... Many conclusion drawn about the technology remain qualitative and subjective. This dearth of quantitative results means as yet it has not been possible to evaluate the potential of the technology and the paradigm" concludes Papaioannou [42] for the area of mobility. Not only in the area of "mobile" agents, increasing efforts are directed to execute and also to support quantitative experiments with agents: efforts of which we will give a short, necessarily incomplete overview.

DESIRE (DEsign and Specification of Interacting REasoning components) represents a formal specification framework for multi-agent system design, which is coined by a hierarchical (de-)composition [10] and thus, exhibits many similarities to the model design of JAMES. As in JAMES, each component has its own input and output interface specification. Components (composed or primitive) can be (re)used as building blocks. The functionality of a composed component is specified by the way in which it is composed from its subcomponents. DESIRE distinguishes between different types of links, each of which has a particular impact on the receiving components. The focus is clearly on designing systems. It is aimed at verifying properties of systems, first steps to combine DESIRE and a proof system have been made [18]. Jung and Fischer criticize the lack of dynamic structures: only knowledge and goals are subject of change and not strategies and operational ingredients, nor does DESIRE support the dynamic creation of components and a variable interaction pattern [31, p.41].

"An advanced, resource adapting perspective such as InteRRaP proposes is thus not straightforward to model". Recent developments in DESIRE take this criticism into account. They support the evolution of multi-agent systems by specifying agents that reason about the properties and structure of agents and thereby, are able to delete and create other agents [12].

A few simple simulation experiments via time slicing document the possibility of interpreting DESIRE models by simulation [11]. However, a time model which would allow one to capture the time boundedness of agents or their concurrent asynchronous behavior is not supported. The same is true for other state based approaches which specify agents based on states and execute this specification within a simulated, e.g. grid based, environment [37]. These efforts appear more focused on the specification of agents and their interaction rather than on experimenting with agents or supporting the construction of experimental frames.

Other test beds are originated in a certain type of knowledge representation or reasoning mechanism around which the simulation system evolved. E.g. SIM_AGENT [51] is based on the forward chaining mechanism of the POPRULEBASE interpreter. The behavior is described as a set of rules. It has originally been built to design single expert systems rather than to experiment with concurrent active autonomous agents. The system is embedded in the POPLOG programming environment, which supports the definition of agents by predefined libraries, e.g. neural nets, theorem provers, planning algorithms, and relational data bases. The simulation algorithm proceeds by time slicing. In each iteration all objects can sense their environments, run processes that interpret inputs, and produce actions. The external actions are executed after all internal changes are completed [51]. Its major strength, as is that of other approaches, e.g. AGENDA [19], DESIRE [11] or TAEMS [58], is its being a toolkit for constructing agents which has been equipped with a simulation engine. Issues in modeling and simulation, e.g. questions of re-usable experimental frames and efficient and sophisticated simulation algorithms have hitherto not found the required attention. However, this is about to change. The developers of TAEMS announced a distributed version of the simulator which currently proceeds by time slicing [58]. Another time slicing multi-agent simulator, i.e. GENSIM, is prepared for running in a distributed setting concurrently as well [1]. Theodoropoulos and Logan have started developing a distributed simulation approach for SIM_AGENT [52] which shall allow an efficient simulation of multiple deliberative agents with explicit time models.

Other test beds for multi-agent systems, e.g. the ROBOCUP simulator, do not employ an explicit time model at all. The simulation engine and the agents communicate with each other asynchronously [41], [28]. E.g. the simulator checks frequently whether the agents have decided on an action that the simulation engine has to take into account, otherwise it proceeds with its own calculation. If the agents are the only source of dynamics, this strategy equals calculating the virtual deliberation time based on the wall-clock time. However, if the simulation has its own dynamics, the execution time of agents and the virtual time of the simulation system have to be put into relation: In this case this is done by controlling the execution time of the simulation system. In distributed environments additional noise will be introduced due to varying loads on the network. As a consequence, the controlling element within the experiment is reduced and experimental results are hardly reproducible. An aspect which is of minor interest in executing contests.

In contrast, JAMES's parallel distributed execution mechanisms are based on explicit time models that represent the time used as a monotonous increasing function of the execution process. The gain in performance is closely correlated to the effort the agents need for deliberation. According to the tests, the algorithm simulates several extensively planning agents close to the cost of a single agent, given that a sufficient number of machines are available [54].

Similar to the soccer simulator other virtual environments exist to which agents can be coupled. One such fictional environment for testing planning agents has been developed, based on the ideas of TRUCKWORLD, by Reece and Tate for the RL/ARPA planning initiative [45], [21]. It is part of the PRECIS environment, which is equipped with a discrete event simulation system. The test scenario refers to the development of plans for non-combatant evacuation operations relative to the virtual island PACIFICA.

The PACIFICA scenario has been re-used to set up the

test bed DIPART: Here agents and simulator are modeled as autonomous units which interact via event queues [44]. The simulator runs as a separate process and represents the actual state of the world. The models kept by agents may be limited or may become out-of-date, they represent the view the agents currently have as they do in JAMES. As in many other approaches the simulator entails the description of the environment. Modeling and execution of the test domain are not separated.

The popular simulation game SIMCITY is used by Hammond and his colleagues to test the real time properties of a case-based planning system [25]. The case-based planner has the role of the major of a city, who has to combine reactive and deliberative capabilities. The major has to attend quickly to fires in the city since otherwise they will spread. Long-term activities, e.g. the layout of an industrial park, require planning. The planning system runs in real time against the simulation system, as does the soccer simulator.

PHOENIX tests planning agents and their capability to fight fires in a virtual forest. The underlying discrete event simulation engine [2] is responsible for synchronizing the processes of the agent's environment, e.g. fire propagation and the agents. Agents are just another type of processes which produce time stamped events which have to be synchronized with the events produced by the environment. The tool box for constructing test beds is complemented by two statistical tools which help setting up experiments and interpreting the produced results [4]. Objects and agents have to be defined as event streams, i.e. processes, which are executed by the simulation engine. A hierarchical, composite, modular modeling of an agent's environment is not supported.

To facilitate the construction of test scenarios, the simulator AFS (Abstract Force Simulator) has been introduced. It supports the modeling of domains that can be described as agents moving in a two-dimensional space and applying force to one another [5]. The model units are called "blobs", circular objects with a set of predefined physical attributes, e.g. *mass*, *velocity*, and basic actions, e.g. *move* and *apply-force*. Higher levels provide goals and context for the lower levels of an agent. The hierarchical agent control unit serves as a frame to design agents and schedules actions to be executed on an internal queue; at each time step all agent actions are executed that are due. Thereby, users are well supported in constructing and testing agents and agent modules, however they are restricted to a certain type of agent and environment.

Many test beds, e.g. PACIFICA, PHOENIX, ROBOCUP, and SIMCITY, confront agents with a complex dynamic environment which exhibits causal structures. Deliberation about the effects of their own activities and anticipation of the dynamics of the environment by its own does pay in these scenarios. Most of the test beds for agents are not designed as general simulation system, though. They support the construction of a certain type of test scenario, i.e. two dimensional grid worlds, where agents dwell in, e.g. [38], [39], [37] and emphasize the view of agents as abstract robots moving around in physical world.

Their purpose is not the construction of test scenarios for testing agents of different types, e.g. planning robots as well as mobile agents that inter-operate and move on the Internet as it is possible in JAMES [55], nor is it their intention to support the testing of multi-agent systems in the large. They provide dynamic often complex "one-way" test scenarios with a clearly defined interface which agents can be easily coupled to.

If we compare JAMES to more general systems like DESIRE, SIM_AGENT, or TEAMS it becomes apparent that those are primarily systems for constructing agents rather than systems for modeling complex dynamic systems, embedding agents and the efficient execution of models. Whereas the model design to describe the experimental frame, the distributed execution, and the firm rooting in a formal discrete event approach constitute the strength of JAMES, little support is offered to develop agents. Its time-triggered reflective automata perspective provides deliberatively only a coarse frame which becomes also obvious in testing mobile agents of the MOLE system in JAMES [55]. Currently JAMES constitutes a prototype, therefore many features one expects from a simulation system are still missing, e.g. tools for input sampling, output analysis, and graphical user interface. As the libraries of models and system features are growing the development of experiments will be facilitated by re-using components.

VII. CONCLUSION

As Jennings and Wooldridge observe, most groups working on agents and multi-agent systems construct their own test environment from scratch, "clearly an undesirable state of affairs" [30]. Thus, simulation systems are needed which are sufficiently expressive, flexible, and easy to use, to provide researchers and students alike with access to experimenting with multi-agent systems designs.

JAMES integrates agents within discrete event simulation. It is aimed at providing an environment for experimentally testing agent architectures, single modules, and interaction strategies. Its main objective is to facilitate testing in the small and testing in the large, equally. The latter of these we associate with the construction of valid models of the agent's environment. Therefore JAMES provides "a clear conceptual framework that enables the complexity of the system to be managed by decomposition and abstraction" [32].

JAMES is not a specification tool for agents presenting yet another agent architecture. Instead, reflective time triggered state automaton are the frame to describe and embed agent architectures in JAMES. Thereby, JAMES supports an abstract but common perception of agents which permeates diverse agent architectures. It provides a general, flexible, modular, and theoretically founded simulation approach with a clear separation between a declarative model design and its efficient concurrent execution. Experiments have demonstrated the suitability of the state automaton metaphor, the ease of integrating different planning systems, and the value of variable structure models to capture

the dynamics of multi-agent systems.

The latter of which are often hard to predict. Even to analyze a single agent's interaction with its dynamic environment might necessitate an experimental approach. Concerning general planning techniques we are still at the beginning of an experimentally founded "big picture" on planning problems and their interaction with the algorithms. The commonly used benchmarks do not cover the problems agents encounter when dwelling in a dynamic world. Furthermore, most (if not all) classical planning systems have so far only been examined in isolation, so it is impossible to predict the performance of an agent architecture that uses them for deliberation. Our experiments gave first hints towards an experimental methodology that tries to find architectural parameters for the use of built-in planning systems. We could show that a first and superficial look on the results of a single test setting (a dynamic benchmark) would have mislead a designer to reduce the planning capability – if not even remove it – for the sake of performance. Only further runs proved such a strategic kind of deliberation to perform better, in terms of score as well as in terms of stability and resource consumption, by finding out how to foster the planner's strengths. Of course our analysis is neither exhaustive nor finished yet. Different planners have to be examined in the same way, in order to find similarities between them. On the other hand we had to discover that even a very abstract and simple domain like the TileWorld has many properties, we would not have thought of without experimentation.

References

[1] J. Anderson. A generic distributed simulaton system for intelligent agent design and evaluation. In H.S. Sarjoughian, F.E. Cellier, M.M. Marefat, and J.W. Rozenblit, editors, *Artificial Intelligence, Simulation, and Planning In High Autonomy Systems*, pages 36–44, San Diego, CA, 2000. SCS.

[2] S.D. Anderson. *MESS User's Manual*. Computer Science Department, University of Armherst, MA, 1995.

[3] S.D. Anderson. Simulation of Multiple Time-Pressured Agents. In *Proc. of the Wintersimulation Conference, WSC'97*, Atlanta, 1997.

[4] S.D. Anderson, D.M. Hart, D.L. Westbrook, and P.R. Cohen. A Toolbox for Empirically Analyzing Artificial Intelligence Programs. *International Journal of Artificial Intelligence Tools*, 4(1):257–279, 1995.

[5] M.S. Atkin, D.L. Westbrook, and G.D. Cohen, P.R. ad Jorstad. AFS and HAC Domain-General Agent Simulation and Control. In *AAAI'98 Workshop Software Tools for Developing Agents*, 1998.

[6] J. Baumann, F. Hohl, K. Rothermel, and M. Strasser. Mole-Concepts of a Mobile Agent System. *WWW Journal - Special Issue on Applications and Techniques of Web Agents*, 1(3):133–137, 1997.

[7] A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[8] M. E. Bratman, D.J. Israel, and M. E. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, (4):349–355, 1988.

[9] M.E. Bratman. *Intentions, Plans, and Practical Reasoning*. Harvard University Press, Cambridge, MA, 1987.

[10] F.M.T. Brazier, B. Dunin-Keplitcz, N.R. Jennings, and J. Treur. DESIRE Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems*, 6(1), 1998.

[11] F.M.T. Brazier, P.A.T. von Eck, and J. Treur. Modelling a Society of Simple Agents: From Conceptual Specification to Experimentation. In *Poster Proc. of the MAAMAW-97*, Ronneby, 1997.

[12] Jonathan Bredin, David Kotz, and Daniela Rus. Trading risk in mobile-agent computational markets. In *Proceedings of the Sixth International Conference on Computing in Economics and Finance*, Barcelona, Spain, July 2000. No proceedings available.

[13] D. Carmel and S. Markovitch. Learning Models of Intelligent Agents. In *International Joint Conference on Artificial Intelligence - IJCAI'97*, 1997.

[14] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine*, 10(3):32–48, 1989.

[15] K.S. Decker and V.R. Lesser. Designing a Family of Coordination Algorithms. In *Eleventh National Conference on Artificial Intelligence*, pages 217–224, 1993.

[16] M. d'Inverno, D. Kinny, and M. Luck. Interaction Protocols in Agentis. In *International Conference on Multi-Agent Systems ICMAS*, 1998.

[17] E. H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers, Boston, 1988.

[18] J. Engelfriet, C.M. Jonger, and J. Treur. Compositional Verification of Mulit-Agent Systems in Temporal Multi-Epistemic Logic. In J.P. Müller, A.S. Rao, and M.P. Singh, editors, *Fifth International Workshop on Agent Theories, Architectures and Languages ATAL'98*, Lecture Notes on Artificial Intelligence, London, 1998. Springer.

[19] K. Fischer, J.P. Müller, and M. Pischel. AGenDA - A General Testbed for Distributed Artificial Intelligence Applications. In G.M.P. O'Hare and N.R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 401–427. John Wiley and Sons, 1996.

[20] E. Gelenbe, E. Seref, and X. Zhiguang. Simulation with learning agents. *Proceedings of the IEEE*, page current issue, 2001.

[21] Y. Gil, M. Hoffman, and A. Tate. Domain Specific Criteria to Direct and Evaluate Planning Systems. Technical Report ISI-93-365, Information Sciences Institute, University of Southern California, Marina del Rey, CA, 1994.

[22] Keith Golden. Leap before You Look: Information Gathering in the PUCCINI Planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages ??–?? AAAI Press, 1998.

[23] G.C. Goldman and J.S. Rosenschein. Emergent Coordination through the Use of Cooperative State-Changing Rules. In *National Conference on Artificial Intelligence*, pages 432–437, Seattle, WA, 1994.

[24] M.L. Greenberg and D.L. Westbrook. The phoenix testbed. Technical Report UM-CS-1990-019, Computer and Information Science, University of Massachusetts at Amherst, 1990.

[25] K.J. Hammond, M.J. Fasciano, D.D. Fu, and T. Converse. Actualized Intelligence: Case-Based Agency on Practice. Technical Report TR-96-06, Computer Science Department, University of Chicago, Chicagor, IL, 1996.

[26] S. Hanks, M. E. Pollack, and P. R. Cohen. Benchmarks, Test Beds, Controlled Experimentation and the Design of Agent Architectures. *AAAI*, (Winter):17–42, 1993.

[27] J. Horty and M.E. Pollack. Option Evaluation in Context. In *7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, Chicago, IL, 1998.

[28] International SRI. *Saphira Software Manual, Saphira Version 5.2*, January 1997.

[29] N. R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Resarch and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):275–306, 1998.

[30] N.R. Jennings and M. Wooldridge. Applications of Intelligent Agents. In N.R. Jennings and M. Wooldridge, editors, *Agent Technology : Foundations, Applications, and Markets*. Springer, 1998.

[31] C.G. Jung and K. Fischer. Methodological Comparison of Agents Models. Technical Report D-98-1, DFKI, Saarbrücken, 1998.

[32] D. Kinny, M. Georgeff, and A. Rao. A Methodology and Modelling Technique for Systems of BDI Agents. In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away*, volume 1038 of *LNAI*, pages 56–71. Springer, 1996.

[33] H. Kitano, S. Tadokor, H. Noda, I Matsubara, T. Takhasi, A. Shinjou, and S. Shimada. Robocup-rescue: Search and rescue for large scale disasters as a domain for multi-agent research. In *Proc. of the IEEE Conference on Systems, Men, and Cybernetics*, 1999.

[34] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The RoboCup Syn-

thetic Agent Challenge 1997. In *International Joint Conference on Artificial Intelligence IJCAI'97*, 1997.

[35] P. Langley and M. Drummand. Towards an experimental science of planning. In K. Sykara, editor, *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 109–114, San Mateo, CA, 1990. Morgan Kaufmann.

[36] B. Logan and G. Theodoropoulos. The distributed simulation of multi-agent systems. *Proceedings of the IEEE*, page current issue, 2001.

[37] M. Luck, N. Griffiths, and M. d'Inverno. From Agent Theory to Agent Construction: A Case Study. In J. Mueller, M. Wooldridge, and N.R. Jennings, editors, *Third International Workshop on Agent theoreies, Architectures, and Languages*, number 1193 in Lecture Notes on Artificial Intelligence, pages 49–63, London, 1997. Springer.

[38] T. A. Montgomery, J. Lee, D. J. Musliner, E. H. Durfee, D. Damouth, Y. So, and the rest of the UM-DIAG. *MICE Users Guide*. Dep. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, January 1992.

[39] J. P. Müller. *The Design of Intelligent Agents*. Number 1177 in LNAI. Springer, Berlin, 1996.

[40] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: to go boldly where no AI system has gone before. *Artificial Intelligence*, 103(1–2):5–47, 1998.

[41] I. Noda. Soccer Server: A simulator for Robo Cup. In *JSAI AI-Symposium 95: Special Session on RoboCup*, 1995.

[42] T. Papaioannou. *On the Structuring of Distibuted Systems: The Argument for Mobility*. PhD thesis, Loughbourough University, 2000.

[43] M. E. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In *AAAI-90*, pages 183–189, Boston, MA, 1990.

[44] M.E. Pollack. Planning in Dynamic Environments: The DIPART System. In A. Tate, editor, *Advanced Planning Technology*. AAAI, 1996.

[45] G.A. Reece, A. Tate, D.I. Brown, and M. Hoffman. The PRECiS Environment. Technical Report ARPA-RL/CPE, Artificial Intelligence Applications Institute, University of Edinburgh, Scotland, 1993.

[46] J. Rintanen. A Planning Algorithm not Based on Directional Search. In A.G. Cohn, L.K. Schubert, and S.C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning. Proc. of the Sixth International Conference KR'98*, pages 617–624, San Francisco, CA, 1998. Morgan Kaufmann.

[47] J.S. Rosenschein and L.P. Kaelbling. A Situated View of Representation and Control. *Artificial Intelligence*, 73, 1995.

[48] T. Sandholm and V. Lesser. Coalitions among Computationally Bounded Agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[49] H. Sarjoughian, B.P. Zeigler, and S.B. Hall. A layered modeling and simulation architecture for agent-based systems development. *Proceedings of the IEEE*, page current issue, 2001.

[50] B. Schattenberg. Agentenmodellierung und -evaluierung im Rahmen eines objekt-orientierten, verteilten Simulationssystems. Master's thesis, University of Ulm, Department of Computer Science, 1998.

[51] A. Sloman and R. Poli. SIM_Agent: A Toolkit for Exploring Agent Designs. In *Intelligent Agents Vol. II (ATLA-95)*, pages 392–407. Springer, 1996.

[52] G. Theodoropoulos and B. Logan. A Framework for the Distributed Simulation of Agent-Based Systems. In H. Szczerbicka, editor, *European Simulation Multi Conference - ESM'99*, pages 58–65. SCS Europe, Ghent, 1999.

[53] A.M. Uhrmacher. A System Theoretic Approach to Constructing Test Beds for Multi-Agent Systems. In F. Cellier and H. Sarjoughian, editors, *A Tapestry of Systems and AI-based Modeling & Simulation Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler*, Lecture Notes on Computer Science. Springer, New York, (to appear 2000).

[54] A.M. Uhrmacher and K. Gugler. Distributed, Parallel Simulation of Multiple, Deliberative Agents. In *Parallel and Distributed Simulation Conference PADS'2000*, Bologna, 2000. IEEE Computer Society Press.

[55] A.M. Uhrmacher and B. Kullick. Plug and Test Software Agents in Virtual Environments. In *Winter Simulation Conference - WSC'2000*, Orlando, FL, December 2000.

[56] A.M. Uhrmacher, P. Tyschler, and D. Tyschler. Modeling and Simulation of Mobile Agents. *Future Generation Computer Systems*, (to appear 2000).

[57] M.M. Veloso, M.E. Pollack, and M.T. Cox. Rationale-Based Monitoring for Planning in Dyanmic Environments. In *Fourth International Conference on Ai Planning Systems (AIPS'98)*, Pittsburgh, PA, 1998.

[58] R. Vincent, B. Horling, and V. Lesser. Experiences in Simulating Multi-Agent Systems. In *Fourth International Conference on MultiAgent Systems ICMAS*, Boston, 2000. AAAI.

[59] D. Weld. Recent advances in AI planning. Technical Report TR-98-10-01, University of Washington, Department of Computer Science and Engineering, October 1998.

[60] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[61] M.J. Wooldridge and N.R. Jennings. Pitfalls of Agent-Oriented Development. In *Proc. 2nd Internation Conference on Autonomous Agents (Agents-98)*, Minneapolis, 1998.

[62] B.P. Zeigler, H. Praehofer, and Kim T.G. *Theory of Modeling and Simulation*. Academic Press, London, 2000.

Biographies

Bernd Schattenberg finished his M.Sc. in Computer Science at the University of Ulm in October 1998. He has been working since 1996 on the design, development, and application of test beds for reactive and deliberative single- and multi-agent systems. He is currently a member of the Artificial Intelligence Department at the University of Ulm, and doing research for his Ph.D. on Hierarchical Planning. His e-mail and web-page addresses are `<schatten@informatik.uni-ulm.de>` and `<www.informatik.uni-ulm.de/ki/schattenberg.html>`.

Adelinde M. Uhrmacher received her M.Sc. and Ph.D. degrees in Computer Science from the University of Koblenz, in 1987 and 1992, respectively. As a fellow of the Alexander von Humboldt foundation she did research at the Department of Electrical and Computer Engineering at the University of Arizona until 1994 when she became an Assistant Professor in the Artificial Intelligence Department at the University of Ulm. In 2000 she joined the Department of Computer Science at the University of Rostock as an Associate Professor. Her research interests are Artificial Intelligence, Modeling and Simulation, particularly the development of agent-oriented modeling and simulation methods. Her e-mail and web-page addresses are `<lin@informatik.uni-rostock.de\>` and `<www.informatik.uni-rostock.de/~lin>`.