

Continual planning and acting in dynamic multiagent environments

Michael Brenner · Bernhard Nebel

Published online: 10 June 2009
Springer Science+Business Media, LLC 2009

Abstract In order to behave intelligently, artificial agents must be able to deliberately plan their future actions. Unfortunately, realistic agent environments are usually highly dynamic and only partially observable, which makes planning computationally hard. For most practical purposes this rules out planning techniques that account for all possible contingencies in the planning process. However, many agent environments permit an alternative approach, namely continual planning, i.e. the interleaving of planning with acting and sensing. This paper presents a new principled approach to continual planning that describes why and when an agent should switch between planning and acting. The resulting continual planning algorithm enables agents to deliberately postpone parts of their planning process and instead actively gather missing information that is relevant for the later refinement of the plan. To this end, the algorithm explicitly reasons about the knowledge (or lack thereof) of an agent and its sensory capabilities. These concepts are modelled in the planning language (MAPL). Since in many environments the major reason for dynamism is the behaviour of other agents, MAPL can also model multiagent environments, common knowledge among agents, and communicative actions between them. For Continual Planning, MAPL introduces the concept of assertions, abstract actions that substitute yet unformed subplans. To evaluate our continual planning approach empirically we have developed MAPSIM, a simulation environment that automatically builds multiagent simulations from formal MAPL domains. Thus, agents can not only plan, but also execute their plans, perceive their environment, and interact with each other. Our experiments show that, using continual planning techniques, deliberate action planning can be used efficiently even in complex multiagent environments.

Keywords Multiagent planning · Continual planning

M. Brenner (✉) · B. Nebel
Institute for Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany
e-mail: brenner@informatik.uni-freiburg.de

B. Nebel
e-mail: nebel@informatik.uni-freiburg.de

1 Introduction

It is an important capability of intelligent agents to be able to determine themselves how to achieve their goals, i. e. to show deliberative goal-directed behaviour. In other words, such agents must be capable of *planning* their actions. Planning has been studied in Artificial Intelligence exhaustively and, particularly in the last decade, algorithms have been developed that in principle are efficient enough to solve realistic planning problems in real time.

However, practical cognitive agents, e.g. robots acting in the real world, are faced with environments that make planning particularly hard: usually, an agent has only very limited knowledge about the current state of the world, mainly because its sensing capabilities are limited. Worse, other agents (including “nature” itself) may change the world without the agent noticing, such that its former knowledge about the world may even become incorrect. In such highly dynamic, partially observable multiagent environments it is impossible for an agent to use the efficient “classical” AI Planning methods that rely on the planner having a complete state description at all times. Planning in such environments can be modelled as a *conformant*, *contingent* or *probabilistic* planning problem. These approaches compute conditional plans or policies for the possible contingencies such that the agent can react adequately when faced with them. Unfortunately, this increased flexibility comes at the cost of being computationally much harder than classical planning [37,47]. Thus, these approaches scale badly in dynamic multiagent environments with large numbers of unobservable features and exogenous events.

Fortunately, in many environments an alternative planning approach is possible: Instead of considering many possible futures in advance, an agent may execute parts of its plan in order to gather additional information, thereby reducing the number of possible contingencies that it has to take into account for the remaining planning. This technique of interleaving planning, plan execution and execution monitoring is called Continual Planning (CP).¹ CP is often advocated as a practical approach to planning in dynamic or incompletely known domains. Yet, only few authors describe how exactly CP can be achieved [1,25]. In particular, the following questions must be answered for any CP approach: If the agent does not have a complete plan when it starts acting, how can it nevertheless behave in a goal-directed manner? How can the agent decide which parts of the problems solving process it should postpone? Do CP agents *plan* their later replanning and if so, how? What is the role of knowledge-gathering actions in CP? In this paper, we present a new principled approach to CP that tries to answer these questions.

Most of the above questions are relevant for single-agent planning in dynamic environments, but become even more important for multiagent environments. Since other agents are the major source of dynamics in such environments, uncertainty can be greatly reduced by monitoring their behaviour and, in particular, by communicating with them. However, CP in multiagent environments raises additional questions: How does the concept of interleaving planning, execution and monitoring generalise to the multiagent cases of collaborative planning and synchronised execution? Which role does communication play there? Are there fundamental differences between action planning and communication planning? The paper sheds some light on these questions by proposing the concept of Collaborative Continual Planning (CCP). We show how CCP agents can behave more successfully than non-collaborative agents in dynamic environments. We also argue for CCP as an interesting new model for

¹ The term *continuous* planning is also often used to describe this form of planning. We prefer the term Continual Planning, since it refers to a repeatedly interrupted and partially postponed planning process rather than a permanently ongoing one. See the survey paper by desJardins et al. for additional reasons [14].

describing the course of *dialogues* between agents and, in particular, interactions that mix communicative and physical actions.

Our CP and CCP approaches are based on the idea of *active knowledge-gathering*: instead of planning for possible contingencies, agents try to learn more about the state of the world directly. In order to enable agents to reason about how they can gather additional knowledge it is necessary to explicitly model the agents' beliefs as well as their sensing capabilities as part of their formal planning domain [21,25,35,43]. Agents can then *plan* how to extend their knowledge. In contrast to Contingent Planning approaches which have used similar ideas, we do not want the planning process to branch over the possible outcomes of sensing actions. Instead we want to enable a planner to *postpone* the decision of what exactly to do with that knowledge to the moment where the actual perception has been made. In other words, we want to “hide” a conditional subplan until the agent has enough information to plan its details. For this purpose, we introduce the concept of *assertions*. Assertions are specific actions, defined normally in the planning domain. However, assertions cannot be directly executed. Instead, the domain designer asserts that the effects of the assertion can be achieved by a subplan if its preconditions are satisfied. In contrast to the similar concept of HTN methods, no decompositions need to be specified for assertions; this is done by the planner itself in later planning phases. Planning with assertions or Assertion Planning (AP) is described in detail in Sect. 3. One important difference to other MA planning techniques is that during planning, assertions are treated like ordinary actions. Consequently, we can make use of the performant classical planning algorithms developed in the last decade to find multiagent plans. In our experiments, we show that complex multiagent problems with a high degree of interference among agents and limited sensing can be efficiently solved by using a state-of-the-art classical planner to produce asynchronous multiagent plans.

To a large degree, our algorithmic approach rests on the capability of agents to explicitly reason about their own and others' sensory and communicative capabilities, their beliefs and mutual beliefs, and about the necessary conditions for joint behaviour. To model these concepts the paper describes the Multiagent Planning Language (MAPL). MAPL plans can freely interleave physical action, sensing and communication, and thus forms the basis for our CP and CCP algorithms. In particular, MAPL plans guarantee that agents autonomously synchronise their behaviour during plan execution.

Since Continual Planning approaches can only be tested in environments where agents can actually interleave planning, execution and sensing, we have developed MAPSIM, a software environment that can automatically generate MA simulations from MAPL domains. In other words, MAPSIM interprets a formal MAPL domain as an executable model of the environment in which agents can perceive, plan, act and engage in dialogues for task-oriented collaboration. While MAPSIM was developed to investigate the approach to Continual Planning presented in this paper, it can also be used for evaluating agents that use different (continual) planning methods.

In summary, the paper makes the following contributions:

- The *multiagent planning language (MAPL)* for modelling MAP domains, including perceptual and communicative actions of agents, their beliefs and mutual beliefs. Plans in this language allow to freely interleave physical action, sensing and communication. Additionally, they guarantee that during execution all agents are provided (by perception or communication) with the necessary knowledge to autonomously, i.e. without a central scheduler or synchronisation component, execute their parts of a MA plan.
- A new principled method for *Continual Planning*, based on active information gathering and the concept of postponing subgoal resolution with assertions.

- A new *Multiagent Planning* algorithm, Continual Collaborative Planning (CCP), which extends the single-agent Continual Planning approach to into a distributed algorithm enabling agents to generate, execute, monitor and revise multiagent plans collaboratively.
- MAPSIM, a *simulation generator* that automatically transforms MAPL domains into multiagent simulations in which agents can plan, act and interact, and which permits the evaluation of CP and CCP approaches.

The paper is structured as follows: In Sect. 2 we present the main concepts of MAPL to model multiagent planning domains, tasks and plans. In Sect. 3 we introduce our approach to Continual Planning. Based on the concept of assertions (Sect. 3.1) we develop the algorithms for Continual Planning (Sect. 3.2) and Continual Collaborative Planning (Sect. 3.3). Section 4 presents MAPSIM, the simulation generator. MAPSIM was used for three different evaluations, presented in Sect. 5: Sect. 5.1 describes the results of an empirical evaluation comparing CP and CCP to classical planning techniques in a dynamic multiagent environment. Section 5.2 analyses a MAPSIM episode where an agent's goals and even the planning domain itself are modified during the CP process. Section 5.3, finally, shows how MAPSIM can be used to produce fairly realistic natural-language dialogues between agents as a result of the CCP process. We discuss related work in Sect. 6. Finally, in Sect. 7 we summarise and discuss future prospects.

2 The multiagent planning language (MAPL)

2.1 Integrating agency into a planning formalism

The term “Multiagent Planning” is an ambiguous one: it has been used both for planning *by* multiple agents, i. e. distributed planning, and planning *for* multiple agents, i. e. planning for multiagent execution. In the most general case, both notions are combined: in a common environments, multiple agents plan and act collaboratively, i.e they synchronise with other agents both during planning and execution.

In this section we discuss a formal language for planning *for* multiple agents, whereas planning *by* several agents will be discussed in Sect. 3. Indeed, the former is a prerequisite of the latter: only when agents can represent their common environment and know what it means to jointly execute plans can they begin to jointly devise such plans.

We will not attempt a definition of what constitutes an “agent” here. Instead, we will only define some necessary (but far from exhaustive) aspects of agency formally and ascribe them to an otherwise unspecified set of agents \mathcal{A} . For example, we will say that agents $a \in \mathcal{A}$ have beliefs, goals and capabilities for acting, sensing and communication. To denote the beliefs, goals, actions, etc. of a specific agent a we will use the superscript notation x^a for any such aspect x and also extend this notation straightforwardly to sets of agents $A \subseteq \mathcal{A}$ where appropriate. Sometimes we will use the functional notation $agt(x) = a$ for the same purpose.

Planning problems and solutions for MA environments differ considerably from their single-agent (or better: agent-unaware) counterparts. The most obvious such difference is that a MA plan can allow agents to act *concurrently*. While concurrent plans have been studied widely, MA plans are more than just that. If the executing agents are to be *autonomous*, i. e. if there is no central scheduler or synchronisation component controlling the joint plan execution, the agents must be enabled to *synchronise* their behaviour at execution time. This is done mainly implicitly by observing others, but can also be done explicitly by means

of communication. In order to assure synchronisation, agents must thus be able to reason about their own and other agents' perceptions, their knowledge and active ways to change it (communication). The MA planning language presented in this section provides explicit representations of beliefs and belief changing actions (perceptions and communication) that permit this kind of reasoning. In fact, MAPL enforces it by defining only those MA plans as valid that guarantee self-synchronised execution (cf. Sect. 2.5).

In addition to the scientific reasons that have led to the MAPL semantics described in this section, there have been practical reasons that have guided the design of its *syntax*. A number of MAP languages have been proposed over time (cf. Sect. 6). Unfortunately, none of these proposals has found widespread use that would be comparable to the success of the now de facto standard for classical single-agent planning, PDDL [18, 23, 40]. We believe that one of the reasons for this is indeed the incompatibility of existing MAP languages to PDDL and, consequently, the difficulty to adapt existing efficient classical planners to MAP. Therefore, the major guideline in the design of MAPL was to keep both its syntax and semantics as close to PDDL as possible. In practice, this means that PDDL and MAPL share a large common subset, i. e. many planning domains and problems are both valid PDDL and MAPL descriptions.² We will not formally specify the MAPL syntax in this paper, but a number of examples in this section will make the similarity to PDDL obvious.

2.2 States and Beliefs

Instead of the propositional representation used in most planning formalisms, our model uses *multi-valued state variables (MVSVs)*. An MVSV v is associated with a finite *domain* dom_v , i. e. a set of possible values it can take. When a state variable takes a particular value from the domain we speak of an *assignment*. For example, the MVSV $colour(ball)$ may be assigned any of the values from its domain $\{red, blue, green, yellow\}$. For a discussion of the SAS⁺ formalism that underlies this model and its relation to propositional planning, see the work by Bäckström and Nebel [2].

There are several motivations for choosing MVSVs as the basis of a multiagent planning formalism:

1. In most multiagent systems, agents have limited knowledge of the environment and limited perceptive capabilities. Even if at some point an agent has perfect knowledge of the world, it cannot be certain of its beliefs at later time points, since actions by other agents may change the world without those changes being perceived by the agent. It is therefore indispensable for planning and acting in MA environments to be able to represent *ignorance* about facts in addition to propositional truth and falsity.
2. Multi-valued state variables provide a natural way to model planning domains even when there is complete information. For example, the position of an object o can be modelled by an MVSV $pos(o)$. This naturally encodes the fact that an object is at exactly one position at a time. This constraint is only implicit in propositional planning languages like PDDL.
3. As shown by Helmert [28] this change in representation is not only convenient for modelling but can be exploited successfully by planning algorithms.

² This common subset mostly corresponds to the ADL part of PDDL 2.1 without conditional effects. Instead of conditional effects, MAPL has so-called Causal Domain Rules [6], but these have been omitted from this paper to simplify the presentation.

4. Multiagent environments are examples of distributed systems and multiagent plans can be seen as (very basic, but automatically synthesised) distributed algorithms. Distributed systems are usually modelled in terms of private and shared variables. Concepts like *read-write conflicts* or *access locks* to shared variables can easily be translated to multiagent planning when using a state variable representation.

Throughout the paper, we will mostly consider MVSVs as atomic entities. However, as indicated by the *functional notation* used in the examples, we assume that *classes* of MVSVs $f(t)$ can be defined similarly to *predicates* in PDDL, i. e. by specifying relational symbols f (with some arity k) and parameter tuples t (also of arity k) which define possible argument types [24]. In this way it is possible, for example, to specify that *colour()* must be defined for all objects of a certain type.

We can now recoin the definitions for basic STRIPS planning in terms of MVSVs.

A *partial variable assignment* (PVA) over \mathcal{V} is a function s on some subset of \mathcal{V} such that $s(v) \in \text{dom}_v$ wherever $s(v)$ is defined. def_s (undef_s) is the set of defined (undefined) variables in s . If a PVA $s(v)$ is defined for all $v \in \mathcal{V}$ then s is called a (complete) *state*. If $s(v)$ is defined (with value x), then the pair (v, x) is called an *assignment* (also written $v = x$). Two PVAs s and s' are called *consistent* if the following holds: if both $s(v)$ and $s'(v)$ are defined, then $s(v) = s'(v)$.

To show the similarities to propositional planning we will use set notation and logical connectives to refer to PVAs where convenient. For example, we will denote the completely undefined PVA by \emptyset and also write $(v = x) \in s$ instead of $s(v) = x$. We define the *union* $s_1 \cup s_2$ of two *consistent* PVAs s_1 and s_2 as the PVA s where $s(v) = x$ if $s_1(v) = x$ or $s_2(v) = x$. Otherwise, $s(v)$ is undefined. Propositions are modelled as MVSVs over the domain $\{\top, \perp\}$. Instead of $f(t) = \top$ we will allow simply writing $f(t)$.

Furthermore, we will write $v_1 = x_1 \wedge \dots \wedge v_n = x_n$ to define a PVA s where $s(v_i) = x_i$ for all $v_i \in \mathcal{V}$, $x_i \in \text{dom}_{v_i}$ appearing in the formula. For all other variables $v \in \mathcal{V}$, $s(v)$ is undefined.

To model beliefs and ignorance with MVSVs, we can extend the domain of a state variable v associated with an agent $a \in \mathcal{A}$ with a special value *unknown*, thus expressing the fact that a does not know the value of v .

Definition 1 A set of MVSVs induces a set of *belief state variables* \mathcal{V}^A for a group of agents \mathcal{A} as follows: For each $v \in \mathcal{V}$ and each non-empty subgroup $A \subseteq \mathcal{A}$ there is a $v^A \in \mathcal{V}^A$ with $\text{dom}_{v^A} = \text{dom}_v \cup \{\text{unknown}\}$. If $|A| = 1$ [$|A| > 1$] then v^A is called an *individual belief (IB) variable* [*mutual belief (MB) variable*]. For convenience we allow to write v^a for IB variables $v^{\{a\}}$.

Based on belief state variables, MAPL describes *belief states* for some agent a simply as PVAs over \mathcal{V}^a . It is obvious that with this pragmatic approach certain forms of beliefs cannot be expressed that would be expressible with a possible world semantics [22]. In particular, constraints that are believed to hold *between* state variables cannot be modelled. For example, the constraint $\text{pos}(a) = x \leftrightarrow \text{pos}(b) \neq x$ could describe that no two agents can be believed to be at the same position at the same time. Often, though, knowledge of such constraints can be modelled by introducing complementary state variables, for example *occupant*(x) would describe who is standing at position x and could have value a or b (plus some dummy unoccupied), but not both. Note also that MAPL allows planners to explicitly restrict the domain of an MVSV v for individual belief states such that disjunctive beliefs (e.g. a is in London or Paris, but not in Berlin) can be described.

It would be possible to also model nested beliefs (up to some fixed level) as belief state variables, e.g. “ a believes that b believes that c believes that $v = x$ ”. Yet, we currently restrict our formalism to individual and mutual beliefs. Note, however, that the “world state”, as known by a planning agent a , is in fact already a belief state of a . Likewise, a ’s belief state variables implicitly are nested beliefs (“ a believes that agent b or group B believes that $v = x$ ”). This one level of nesting plus mutual beliefs is sufficient for the domains we are currently interested in. An additional explanation is provided by our communication model (cf. Sect. 2.4.1).

A state s is *belief consistent* if the following holds: if $v^A = x$, then $v^{A'} = x$ for all $A' \subseteq A$. In words, mutual beliefs imply mutual beliefs by all subgroups (in particular: individual beliefs). It is easy to see that belief consistency implies that agents cannot have contradictory mutual beliefs within different groups of agents. Formally: If s is a belief consistent state, then $s(v^A) = s(v^B)$ for all $A, B \subseteq \mathcal{A}$ where $A \cup B \neq \emptyset$ and both v^A and v^B are defined in s .

For the rest of the paper, we assume that belief states are belief consistent. This means that we assume that whenever an agent updates the value of a belief variable $v^A = x$ it also updates $v^{A'} = x$ for all $A' \subseteq A$ and marks v^B as undefined for all $B \not\subseteq A$.

It is often important that agents can reason about their own future knowledge or the beliefs of other agents without knowing the exact value of a belief variable. For example, in Continual Planning it is essential that an agent can assert that at some future point in time, it will know the value of a state variable, regardless of which value this will be—because it can then enter a new planning phase based on its now more detailed knowledge. To this end, MAPL provides the concept of *Know-If (KIF) variables* v_{KIF}^A . KIF variables describe *whether* the state of a variable is known or not. As such KIF variables are boolean, i.e. classical propositions. Whenever $v^A \neq \text{unknown}$ in a state, then $v_{KIF}^A := \top$ is set automatically. However, KIF variables can also be made true explicitly in a plan, e.g. by application of *sensor models* as described in the next subsection.

2.3 Multiagent planning domains

We will now describe what constitutes a multiagent planning domain.

Definition 2 A *multiagent planning domain* is a tuple $\mathcal{D} = (\mathcal{A}, \mathcal{V}, \mathcal{C}, \mathcal{E})$ consisting of agents \mathcal{A} , state variables \mathcal{V} , constants \mathcal{C} , and events \mathcal{E} . For each state variable $v \in \mathcal{V}$ there is a finite domain $dom_v \subseteq \mathcal{C}$.

Events $e \in \mathcal{E}$ have the form $e = (a, pre, eff)$ where $a \in \mathcal{A}$ is the *controlling agent*, *pre* is a PVA over \mathcal{V} called the *precondition* of e , and *eff* is a PVA over \mathcal{V}^A called the *effect* of e .

MAP domains are very similar to classical *grounded* STRIPS-like planning domains, i.e. we assume that schematic definitions using variables (e.g. for state variables and operators, as in the example shown in Fig. 1) have been replaced by the sets of their possible instantiations. Thus, we will regard events and state variables as atomic for the most part, just as in propositional planning. Note, however, that agents \mathcal{A} and constants \mathcal{C} may appear implicitly in state variables, too. For example, a state variable $v = (posrobot)$ may refer to an agent $robot \in \mathcal{A}$ or a constant $robot \in \mathcal{C}$.

MAPL *events* correspond to instantaneous *actions* in PDDL, and we will use both terms interchangeably. However, the more neutral term “event” hints at the fact that the same action that one agent executes on purpose constitutes an *uncontrollable* event for another one. A crucial difference to PDDL actions (apart from the use of MVSVs) is the explicit inclusion of

```

(:action move
 :parameters (?a - agent ?to - location ?d - door)
 :variables (?from - location)
 :precondition (and
  (pos ?a : ?from)
  (doorstate ?d : open)
  (entrance ?d ?from) (entrance ?d ?to))
 :effect (pos ?a : ?to))

```

Fig. 1 A MAPL operator describing movement from one room to another through a specific door

the *controlling agent* in the definition of an event. This is necessary to enable reasoning about the subjective perspective of individual agents on plans and their execution. The distinction between *objective* (i.e. outside) and *subjective* perspectives on plans is crucial for many aspects of multiagent planning we discuss. Whenever a particular perspective is adopted in this paper, it will be explicitly mentioned.

Since Def. 2 describes the objective perspective on a MAP domain, event preconditions range over \mathcal{V} , thus describing only the physical conditions for executing actions. Knowing the controlling agent a of an action, we can also reason about this agent's *subjective* perspective, in particular about the induced *knowledge preconditions* (ranging over \mathcal{V}^a) that must be satisfied before the agent will autonomously try to execute the action (cf. Sect. 2.5). Note that action *effects* may change beliefs of agents directly. This is necessary for modelling the effects of communication and sensing.

An example for the definition of a MAPL operator in the PDDL-like syntax used in our implementation (cf. Sect. 4) is shown in Fig. 1. Several things are notable: most visibly, MAPL uses colons to signify assignments to state variables. However, assignments to boolean state variables can either be written $(svar : true)$ [$(svar : false)$] or $(svar)$ [$(not\ (svar))$], thereby preserving compatibility with PDDL. Note also that a MAPL state variable is guaranteed to have exactly one value (or is undefined) in any state. This allows MAPL actions to be specified with less parameters, since the planner can safely fill in the missing values from the context of the current state. For example, the move action has no parameter referring to the current location of the agent, because this can be determined by the planner as the current value of the state variable $(pos\ ?a)$. A similar concept was originally proposed for PDDL (cf. [40]), but could not be guaranteed to work in a propositional planning formalism, because multiple propositions of the form $(pos\ ?a\ ?from)$ could be in principle be true in the same state for different values of $?from$.

Another important difference to standard planning formalisms is that events can depend on or affect the belief states of (other) agents. In the following we will model perception and communication by means of such events. Usually, planning domains heavily restrict which belief state variables the controlling agent of an event can access.

2.4 Perception and communication

MAPL describes three ways for agents to update their own beliefs and those of others: individual sensing, communication and copresence (joint sensing). Using these constructs, a MAPL domain designer can describe the perceptive capabilities of agents, the conditions for successful communications as well as its exact effects, and the circumstances under which agents need not communicate about certain facts in a situation at all because they realize that they all make the same perceptions.

Fig. 2 The conditions for perceiving the state of a door as described by a *sensor model*

```
(:sensor sense-door
:agent (?a - agent)
:parameters (?d - door ?l - location)
:precondition
  (and (pos ?a : ?l) (entrance ?d ?l))
:sense (doorstate ?d))
```

2.4.1 Sensing

Sensor models describe the circumstances *cond* under which an agent *a* can perceive the current value of a state variable *v*. Objectively, i. e. from the outside perspective, this is defined as follows:

Definition 3 An *objective sensor model* $\text{sensor}(a, v, \text{cond})$ of an agent *a* for a state variable $v \in \mathcal{V}$ is a set of events where for each $x \in \text{dom}_v$ there is an event $e \in \text{sensor}(a, v, \text{cond})$ with $\text{agt}(e) = \text{env}$, $\text{pre}(e) = \text{cond} \wedge (v = x)$ and $\text{eff}(e) = (v^a = x)$.

Since there are no conditional effects in MAPL, objective sensor models are defined as *sets* of events where for each possible value *x* of *v* there is a specific event which can occur only if $v = x$ holds.³ Thus, in any world state at most the single event referring to the actual value *x* of *v* can be “executed” by the environment agent *env* and will then make $v^a = x$ true, i. e. it will make $v = x$ known to *a*. Whether this perception takes place depends on the condition *cond* which usually includes state variable assignments that refer to the state of the perceiving agent *a* directly.

For example, Fig. 2 shows a sensor model that describes under which circumstances a robot will perceive whether a door is open or closed. It completely abstracts from those details of the robot’s sensor systems that are irrelevant for the planning domain (but, of course, these can be as detailed as necessary). In this case the sensor model simply states that the robot will sense the door state when it is situated in any room to which the door is an entrance. The formal sensor model defined by Fig. 2 is $\text{sensor}(a, \text{doorstate}(?d), \{\text{pos}(?a) = ?l, \text{entrance}(?d, ?l)\})$, i. e. the state variable to be sensed is *doorstate*(?*d*) and the condition under which it can be perceived is $\text{pos}(?a) = ?l \wedge \text{entrance}(?d, ?l)$.

When an agent *a* reasons about its future perceptions, e. g. when it plans a sensing action, it (usually) does not know what it is going to perceive and thus cannot use objective sensor models. What *a* does know, however, is that after sensing it will know the value of *v* (whichever it is). This is captured by the following definition:

Definition 4 The *subjective sensor model* $\text{rsensor}(a, v, \text{cond})$ is an event *e* with $\text{agt}(e) = a$, $\text{pre}(e) = \text{cond}$ and $\text{eff}(e) = (v^a_{KIF} = \top)$.

Note that Fig. 2 defines both an objective and a subjective sensor model. Usually planning agents will only be able to employ subjective sensor models in their plans. However, they may use objective sensor models to reason about what *other* agents can perceive. If an agent *a* knows that $v = x$ and also knows that for another agent *b* *cond* holds for a sensor model $\text{sensor}(b, v, \text{cond})$, then *a* can apply $\text{sensor}(b, v, \text{cond})$ and thus infer $v^b = x$.

The set of all (objective and subjective) sensor models is denoted by $\mathcal{E}_{\text{sense}} \subseteq \mathcal{E}$.

³ Note that, based on this definition, MAPL agents can only make exact perceptions of a state variable or none at all. This is clearly an unrealistic assumption (although one shared by most previous approaches to planning with sensing). Therefore, we are currently extending the concept of sensor models such that they can arbitrarily *constrain the domain* for the perceived state variable. The current definition would just be the most restrictive special case of this semantics where the domain is reduced to a singleton.

```

(:action tell_val ??svar
 :agent (?speaker - agent)
 :parameters (?hearer - agent)
 :variables (?room - room)
 :precondition (and
  (K ?speaker (??svar ??args))
  (pos ?speaker : ?room) (pos ?hearer : ?room))
 :effect (and
  (K ?hearer (??svar ??args))
 ))

```

Fig. 3 A speech act template. In this domain an agent may inform another about the value of state variable known to him whenever they are in the same room

2.4.2 Speaking

MAPL *speech acts* do not necessitate a separate definition; we call all those actions speech acts that affect the belief states of *other* agents. However, the MAPL syntax support speech acts in a specific way: while the general rules of communication may be specific for a domain (for example one might want to specify that agents must be in the same room to communicate), the *content* of communication is usually very free. Therefore MAPL permits to describe speech act *templates*, i.e. speech act definitions that take state variables as *parameters*. This is exemplified in Fig. 3 which describes a general way for agents to tell each other the value of a state variable when they are in the same room. The state variable (or rather: the ungrounded state variable schema) is not fixed. Instead, the template parameter *??svar* indicates that grounding should not only range over the domains of the normal parameters, but over the set of state variables of the domain. Since state variables may have different arities, argument types, and domains, MAPL provides syntactic support to refer to these, too. Since the processing of such templates is in essence not different from standard schema grounding, we will not detail this process further here.

We have explicitly not included nested beliefs in our framework; yet the perceptive reader will notice that the speech act in Fig. 3 should lead to one: assuming that an agent *a* communicates a fact $p = (v = x)$ to agent *b*, the effect $v^b = x$ could be expressed as $Bel^b p$ in some standard epistemic logic. However, since *a* knows this to be the effect of his action also $Bel^a Bel^b p$ will be true. Instead of extending our framework accordingly, we make the additional assumption that the hearer always can detect *who* spoke. If this is the case, then *b* could in principle infer $Bel^b Bel^a Bel^b p$, which in turn *a* may infer, etc. In short, under the assumption of perfect communication and speaker detection, our modelling of speech acts induces mutual belief. This is not surprising [22], yet welcome, since it allows us to replace simple knowledge effects with mutual belief effects (among the speaker and hearer) in speech acts. For the rather abstract model of communication chosen for MAPL, these assumptions seem reasonable and were one of the main reasons for not supporting nested beliefs at all in MAPL.

2.4.3 Copresence

Communication is not the only way to achieve mutual belief. Another possibility, *copresence* (or Co-perception) was already described by Lewis [36]. Informally, agents are copresent when they are in a common situation where they cannot only perceive the same things but also each other. Such a situation can lead to mutual belief since the agents can mutually infer their perceptions, the beliefs about other agents' perceptions, etc.

```

(:sensor copres-pos
 :agent (?x ?y - agent)
 :parameters (?l - location)
 :precondition (and (pos ?x : ?l) (pos ?y : ?l))
 :sense (pos ?y))

```

Fig. 4 A *copresence model* specifying that agents achieve mutual belief about their locations when those are identical

We can describe copresence situations as special kinds of sensor models $\text{sensor}(A, v, \text{cond})$ that have effects on a mutual belief variable v^A for a group of agents A . A basic example could be a copresence model stating that agents achieve mutual belief about their respective locations whenever those are identical. We can use exactly the same syntax as in Fig. 2 to describe the copresence model, but with a list of agent parameters instead of a single one, as shown in Fig. 4. The reader will note that only the location of agent $?y$ will become common knowledge. However, in any situation where an instantiation of this copresence model for some agents x and y , and a location l can be triggered, a symmetric instantiation where the roles of the agents are swapped can also be triggered that will make the location of the other agent commonly known.⁴

It is often reasonable to assume that the agents know about their respective sensing capabilities; formally, this means to assume common knowledge about the sensor models. If this is the case, common knowledge can be inferred even from single-agent sensor models when the perception condition already is common knowledge. In that situation all copresent agents could infer the perceptions of the others, plus their inferences, etc. A more formal treatment of this topic will be given in a future publication. There we will also describe an approach to automatically deriving copresence models from individual sensor models.

2.5 Multiagent plans, problems, and solutions

We will now define the semantics of MA plans. Based on this, we can describe MAP tasks and what it means for a plan to solve them.

As usual, we will base our definition of the semantics of plans on the semantics of individual actions. The classic STRIPS-like semantics of actions can be expressed in our formalism as follows:

Definition 5 An event $e = (a, \text{pre}, \text{eff})$ is *enabled* in a state s if $(v = x) \in s$ for all $(v = x) \in \text{pre}$.

The *occurrence* of an enabled event e in state s results in state $\text{app}(s, e)$ where $(v = x) \in \text{app}(s, e)$ iff $(v = x) \in \text{eff}$ or $[(v = x) \in s \text{ and } v \in \text{undef}_{\text{eff}}]$.

The occurrence of a sequence of events, called its *symbolic execution*, is defined inductively as follows: $\text{res}(s, \langle \rangle) := s$, and $\text{res}(s, \langle e_1, \dots, e_n \rangle) := \text{app}(\text{res}(s, \langle e_1, \dots, e_{n-1} \rangle), e_n)$ if e_n is applicable in $\text{res}(s, \langle e_1, \dots, e_{n-1} \rangle)$; otherwise $\text{res}(s, \langle e_1, \dots, e_n \rangle)$ is undefined.

We are, however, not interested in sequential plans, but in concurrent multiagent plans. For this paper, we use a very simple form of concurrency, since our focus is on Continual

⁴ If there are more than two agents at a location, all two-agent instantiations of the sensor model will be triggered, such that pairwise common knowledge is achieved about the respective locations. Unfortunately, this is not identical to common knowledge among the complete group. Currently, the only way to describe this in our framework would be to specify explicit copresence models for three, four, five, etc. agents. However, for the most tasks pairwise common knowledge seems sufficient.

Planning, acting and interacting rather than on the subtleties of synchronous and asynchronous concurrency. Thus, we settle with the following intuition, well known from distributed systems and non-linear planning research: two events can happen concurrently if none of them changes the value of a state variable that the other relies on or affects, too. See [5, 6, 13] for more involved concurrency models that could be combined with this one.

Definition 6 An effect $v = o$ of some event e *threatens* any assignment $v = o'$ where $o' \neq o$. If any effect of e threatens a precondition or effect of another event e' , we also say that e *threatens* e' . Two events e and e' are mutually exclusive (or *mutex*) if e threatens e' or e' threatens e .

We define asynchronous plans as partially ordered plans without mutex conflicts:

Definition 7 An *asynchronous plan* is a pair $P = (E, <)$ where E is a set of events and $< \subseteq E \times E$ defines an ordering relation among the events. If two unequal events e_1 and e_2 are unordered in P , i. e. $e_1 \not< e_2$ and $e_2 \not< e_1$, then e_1 and e_2 must not be mutex.

Note that since plans are intended to be executed by multiple agents asynchronously, non-mutex events may not only occur in every possible total order, but also in parallel (cf. our previous work for an extended definition that also includes enforced *synchronous* concurrency [6]).

We can reduce the semantics of asynchronous plans to the semantics of totally ordered ones, as defined in Def. 5. This is possible because of the following Lemma.

Lemma 1 Let P_1, P_2 be sequential plans corresponding to different total orderings of an asynchronous plan P . Then $res(I, P_1) = res(I, P_2)$ for all possible states I .

Proof It is easy to show that two non-mutex actions will lead to the same state regardless of their application order and prior state. Therefore, both total-order plans can be transformed into each other by repeated pairwise swapping of subsequent non-mutex actions without changing the resulting state. \square

Since the execution order is irrelevant, we can define:

Definition 8 The state $res(s, P)$ resulting from the *symbolic execution* of an asynchronous plan P in a state s is defined as $res(s, P) = res(s, P_{TO})$ where P_{TO} is an arbitrary total order of P .

If $res(s, P)$ is defined for some state s and plan P , we say that P is *valid* in s .

Just like in other planning formalisms, multiagent planning tasks are described by a world state and some goal description that must be achieved by a plan. The main extension in the following definition is the incompleteness of initial states and the possible reference to beliefs in both the initial state and the goal.

Definition 9 A *multiagent planning task* for a multiagent planning domain \mathcal{D} is a pair $T = (I, G)$ consisting of an initial state I and a goal state G , both possibly incomplete and defined over \mathcal{V}^A .

An asynchronous plan P is a (global) *solution* to a planning task $T = (I, G)$ if $res(I, P) \supseteq G$.

The initial state of a MAP task is allowed to be incomplete. However, its solution is an asynchronous, but nevertheless non-conditional plan. Therefore the MAP problem as defined

here is in fact a *conformant* planning problem [31, 50], i. e. a plan must guarantee to achieve a goal under all circumstances. However, in Sect. 3 we will show how contingencies can be ignored in early planning phases of the Continual Planning process and how execution monitoring can lead to non-conformant solutions in later planning phases.

Asynchronous plans can already be regarded as multiagent plans. However, if the executing agents are to be truly autonomous they must have means to *coordinate* the execution of their parts of the plan with others. In particular, this means that they must wait for others to provide the preconditions for their own actions. Sometimes, coordination may also mean that an agent has to *inform* another one about having provided such a precondition. This kind of behaviour can be achieved by making agents reason about the *knowledge preconditions* (KPs) and *knowledge effects* of their actions and those of others. We first define:

Definition 10 A MAP domain $\mathcal{D} = (\mathcal{A}, \mathcal{V}, \mathcal{C}, \mathcal{E})$ is said to be *KP-respecting* if for each event $e = (a, pre, eff)$ in \mathcal{E} the following holds: if $(v = x) \in pre$, then also $(v^a = x) \in pre$.

A MAP domain can be made KP-respecting by extending it with appropriate preconditions. We call the domain \mathcal{D}^{KP} resulting from the minimal such extension the *KP-respecting extension* of \mathcal{D} .

When planning is done using the KP-respecting version of a domain directly, it will guarantee that plans are created that satisfy the additional knowledge preconditions. This means that the plan will have to “explain” the sensing or communication that made individual agents aware of changes in the world. Often matters can be simplified by assuming that an agent is aware of the changes it brought about itself. If this is a reasonable assumption for a domain, its KP-respecting extension can be created ensuring that for each *effect* eff of an event $e = (a, pre, eff)$ the following holds: if $(v = x) \in eff$, then also $(v^a = x) \in eff$.

Definition 11 An asynchronous plan P is a *self-synchronising* solution to a planning task $T = (I, G)$ in a MAP domain $\mathcal{D} = (\mathcal{A}, \mathcal{V}, \mathcal{C}, \mathcal{E})$ if P is a solution (in the sense of Def. 9) to T in the KP-respecting domain \mathcal{D}^{KP} .

In a self-synchronising plan, an agent can only execute an action if not only its usual preconditions are satisfied, but if the agent a also knows about this. The necessity of knowledge preconditions for successful action execution is widely accepted [29]. However, it may also be argued that KPs force agents to act in an overly prudent manner [26]. Sometimes one may want instead adopt a “leap-before-you-look” approach [25], i. e. one may want to determine whether preconditions of an actions were satisfied only by the success or failure of its execution. In a sense, Continual Planning as presented in Sect. 3 is such an approach, but with respect to *plans*, not individual actions.

3 Continual planning

The term “Continual Planner” is often used simply to describe a system that replans in light of state changes which have rendered its previous plan invalid, i. e. a system that does execution monitoring and replanning. While these are indeed important aspects of any continual planning approach, many such systems will only start acting when they have found a *complete* plan. In other words, while the execution process can be interrupted, planning is monolithic.

In this paper, we argue for an extended notion of continual planning where agents can *suspend* the planning process and start acting. But when and why should they do this? How can they decide what to do when they have not finished planning yet? And when will they resume planning again?

We will answer these questions in this section. The key idea of our approach is that agents will deliberately *postpone* those parts of their planning process that concern currently indeterminable contingencies. However, postponing subproblem resolution will only be allowed if instead they engage in *active information gathering*. This ensures that, as soon as the additional knowledge is available, the planning process can be resumed and the plan can be further refined.

3.1 Assertions

In Continual Planning, we want to allow the planning agent to deliberately postpone parts of its planning to later phases in the plan-execution-monitoring cycle when it has gathered more information. In other words, we want to “hide” a conditional subplan until the agent has enough information to resolve the contingency. For this purpose, we introduce the concept of *assertions*.

Assertions are virtual MAPL actions that a planner can reason about and include in a plan as usual, but that can never be executed. The name “assertion” comes from the role these actions play for continual planning: they *assert* that, once their precondition will have been achieved, their effects will be achievable, too—although not by executing the assertion, but by means of a new plan. Therefore, when during plan execution an assertion actually becomes executable, it is *expanded*, i. e. a new planning phase is triggered. In this planning phase the planner can make use of the information gained during the last execution phase and can replace the assertion with concrete, executable actions. In fact, a planner need not even wait to expand an assertion until it is executable. It is sufficient that a specific subset of preconditions of the assertion is satisfied in the current state to inform the planner that replanning is now possible. These special preconditions are called the *replanning conditions*. Formally we define assertions as follows:

Definition 12 An *assertion* is an event e with a distinguished, non-empty set of preconditions $repl(e) \subseteq pre(e)$, called the *replanning conditions*. Preconditions $p \notin repl(e)$ are called the *ordinary preconditions* of e . If $repl(e) = \emptyset$, then e is an *ordinary action*. We denote the set of assertions with $\mathcal{E}_{ass} \subseteq \mathcal{E}$.

When the replanning condition of an assertion has been satisfied, it can be *expanded*, i. e. it can be replaced by a subplan that achieves the asserted effects.

Definition 13 An assertion e is *expandable* in a state s if $repl(e) \subseteq s$. For a plan P , an assertion $e \in P$ is said to be *permanently expandable* in a state s if e is expandable in s there is no event $e' \in P$ with $e' \prec e$ that threatens any replanning condition $c \in repl(e)$.

Usually, assertions that become expandable during the execution of a plan P will also be permanently expandable. However, sometimes this is only the case temporarily. For example, an agent may know that another agent will soon destroy the replanning condition again. In such a case, it will not make sense yet to devise a concrete plan for the assertion. Therefore, the semantics of plans with assertions (or *assertional plans*) will rest on permanently expandable assertions.

The power of assertions for Continual Planning results from the following change in the semantics of plans:

Definition 14 The state $res(s, P)$ resulting from the *symbolic execution* of an assertional plan P in the current state s is defined as follows:

If any assertion $e \in P$ is permanently expandable in s , then $res(s, P)$ is undefined. Otherwise, $res(s, P)$ is defined as in Def. 8.

If $res(s, P)$ is defined for some state s and plan P , we say that P is *valid* in s .

This definition describes how assertions will trigger a new planning phase: as soon as they are permanently expandable, they will render an otherwise valid plan obsolete, thereby forcing the continual planner to switch back into planning mode.

Definition 14 makes an important distinction that is not present in other planning formalisms: it distinguishes between the current state s and the projected future states $res(s, P)$. In particular, replanning conditions work as normal action preconditions as long as they do not hold in s . When, however, the plan has been executed until the point where an assertion is actually permanently expandable in s , the semantics of assertions changes and they will trigger replanning.

This unusual semantics induces an interesting interplay between planning and execution: Assertions will first enable the planner to find a plan and start its execution. Later, however, the same assertions will render the plan obsolete again and force the planner to switch back into planning mode. This process will be specified in detail by Alg. 2 in next section.

Note that all directly executable actions in a valid plan P , i.e. those events $e \in P$ whose preconditions are satisfied in the current state s , can never be assertions (otherwise their replanning conditions would be satisfied, too, and the plan would no longer be valid). This is important, because it means that when an agent selects an action to execute among the ones currently enabled in the plan, it can never be an assertion. Thus, assertions are completely *virtual* actions that only appear in plans, but never in execution.

Example Consider a scenario in which a robot has to explore an apartment, i.e. it must enter all rooms at least once. The robot has a map of the apartment, but does not know which of the doors between adjacent rooms are open. It can sense the state of all doors in a room as soon as it enters it (using the sensor model of Fig. 2). A *contingent* plan for this scenario modelled as a *tree* where each possible perception during plan execution spawns a new plan branch could result in a number of branches exponential in the number of doors in the worst case (depending on the layout of the map) [43,30]. This is due to the fact that, while a contingent planning algorithm will *reason* about future sensing, it cannot actively try to reduce uncertainty by actually *executing* sensing actions. In contrast, this kind of *active information gathering* is encouraged by the assertion `move_a` shown in Fig. 5. Note that, in contrast to the `move` action of Fig. 1, this assertion requires only that the robot knows *whether* a door is open or not, i.e. $(KIF ?a (doorstate ?d))$ or, formally, $doorstate(?d)_{KIF}^a = \top$. However, in order to satisfy this condition, the robot will have to plan a `sense-door` sensing action which, when executed, will provide the robot with the information really needed, i.e. the real value of $doorstate(?d)^a$, which it can then use in the next planning phase to concretise its plan for exploring the apartment.

In summary, assertions are virtual actions that can be used to enable planning for active, *goal-directed information gathering*. Instead of branching on possible perceptions, asser-

Fig. 5 Assertion for planning movements between rooms that ignore the state of doors until perceived during execution

```
(:action move_A
:agent (?a - agent)
:parameters (?to - location)
:variables (?from - location ?d - door)
:replan (KIF ?a (doorstate ?d))
:precondition (and (pos ?a : ?from)
(entrance ?d ?from) (entrance ?d ?to))
:effect (pos ?a : ?to))
```

tions help to deliberately postpone contingency resolution until the missing information is available. Thus, assertions not only abstract from subproblems in a plan (in a way similar to hierarchical task networks; cf. Sect. 6), but additionally lead to a *temporal* decomposition of the planning *process*. The next section show this is accomplished in a planning algorithm.

3.2 Continual planning algorithm

Basically, Continual Planning consists of three interleaved phases:

1. (Re-)planning in order to determine how to achieve the current goals from the current state
2. Plan execution
3. Perception of self-induced or exogenous changes to the world

A continual planning agent that tightly integrates these phases is described in Alg. 1. Execution and monitoring of expected changes are particularly interdependent. Therefore Alg. 1 has only two major subprocesses, which are detailed in Algs. 2 and 3. In Sect. 3.3 we will further extend this model with *collaborative* capabilities.

Algorithm 1 CONTINUAL PLANNING AGENT(S, G)

```

 $P = \emptyset$ 
while  $S \not\supseteq G$  do
   $P = \text{MONITORINGANDREPLANNING}(S, G, P)$ 
  if  $P = \emptyset$  then
    return "cannot achieve goal  $G$ "
  ( $S, P$ ) =  $\text{EXECUTIONANDSTATEESTIMATION}(S, P)$ 
return "goal reached"

```

Algorithm 2 shows how a new planning phase is triggered: Step (1) describes *plan monitoring*, i. e. checking whether a changed current state or an expandable assertion makes the current plan invalid. If this is the case, Alg. 2 first determines the prefix of the asynchronous plan that is still executable (step 2), then extends it with a newly planned suffix. Note that since a multiagent plan is only partially ordered, the “obsolete suffix” consists only of those actions that *causally* rely on preconditions that have become unachievable. Of course, simple replanning is also possible if *plan stability* is not an issue. However, especially in collaborative settings where agents need to reach agreement over their plans, breaking plan stability is costly since it may lead to *asynchronous backtracking*, i. e. plan revision recursively concerning other agents [55].

Algorithm 2 MONITORINGANDREPLANNING(S, G, P)

```

if  $res(S, P) \not\supseteq G$  or any assertion  $e \in P$  is permanently expandable then (1)
   $E = \mathcal{E} \setminus \{e \in \mathcal{E}_{ass} \mid repl(e) \subseteq S\}$ 
   $P' = \text{REMOVEOBSOLETE_SUFFIX}(P)$  (2)
   $P'' = \text{PLANNER}(E, res(S, P'), G)$  (3)
   $P = \text{CONCAT}(P', P'')$  (4)
return P

```

In order to exploit the power of state-of-the-art planning systems, step (3) of Alg. 2 uses calls to an unspecified classical planner PLANNER (supporting the STRIPS subset of PDDL)

as a subroutine. In order to allow this modularisation and still be compliant with the semantics of plans with assertions, the algorithm *removes* all expandable assertions from the set of possible events before calling the PLANNER subroutine. However, it may be more convenient to modify the planner (as we have done in our implementation) so that it assures itself that expandable assertions are not included in a plan. Since most modern non-temporal planners produce totally-ordered plans, the generated must be converted to asynchronous plans. We use a straightforward algorithm for doing this that adds new actions to an asynchronous plans as early as possible but making sure that no conflict arise. Cf. Bäckström's work for a discussion of several variants of this approach [3].

Algorithm 3 EXECUTIONANDSTATEESTIMATION(S, P)

```

 $e = \text{choose a first level event from } P$ 
 $S' = \text{app}(S, e)$  (1)
 $\text{exp} = \text{EXPECTEDPERCEPTIONS}(S', \mathcal{E}_{\text{sense}})$  (2)
if  $\text{agt}(e) = \text{self}$  then
  EXECUTE( $e$ ) (3)
 $\text{perc} = \text{GETSENSORDATA}()$  (4)
if  $\text{perc} \supseteq \text{exp}$  then (5)
  remove  $e$  from  $P$ 
 $S = \text{FUSE}(S', \text{perc})$  (6)
return ( $S, P$ )

```

Algorithm 3 describes the execution phase of Continual Planning: First, it non-deterministically chooses an event e on the initial level of the plan, i.e. one whose preconditions are satisfied in the current state. Note that, due to Def. 14, e can *not* be an assertion. Steps (1–2) first compute the state and perceptions expected as a result of executing event e . If the planning agent is also the executing agent, he will execute e (step 3), otherwise the agent will do nothing but try to *observe* e .

Steps (4–5) try to match the expected results with the real perceptions after executing e . This is also the case if the planning agent was not the executing agent of e , but wants to determine if another agent has executed e . Note that by not removing e from the plan until its occurrence is perceived the CP algorithm enters a waiting loop: Alg. 3 will be executed repeatedly until Alg. 2 detects that external events have made P invalid. Step (6) tries to estimate the next state by fusing old knowledge, new perceptions and expected changes (that might not have been observable, but predicted by e). Basically, FUSE applies those expected effects that do not contradict perc . Contradiction is defined in terms of mutually exclusive state variable assignments.

3.3 Continual collaborative planning

For Continual *Collaborative* Planning (CCP) we must incorporate the Continual Planning algorithm (Algs. 2 and 3) into a *distributed* algorithm that allows communication between agents. The basic CCP algorithm is shown in Alg. 4. As is customary in Distributed Systems, the current state of the algorithm depends on the messages received from other agents [39].

As long as no message is sent or received the basic Continual Planning algorithm is executed as discussed above. However, during his individual CP process the agent may now also ask others to achieve subgoals. This arises naturally whenever an agent devises a plan that includes actions by another agent. Instead of executing such an action himself, the planning

Algorithm 4 CCP AGENT(S, G)

```

 $P = \emptyset$ 
Received no message:
  if  $S$  satisfies  $G$  do
    return "goal reached"
   $P = \text{MONITORINGANDREPLANNING}(S, G, P)$ 
  if  $P = \emptyset$  then
    return "cannot achieve goal  $G$ "
   $(S, P) = \text{EXECUTIONANDSTATEESTIMATION}(S, P)$ 
Received request( $e$ ) from agent  $a$ :
   $sg = \text{TRANSLATEREQUESTTOGOAL}(e)$  (1)
   $P = \text{MONITORINGANDREPLANNING}(S, sg, \emptyset)$  (2)
  if  $P = \emptyset$  then
    send "cannot execute request  $e$ " to  $a$ 
  else
    send "accept request  $e$ " to  $a$ 
    add  $sg$  to  $G$  as temporary subgoal (3)
Received (tell-val  $vx$ ) from agent  $a$ :
  add  $v = x$  to  $S$  (4)
Received "cannot execute request  $e$ " from agent  $a$ :
  add  $e$  to blacklist (5)
Received "accept request  $e$ " from agent  $a$ :
  add  $e$  to whitelist (6)

```

agent must send an appropriate *request* to the other agent. The planning agent will keep a *whitelist* for requests that have been accepted and a *blacklist* for those which have been denied.

To incorporate the generation of speech acts into continual planning, we slightly extend Algs. 3–Alg. 5. The only visible changes occur in steps (4–5) in which requests are determined and communicated. Note, however, that already step (3), i.e. the execution of a planned action, may refer to a communicative action. For example, MAPL domains generally define *tell-val* actions, i.e. actions for informing another agent about a state variable value. If an agent receives such a speech act (step 4 of Alg. 4) it simply adds this information to its knowledge base, i.e. it believes in to the sincerity of other agents.

The specific request is computed by the function SELECTBESTREQUEST in step (4) of Alg. 5. This is a black-box function whose realisation may differ between CCP implementations and applications. SELECTBESTREQUEST can take into account the *whitelist* and *blacklist*

Algorithm 5 EXECUTIONANDSTATEESTIMATION(S, P)

```

 $e = \text{choose a first level event from } P$ 
 $S' = \text{app}(S, e)$  (1)
 $exp = \text{EXPECTEDPERCEPTIONS}(S', \mathcal{E}_{sense})$  (2)
if  $agt(e) = \text{self}$  then
  EXECUTE( $e$ ) (3)
else
   $e' = \text{SELECTBESTREQUEST}(e, P, \text{blacklist}, \text{whitelist})$  (4)
  send request( $e'$ ) to  $agt(e)$  (5)
   $perc = \text{GETSENSORDATA}()$  (6)
  if  $perc \supseteq exp$  then (7)
    remove  $e$  from  $P$ 
   $S = \text{FUSE}(S', perc)$  (8)
  return (S,P)

```

to determine requests that have not already been accepted or refused or which would be *subsumed* by such earlier requests.

SELECTBESTREQUEST also takes into account not only the event in question itself, but also its context in the agent's plan: If the plan contains several actions by another agent, i. e. a whole subplan, it is often best not to request execution of the actions individually, but to ask for the achievement of its end result. This will give the executing agent the liberty to find the best way to combine execution of the request with its individual goals. In other situations or applications, in contrast, it might be crucial for the requesting agent that its original plan is executed as precisely as possible.

Since the decision about the most appropriate request is highly dependent on the domain of application and, in particular, the form of cooperativity among agents, the general CCP framework does not commit to a specific algorithm for SELECTBESTREQUEST. Our standard implementation uses a simple algorithm that determines the maximal asynchronous subplan of P that uses only *one* specific agent. Then SELECTBESTREQUEST will request all actions on the final level of this plan. However, we are currently investigating variants that determine where it is reasonable to “outsource” larger subproblems to *groups* of other agents.

CCP process flow CCP agents are first and foremost individual CP agents. Collaboration among agents arises only when one CP agent includes actions by another agent into his plan, i. e. when it wants to be helped. Whether collaboration is generally desirable or only a last resort largely depends on the domain of application: sometimes collaboration can raise a plan's utility dramatically, e. g. because of concurrent execution, but often the need for synchronisation and negotiation outweighs these advantages. We abstract from this issue here and assume an appropriate cost model that PLANNER can exploit exists for each planning domain, e. g. one that assigns additional costs to actions intended to be performed by other agents.

When an agent receives a request to perform an action, it first translates this request to a new subgoal (step 1 of Alg. 4). If the request corresponds directly to a grounded MAPL action, the subgoal is determined by the *effects* of this action. However, we want to allow for other forms of requests, too; in particular we want to be able to model requests posed by *human* users, e. g. during Human–Robot–Interaction. The reason for translating such requests to goals is twofold: what matters to the requesting agent is usually not the exact action, but the end result achieved, i. e. the achievement of a goal or of a precondition for a subsequent action of the requesting agent.⁵ Additionally, agents may want to use *referring expressions* in their requests, because they cannot make an *unique name assumption* for objects they talk about. This is of particular importance when interaction happens in natural language, e. g. in Human–Robot–Interaction. Referring expressions can be modelled as constraints on the goal state to achieve [7]. The translation process has been described in our previous work [7]. In a nutshell, it maps a request to the effect of the corresponding operator scheme. If referring expressions are used, a more complex formula describing the constraints on the operator parameters is produced.

In the basic form of CCP described here, agents will check whether they can achieve the new subgoal (step 2 of Alg. 4) and, if possible, are always willing to adopt it (step 3). If the request is conflicting with their previous goals, they will reject it. The requesting agent will then add this request to a *blacklist* which is taken into consideration when requests are chosen again (step 4 of Alg. 5).

⁵ This is also where *assertions* play a central role again. By referring to an assertion in a request, an agent a can ask another agent b for some behaviour even without knowing what b knows about the current situation, i. e. a does not need to reason about the abstraction level of b 's ensuing continual planning process. Instead, b translates the request to a goal corresponding to the effects of the assertion and will then use the requested assertion, another one or none at all, depending on its own degree of knowledge.

In practice often a more complex strategy of subgoal acceptance will be necessary. For instance, we have experimented with fixed and dynamic hierarchies among agents where higher-ranking agents will not accept requests from lower-ranking ones. Evaluating the resulting CCP variants is an important topic for future work. The aim of this paper, however, is to introduce CCP as a general framework for collaborative planning; we therefore restrict our attention to the setting with maximal cooperativity among agents.

Temporary subgoals A request is adopted as a *temporary* subgoal (step 3 of Alg. 4), i. e. the agent commits to achieving it at some point during the CCP run, but it need not necessarily stay true later on. Temporary subgoals (TSG) are not first-class constructs in MAPL; instead we exploit the fact that new goals and actions can be added to the planning domain on-the-fly during CP. Essentially, for each TSG sg a unique constant c_{sg} is generated and (*achieved* c_{sg}) is added to G as a conjunct. Then an artificial action (*achieve* c_{sg}) is created whose precondition is sg . Since only (*achieve* c_{sg}) can achieve (*achieved* c_{sg}) this enforces the planner to satisfy the precondition, i. e. sg . However, once (*achieved* c_{sg}) has been made true sg can become false again without preventing G from becoming satisfied. Thus sg is only enforced to be temporarily true.

Requests and TSGs add another important element to CCP: just as the beliefs of an agent are continually revised during the CP process, the use of TSGs lead to continual *goal revision*. Although Alg. 4 only *adds* new TSGs to G , we can in practice remove TSGs that are already satisfied from G . During CCP, agents thus try to achieve continually expanding and shrinking goal sets. In Sect. 5.3 we will discuss variants of CCP that deal with requests and TSGs in particular ways.

4 MAPSIM

Plan execution and plan monitoring are essential parts of Continual Planning algorithms. Therefore continual planning must, in contrast to classical planning, be implemented and tested in some sort of “reality”. Providing such a reality, e. g. a simulation, is not a trivial task. Being able to provide realities in a domain-independent way, such that continual planning can be evaluated for arbitrary planning domains, is even harder. This might be a reason for the comparatively few systematic approaches to continual planning in the literature (cf. Sect. 6).

For evaluating CCP across varied MAPL domains and scenarios, we have developed a simulation testbed for multiagent planning, called MAPSIM. MAPSIM is a *simulation generator* that automatically transforms MAPL domains into multiagent simulations. MAPSIM parses and analyses a MAPL domain and turns it into perception, action, and communication models for CCP agents. During the simulation, MAPSIM maintains and updates the global world state and it uses the sensor models to compute individual and joint perceptions of agents. In other words, MAPSIM interprets the planning domain as an *executable model* of the environment. Thus, MAPSIM allows designers of DCP algorithms to evaluate their approaches on various domains with minimal effort.

Agents interact with MAPSIM by sending *commands* that directly derived from MAPL actions selected for execution during CCP. The simulator then executes the action, i. e. it checks the preconditions and applies effects as specified in the MAPL domain. If the controlling agent of a command is not identical to the agent who sent it to the simulator this is interpreted as a *request* which, of course, is not directly executed but passed on to the corresponding agent. MAPSIM also accepts some specific commands for acknowledging subgoal acceptance and subgoal achievement. When a command corresponds to a speech

act, it is “heard” by the addressees, i. e. its effects are added to the beliefs of the addressees. In short, MAPSIM allows agents to directly implement Alg. 4 without having to care about the internals of the “world” they are acting in. However, MAPSIM does not enforce the use of CCP. Agents can use arbitrary deliberative or reactive methods to determine their behaviour and their reactions to requests. We believe that this can make MAPSIM a valuable evaluation tool even when a (distributed) continual planning algorithm is used that differs significantly from CCP.

It is also possible to extend the auto-generated simulation with (parts of) a more detailed world model, i. e. to specify additional action effects that are not (intended to be part) of the MAPL domain as the agents know it. This can be used to model *non-deterministic* (at least to the agents) events or special action effects, e.g. for verbalising speech acts or extending an agent’s goals or planning operators (see Sect. 5).

Interaction is very common during CCP runs in MAPSIM: agents plan and execute speech acts in order to gather information, negotiate requests and ensure self-synchronised plan execution by informing each other about state changes. Thus, CCP can be regarded not only as multiagent planning approach, but also as a model of collaborative, situated *dialogue*. Figure 6 shows parts of one such interaction in a scenario where one agent, Anne, wants another, R2D2, to bring her coffee. Since it is hard to study dialogue behaviour in different variants of CCP based only on the logging format shown in Fig. 6, we have extended MAPSIM with a verbalisation module, called the *reporter*. The reporter observes all physical and communicative events in the simulation and verbalises them in English. All dialogues shown in the remainder of the paper are unaltered outputs of this module. Figure 7 shows the beginning of the MAPSIM run of Fig. 6 using the *reporter*.

The reporter is a simple template engine that first determines an appropriate pattern depending on the command type currently executed, then recursively replaces templates with concrete arguments until a template-free sentence is generated. Base values for arguments are generated directly from analysing the MAPL domain. For example, operator names are assumed to directly correspond to verbs. Standard templates can be overridden by domain-specific patterns, but, surprisingly, this is often not even necessary to generate fairly natural English phrases. While, compared to “real” natural-language processing systems, this is a simplistic approach with obvious limits, the minimal effort needed to give MAPSIM basic

Fig. 6 A MAPSIM interaction in the *Household* domain

MAPSIM run starts. There are 2 agents: Anne and R2D2.

- (1) Anne: request R2D2 'give R2D2 coffee Anne'.
- (2) R2D2: accept request 'give R2D2 coffee Anne'.
- (3) R2D2: request Anne 'tell val Anne R2D2 pos(coffee)'.
- (4) Anne: execute 'tell val Anne R2D2 pos(coffee)'.
- (5) R2D2: ack achieved 'tell val Anne R2D2 pos(coffee)'.
- (6) ...

Fig. 7 Verbalisation of the interaction of Fig. 6 by the MAPSIM *reporter* agent

MAPSIM run starts. There are 2 agents: Anne and R2D2.

- (1) Anne: "Please bring me the coffee, R2D2."
- (2) R2D2: "Okay."
- (3) R2D2: "Where is the coffee, Anne?"
- (4) Anne: "The coffee is in the kitchen."
- (5) R2D2: "Thanks, Anne."
- (6) ...

linguistic capabilities is a noteworthy indication of the similarity between the MAPL representation and language.

Currently, dialogues are only *verbalised* by the reporter; the “real” communication between agents is performed as direct update of mental states as described by the MAPL speech acts. See Sect. 7 for a discussion of how agents can be enabled to communicate completely in a natural language, thereby enabling human-in-the-loop collaboration and dialogues.

MAPSIM is implemented in Python. The generic planner currently used by the CCP agents is a modified version of Axioms-FF [51] that prevents the use of assertions enabled in the current state, as required by Def. 14 and allows to express belief introspection with derived predicates. To enable the use of a classical PDDL planner like FF, MAPL is first compiled to PDDL: induced state variables are explicitly generated; speech acts that use state variables as parameters in MAPL are translated to a specific PDDL action for each such state variable; derived predicates for introspection of belief state variables are added to the PDDL domain; and MVSVs are translated to PDDL propositions.

5 Evaluation

In this section, we will present several rather different applications of continual planning. Section 5.1 presents some empirical data showing the effectiveness of our approach in a prototypical highly-dynamic multiagent system. Section 5.2 shows an example of how continual planning agents may deliberately extend not only their knowledge about the current state, but even their capabilities, i.e. their planning operators, and other parts of their planning *domain*. Finally, in Sect. 5.3 we discuss continual collaborative planning as a natural model of human interaction and show how, with minimal effort, MAPSIM agents can engage in natural-language *dialogues*.

5.1 Empirical evaluation

For this proof-of-concept study, we wanted a domain that is fairly prototypical for dynamical multiagent environments. Concretely, we needed a domain

- in which multiple agents can act autonomously, but
- where there is potential for action conflicts between agents, and
- where sensing can be limited. Finally,
- all of the above should be *scalable*.

We chose a simple grid world domain where agents must find their way to a goal position. Figure 8 presents an example of such a problem: four agents need to get to their respective goal positions, they must avoid the obstacles and react to the movements of other agents.

It is obvious that domains as this one are best reasoned about with specialised algorithms, not with a continual planning framework as generic as ours. However, the grid world provides us with an environment where parameters (the number of agents or obstacles, the sensing or communication range, etc.) can easily be scaled independently of each other to provide arbitrarily complex scenarios. More importantly, we believe that all these scenarios will still retain an important property that seems to be characteristic of many multiagent environments: the *situatedness* of agents. Intuitively, being situated means that each agent has a “vicinity” to which its immediate activities (sensing, communication, physical actions) are limited. Conflicts between agents can only arise where their respective situations overlap somehow—but this often also means that these agents will become copresent and able to

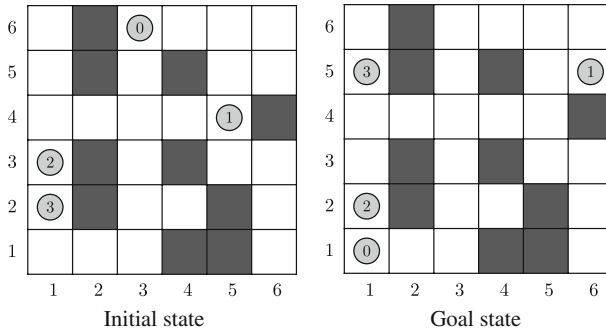


Fig. 8 A multiagent planning problem in the grid world

communicate. Thus, at the point where agents *need* to coordinate their actions, they will be able to do so (either explicitly through communication or implicitly by observing each other). In such environments, continual planning seems to be the approach of choice: despite the high dynamics of the MAS as a whole, agents can safely restrict detailed planning to those parts of their problem that are likely to cause problems immediately if not planned carefully, but which are also best known, because they can be perceived currently.

To investigate CP in the grid domain we have created a test suite of 50 random problems varying in the number of agents (2–10), the grid size and the number of blocked cells. Figure 8 shows one such problem from an omniscient perspective, i. e. the initial and goal position for all agents are shown in the same grid. Each agent, implemented as an individual MAPSIM thread, knows only about its own goal state and its own perceptions. For a centralised planner with complete knowledge, each problem would be solvable, i. e. there is a sequential or asynchronous plan leading each agent into its goal position.

The purpose of our experiments was to study the success rate of agents planning and acting distributedly under different configurations. We have run the simulation for each problem under several such configurations that vary and combine the following parameters:

- the sensor range of each agent
- the memory duration, i. e. the time for which perceived facts are believed to still be true in the world before they are declared unknown again
- allowing agents to request movements from others

The only domain-level action that agents can perform is *move*, shown below, describing that agents can move to a connected grid cell if is free. MAPSIM will determine whether this precondition is satisfied at runtime. In particular, if several agents want to access the same grid cell at the same moment, one is selected randomly to be successful whereas the other actions are simply not executed. This can lead to all kinds of interesting behaviours, e.g. agents getting stuck permanently behind each other or looping behaviour among several agents.

```
(:action move
:agent (?a - agent)
:parameters (?c - gridcell)
:variables (?ca - gridcell)
:precondition (and
  (occupant ?ca : ?a)
```

```

    (occupant ?c : empty)
    (connected ?c ?ca))
:effect (and
    (occupant ?c : ?a)
    (occupant ?ca : empty))
)

```

Depending on the sensing distance chosen for each experiment, a binary relation *in-sensing-distance* among grid cells is defined which is used in the agents' only sensor model, as shown below. It describes that the occupant of every grid cell in sensing distance of the agent will be perceived by the agent.

```

(:sensor sense-gridcell
:agent (?a - agent)
:parameters (?c - gridcell)
:variables (?ca - gridcell)
:precondition (and
    (occupant ?ca : ?a)
    (in-sensing-distance ?ca ?c))
:sense (occupant ?c)
)

```

Whenever sensing is limited in the grid agents must resort to Continual Planning with assertions. The planning domain contains the single assertion shown below which asserts that an agent will be able to find a plan to reach a grid cell once it has reached a position in sensing distance. Notice that the difference to the standard *move* action is subtle: instead of having to know that a grid cell is empty the agent only has to know *whether* it is empty

```

(:action move_A
:agent (?_pa - planning_agent)
:parameters (?a - agent ?c - gridcell)
:variables (?ca - gridcell)
:precondition (and
    (occupant ?ca : ?a))
:replan
    (KIF ?_pa (occupant ?c))
:effect (and
    (occupant ?c : ?a)
    (occupant ?ca : empty))
)

```

The experiments were run on a 1.8 GHz Intel Pentium with 1 GB RAM. If after at most 10 min all agents had achieved their goals, the run was counted as successful, otherwise as a failure. Since each run consisted of many planner calls by several agents, individual planner calls were timed out after 10 s.⁶

⁶ It is instructive to compare these time constraints to the time limits used in the 2008 International Planning Competition (<http://ipc.informatik.uni-freiburg.de>). There a single planner run is allowed to take up to 30 min! While of course this gives no indication about actual planner runtimes, it nevertheless shows a research focus on particularly hard problems that is quite different from the need for real-time planning in realistic dynamic environments that motivates our continual planning approach.

Agents with better sensing capabilities, i. e. higher sensor range, should have a clear advantage over ones with weaker sensors. In particular, the pro-active CP agents should face dead ends more often which would force them to replan more often. Additionally, sooner or later, assertions used in the initial plans must be expanded, i. e. repeated replanning is enforced by the CP algorithm. However, in domains with such high dynamics as the grid world, better sensing is not always helpful: Agents who can perceive the whole grid, i. e. are omniscient at all times, realize immediately when their fully realized plan is no longer valid. Since this is often the case when other agents move through the anticipated path of an agent, the number of replannings in the omniscient settings is almost as high as in the CP settings. In other words, ignorance can indeed be indeed bliss!

This is also true for the amount of time agents spend on planning. The average planning time, i. e. the average time spent in the PLANNER subroutines, is much higher for the configurations with full observability (often more than twice as high). This is of course due to the use of assertions which hide subplans that must be explicitly searched for in the full observability configurations. Obviously, CP agents solve a different planning problem or, to put it differently, are given important additional information about the environment they act in (in form of assertions). Comparing the different planning times therefore is of no value algorithmically. Nevertheless, the increase in speed shows that assertions cannot only compensate for the limited perceptions of agents, but can also provide helpful search guidance for a CP algorithm.

It would be a natural to assume that fully observant agents find better, even optimal, paths through the grid. The data shows this is not the case. This can be partly explained by the fact that fully observant agents recognise other agents blocking the shortest paths very early and plan *detours* around them. The “safe” plans thus generated often remain valid throughout the simulation. Thus, if other agents move as well and a better path would be possible, the agent does not realize this.⁷

The results discussed so far suggest that CP with assertions is suitable for MA environments with limited sensing and high dynamics. The following analysis gives further indications about *how* it should be used in such environments. The experiment relates the perceptive and mnemonic powers of agents, i. e. in our domain their sensor range and their memory duration, i. e. the time for which perceived facts are believed to still be true in the world before they are declared unknown again. The reason for having memories have limited duration at all is the following: when sensing is limited, agents have to rely on their memory for things they cannot currently see. However, in environments with high dynamics old beliefs may have become obsolete by the time they are used by the planner. Since an agent cannot know which beliefs have become obsolete it would be best to attribute them confidence values. However, in our non-probabilistic planning setting, we can only choose to simply *forget* them after some time.

Our assumption was that forgetting would give the CP algorithms the chance to use assertions again: replanning conditions that might have been true in an agent’s knowledge base and thus being prevented from being used in a plan (cf. Def. 14) would become usable again. The newly re-enabled assertion would then lead the agent to re-investigate areas of the environment that were promising for achieving his goals, but had been unreachable before.

To test this assumption, we compared MAPSIM runs on a 10×10 grid, varying three different memory durations (M_{perm} is permanent memory, M_5 is a memory duration of 5

⁷ The only way to detect this kind of serendipity would be to replan after each new perception. However, this would turn the agent’s behaviour even more reactive than the one of the CP agent. In particular, the number of replannings would be equal to the number of actions executed whereas the agents in our experiments have to replan only in every second or third step – despite the high dynamics of their world.

Table 1 Memory span versus sensor range

	M_0	M_5	M_{perm}
S_1	37	62	54
S_2	84	88	83
S_5	96	99	90
S_{10}	<i>100</i>	–	–

Entries show success rate in %.
Italic values indicate the dominating configuration

cycles, and M_0 means that only the current perceptions are taken into account) to four different sensor ranges (1, 2, 5, and 10, i.e. all, grid cells). The baseline for our test was the behaviour of agents with full observability. For these agents remembering old perceptions is unnecessary; we have therefore restricted the test to S_{10}/M_0 there. Since we were not interested in other configuration aspects in this experiment, we chose only scenarios where this configuration led to all agents achieving their goals (i.e. we ignored scenarios that led to looping behaviour or stalls even among fully observant agents). The success rates for all configurations are shown in Table 1.

The main result of this experiment is that controlled forgetting (M_5) is the best choice in all scenarios with limited sensing. Relying on possibly obsolete memory is often even worse than not using past perceptions at all. This confirms our hypothesis that it's better to resort to active information gathering (induced by assertions with KIF replanning or preconditions) than to stick to obsolete beliefs. It is also encouraging that Assertional Planning with heavily restricted sensing and memory duration (M_5/S_5) leads to a performance that is almost as good as under full observability.

5.2 Modifying domain knowledge during continual planning

As already mentioned, continual panning with assertions cannot only be used for classical knowledge gathering behaviour, but also lends itself to changing other aspects of the planning problem. In this section, we discuss a worked example about how a planning agent can use it to actively extend its planning *domain* in the middle of the continual planning process. Thus, the notion of active information gathering is extended to forms of knowledge that is usually considered pre-defined and unchangeable in planning research.

Consider the example of a household robot who was just bought and is now told by his owner, for the first time, to clean the dishes. The robot knows that there is a dishwasher in the kitchen, but does not know of which brand. Hence, the robot does not have exact information about how to operate this specific dishwasher. However, the robot's planning domain includes the following assertion:

```
(:action operate_dishwasher
:agent (?robot - agent)
:parameters (?dw - dishwasher)
:variables (?r - room)
:precondition (and
  (pos ?robot : ?r)
  (pos ?dw : ?r))
:replan (canOperate ?robot ?dw)
:effect (dishesClean))
```

As can be seen, this assertion describes only very general knowledge about the usefulness of dishwashers, namely that they can get dishes clean when one stands next to them and knows how to operate them. Like all assertions, this one will be used to postpone planning for subproblems due to a lack of knowledge. Here, however, the knowledge missing is *procedural*, i.e. it is knowledge about *how to bring about* intended effects. Most importantly, the planning domain used by the robot does not contain any concrete actions that may achieve the effect *dishesClean*. How then, with assertions not being executable, can the robot ever achieve its goal?

The initial plan generated by the robot looks as follows:

```
move livingroom kitchen
senseBrandOf dishwasher
downloadManual dishwasher
operateDishwasher
```

In this plan, *operateDishwasher* is enabled by action *downloadManual dishwasher*. The corresponding MAPL operator is defined as follows.

```
(:action downloadManual
:agent (?robot - agent)
:parameters (?t - tool)
:precondition (K ?robot (brandOf ?t))
:effect (canOperate ?robot ?t))
```

Note that the effect specified provides only a crude approximation of what will really happen during the execution of *downloadManual dishwasher*: the robot will contact a web service provided by the company producing the tool in question and download the manual from there. As a result, its planning domain will be extended with operators describing how to operate this particular dishwasher.

It is crucial that (*canOperate robot dishwasher*) is not an ordinary precondition of *operateDishwasher*, but a replanning condition. This will force the robot to actually execute *downloadManual* before even thinking in detail about how to operate the dishwasher. As soon as *downloadManual* has been performed, though, replanning is triggered because the replanning condition is now satisfied. During this planning phase, the continual planning algorithm will be able to use its newly acquired operators to find a concrete plan for actually operating the dishwasher successfully. The robot has *learned* new capabilities *on the fly* and used them immediately.

There are several things to note about this example. It shows how changes in a planning domain can occur during continual planning. Even more importantly, the changes can be provoked by the agent itself in a *goal directed* manner, i.e. the agent *actively* extends its procedural knowledge in order to be able to solve the problem at hand. This is reminiscent of the notion of *goal-driven learning* [45].

The example also shows how assertions can be used as modelling tool. In particular, they permit building *lightweight* planning domains (a similar concept has recently been advocated by Kambhampati as model-lite planning [32]). In the beginning such a domain will contain only few real actions, but mostly assertions which abstractly describe the capabilities of agents, but which can be extended “on demand”. This modelling approach is interesting

- whenever the set of operators is very large in principle, but small in practice for any given problem (adding all possible operators would unnecessarily blow up the size of the planning domain in this case)

- when, in contrast, some concrete operators are not yet available when the domain is designed (e. g. operators for a new dishwasher models are added to the database later)
- when operator description themselves are due to change, e. g. because the agent is in the process of *learning* them

The idea of extending lightweight domain models on the fly can also be applied to other forms of knowledge. For example, an agent may extend its *ontology* of object classes during continual planning. The next section discusses how agents' *goals* can be updated during distributed planning.

5.3 Situated dialogue as continual collaborative planning

In this section, we will explain the use of CCP for scenarios that interleave planning, acting and sensing as well as action and interaction between multiple agents, i. e. for scenarios where agents engage in *situated dialogue* [9, 34].

Figures 9, 10, and 11 show situated dialogues between MAPSIM agents in different MAPL domains, generated using the CCP algorithm and automatically verbalised by the MAPSIM *reporter*. This shows how, due to the domain independence of both the CCP algorithm and the MAPSIM implementation, it is easily possible to evaluate different CCP variants and dialogue strategies across different domains and scenarios.

It is important to realize that none of the sample runs shows the execution of a single multiagent plan, but a *series* of plans, devised, partly executed and revised several times according to Alg. 4.

In Fig. 9 the necessity for collaboration stems from the fact that only MrChips can move to the kitchen to get coffee, but only MrData can open the kitchen door. At the beginning, both agents have different goals: MrData wants to have coffee and MrChips wants to be at MrData's service. Basically, this is a master-slave scenario; however, since both agents have individual goals, incidentally it is MrChips that takes the initiative. MrChips' original goal is very simple: he wants to achieve (*has-goal MrChips*). This goal is directly achieved by steps (1) and (2) of the dialogue. However, step (2) already is a request by MrData that

```

MAPSIM run starts. There are 2 agents: MrChips and MrData.
(1) MrChips: "What can I do for you, MrData?"
(2) MrData: "Please bring me the coffee, MrChips!"
(3) MrChips: "Where is the coffee, MrData?"
(4) MrData: "The coffee is in the kitchen, MrChips!"
(5) MrChips: "Please open the kitchen door, MrData!"
(6) MrData opens the kitchen door.
(7) MrChips moves to the kitchen.
(8) MrChips takes the coffee.
(9) MrChips moves to the livingroom.
(10) MrChips: "I have the coffee, MrData!"
(11) MrChips: "Please take the coffee, MrData!"
(12) MrData takes the coffee.
(13) MrData: "Thanks for the coffee, MrChips!"
MAPSIM terminates successfully.

```

Fig. 9 Mixed-initiative dialogue between two artificial agents in the MAPSIM (*Household* domain)

Fig. 10 Long-term request in the *object manipulation domain*

- (1) Boss: "Please put obj1 on obj2, Robot."
- (2) Robot picks up obj1.
- (3) Robot puts obj1 on obj2.

- (1) Ambulance: "Please extinguish house1, Firebrigade."
- (2) Firebrigade: "Okay."
- (3) Firebrigade refills watertank.
- (4) Firebrigade extinguishes house1.
- (5) Ambulance: "Thanks for extinguishing house1, Firebrigade."

Fig. 11 Acknowledgements for subgoal adoption and achievement

provides MrChips with a new temporary subgoal that he will try to achieve during the rest of the dialogue.

To see why and how MrData generated this request, consider MrData's individual planning process: knowing that the coffee is in the kitchen, he can generate a plan that involves MrChips going to the kitchen and bringing back the coffee. Algorithm 4 uses the function `SELECTBESTREQUEST` to determine the appropriate subgoal that MrData will request MrChips to achieve. Our standard implementation uses a simple algorithm `FIND INDIVIDUAL SUBPLAN` (omitted from the paper) that determines the largest subplan of P that uses only *one* specific agent. Then `SELECTBESTREQUEST` chooses an action on the final level of this plan as the best request. In our example, MrChips can directly ask for the last action in his plan, namely MrData giving him the coffee.

The strategy of requesting long-term effects rather than immediately possible actions is also exemplified in Fig. 10. Its main advantage is that it abstracts from the level of detail on which the individual agents can plan (due to the differences in their knowledge). In Fig. 10, the "Boss" agent may not know whether the preconditions for the "put" action are already satisfied, i. e. his own plan may have included knowledge-gathering actions and assertions. However, by asking for the expected result, he gives the robot the opportunity to find its own appropriate solution.

In the *household* scenario, MrChips adopts the new goal to provide MrData with coffee (according to step 3 of Alg. 4). However, since he does not know where the coffee is his next plan must resort to a fairly abstract assertion, (*fetch-A MrChips coffee*), that completely hides the position of the target object as well as the possible complex planning necessary to reach that position. Although very abstract, the assertion guides MrChips' planning by means of its *replanning condition* ($KIF\ MrChips\ (pos\ coffee)$) which, in words, says that in order to fetch the coffee he must at least find out where it is. This leads to the following multiagen plan by MrChips. Here, MrChips takes the initiative again by asking MrData about where the coffee is:

```
MrChips: request MrData 'tell_val MrData MrChips pos(coffee)'
MrData: execute 'tell_val MrData MrChips pos(coffee)'
MrChips: execute 'fetch-A MrChips coffee'
MrChips: execute 'give MrChips MrData coffee'
```

Algorithm 2 declares MrChips' plan as valid, so Alg. 3 executes the first action: MrChips requests MrData to tell him the position of the coffee. The domain-specific verbalisation template maps a request for telling the value of state variable pos to the wh-question "where?" as shown in step 3 of the Fig. 9.

Note that in the further course of the dialogue both agents switch seamlessly between communicative and physical actions. However, unnecessary verbalisation of action effects is avoided, because both agents also reason about their mutual perceptions. For example, MrData does not verbalise having opened the door, because in his plan he can apply the sensor model of Fig. 2 for MrChips and thus deduce that he will perceive the opening of the door himself. In this manner, CCP both enforces knowledge preconditions, but also avoids unnecessary communication about them. In many applications, however, communication is necessary for *grounding* the collaborative process or simply acknowledging understanding [52]. Figure 11 shows an example (from a Search and Rescue planning domain) where CCP was run with enforced acknowledgements upon subgoal adoption and achievement. Especially in those Collaborative Planning environments that include both artificial and human agents such acknowledgements are crucial.

6 Related work

This work integrates ideas from several subfields of AI, in particular Classical and Distributed Planning, Multiagent Systems, Epistemic Logic, and Reasoning about Actions and Change.

Our asynchronous plans are similar to Boutilier and Brafman's [5] multi-actuator plans. They model interacting effects of concurrent actions by specific kinds of conditional effects of the individual agents. A plan must provide simultaneity constraints ensuring that the interaction really takes place as planned. The authors assume that an external synchronisation mechanism will ensure that during execution the constraints are met by the agents. Cox and Durfee's [13] and Clement and Barrett's [11] coordination algorithms provide such mechanisms. Our approach, however, rests on the assumption that executing agents are truly autonomous and there is no external instance to synchronise them. Therefore it must allow agents to synchronise plan execution on their own. This is achieved by explicitly including the knowledge and perceptions necessary for coordinated execution into the plans of agents. Since only plans that include these information are valid multiagent plans, the planning algorithm itself can (and is forced to) ensure synchronised execution. It is worth studying, however, how special-purpose coordination algorithms like those of Cox or Clement could be integrated with our Continual Planning approach, thereby potentially simplifying the actual planning process.

Continual Planning is often advocated as a practical approach to planning in dynamic or incompletely known domains [48]. In practice, this often amounts to not more than repeatedly switching between planning and execution. Previous work that more tightly integrates planning, monitoring, execution and information gathering includes [1,20,21,25,33]. Similarly to our work, these approaches explicitly model knowledge and knowledge-gathering actions. Instead of the concept of assertions which enables us to postpone parts of the planning process, yet reason about its outcomes, these approaches use runtime variables to represent unknown sensing results. Runtime values can be used as action parameters in the remainder of plan and thus allow for reasoning about unknown future knowledge, although this reasoning is heavily limited because nothing is known about the variable beside the fact that has been sensed. Because of the limitations of runtime variables, MAPL does not yet support them. Instead, our use of MVSVs enables a planner to non-deterministically *guess* one of the possible value of the MVSVs domain if this is desired by the domain modeller.

Planning with sensing actions has often been described in the planning literature [26, 35,43,44,53]. To our knowledge, none of these models extends planning for sensing to the

philosophical concept of *copresence* [36]. Copresence has been much discussed in literature on pragmatics, e.g. by Clark and Marshall [10]. To our knowledge, ours is the first work that allows agents to explicitly reason about copresence in order to self-coordinate multiagent plan execution without explicit communication.

The explicit inclusion of beliefs and mutual beliefs in our planning approach follows BDI models of multiagent planning, e.g. the SharedPlans model of Grosz and Kraus [27] that describes the role of (mutual) beliefs as necessary conditions for planful MA behaviour. Our formalism and implementation does not cover all aspects of these models (yet); in particular, we do not model *intentions* explicitly (yet). However, by explicitly modelling perception and copresence our approach complements existing BDI approaches to MA plans, since it can explain how knowledge conditions for joint behaviour can be achieved during plan execution.

The need for Distributed Continual Planning (DCP), i. e. Continual Planning in multiagent settings, was pronounced clearly by desJardins and colleagues in [14]. Most work within this field is based on hierarchical representations of multiagent plans [12, 15–17, 41]. Indeed, the expansion of assertions is similar to the decomposition of HTN schemata [19, 42, 54]. In our approach, however, the abstraction hierarchy need not be explicitly given by the domain designer, but is resolved by the planner itself. Also the purpose of the abstraction is different from HTN planning: while HTN decompositions embody knowledge about how to solve subtasks, assertions essentially represent a way to postpone parts of the planning process. Thus Continual Planning with assertions produces a *series* of *non-hierarchical* plans, whereas HTN produces one abstraction hierarchy. Note also that while it has been proposed in textbooks that HTN planners may leave parts of the plan hierarchy unexpanded until a plan has been partially executed, we are not aware of work that describes how exactly an HTN planner should make such decisions.

Our CCP approach to *dialogue* is close in spirit to existing frameworks for collaborative dialogue, in particular the one of Lochbaum [38]. In contrast to other approaches to dialogue planning that use formal state descriptions mainly for the specification and validation of the computational approaches, e. g. [4, 46, 52], we directly reason on the formal logical representations of the agents' beliefs (i. e. the MAPL states). In this respect, our work mostly resembles Sadek's [49] approach to dialogue planning. Due to our use of a general-purpose planning method, other approaches can deal with more elaborate linguistic phenomena. However, CCP seems to be able to explain pragmatic aspects of a dialogue better (or at least more naturally) than other approaches, because of the inherently causal reasoning underlying all dialogue. In particular, CCP is suited for *situated* dialogue planning, because the generated dialogues directly depend on the agent's knowledge about the current situation and its physical actions in the world.

7 Summary and discussion

Acting deliberately is hard in realistic dynamic environments that cognitive agents can only partially observe and influence. For all practical purposes, deliberation must be combined with reactive behaviour that takes newly perceived changes into account quickly. Often, however, dynamic environments demand an even more *proactive* behaviour: agents must actively try to gather new or reconfirm outdated information before they can determine appropriate solutions to their problems. To that end, they must be able to reason about how and when to extend or update their knowledge, and then plan their own cycle of planning, acting and replanning.

In this paper, we have presented a new principled approach to this kind of Continual Planning. Based on the novel concept of *assertions* agents can decide autonomously which parts of of planning problem they can already solve in detail and for which they must gather additional information. Decision cannot be postponed arbitrarily: the semantics of assertions enforces the achievements of their so-called replanning conditions in a plan, but prohibits their use as soon as these are actually satisfied. Thus, whether an assertion may (still) be used in a plan is highly *context-dependent*. Since this context is given by the agent's previous knowledge and expectations about the world, but also includes its current perception, Continual Planning with assertions naturally leads to the combination of proactive, reactive and deliberative behaviour that seems to be crucial in dynamic agent environments.

It is evident that, by postponing solutions to subproblems, agents run the risk of getting caught in dead ends—both metaphorically in their planning search space, but also in very concrete ones in the real world. However, in dynamic, partially observable domains, the alternative often is not being able to act at all, for lack of information and because of the computational intractability of contingent planning. As a solution to this dilemma, our approach offers domain modellers the possibility to express their knowledge about the achievability of subgoals by means of assertions, thereby enabling agents to behave more proactively than in purely deliberative planning approaches, yet only where deemed appropriate by the domain modeller and when more detailed information is still missing. Because of the strict semantics of assertions, domain modellers need not worry about inappropriately “oversimplifying” the planning problem by having too many or too abstract a set of assertions: the Continual Planning algorithm will only postpone subproblems when and as long as there is information missing for their solution. In other words, it automatically chooses the correct level of abstraction depending on the current situation and makes sure that its plan will get more and more concrete during the planning-execution cycle. In the end, agents can only achieve their goals by executing ordinary domain actions.

As already said, the approach to Continual Planning presented in this paper will lead to agent behaviour that combines deliberation, proactivity, and reactivity. This form of behaviour is particularly helpful in multiagent environments, especially in the form of active information gathering by means of communication. The formal planning language MAPL presented in this paper was chiefly designed to support planning and acting in such environments. In particular, communicative actions are regarded, both in the formalism and in the planning algorithms, as just a special case of standard action planning. In other words, communication planning happens on the *pragmatic* level, i. e. a communicative action describes how to achieve communicative goals without specifying their linguistic realization. As a result, the Collaborative Continual Planning algorithm presented is able to seamlessly integrate action and communication planning. This opens up many possibilities for future research on *dialogue* as a form of Collaborative Continual Planning. In this paper, we have briefly exemplified the potential of this idea in Sect. 5.3, showing how fairly natural dialogue can arise from CCP using even very simplistic rules for the verbalisation of communicative actions. For future work on dialogue, we are currently implementing algorithms for generating and interpreting referring expressions. This will allow artificial agents to interact with *humans* in domain-specific restricted natural language.

As discussed in Sect. 5.2, our notion of “information gathering” transcends the usual meaning of the term in AI Planning: during the Continual Planning process, agents can plan a change not only to their factual knowledge, but also to their goals or their procedural knowledge, i. e. their planning operators. This opens up intriguing new possibilities for the design of planning agents: instead of being provided with huge planning domains containing a large

set of operators, the agents can start with a more “lightweight” limited set of executable operators, extended with assertions representing additional procedural knowledge they may actively “learn” if necessary. The agent will then determine themselves, by means of the Continual Planning process, which additional competences they will need for the specific problems at hand.

In this paper, our new approach to Continual Planning has been presented in theory and via results using the simulation testbed MAPSIM. However, it is also being integrated into a real robot system where it is used not only for continual action planning, but also, e.g., for disambiguation of natural language utterances and predicting behaviour of other agents [8]. By extending the robot implementation to use the Collaborative Continual Planning algorithm, we ultimately hope to achieve mixed-initiative human–robot interaction and cooperation.

Acknowledgements This work has been supported by the EC under contract number FP6-004250-CoSy and by the German Federal Ministry of Education and Research (BMBF) under Grant 01IME01-ALU (DESIRE). We thank the anonymous reviewer for detailed and helpful comments.

References

1. Ambros-Ingerson, J. A., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of AAAI-88* (pp. 83–88), Saint Paul, MI.
2. Bäckström, C., & Nebel, B. (1995). Complexity results for SAS⁺ planning. *Computational Intelligence*, *11*, 625–655.
3. Bäckström, C. (1998). Computational aspects of reordering plans. *JAIR*, *9*, 99–137.
4. Blaylock, N., Allen, J., & Ferguson, G. (2003). Managing communicative intentions with collaborative problem solving. In *Current and new directions in dialogue*. Kluwer.
5. Boutilier, C., & Brafman, R. (2001). Partial order planning with concurrent interacting actions. *JAIR*, *14*, 105–136.
6. Brenner, M. (2005). Planning for multiagent environments: From individual perceptions to coordinated execution. In *Wsh. Multiagent Planning and Scheduling, ICAPS '05*, Monterey, USA.
7. Brenner, M. (2007). Situation-aware interpretation, planning and execution of user commands by autonomous robots. In *Proceedings of IEEE RO-MAN 2007*.
8. Brenner, M., Hawes, N. A., Kelleher, J. D., & Wyatt, J. (2007). Mediating between qualitative and quantitative representations for task-orientated human–robot interaction. In *Proceedings of IJCAI 2007*.
9. Brenner, M., & Kruijff-Korbayová, I. (2008). A continual multiagent planning approach to situated dialogue. In *Proceedings of the 12th workshop on the semantics and pragmatics of dialogue (semidl)*, London, UK.
10. Clark, H. H., & Marshall, C. R. (1981). Definite reference and mutual knowledge. In *Elements of discourse understanding*. Cambridge University Press.
11. Clement, B., & Barrett, A. (2003). Continual coordination through shared activities. In *Proceedings of AAMAS '03*.
12. Clement, B., & Durfee, E. (1999). Top-down search for coordinating the hierarchical plans of multiple agents. In *Proceedings of AGENTS '99*.
13. Cox, J. S., & Durfee, E. H. (2005). An efficient algorithm for multiagent plan coordination. In *Proceedings of AAMAS '05*.
14. DesJardins, M., Durfee, E., Ortiz, C., Jr., & Wolverton, M. (1999). A survey of research in distributed, continual planning. *The AI Magazine*, *20*(4), 13–22.
15. DesJardins, M., & Wolverton, M. (1999). Coordinating a distributed planning system. *The AI Magazine*, *20*(4).
16. Durfee, E., & Montgomery, T. (1991). Coordination as distributed search in hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(6), 1363–1378.
17. Durfee, E. H. (1999). Distributed continual planning for unmanned ground vehicle teams. *AI Magazine*, *20*(4), 55–61.
18. Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany.

19. Erol, K., Hendler, J., & Nau, D. (1996). Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence*, 18, 69–93.
20. Etzioni, O., Golden, K., & Weld, D. S. (1997). Sound and efficient closed-world reasoning for planning. *Artificial Intelligence*, 89(1–2), 113–148.
21. Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., & Williamson, M. (1992). An approach to planning with incomplete information. In *Proceedings of KR-92* (pp. 115–125).
22. Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning about knowledge*. MIT Press.
23. Fox, M., & Long, D. (2003). PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *JAIR*, 20, 61–124.
24. Geffner, H. (2000). Functional STRIPS: A more flexible language for planning and problem solving. In Minker, J. (Ed.), *Logic-based artificial intelligence*. Dordrecht, Holland: Kluwer.
25. Golden, K. (1998). Leap before you look: Information gathering in the puccini planner. In *Proceedings of AIPS-98* (pp. 70–77).
26. Golden, K., & Weld, D. (1996). Representing sensing actions: The middle ground revisited. In *Proceedings of KR '96*.
27. Grosz, B. J., & Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86.
28. Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26, 191–246.
29. Hobbs, J. R., & Moore, R. C. (Eds.). (1985). *Formal theories of the commonsense world*. Norwood, NJ: Ablex.
30. Hoffmann, J., & Brafman, R. (2005). Contingent planning via heuristic forward search with implicit belief states. In S. Biundo, K. L. Myers, & K. Rajan (Eds.), *ICAPS* (pp. 71–80). AAAI.
31. Hoffmann, J., & Brafman, R. I. (2006). Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170, 507–541.
32. Kambhampati, S. (2007). Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of AAAI-07*.
33. Knoblock, C. A. (1995). Planning, executing, sensing, and replanning for information gathering. In C. Mellish (Ed.), *Proceedings of the fourteenth international joint conference on artificial intelligence* (pp. 1686–1693). San Francisco: Morgan Kaufmann.
34. Kruijff, G.-J., & Brenner, M. (2007). Modelling spatio-temporal comprehension in situated human–robot dialogue as reasoning about intentions and plans. In *AAAI spring symposium on intentions in intelligent systems*, Stanford, CA.
35. Levesque, H. J. (1996). What is planning in the presence of sensing? In *Proceedings of AAAI-96* (pp. 1139–1146). MIT Press.
36. Lewis, D. (1969). *Convention. A philosophical study*. Cambridge, MA: Harvard University Press.
37. Littman, M., Goldsmith, J., & Mundhenk, M. (1998). The computational complexity of probabilistic planning. *JAIR*, 9, 1–36.
38. Lochbaum, K. E. (1998). A collaborative planning model of intentional structure. *Computational Linguistics*, 24, 525–572.
39. Lynch, N. (1996). *Distributed algorithms*. San Francisco, CA: Morgan Kaufmann.
40. McDermott, D. (1998). PDDL—the planning domain definition language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
41. Myers, K. L. (1999). Cpef: A continuous planning and execution framework. *The AI Magazine*, 20(4).
42. Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In T. Dean, (Ed.), *Proceedings of the 16th international joint conference on artificial intelligence (IJCAI-99)*. Stockholm, Sweden: Morgan Kaufmann.
43. Petrick, R., & Bacchus, F. (2002). A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS-02*.
44. Petrick, R. P. A., & Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of ICAPS 2004* (pp. 2–11).
45. Ram, A., & Leake, D. B. (Eds.). (1995). *Goal-driven learning*. Cambridge, MA: MIT Press.
46. Rich, C., & Sidner, C. L. (1998). Collagen: A collaboration manager for software interface agents. *UMUAI*, 8(3–4), 315–350.
47. Rintanen, J. (1999). Constructing conditional plans by a theorem-prover. *JAIR*, 10, 323–352.
48. Russell, S., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.), Englewood Cliffs, NJ: Prentice-Hall.
49. Sadek, M. D. (1991). Dialogue acts are rational plans. In *Proceedings of the ESCA/ETR workshop on multi-modal dialogue*. Italy: Maratea.
50. Smith, D. E., & Weld, D. S. (1998). Conformant graphplan. In *AAAI/IAAI* (pp. 889–896).

51. Thiebaux, S., Hoffmann, J., & Nebel, B. (2003). In defense of axioms in PDDL. In *Proceedings of IJCAI*.
52. Traum, D. (1994). *A computational model of grounding in natural language conversation*. PhD thesis, University of Rochester.
53. Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending graphplan to handle uncertainty and sensing actions. In *AAAI/IAAI* (pp. 897–904).
54. Yang, Q. (1997). *Intelligent planning: A decomposition and abstraction based approach*. Springer.
55. Yokoo, M., & Hirayama, K. (2000). Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2).