Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The actor's view of automated planning and acting: A position paper

Malik Ghallab[a], Dana Nau[b,*], Paolo Traverso[c]

[a] LAAS/CNRS, University of Toulouse, France
[b] Dept. of Computer Science and Inst. for Systems Research, University of Maryland, College Park, MD, USA
[c] FBK ICT IRST, Trento, Italy

ARTICLE INFO

Article history:
Received 20 June 2013
Received in revised form 31 October 2013
Accepted 1 November 2013
Available online 7 December 2013

Keywords:
Automated planning and acting

ABSTRACT

Planning is motivated by acting. Most of the existing work on automated planning underestimates the reasoning and deliberation needed for acting; it is instead biased towards path-finding methods in a compactly specified state-transition system. Researchers in this AI field have developed many *planners*, but very few *actors*. We believe this is one of the main causes of the relatively low deployment of automated planning applications.

In this paper, we advocate a change in focus to actors as the primary topic of investigation. Actors are not mere plan executors: they may use planning and other deliberation tools, before and during acting. This change in focus entails two interconnected principles: a *hierarchical structure* to integrate the actor's deliberation functions, and *continual online planning and reasoning* throughout the acting process. In the paper, we discuss open problems and research directions toward that objective in knowledge representations, model acquisition and verification, synthesis and refinement, monitoring, goal reasoning, and integration.

1. Introduction

Automated planning is a rich technical field. It has benefitted from the work of a very active and growing research community that has produced a number of already mature technologies. However, some areas in this field are over-explored, and there are other areas where further investigation is critically needed if automated planning is to have a wider impact on practical problems. One such area is the *integration of planning and acting*.

Planning is motivated by acting. It has been described as the "reasoning side of acting" [48]. However, most of the literature on domain-independent automated planning is biased toward part of that reasoning side: there are numerous *planners*, but very few *actors*.[1] The importance of the acting part has been largely underestimated, and insufficient attention has been given to the numerous AI problems that emerge when attempting to integrate acting with planning. Consequently, in contrast with successful AI fields such as machine learning, the deployment of automated planning techniques into fielded applications has remained relatively low despite its large potential.

Although this situation is unfortunate, there are some good reasons why it has occurred:

- Planning, informally, consists of choosing and organizing a set of actions for some purpose [100]. This requires an ability to predict what effects various actions will have in various states. However, making good predictions can be

* Corresponding author.
E-mail addresses: malik@laas.fr (M. Ghallab), nau@cs.umd.edu (D. Nau), traverso@fbk.eu (P. Traverso).

[1] We use the term "actor" instead of the related term "agent" to underline the acting functions and distinguish from the broader meaning of agents; the distinction is discussed in Section 2.

0004-3702/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.artint.2013.11.002

very difficult. The widely-used "preconditions-and-effects" representations[2] provide predictive models of actions that are often too abstract to be of great use to actors. The field of automated planning has focused more on improving the performance of search algorithms in this abstract space than on improving the predictive capabilities of its models.

- Planning is nicely formalized from an AI point of view, e.g., as a mapping from abstract domains and problems into plans, while acting is much harder to formalize.
- Planning is abstracted away from the intricacies of observing and changing the world. There is a clear frontier between *planning* and *executing*, i.e., execution functions that face no deliberation making problems. But the borderline between *acting* and *executing* is much more blurred. The distinction between reasoning functions and sensory-motor functions is more intricate for an actor than for a planner. It may even vary over time, e.g., with learning.
- Providing a plan as a service is in principle an easily packaged input–output function. But in open and dynamic environments, helping actors with their reasoning requires many different information gathering, processing and decision-making functions. The *closed-world* assumption can make sense for a planner, but much less so for an actor. These actor's deliberation functions are difficult to integrate.

One should certainly applaud the remarkable success of the search techniques developed in planning. They have led to a mature technology for solving path-finding problems in huge state transition systems, compactly specified by precondition-effect operators; a technology that has been applied to several areas apart from planning, such as genomics [37,55] and diagnosis [98,56,52]. But at the same time, numerous planning problems remain open that request more attention.

Many papers before this one have recognized and discussed the problems of integrating planning and acting [31,88,86,78]. These problems are particularly critical in areas such as robotics where they have been addressed through numerous publications [94,15,62,33,14,58,104]. The multi-agent community is also very active on the issues of integrating planning and acting [32,35,81,22]. However, these and similar contributions have not changed the main focus of the field. Numerous acting problems remain open, not addressed by the remarkable advances in planning and the significant scaling up of search techniques. We believe that the bottleneck is not anymore on search issues; it is time to change our focus.

We are advocating here to concentrate research efforts on *actors* as the primary topic of investigation in automated planning and acting, not on *planners* that actors may rely upon. Our focus is on acting from an AI point of view. We are not concerned here with the design of a variety of systems needed to exert and control forces on the environment in order to act (the *executor*'s functions). However, explicit models of these devices are often needed in the actor's deliberations. Our actor is a cognitive agent; we are interested in its deliberation capabilities.

A system that interacts with its environment through sensory-motor functions may or may not need deliberation. Tele-operated devices, automated vacuum cleaners, lawn mowers and similar systems that deal with a fixed environment and/or a single task do not strictly need deliberation. Automation engineers have deliberately (and wisely) engineered such problems out of the environment. In contrast, a system that performs a variety of tasks and interactions autonomously in a diversity of open environments must reason and make decisions about its activity. Deliberation is required for autonomy and diversity.

We are proposing to focus on the representations, modeling and reasoning techniques that a deliberate actor needs in order to perform various tasks. Our actor may use several planning and decision-making techniques as well as other computational tools, both before and while acting. Our proposed focus concerns automated planning integrated with these other reasoning tools: a focus that enriches the field with a number of research challenges and allows it to address significantly more relevant problems for real-world applications.

As an illustration, consider the reasoning activities of a traveler who is planning and making a trip, e.g., from Trento, Italy, to Canberra, Australia:

- At the travel-preparation stage, the traveler decomposes the trip into several steps: (i) planning the intercontinental flight to Sydney from a hub close to Trento, taking into account several criteria (cost, flight time, preferences, etc.); (ii) extending this plan to include the tasks of traveling from Trento (e.g., by car or by train) to a travel hub such as Verona or Milano, and traveling (e.g., by bus or by air) from Sydney to Canberra; and (iii) buying some of the tickets. Buying the tickets may seem like a routine activity, but it nonetheless refines into several steps that follow different procedures depending on the mode of transportation (bus, train, airplane) and the broker (travel agency, web service etc.). Several of these travel-preparation steps may require planning, possibly by some other actors than our traveller.
- At this point the trip preparation stops, although it is not complete. The plan is partial because the traveler's models are necessarily incomplete and many of the steps cannot afford or do not require early anticipation, e.g., where to have lunch in Milano airport before the flight, or how to go from the International Terminal in Sydney to take a bus to Canberra. The actor needs to be confident that the missing steps can be completed while traveling.
- Now the trip starts. Planned actions are refined into more concrete ones, e.g., "take the train" decomposes into buy the train ticket, find the platform, get to the platform, wait for the train, find the train car, get into the car, find the seat, etc. Other anticipated actions will be revised given the current context: e.g., skip lunch at Milano airport because there is no time. New plans will be generated using information available only while acting, e.g., what to do on a 12 hours

[2] Also called "STRIPS-style" or "classical" representations; see Section 3.1.1 for details.

flight will also depend on services proposed onboard. Finally, contingent events may break the plan: e.g., a glitch in Hong Kong delays the arrival in Sydney beyond the last connection to Canberra, forcing the actor to revise the plan.

This simple example shows that beyond the trip-preparation stage, the actor's reasoning goes through numerous interconnected steps, at different hierarchical levels. In order to be chosen and carried out, *all these steps require prediction*, but possibly with different representations and models. Some of these steps need refinement into lower level steps (e.g., take the train), possibly using different planning techniques, others are primitives that the actor manages without explicit reasoning (e.g., sitting once at the seat).

The focus on the actor as the primary investigation topic entails two interconnected principles:

- *Hierarchically organized deliberation*. Regardless of the actor's internal architecture, it will view, prepare, and carry out its actions hierarchically. Each action in a plan may be a task that may need further refinement and planning, which will usually be done online (see below) and may require different representations, tools, and techniques from the ones that generated the task. Such a hierarchical deliberation process differs significantly from hierarchical planning techniques (e.g., HTN planning) for reducing the search complexity of offline, non-heterogeneous plan generation.
- *Continual online planning and reasoning*. Only in exceptional circumstances will the actor do open-loop execution of plans synthesized offline. Instead, it will monitor, refine, extend, update, change, and repair its plans throughout the acting process. The actor will need to generate activities dynamically at run-time in order to carry out other higher-level activities that it is currently performing. Planning will need to be tuned for *plan repair*. An activity preparation stage will certainly be needed, but with the objective of allowing flexible repair. The actor's plans will remain partial as long as the cost of possible mistakes is lower than the cost of modeling, information gathering and thorough planning.

These arguments are developed in the rest of this paper which is organized as follows. Section 2 presents two detailed examples that are close to actual applications, and supports our plea by illustrating the actor's viewpoint and explaining the relevance of its perspective. Section 3 develops open problems from the actor's viewpoint in knowledge representation, model acquisition and verification, synthesis and refinement, and monitoring and hierarchical integration; and Section 4 concludes.

2. Examples

The two illustrative examples discussed in this section are quite different but they face similar open problems. Both are organized as hierarchies of components (see Figs. 1 and 2). In such a hierarchy, a component receives tasks from the component above it, and decides what activities need to be performed in order to perform those tasks. Performing a task may involve refining it into lower levels steps, issuing subtasks to other components below in the hierarchy, executing commands in the real world, and reporting to the component that issued the task. In general, tasks at different levels of the hierarchy may involve different state spaces and action spaces.

In order to carry out a task, a component may perform a variety of deliberation functions. These may include plan synthesis or revision, plan verification, scheduling, monitoring, diagnosis, information gathering and analysis, and so forth. Software packages are already available for some of these deliberation functions; additional ones are being developed. We will call these software packages *enablers*.

To clarify the following discussion, let us state informally our terminology:

- An *action*, conceptually, is a world-transformation step that can be used to perform a task. Since components at different levels may have different state spaces and action spaces, a primitive action at some level may be considered at lower level as a complex task to be further refined. In Fig. 1, for example, "Store" is an action at the highest level of the hierarchy, but at a lower level it is refined into several interacting processes.
- A *plan* is a collection of actions organized into a simple structure, such as a totally or partially ordered sequence, a policy, or a temporal network. Plans may be synthesized offline or online, or repaired online, by a planner from the specification of a planning domain and problem. Plans may even be written by hand, e.g., the six-step plan at the top of Fig. 1. In this paper, *planning* means synthesizing or repairing a plan; a *planner* is the software package (i.e., an enabler) that performs the synthesis or repair.
- A *skill* is also an organized collection of steps but it may have a more complex structure than a plan. A skill such as the navigation skill in Fig. 1 may involve sensing and actuating functions, loops and conditional steps, concurrent interacting processes, etc. Hence, a plan can be seen as a special case of a skill.
  An actor chooses and retrieves appropriate skills from a library that was developed offline; it instantiates and adapts them to the current context. The offline development of a library of skills may use any combination of specification and synthesis techniques, hand-programming, offline planning and learning.
- A *component* is an element in the hierarchical organization of an actor; some examples include each of the oval boxes in Fig. 1. A component uses enablers to perform its deliberation functions. In particular, it refines a task either through online planning or through the retrieval and adaptation of an available skill.

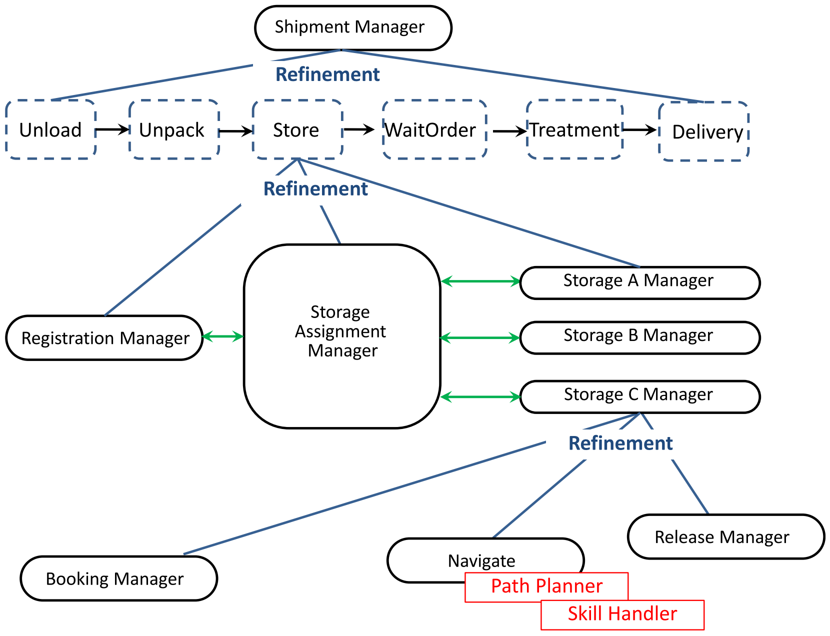

Fig. 1. Deliberation components for a Harbor Organization Management facility.

- An *actor* is the integrated hierarchy of deliberation components with their enablers: for example, Fig. 1 depicts a single actor. We use the term “actor” instead of “agent” to underline its acting functions and differentiate it from the broader meaning of agents.

2.1. The Harbor Operation Management facility

The example in this section is inspired by a Harbor Operation Management (HOM) facility for the sea port of Bremen, Germany, originally presented in [18,24]. The Bremen harbor is one of the most important European ports for the transportation of cars. It covers an area of about 400,000 m^2, with over 120,000 storage places and a dozen technical treatment stations, servicing 6 Million cars of 18 different brands every year.

The task of the HOM facility is to supervise and control all the different operations performed in the harbor: unload cars from ships, park them in storage areas, move them from one area to another or to a treatment area, perform the treatment (e.g., repair a damaged car), process retailers’ orders and prepare the delivery of cars according to the orders’ requirements, e.g., by loading them to trucks when they arrive at the harbor. Some of these operations are performed by humans workers, others automatically by machines. For instance, the moving of cars from a storage area to a treatment area can be performed by a truck manually driven, or by an Automated Ground Vehicle. Here, we are not concerned with the automated operations (see next section). HOM focuses on the management of the harbor, i.e., whether, when and how to perform a task, such as unloading cars from a ship, or moving cars from a storage area to a treatment area, when and how to repair a car if damaged, what to do if a selected storage area is not available, etc.

While this environment is rather structured, it has several characteristics that make the operation management complicated:

- It is *customizable*: e.g., the procedures of a delivery process can be customized according to the car brand, model, or retailer-specific requirements.
- It is *variable*: procedures for unloading/loading cars at different gates depend on the car brands; storage areas have different parking procedures, etc.
- It is *dynamic*: ships, cars, or trucks join the system dynamically, cars get moved to storage areas depending on space availability, treatment procedures before delivery depend on the order requirements that arrive dynamically.
- It is *unpredictable*, e.g., a car may be damaged and may therefore need to be repaired, storage areas may or not be available, some retailer orders have unpredictable requirements, there may be unpredictable delays.

At a very high level, the whole operational management of the harbor can be described as a simple plan of abstract actions (see Fig. 1): ⟨unload, unpack, store, wait-for-order, treatment, delivery⟩. Generating such a plan is obvious and unnecessary: it is invariant and easily specified by hand. The actor’s problem in HOM is the following: given such a simple high-level sequential plan of abstract actions, refine it dynamically into concrete operations. In Fig. 1, HOM refines the abstract action store into more detailed (and possibly executable) skills, e.g., for registering a car to be stored, for moving it, etc. The different components implementing the skills interact among themselves by message passing.

HOM has to deal with objects that dynamically appear in the environment, such as ships and trucks dynamically entering and leaving the harbor. It has to deal with entities that can change their status dynamically, e.g., a car may get damaged, a gate or an area for storing or treating cars may become available or unavailable, etc. Some of these changes of status are *controllable* (e.g., cars can be moved from/to areas, they can be repaired). Others are caused by exogenous events that are *uncontrollable* (e.g., a car can get damaged, an area can become unavailable, a gate may not be available to unload some cars, a truck may not arrive on time to load cars to be delivered).

Moreover, management operations are carried out by different *components* (e.g., ships, gates, storage areas, vehicles) that must interact among each other in order to accomplish a task. For instance, each ship has its own procedure to unload cars to the gate. A gate has its own procedure to accept cars that are unloaded to the deck. These two procedures interact in order to get cars into the harbor. Similarly, a gate may interact with the equipment on a truck to inform the driver that some cars need to be moved to some storage areas. The truck has to interact with the procedure of the storage area to register, accept and park cars (see Fig. 1).

A possible advantageous design choice is therefore to model the domain in a distributed way, as a set of procedures that each component performs by interacting with the procedures of other components. The interactions between the ships and the gates, the gates and the trucks, the trucks and the storage areas, must be handled by the HOM facility that manages, synchronizes, and controls the different distributed procedures. In order to do this task, HOM must deal with a high degree of uncertainty and nondeterminism. This is not only due to exogenous events, but also to the fact that each procedure may—from the point of view of the management facility—behave nondeterministically. For instance, in the task to synchronize a ship with a gate to unload cars, the ship may send a request for unloading cars to the unloading manager and the gate may reply either that the request meets its requirements and the unloading operation can proceed according to some unloading specifications, or that the request cannot be handled. The management facility cannot know *a priori* what the request, the unloading specifications, and reply will be.

All the available procedures described so far constitute the basic building blocks of HOM. Given a high-level plan of abstract actions, and the set of available distributed and interactive procedures for each object that enters the system, we need a *refinement mechanism* that transforms abstract actions into a composition of the available procedures, and an *adaptation mechanism* that reacts to exogenous events and unforeseen situations. Both the refinement and adaptation mechanisms can be designed through an approach based on our vision, where the HOM facility is an *actor* that can be organized into a hierarchy of reasoning functions, where each abstract action can be a task to be further refined and planned for, and where online planning and acting are performed continually in order to adapt and repair plans. We need to embed one or several planners as enablers of components, which are called at run-time, when the system has to refine an abstract action to adapt a new context. We need to provide refinement mechanisms that can be triggered at run-time whenever an abstract action in a procedure needs to be refined or an adaptation needs to be taken into account, in agreement with our idea that the actor (integrating planning and acting) should be the primary topic of investigation.

In conclusion, here are some of the things needed in order to apply state-of-the-art approaches and techniques in the HOM example:

- Effective and efficient techniques for supporting the refinement at run-time of high-level actions and plans into lower-level components. For example, the simple sequence of high-level actions ⟨unload, unpack, store, wait-for-order, treatment, delivery⟩ should be refined at run-time into components such as the registration manager, the storage assignment manager and the storage manager. These components refine the action store by interacting among themselves (see Fig. 1). Notice that in many cases the refinement can be done only online, since in such dynamic and uncertain environment HOM needs to gather information at run-time.
- Effective and efficient run-time adaptation mechanisms for reacting dynamically, in a context-dependent way, to exogenous events and unforeseen situations. Current approaches in the planning literature either tend to foresee all possible events and situations, which is unpractical in realistic complex domains, or they tend to replan any time something unexpected occurs, which is hard to do in practice at run-time. Some adaptation approaches (e.g., [12,29,72]) require analyzing all the possible adaptation cases at design time, and handcoding the corresponding adaptation activities. This is hard to do *a priori* in a very dynamic and uncertain environment.
- Effective techniques for representing and reasoning about heterogeneous models, as well as effective techniques for mapping high-level representations into finer-grained state spaces and action spaces. These are needed, for example, in the refinement of the storage manager into the *navigate* component in Fig. 1.
- Techniques for effectively combining different reasoning techniques at different levels of the hierarchical structure. For instance, at one level it may be convenient to abstract away certain sources of uncertainty, e.g., exogenous events, and use efficient planners for deterministic domains. At a lower level, we may need instead to consider exogenous events. While some approaches address the problem of combining hierarchical reasoning with planning in nondeterministic domains (e.g., [63,64]), how to combine different forms of reasoning at different levels of a hierarchical system is still an open issue.

2.2. A factotum robot

A few harbors around the world are already equipped with Automated Ground Vehicles (AGVs) performing some of the transshipment operations discussed in the previous section. One of the earliest developments in this robotics area took place in Rotterdam, the experimental site of the Martha project (see [2,105] and Section 20.4 of [48]). For the sake of our argument, let us abstract here the issues of multi-robots task and resource sharing, which are important in this application. Let us focus on a single kind of robot that we will call a *factotum*, whose purpose is to carry out a variety of transportation and logistics missions autonomously. Such a factotum is a mobile platform equipped with several sensors (lasers, cameras, etc.) and actuators (forklift, arms, etc.) that can move around in open environments such as harbors or warehouses. It may need to perform tasks such as fetching objects, putting them into boxes, assembling small boxes into bigger ones or into containers to prepare orders. It may also need to move boxes and containers around, deliver them or pile them up in storage areas.

Such tasks can be performed with a few parameterized actions, such as these:

- map the environment (extend or update the map);
- localize itself in the environment map;
- move along a path;
- detect and avoid obstacles;
- identify and locate items (an object, a box, a shelf);
- grasp and ungrasp items;
- push items.

These actions are far too complex to be modeled only as finite-state transitions with preconditions-and-effects operators. One needs more detailed descriptive and operational models. For example, move works if it is given waypoints in free space or an obstacle-free path that meet kinematics and localization constraints, e.g., the path has visual landmarks or geometric features as required by localize, path segments without external localization capabilities should be small enough to allow for a drift error in the position estimate below an acceptable threshold; avoid can handle moving obstacles, provided that their velocity relative to the robot is not too high and the environment in front of the robot is not too cluttered. These conditions need to be checked and monitored while performing the actions. Further, concurrency has to be managed. For example, move should run in parallel with detect, avoid and localize.

These actions need domain-specific planners, e.g., a motion planner for move, a manipulation planner for grasp (with possibly several locate, push, grasp and move steps). Corresponding plans are more than a sequence or a partial order of actions; they require closed-loop control and complex monitoring.

A factotum robot has to choose and reason about its actions in order to perform its activities efficiently. At this abstract level, automated planning techniques should hold the stage. Instead of addressing this need, a significant part of the field has adopted implicitly a position akin to the following: "let us assume that factotum manufacturers will package low-level commands into actions such as goto, put and take that are amenable to our classical representations; we will carry on from there." But this position should be revised.

Indeed, given the advances of automated planning, the hard research issues are now precisely at the actor's level, rather than at the planner's level. Furthermore, robotics manufacturers have not packaged low-level commands into well behaving state-transition actions. They wisely went around hard problems and put together nicely engineered environments such as automated warehouse solutions.[3] But these hard AI problems remain open and their solutions are needed urgently for more versatile and easily deployable factotum robots. They correspond to the actual challenges of the field and provide a rationale for the change of focus we are advocating.

At the mission-preparation stage (the root node in Fig. 2), it is legitimate to view a logistics task as an organized set of abstract subtasks for collecting, preparing, conveying and delivering the goods. It can be reasonable to further decompose each subtask into a sequence of still abstract actions such as goto, take, goto, put, etc. We believe that the state of the art for solving the mission preparation stage at this abstract level is satisfactory.

However, the mission preparation stage is just the visible part of an "iceberg of deliberation problems" faced by our factotum robot in order to perform the entire task correctly. Consider that a goto action can be performed in many different ways depending on the environment properties: it may or may not require a planned path, it may use different localization, path following, motion control, detection and avoidance methods ("Move" node in Fig. 2). A goto after a take is possibly different from the one before (because of the held object). Let us assume that the robot has a set of navigation *skills* into which a goto can be refined. The same goto may start with a skill then switch to more adapted skills when required by the environment and the context. Such a switch between skills may be a normal progression of the goto action or a retrial due to complications. Our factotum robot will also need a collection of skills for take, put, open, close, etc., and any other action it may need to perform. All these skills should come with predictive models and possibly with additional knowledge to allow the robot to choose the best one with the right parameters.

[3] See for example http://www.kivasystems.com.

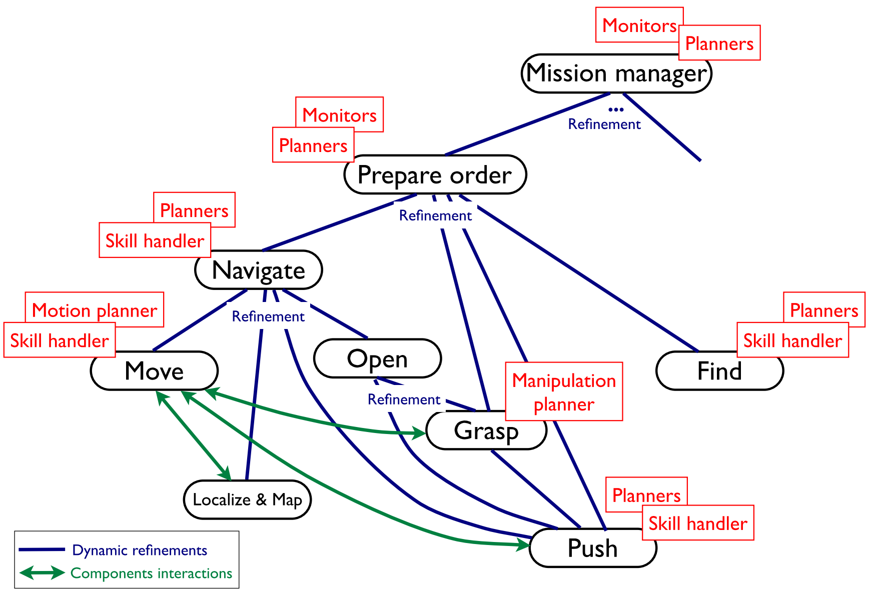

Fig. 2. Deliberation components for a factotum robot.

These skills may use complex control constructs with concurrent processes (loops, conditionals, semaphores, multi-thread and real-time locks). They can be developed from high-level specifications in some representation and/or with plan synthesis techniques. For example, [76] illustrates the use of HTN techniques for generating candidate navigation skills, while [101] uses probabilistic planning methods. Most often, skills in actual robotics platforms are developed from specifications in procedural languages, such as PRS [59] or TCA [95], or in Petri nets and automata-based representations, e.g., PLEXIL [104] or SMACH [19]. Different representations and methods may be useful to cover the skills needed by a factotum robot. Machine learning techniques are mandatory for improving the skills, acquiring their models, and adapting the factotum to a new trade.

Coming up with the needed skills and their explicit models does not solve all the problems in the hidden part of the already mentioned deliberation iceberg, it solves only the issue of *refining* actions into lower level commands. Several other problems remain for using these skills online, e.g., instantiation of the parameters from sensor readings, constraint propagation, synchronization, time management, and action revision.

In addition to acting appropriately with the right skills, there are *monitoring* problems at various levels, from the primitive and skill levels to the task and mission levels, and possibly to the goal level. Monitoring compares predictions to observations; it diagnoses discrepancies and triggers appropriate recovery actions. In order to do that, predictions of what is needed to correctly perform ongoing activities should be made beforehand. Making the right predictions from the combined models of actions and skills, as well as from additional models of the environment, is a difficult problem that involves heterogeneous representations.

Finally, autonomous acting requires extended *perception* capabilities: sensing actions as well as querying and information gathering steps, combined into a consistent interpretation of the environment. Open perception problems are not limited to signal processing and pattern recognition issues. They include in particular AI problems such as anchoring and symbol grounding (i.e., establish, maintain and revise signals-to-symbols relations), reasoning on what is observable and what is not, integrating knowledge-gathering actions to environment changing actions, acting in order to maintain a consistent interpretation of self and the world, dynamic situation and plan-recognition problems.

In summary, this example illustrates some of the challenges of deliberate action. It reveals that the bottleneck for the design of a factotum robot is not at the level of more sophisticated path-finding techniques in compactly specified state-transition systems. Coming up with better domain-independent heuristics and more efficient search algorithms will not solve what we referred to as the actor's deliberation problems. Instead, one needs in particular:

- descriptive models of actions and environments that are elaborate enough to allow for the synthesis of complex skills, possibly making use of domain-specific knowledge;
- operational models that make it possible to run these skills, make context-dependent choices, and monitor their execution;
- online algorithms that interleave reasoning and acting, to make it possible to refine and monitor a task at different hierarchical levels and to revise dynamically its lookahead predictions.

Next, we will discuss these and other open problems of a research agenda focused on actors.

3. Open problems

As mentioned in previous sections, the change of focus from planners to actors stresses *hierarchical online reasoning*. The main cognitive functions an actor has to perform seem to be the following:

- *refine* a task or an abstract action into lower level actions and primitives;
- *monitor* current state of the world it perceives with respect to predictions;
- *revise* its previous refinements.

Refinement and revision may involve the online synthesis of plans and more complex programs. They may also involve the optimized choice of skills, previously synthesized and/or learned, and their parameters. These functions require predictive domain models that may use several knowledge representations.

In the sequel of this section we will discuss open problems that we foresee for the design of actors regarding the following issues:

- knowledge representations for needed models;
- modeling, knowledge acquisition and verification techniques;
- synthesis techniques required;
- monitoring techniques;
- integration issues.

Needless to say, this decomposition does not reflect the steps of a research roadmap. Its purpose is mainly to structure the argument and present the problems that need to be tackled.

3.1. Knowledge representations

An actor needs to *reason* about its actions and to *perform* them. It needs models for choosing, organizing, monitoring, repairing and revising its actions and plans. It also needs operational specifications for refining its actions into lower-level executable steps or mapping them into skills. The specifications needed for reasoning, referred to here as *descriptive models*, address the "what", the "when", and possibly the "why" issues, i.e., conditions, effects, motivations, rewards and costs of actions. The specifications needed for achieving actions, referred to as *operational models*, address the "how" issues, i.e., modus operandi for actions in their context. Some models may offer both descriptive and operational specifications. The knowledge representations required by actors should facilitate the specification of the "know-what" as well as the "know-how" models of actions in a general and consistent way. They should allow for efficient deliberation algorithms.

Note that both types of models need to take into account the actor's environment. They must either describe it directly, or be integrated with separate models (see Section 3.1.3) in order to account for the environment's dynamics, exogenous events, causal relationships, effects and possible interference with the actor's actions.

3.1.1. Descriptive models

Descriptive models for domain-independent planning (for background on this topic, see [48,89]) can be specified using various representations, among which are these:

- Precondition-and-effect representations such as Strips, ADL, and the PDDL family of languages [41,42];
- Formal action languages, which stem mainly from the situation calculus and the event calculus in first-order logic, e.g., Golog [68], and related Action Description languages, such as $\mathcal{A}, \mathcal{C}+, \mathcal{K}$ [51,50,49], etc.;
- Timelines over state-variables, e.g., the planners [47,43,45] and the ANML language [97];
- Nondeterministic and probabilistic representations, e.g., the PPDDL, and RDDL [91] languages.

There are several open problems regarding the comparative expressiveness of these representations for planning. From the actor's viewpoint other urgent issues need to be addressed. All of these representations are designed for allowing an efficient computation of the state resulting from an action, but at a too abstract level. It is unclear how relevant these representations remain for expressing detailed actions with links to sensing, control and monitoring.

Furthermore, these representations need to be integrated with domain-specific representations and algorithms. An actor should be able to query specialized enablers for checking feasibility of actions and making its predictions. For example, a factotum robot needs among other specialized enablers a motion planner, a manipulation planner, a perception planner (visibility, sensing modalities, etc.); all these need to be consistently integrated with the other enablers.

Most of the planning representations listed above assume a static environment or can handle some limited extensions of this assumption. Several problems remain for grasping the proper dynamics of the environment. Exogenous events need to be among the main building blocks of descriptive models. Furthermore, these representations generally rely on the *closed world* assumption, i.e., the assumption that all facts about the world are either explicitly stated, or derivable from explicitly stated facts. This assumption may be reasonable for planning at an abstract level, but it is hardly acceptable for acting. An actor should be able to reason about unknown facts, in particular about missing information that it can arrange to acquire with observation and information gathering actions. Some work has been done along this line, e.g., [17,13,4,74,92]. However, most of these works rely on assumptions that can be applied only in specific domains. For instance, [74] assumes that the gathered information persists for a reasonable period of time, and none of the actions causes this assumption to be violated.

We already mentioned that planning for an actor is to be viewed as a continual online process giving partially refined plans, provided that the actor is confident that the missing steps in its plans can be completed while acting. This confidence should be derivable from the descriptive models. Adequate representations should be added to available formalisms to express when and which steps are critical and need to be refined down to some level before committing to a plan and starting to act. The level of criticality of a step may vary widely depending on the foreseen context of its execution.

3.1.2. Operational models

Operational models for acting can be specified with representations such as:

- Procedural representations, e.g., PRS [59], RAP [39], TCA [94], RMPL [58], etc.;
- Automata, Petri nets, workflow and distributed processes, e.g., PLEXIL [104], RMPL, BPEL [6,109], APFL [23], SMACH [19], etc.;
- Adaptive and partial programming languages and systems, such as Alisp [5] or A2BL [96], that couple specification with reinforcement learning techniques.

In robotics for example, procedural representations are quite popular for refining actions into hand specified skills, possibly with monitoring and repair actions. But these kind of specifications are not very generic over a variety of actions and environments. They seldom provide models of actions and causal relationships for the dynamics they specify.

There is a significant overlap between operational models for low-level reactive systems and those for deliberative actors. The same representations can be used for both, e.g., state charts [53], and their more recent extensions toward behavioral programming [54], can be used for specifying the know-how for simple embedded systems as well as for more complex behaviors. The specifics of the actors we are interested in stem from the diversity in tasks and environments. Representations for operational models should allow for handling such diversity.

It is interesting to note that *methods* in HTN planning, i.e., directives for refining a task into subtasks, offer in principle an operational component in a descriptive model. However, these constructs have been used mostly for reducing the search complexity through heuristic domain knowledge. More is needed on how to extend HTN methods to acting functions.

The relationships between the operational and descriptive specifications of an actor's models cover several open problems. For example, one would like to be able to extend a descriptive model towards its operational counterpart. Conversely, it should be possible to synthesize the descriptive specifications of an operational model, making it possible to reason about it and perhaps to generalize it and use it more generically. Here also there are open problems regarding the environment dynamics, usually taken into account in operational models in hard-coded and ad-hoc ways. An interesting perspective would be to combine a generic model of an environment with an actor's operational models, to verify their consistency, to synthesize knowledge needed for monitoring, etc.

3.1.3. Environment models

Different planning and acting problems may require different kinds of models of the same environment. For example, a factotum robot (see Section 2.2) needs environment maps at the geometric, topologic and semantic levels. It would need object models describing the geometry and shape of objects, their functions and possible uses. Both the robot and the HOM facility would also require models of exogenous events that normally take place in the environment. Their interaction with humans would demand an additional range of models.

The traditional approach in automated planning is to integrate the relevant part of the environmental model, viewed at an abstract relational level, within the planning problem specifications. This approach does not work for open environments, such as the environments in which the robot or the HOM facility would operate. An actor should be able to access additional models when needed, e.g., by querying online ontologies or downloading additional object and event descriptions. For example, the Bremen harbor system may need to issue queries for information about incoming ships or new cars. An actor should be able to acquire new skills when needed, e.g., how to operate a device that the robot is not familiar with, or how to deal with a new unloading or storage procedure. The integration of these additional models with the other actor's models is not straightforward, even with standardized representations.

We mentioned that in principle, the same model may be used as both a descriptive and an operational specification. This interesting possibility is related to the preceding set of open problems. Complex environments are often modeled operationally through *simulation models*; the results of simulations can be used—e.g., by sampling techniques—to provide descriptive predictions for use in reasoning, planning and optimization. This is illustrated, for example, by the sampling techniques in the configuration space for motion and manipulation planning that are the basis of the powerful probabilistic roadmaps and rapidly-exploring random tree algorithms. This avenue of research, not enough explored, opens promising perspectives.

3.2. Model acquisition and verification

The acquisition of descriptive and operational models of an actor is a critical issue that can determine the feasibility, the effectiveness, and the correctness of automated acting and planning techniques. A fundamental step supporting model acquisition is *model verification*. Informally, model verification is the task that checks whether the model is "good", that is, it provably meets some properties.[4]

Model acquisition and verification can be carried out in different ways. Model acquisition can be done manually, through the definition of a model in a proper specification language (see the previous section), with the support of software engineering tools (e.g., for the definition of requirements), or it can be supported by tools for learning from examples and interactions with a teacher.

Machine learning techniques are critical components, possibly with simulators, for the acquisition of operational models with adaptive and partial programming approaches. Reinforcement learning, learning from demonstration and active imitation offer promising perspectives in that direction, e.g., [7,60,107] and approaches mentioned in Section 3.1.2. Learning descriptive models of actions and skills is an active area of research, e.g., [112,61,66,110,111], in which many challenges remain to address in the integrative actor's requirements.

Model verification can be done through testing, simulation, or through automated techniques based on formal methods (e.g., model checking or theorem proving). Extensive work has been done in the computer automated verification community using approaches such as automated theorem proving and model checking [36,26]. Different techniques have been used for modeling and verification of models of different kinds of environments (e.g., continuous variables in CAD models, discrete variables in state-transition models). In planning, some work has been done to address the problem of verifying properties of models [46] and verifying the correctness of plans. Properties can include temporal conditions on states predicted by plans (e.g., never get into a dangerous state, maintain some desired requirements) but also some properties related to reliability, robustness, possibility to recover from failure. In the case of nondeterministic or probabilistic models properties should be verified on sets of histories or other complex structures.

Our vision, which concentrates on actors as the primary topic of investigation, poses a variety of new research challenges in model acquisition and verification. These are summarized below.

Hierarchical structure and heterogeneous models. Some new challenges are due to the hierarchical structure of an actor as a fundamental integrative organization. Model acquisition and verification must take into account such hierarchical structure and, as a consequence, the need to provide mappings between different representations at different levels. While verification is helpful for acquiring abstract models in general, it becomes mandatory when we rely on a hierarchy of several heterogeneous models. Model verification must be able to check whether models at one level satisfy some conditions at another level. Moreover, at different levels we may have different representation mechanisms and different descriptive and/or operational representations. For instance, in order to navigate autonomously, a factotum robot needs different environment maps using different spatial representations. This heterogeneity may result in the need for different tools for supporting model acquisitions, e.g., tools supporting the construction of topological mapping systems combined with localization, tools for metric and topological simultaneous localization and mapping and semantic labeling techniques. Similarly, different verification techniques may be required to check properties of the acquired model. In some cases, hybrid models and hybrid verification techniques can be required, e.g., when we have discrete variables at one level and continuous variables at another.

The heterogeneity induced by the hierarchical structure is not only a matter of representation. At different levels there may be different models of the environment, and different ways of obtaining these models. Models may be generated with tools that support their construction, may be defined manually, or may be developed with a mix of automated and manual (e.g., editing) techniques. Alternatively, models may be given by external systems or inherited from a different hierarchical level.

Assumption-based modular acquisition and verification. In a complex planning and acting system based on a hierarchical structure, it is possible to verify that one module or part of the system satisfies some properties, provided that some assumptions about different modules or parts of the system guarantee some behaviors. This idea resembles some "assume guarantee" techniques used in model checking [82] and techniques for assumption-based automated composition, e.g., [85]. In the HOM example, we can assume an action for storing cars completes its task successfully, and on this base plan for the delivery step. This challenge in verification is highly related to model acquisition.

Semantic annotations and dynamic models. A further challenge is the model acquisition and verification with respect to domain ontologies. The knowledge representations discussed in the previous section are suited to describe dynamic domains, i.e., the evolution of a system through actions and events. One also needs to represent static properties of a domain. These are conveniently grasped with ontologies and description logic expressing the semantics of terms [11]. It is clear that both

[4] As explained in the rest of this section, model verification becomes even more vital if the models are hierarchical and heterogeneous and if online planning and acting is required.

types of representations are needed by an actor. Several attempts have been made to extend languages suited to define dynamic models with languages used to describe static properties, e.g., see the work on WSDL-S, METEOR-S, and BPEL-S [69,93,83,84]. However, extending the expressiveness of languages in one or the other direction results in both conceptual and practical problems. A different approach can be based on the idea to keep separate the description of the behavior of the domain and the semantic description of objects in the domain and bridge the two descriptions with semantic annotations of the dynamic models of behaviors. For instance, in the HOM example, we can have an ontology describing the different types of cars and the treatment needed, and a plan can query such ontology to understand the next steps to be performed.

Run-time vs. offline verification. Further challenges are due to the need for continual online planning and reasoning. Due to the need of continual planning and monitoring, important information is available only in certain moments and in some situations. Planning and reasoning must be done according to some assumptions. If we explicitly represent the assumptions under which our model has been constructed, then we can verify that such assumptions are maintained, e.g., at run-time or in different contexts.[5]

Moreover, some verification can be done only at run-time [17]. Run-Time Verification [67,25] takes place while the actor is planning and/or acting. Properties to be verified at run-time can be generated by planning for monitoring purposes.

Finally, there is a balance between defining detailed models that are used to generate plans that are guaranteed to satisfy some conditions (these models are supposed to be "complete" and plans are correct by construction), and having models that allow us to generate plans with more relaxed constraints, and then verify (either at design or at run time) whether those plans satisfy some conditions. Since the actor is continuously monitoring, refining and repairing plans, relaxing some conditions and abstracting away some details from the model may be important in order to determine quickly and dynamically a useful plan. For instance, in HOM we can abstract away possible effects of exogenous events (such as a car being damaged, or a storage area being not available) and plan accordingly to a more simple model than the nondeterministic one.

3.3. Synthesis and refinement

For the synthesis of plans and skills in a dynamically changing world, several important issues need to be further explored. Some of them include the integration of domain-independent planners with task planners such as motion and manipulation planners, replanning and plan revision, temporal planning, external information sources, and how to use domain knowledge.

Flexibility and robustness. In a world that is dynamically changing and whose properties are only partially known, states of the world may occur during execution of a plan or skill that do not match what was predicted. In such situations, robustness and flexibility are quite important. The mismatch may make the actor's objectives easier to perform, making it desirable to revise the plan to take advantage of this—or it may interfere with the successful achievement of those objectives, making it necessary to revise the plan or to *replan* (i.e., discard the rest of the current plan, and synthesize a completely new one). Despite a number of recent works on replanning and plan revision, e.g., [102,103,40,71,10,9,22], substantial open problems remain. For example, much more work is needed to understand and address the tradeoffs between constructing a large and complicated plan that will succeed under all known contingencies, or postponing part of the planning process until some of the plan has been executed, to get a better idea what contingencies will actually occur. Similar tradeoffs between very cautious plans that are costly and optimized plans that are risky, as in the branched conditional plans of [27], need to be explored.

Temporal planning. Temporal planning is needed because actions are naturally durative and take place concurrently with other actions and events. Temporal planning opens several expressivity and algorithmic problems for which researchers have developed extensions of state-based representations, such as in [70,28], or timelines on state variables. From the actor's viewpoint and in the context of continual online planning, time has a triple role as:

- a resource that is needed by every action, and requires specific representations and reasoning methods;
- a computational resource that is needed for deliberation;
- a real-time constraint on the execution dynamics as well as on the actor's deliberations.

Although research has been done on each of these characteristics individually, synthesis algorithms need to integrate these three characteristics of time. The state of the art does not yet allow such integration.

Information queries, open worlds. Some AI planning formalisms, such as POMDPs, include the notion of making observations during plan execution, but these formalisms have several limitations. For example, they typically do not model the time delays that can occur before information is returned from a sensor, web service, or other information source, nor the notion

[5] This run-time verification is related to monitoring.

that such information may remain valid only for a limited period of time, and they often assume a *closed world* in which all of the possible objects, states, events, and action outcomes are given in advance. In an open world, it may be necessary to perform some of the information-gathering at planning time rather than postponing it until plan-execution time. Although some work has been done on this, the existing models are still limited [65,99], or quite specific [90]. If the world can change while planning is going on, this can present problems that in the worst case are complicated or even undecidable [8,9], but still must somehow be dealt with.

Effective use of domain knowledge. AI planning research has traditionally striven to develop domain-independent planning techniques. These techniques need to be combined with techniques that provide a more explicit role for domain knowledge. Partly this can be done by paying more attention to the notion of hierarchy, especially in regard to online planning.

To the extent that AI planning research has dealt with hierarchy, it has usually assumed that every level of the hierarchy has the same state space and action space. However, in practical hierarchical systems, the lower levels generally require different state and action spaces, in order to reason with the appropriate representations about details that would be both irrelevant and problematic at the higher, more abstract levels.

3.4. Monitoring and goal reasoning

The motivations for monitoring and goal reasoning derive naturally from the principle of continual online planning and plan revision. Deliberate action relies on a *feed-forward* control model, i.e., make predictions then take corrective actions triggered by monitoring the differences with observations. The precise role of monitoring is thus (i) to detect discrepancies between predictions and observations, (ii) to assess and diagnose these discrepancies, and (iii) to decide about first recovery actions while plan repair is triggered. The problems to address here relate in particular to the following issues:

- the knowledge that is needed to do monitoring;
- focus-of-attention mechanisms;
- diagnosis techniques;
- recovery mechanisms;
- integration of monitoring into acting.

Let us discuss these issues successively.

Some of the conditions that need to be monitored are in principle derivable from descriptive and operational models of actions, but there may be an overwhelming number of such conditions. Their dependency relationships may not be derivable from action models, and neither would the effects of exogenous events that can interfere with actions. Moreover, conditions not specific to any action, such as the internal functioning status of the actor, have also to be surveyed.

For all these reasons, monitoring knowledge has to be specified jointly with planning knowledge. What can be derived from the latter needs to be combined with the former. This is the case, for example, with plan-invariant conditions, i.e., conditions predicted and required to hold for part of the plan, as in [44,21,34,16]. The consistency of the two types of knowledge has to be verified, preferably within the knowledge specification and acquisition process. This leads to the issue of knowledge representations allowing such verification while permitting to take into account heterogeneous hierarchical levels. There are numerous open problems here and possible trade-offs, insufficiently investigated.

Combining monitoring and planning knowledge should allow the synthesis of plans with both functional properties (i.e., achieving the goals) and nonfunctional properties such as permitting monitoring and diagnosis. In many cases specific actions are needed to allow errors to be traced. This has been illustrated in the approach of "safe assumption-based plans" of [3] for nondeterministic planning where explicit assumptions are made to restrict plan search.

Monitoring a dynamic evolution requires a close link to perceiving and information gathering, as illustrated in [57]. Focus of attention mechanisms should give the environment features to perceive and the information to acquire. They should provide the properties and constraints that need to be checked and the correct timing for that. Focus of attention should allow a look-ahead in monitoring (detect forthcoming problems as early as possible), supported by a cost-benefit analysis with respect to ongoing activities. Finally, these focus mechanisms rely on monitoring knowledge as well as on other domain models, such as perception models. This adds additional requirements on the design of such models.

The interpretation of discrepancies between predictions and observations is a major monitoring function. Diagnosis mechanisms cannot rely on nominal models alone. To get to the causes of a detected anomaly requires more general causal models. Model-based diagnosis techniques address this issue. They have been applied successfully to autonomous agents, e.g., in the Deep-Space-One probe [77,106]. However, these techniques are limited to the diagnosis of the internal state of the actor (*proprioceptive monitoring*). They need to be extended to the diagnosis of the actor–environment interactions. This is very hard. But fortunately, the actor does not need a deep explanation of every detected anomaly. Its purpose is seldom to make a full repair, but mainly to explain enough the anomaly in order to go ahead with its activity.

Consider a factotum robot performing an open(*door*) action. The action refines into a skill with steps such as finding the door, identifying the door's opening direction (left/right, in/out), locating the handle, identifying the type of handle, grasping the handle, turning it, and moving the robot arm and base while holding the handle to open the door. Complex monitoring and diagnosis have to take place throughout the performance of these steps. For example, the last step may fail in several different ways. A failure may be due to a locked door. If the door started to turn, the failure may be due to an obstacle. If the door turns inward, then the robot may try to locate the obstacle and remove it from the path.

Recovery actions are needed to handle a detected failure temporarily, in order to allow for a proper plan repair. These reactive recovery actions will need to be adapted to diagnosed situations. Response time constraints may require offline synthesis and caching of possibly parametrized state-action pairs (as a recovery policy or a universal plan). Here also the research agenda of what needs to be specified and synthesized at each level of an actors hierarchy is rich in open problems.

Finally, the monitoring functions of focus, detection, diagnosis and recovery need to be integrated into the hierarchy of Fig. 2. If monitoring is centralized, it has to map its functions to the hierarchy. If it is distributed along the hierarchy, then there is a local monitoring at the level of each actor node and some propagation mechanisms to allow for coordinated monitoring. Questions to be addressed are for example: what each enabler (planner, scheduler etc.) gives to the corresponding monitor; how that monitor changes the performed actions, locally and elsewhere, to allow for the focus and information gathering mechanisms; how the recovery actions are propagated.

Goal reasoning. Goal reasoning (also called *goal-driven autonomy* [75]) is an abstract form of monitoring, at the level of the actor's objectives. The idea is to keep the actor's goals in perspective with respect to the evolution of the environment. When failures, conflicts or opportunities arise, the actor has to assess what is feasible, what compromises are possible, how to reassess its previous commitments [86], and whether to make new commitments or goals [30]. A simplified view of an actor's motivations can be given by a goal state or a cost criterion. A more general view would consider several possibly conflicting objectives, with preferences and a hierarchical scale.

For example, a factotum robot has to prepare a set of delivery orders (items packaged into boxes); it has also received a few demands from persons it is servicing; this in addition to background chores such as maintaining the environment in good order. The robot manages its to-do list as a dynamic priority scheme. A request for a coffee can be postponed after higher priority jobs while informing the requester; if the latter happen to leave the place then that coffee request is no longer relevant.

The agenda of open problems here includes in particular the following:

- How to specify and formalize this dynamic hierarchy of motivations;
- How to monitor the current goals, adapt and drop if needed some objectives;
- How to generate new goals given a higher level motivation.

3.5. Integration

We have argued that the change of focus from planners to actors stresses *hierarchical online reasoning*. The design and implementation of the deliberation functions needed for example by the HOM system or a factotum robot involve several components and enablers at different levels of the hierarchy. This naturally brings up more complex organization and integration problems than the design of a task planner. Among these problems there are in particular the following issues:

- How to organize the hierarchy of components implementing an actor. The hierarchy may include parts that are static, e.g., parts in which some components can trigger execution functions (have executors among their enablers). But other parts may change over time, or in different domains and problems. For example, learning a frequently performed task could provide a skill that could be used instead of repeatedly generating plans in slightly different contexts.
- How to handle distribution and concurrency of components.
- How to integrate observed real time constraints for reasoning and acting. These issues are the consequences at the integration level of the problems discussed in Section 3.3.

Several organizational paradigms from other areas of computer science may be useful for addressing these problems. Some examples include networked message-passing organization, data- and control-flow streaming mechanisms, blackboards and other shared memory architectures, and various combinations of the above. It is unclear whether there is a dominant architecture for an actor's deliberation issues. Different domain-specific considerations as well as engineering and community development issues (e.g., software reuse) play a critical role.

The HOM system for example is designed around a message-passing architecture. In Fig. 1, the top part shows the procedure for the whole operational management as a simple sequential plan of abstract actions: ⟨unload, unpack, store, wait-for-order, treatment, delivery⟩. The operation management facility refines these abstract actions into more detailed (and possibly executable) skills, e.g., for registering a car to be stored, for moving it, etc. The different components implementing the skills interact among themselves by message passing.

Several hierarchical online reasoning problems have been addressed in the area of architectures and languages for multi-agent systems, e.g., [108,80,81,35]. At a high level, some of these architectures can be applied to systems such as the HOM example. However, many open issues remain, e.g., how to refine automatically at run-time a high-level action such as store into a composition of lower-level components, or how to adapt such components to exogenous events and unforeseen situations.

Many experimental robot platforms implement sophisticated architectures, e.g., hierarchical (or three-tier) architectures [20,1,79], teleo-reactive architectures [38,73], reactive architectures extended for deliberation [62], and shared-memory architectures [87]. It is very reasonable to design an actor-based deliberation system with one of the first two categories, and possibly even with the other types. For our factotum robot, one may devote a component to each type of action, e.g., a navigation component, a simple pick-and-place manipulation component, an object finding component, an appliance or furniture manipulation component, a dialogue and communication component. These will provide services to components dealing with higher level tasks such as an order preparation component, or an assembly component. On top of these, a mission preparation and following component will be in charge of the overall jobs of the robot.

Finally, let us mention that we focused the above discussion on actors with autonomous deliberation functions and their architectural needs. Some applications may require also addressing environments with high dynamics and strong real-time constraints, e.g., for a flying robot. For such domains, techniques from real-time systems would be needed not only for the executor's level but possibly also for the low-level components. One may imagine an organization into nested loops with the higher-frequency dynamics at the lowest level. Furthermore, some domains may have critical safety constraints. A natural design is to circumscribe these constraints into specific enablers and components that are developed using adaptations of methods that were initially designed to prove safety properties, e.g., [54].

4. Conclusion

Although automated planning has been very successful in developing search techniques in abstract "preconditions-and-effects" state-transition spaces, these achievements have not had as much application impact as one might desire. Our analysis of this situation stresses several issues that others have previously recognized:

- "preconditions-and-effects" operators remain too abstract to model concrete actions adequately;
- there is more to deliberate action than just planning;
- it is important to address in a systematic and integrative way the problems involved with acting deliberately, which remains one of the main objectives of AI.

In this paper we have advocated a focus on the design and development of *actors*, as opposed to *planners*, *executors* or other *enablers* that an actor may use to perform its activities. Our two motivating examples, in service robotics and management of a large facility, illustrate how relevant the actor's viewpoint is for most planning applications.

In summary, our proposed focus entails two principles:

- *Hierarchically organized deliberation*. This principle goes beyond existing hierarchical planning techniques; its requirements and scope are significantly different. The actor performs its deliberation online; it requires heterogeneous models and mappings to perform refinements into finer-grained state spaces and action spaces. Methods and tools are needed to support the construction of models and mappings and the run-time reasoning.
- *Continual planning and deliberation*. The actor monitors, refines, extends, updates, changes and repairs its plans throughout the acting process, using both *descriptive* and *operational* models of actions. Ways are needed to acquire and verify these models, automatically extend them so that the actor can operate in an open environment, and do monitoring and goal reasoning.

We believe the time is ripe for the change of focus that we are advocating. We hope this paper will be useful in helping readers to understand the need for this change and the pressing problems to be addressed.

Acknowledgements

We are grateful to the anonymous reviewers for their thorough reading and insightful comments, which contributed to improving this paper.

This work was supported in part by EU FP7 SAPHARI under grant agreement no. ICT-287513, ARO grant W911NF1210471, ONR grants N000141210430 and N000141310597, and a UMIACS New Research Frontiers Award. The information in this paper does not necessarily reflect the position or policy of the funders; no official endorsement should be inferred.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An architecture for autonomy, Int. J. Robot. Res. 17 (1998) 315–337.
[2] R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert, Multi robot cooperation in the Martha project, IEEE Robot. Autom. Mag. 5 (1) (1998) 36–47.
[3] A. Albore, P. Bertoli, Generating safe assumption-based plans for partially observable, nondeterministic domains, in: AAAI, 2004, pp. 495–500.
[4] J.L. Ambite, C.A. Knoblock, M. Muslea, S. Minton, Conditional constraint networks for interleaved planning and information gathering, IEEE Intell. Syst. 20 (2005) 25–33.
[5] D. Andre, S.J. Russell, State abstraction for programmable reinforcement learning agents, in: AAAI, 2002.
[6] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weeravarana, Business process execution language for Web services, http://msdn.microsoft.com/en-us/library/ee251594(v=bts.10).aspx, 2003.

[7] B.D. Argall, S. Chernova, M.M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (2009) 469–483.
[8] T.C. Au, D.S. Nau, The incompleteness of planning with volatile external information, in: ECAI, 2006, pp. 839–840.
[9] T.C. Au, D.S. Nau, Reactive query policies: A formalism for planning with volatile external information, in: IEEE Symp. on Computational Intelligence and Data Mining (CIDM), 2007, pp. 243–250.
[10] N.F. Ayan, U. Kuter, F. Yaman, R.P. Goldman, Hotride: Hierarchical ordered task replanning in dynamic environments, in: Proc. 3rd Workshop on Planning and Plan Execution for Real-World Systems, 2007.
[11] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003.
[12] L. Baresi, S. Guinea, A3: self-adaptation capabilities through groups and coordination, in: India Software Engr. Conf. (ISEC), 2011, pp. 11–20.
[13] G. Barish, C.A. Knoblock, Speculative plan execution for information gathering, Artif. Intell. 172 (2008) 413–453.
[14] M. Beetz, Structured reactive controllers: controlling robots that perform everyday activity, in: Proceedings of the Third Annual Conference on Autonomous Agents, ACM, 1999, pp. 228–235.
[15] M. Beetz, D. McDermott, Improving robot plans during their execution, in: Internat. Conf. on AI Planning Systems (AIPS), 1994.
[16] S. Bernardini, D. Smith, Finding mutual exclusion invariants in temporal planning domains, in: Seventh International Workshop on Planning and Scheduling for Space (IWPSS), 2011.
[17] P. Bertoli, A. Cimatti, P. Traverso, Interleaving execution and planning for nondeterministic, partially observable domains, in: ECAI, 2004, pp. 657–661.
[18] F. Boese, J. Piotrowski, Autonomously controlled storage management in vehicle logistics applications of RFID and mobile computing systems, Int. J. RF Technol. Res. Appl. 1 (2009) 57–76.
[19] J. Bohren, S. Cousins, The SMACH high-level executive, IEEE Robot. Autom. Mag. 17 (2010) 18–20.
[20] R.P. Bonasso, R.J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, Experiences with an architecture for intelligent, reactive agents, J. Exp. Theor. Artif. Intell. 9 (1997) 237–256.
[21] A. Bouguerra, L. Karlsson, A. Saffiotti, Semantic knowledge-based execution monitoring for mobile robots, in: IEEE Internat. Conf. on Robotics and Automation (ICRA), IEEE, 2007, pp. 3693–3698.
[22] M. Brenner, B. Nebel, Continual planning and acting in dynamic multiagent environments, Auton. Agents Multi-Agent Syst. 19 (2009) 297–331.
[23] A. Bucchiarone, A. Lluch-Lafuente, A. Marconi, M. Pistore, A formalisation of adaptable pervasive flows, in: Web Services and Formal Methods, 2009, pp. 61–75.
[24] A. Bucchiarone, A. Marconi, M. Pistore, H. Raik, Dynamic adaptation of fragment-based and context-aware business processes, in: Internat. Conf. on Web Services, 2012, pp. 33–41.
[25] A. Cimatti, F. Giunchiglia, P. Pecchiari, B. Pietra, J. Profeta, D. Romano, P. Traverso, B. Yu, A provably correct embedded verifier for the certification of safety critical software, in: Internat. Conf. on Computer Aided Verification (CAV), 1997, pp. 202–213.
[26] E.M. Clarke, J.M. Wing, Formal methods: State of the art and future directions, ACM Comput. Surv. 28 (1996) 626–643.
[27] A.J. Coles, Opportunistic branched plans to maximise utility in the presence of resource uncertainty, in: ECAI, 2012, pp. 252–257.
[28] A.J. Coles, A. Coles, M. Fox, D. Long, COLIN: planning with continuous linear numeric change, J. Artif. Intell. Res. 44 (2012) 1–96.
[29] M. Colombo, E.D. Nitto, M. Mauri, Scene: A service composition execution environment supporting dynamic changes disciplined through rules, in: Internat. Conf. on Service Oriented Computing (ICSOC), 2006, pp. 191–202.
[30] M.T. Cox, Perpetual self-aware cognitive agents, AI Mag. 28 (2007) 32–46.
[31] T.L. Dean, M. Wellman, Planning and Control, Morgan Kaufmann, 1991.
[32] M. desJardins, E.H. Durfee, C.L. Ortiz, M. Wolverton, A survey of research in distributed, continual planning, AI Mag. 20 (1999) 13–22.
[33] O. Despouys, F. Ingrand, Propice-plan: Toward a unified framework for planning and execution, in: European Workshop on Planning, 1999.
[34] P. Doherty, J. Kvarnström, F. Heintz, A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems, Auton. Agents Multi-Agent Syst. 19 (2009) 332–377.
[35] E.H. Durfee, Distributed problem solving and planning, in: European Agent Systems Summer School (EASSS), 2001, pp. 118–149.
[36] E.A. Emerson, Temporal and modal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier, 1990, pp. 995–1072, Chap. 16.
[37] E. Erdem, E.R.M. Tillier, Genome rearrangement and planning, in: AAAI, 2005, pp. 1139–1144.
[38] A. Finzi, F. Ingrand, N. Muscettola, Model-based executive control through reactive planning for autonomous rovers, in: IEEE/RSJ Internat. Conf. on Intell. Robots and Systems (IROS), 2004, pp. 879–884.
[39] R.J. Firby, An investigation into reactive planning in complex domains, in: AAAI, AAAI Press, 1987, pp. 202–206.
[40] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Internat. Conf. on Automated Planning and Scheduling (ICAPS), 2006, pp. 212–221.
[41] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (2003) 61–124.
[42] M. Fox, D. Long, Modelling mixed discrete-continuous domains for planning, J. Artif. Intell. Res. 27 (2006) 235–297.
[43] J. Frank, A.K. Jónsson, Constraint-based attribute and interval planning, Constraints 8 (2003).
[44] G. Fraser, G. Steinbauer, F. Wotawa, Plan execution in dynamic environments, in: Innovations in Applied AI, Springer, 2005, pp. 208–217.
[45] S. Fratini, A. Cesta, R. De Benedictis, A. Orlandini, R. Rasconi, APSI-based deliberation in Goal Oriented Autonomous Controllers, in: Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), 2011.
[46] A. Gerevini, L. Schubert, Inferring state constraints for domain-independent planning, in: AAAI, 1998, pp. 26–30.
[47] M. Ghallab, H. Laruelle, Representation and control in IxTeT, a temporal planner, in: Internat. Conf. on AI Planning Systems (AIPS), 1994, pp. 61–67.
[48] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.
[49] E. Giunchiglia, Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism, in: Internat. Conf. on Principles of Knowledge Representation and Reasoning (KR), 2000, pp. 657–666.
[50] E. Giunchiglia, G.N. Kartha, V. Lifschitz, Representing action: Indeterminacy and ramifications, Artif. Intell. 95 (1997) 409–438.
[51] E. Giunchiglia, V. Lifschitz, An action language based on causal explanation: Preliminary report, in: AAAI, 1998, pp. 623–630.
[52] A. Grastien, P. Haslum, S. Thiébaux, Conflict-based diagnosis of discrete event systems: Theory and practice, in: Internat. Conf. on Principles of Knowledge Representation and Reasoning (KR), 2012.
[53] D. Harel, Statecharts: A visual formalism for complex systems, Sci. Comput. Program. 8 (1987) 231–274.
[54] D. Harel, A. Marron, G. Weiss, Behavioral programming, Commun. ACM 55 (2012) 90–100.
[55] P. Haslum, Computing genome edit distances using domain-independent planning, in: ICAPS Scheduling and Planning Applications Workshop, 2011.
[56] P. Haslum, A. Grastien, Diagnosis as planning: Two case studies, in: ICAPS Scheduling and Planning Applications Workshop, 2011.
[57] F. Heintz, J. Kvarnström, P. Doherty, Bridging the sense-reasoning gap: DyKnow – Stream-based middleware for knowledge processing, Adv. Eng. Inform. 24 (2010) 14–26.

[58] M.D. Ingham, R.J. Ragno, B.C. Williams, A reactive model-based programming language for robotic space explorers, in: Internat. Symp. on AI, Robotics and Automation in Space (ISAIRAS), 2001.
[59] F. Ingrand, R. Chatilla, R. Alami, F. Robert, PRS: A high level supervision and control language for autonomous mobile robots, in: IEEE Internat. Conf. on Robotics and Automation (ICRA), 1996, pp. 43–49.
[60] K. Judah, A.P. Fern, T.G. Dietterich, Active imitation learning via reduction to IID active learning, in: Proc. Uncertainty in AI, 2012, pp. 428–437.
[61] S. Kambhampati, S.W. Yoon, Explanation-based learning for planning, in: C. Sammut, G.I. Webb (Eds.), Encyclopedia of Machine Learning, Springer, 2010, pp. 392–396.
[62] K. Konolige, K. Myers, E. Ruspini, A. Saffiotti, The Saphira architecture: a design for autonomy, J. Exp. Theor. Artif. Intell. 9 (1997) 215–235.
[63] U. Kuter, D.S. Nau, M. Pistore, P. Traverso, A hierarchical task-network planner based on symbolic model checking, in: Internat. Conf. on Automated Planning and Scheduling (ICAPS), 2005, pp. 300–309.
[64] U. Kuter, D.S. Nau, M. Pistore, P. Traverso, Task decomposition on abstract states, for planning under nondeterminism, Artif. Intell. 173 (2009).
[65] U. Kuter, E. Sirin, D.S. Nau, B. Parsia, J. Hendler, Information gathering during planning for web service composition, J. Web Semant. 3 (2005) 183–205.
[66] T. Könik, P. O'Rorke, D.G. Shapiro, D. Choi, N. Nejati, P. Langley, Skill transfer through goal-driven representation mapping, Cogn. Syst. Res. 10 (2009) 270–285, http://dblp.uni-trier.de/db/journals/cogsr/cogsr10.html#KonikOSCNL09.
[67] A. Legay, S. Bensalem (Eds.), Proc. Internat. Conf. on Runtime Verification (RV), Lect. Notes Comput. Sci., vol. 8174, Springer, 2013.
[68] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. Scherl, Golog: A logic programming language for dynamic domains, J. Log. Program. 31 (1997) 59–84.
[69] K. Li, K. Verma, R. Mulye, R. Rabbani, J.A. Miller, A.P. Sheth, Designing semantic web processes: The WSDL-S approach, in: Semantic Web Services, Processes and Applications, 2006, pp. 161–193.
[70] I. Little, D. Aberdeen, S. Thiébaux, Prottle: A probabilistic temporal planner, in: Proc. AAAI, AAAI Press/MIT Press, Menlo Park, CA/Cambridge, MA, 2005, pp. 1181–1186.
[71] I. Little, S. Thiébaux, Probabilistic planning vs. replanning, in: ICAPS Workshop on IPC, 2007.
[72] A. Marconi, M. Pistore, A. Sirbu, H. Eberle, F. Leymann, T. Unger, Enabling adaptation of pervasive flows: Built-in contextual adaptation, in: Internat. Conf. on Service Oriented Computing (ICSOC/ServiceWave), 2009, pp. 445–454.
[73] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, R. McEwen, A deliberative architecture for AUV control, in: IEEE Internat. Conf. on Robotics and Automation (ICRA), 2008, pp. 1049–1054.
[74] S.A. McIlraith, T.C. Son, Adapting Golog for composition of semantic web services, in: Internat. Conf. on Principles of Knowledge Representation and Reasoning (KR), 2002, pp. 482–496.
[75] M. Molineaux, M. Klenk, D. Aha, Goal-driven autonomy in a Navy strategy simulation, in: AAAI, 2010, pp. 1548–1554.
[76] B. Morisset, M. Ghallab, Learning how to combine sensory-motor functions into a robust behavior, Artif. Intell. 172 (2008) 392–412.
[77] N. Muscettola, P.P. Nayak, B. Pell, B.C. Williams, Remote agent: to boldly go where no AI system has gone before, Artif. Intell. 103 (1998) 5–47.
[78] K.L. Myers, CPEF: A continuous planning and execution framework, AI Mag. 20 (1999) 63–69.
[79] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, CLARAty and challenges of developing interoperable robotic software, in: IEEE/RSJ Internat. Conf. on Intell. Robots and Systems (IROS), 2003.
[80] M. Paolucci, O. Shehory, K.P. Sycara, Interleaving planning and execution in a multiagent team planning environment, Electron. Trans. on Artif. Intell. 4 (2000) 23–43.
[81] P.M. Pappachan, E.H. Durfee, Interleaved plan coordination and execution in dynamic multi-agent domains, in: ICMAS, 2000, pp. 425–426.
[82] C.S. Pasareanu, M.B. Dwyer, M. Huth, Assume guarantee model checking of software: A comparative case study, in: The Spin Workshop, 2013.
[83] A.A. Patil, S.A. Oundhakar, A.P. Sheth, K. Verma, METEOR-S web service annotation framework, in: WWW, 2004, pp. 553–562.
[84] M. Pistore, L. Spalazzi, P. Traverso, A minimalist approach to semantic annotations for web processes compositions, in: Euro. Semantic Web Conf. (ESWC), 2006, pp. 620–634.
[85] M. Pistore, P. Traverso, Assumption-based composition and monitoring of web services, in: Test and Analysis of Web Services, 2007, pp. 307–335.
[86] M.E. Pollack, J.F. Horty, There's more to life than making plans: Plan management in dynamic, multiagent environments, AI Mag. 20 (1999) 1–14.
[87] S. Rockel, B. Neumann, J. Zhang, K.S.R. Dubba, A.G. Cohn, S. Konecny, M. Mansouri, F. Pecora, A. Saffiotti, M. Gunther, S. Stock, J. Hertzberg, A.M. Tomé, A. Pinho, L. Seabra Lopes, S. von Riegen, L. Hotz, An ontology-based multi-level robot architecture for learning from experiences, in: AAAI Spring Symposium, 2013, pp. 1–6.
[88] S.J. Rosenschein, L.P. Kaelbling, A situated view of representation and control, Artif. Intell. 73 (1995) 149–173.
[89] S. Russell, P. Norvig, Artificial Intelligence, A Modern Approach, 3rd edn., Prentice-Hall, Upper Saddle River, NJ, 2009.
[90] M. Samadi, T. Kollar, M. Veloso, Using the web to interactively learn to find objects, in: AAAI, 2012, pp. 2074–2080.
[91] S. Sanner, Relational Dynamic Influence Diagram Language (RDDL): Language description, Technical Report, NICTA, 2010.
[92] S. Sardiña, G.D. Giacomo, Y. Lespérance, H.J. Levesque, On the semantics of deliberation in Indigolog – from theory to implementation, Ann. Math. Artif. Intell. 41 (2004) 259–299.
[93] A.P. Sheth, K. Gomadam, A. Ranabahu, Semantics enhanced services: METEOR-S, SAWSDL and SA-REST, IEEE Data Eng. Bull. 31 (2008) 8–12.
[94] R. Simmons, Concurrent planning and execution for autonomous robots, IEEE Control Syst. Mag. 12 (1992) 46–50.
[95] R. Simmons, Structured control for autonomous robots, IEEE Trans. Robot. Autom. 10 (1994) 34–43.
[96] C. Simpkins, S. Bhat, C. Isbell Jr., M. Mateas, Towards adaptive programming: integrating reinforcement learning into a programming language, in: Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, ACM, 2008, pp. 603–614.
[97] D.E. Smith, J. Frank, W. Cushing, The ANML language, in: Internat. Conf. on Automated Planning and Scheduling (ICAPS), 2008.
[98] S. Sohrabi, J.A. Baier, S.A. McIlraith, Diagnosis as planning revisited, in: Internat. Conf. on Principles of Knowledge Representation and Reasoning (KR), 2010.
[99] S. Sohrabi, S.A. McIlraith, Preference-based web service composition: A middle ground between execution and search, in: The Semantic Web–ISWC 2010, Springer, 2010, pp. 713–729.
[100] A. Tate, Planning, in: MIT Encyclopedia of the Cognitive Sciences, 1999, pp. 652–653.
[101] F. Teichteil-Königsbuch, U. Kuter, G. Infantes, Incremental plan aggregation for generating policies in MDPs, in: Autonomous Agents and Multi-Agent Systems (AAMAS), 2010, pp. 1231–1238.
[102] H. Tonino, A. Bos, M. de Weerdt, C. Witteveen, Plan coordination by revision in collective agent based systems, Artif. Intell. 142 (2002) 121–145.
[103] R. Van Der Krogt, M. De Weerdt, Plan repair as an extension of planning, in: Internat. Conf. on Automated Planning and Scheduling (ICAPS), 2005.
[104] V. Verma, T. Estlin, A.K. Jónsson, C. Pasareanu, R. Simmons, K. Tso, Plan execution interchange language (PLEXIL) for executable plans and command sequences, in: Internat. Symp. on AI, Robotics and Automation in Space (ISAIRAS), 2005.
[105] T. Vidal, M. Ghallab, R. Alami, Incremental mission allocation to a large team of robots, in: IEEE Internat. Conf. on Robotics and Automation (ICRA), 1996.
[106] B.C. Williams, P.P. Nayak, A model-based approach to reactive self-configuring systems, in: AAAI, 1996, pp. 971–978.
[107] A. Wilson, A.P. Fern, P. Tadepalli, A Bayesian approach for policy learning from trajectory preference queries, in: Adv. Neural Inf. Process. Syst., 2012, pp. 1142–1150.

[108] M.J. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed., Wiley, 2009.
[109] WSBPEL Technical Committee, Web Services Business Process Execution Language, Version 2.0 – OASIS Standard, 2007.
[110] H.H. Zhuo, D.H. Hu, C. Hogg, Q. Yang, H. Muñoz-Avila, Learning HTN method preconditions and action models from partial observations, in: C. Boutilier (Ed.), IJCAI, 2009, pp. 1804–1810.
[111] H.H. Zhuo, Q. Yang, R. Pan, L. Li, Cross-domain action-model acquisition for planning via web search, in: F. Bacchus, C. Domshlak, S. Edelkamp, M. Helmert (Eds.), Internat. Conf. on Automated Planning and Scheduling (ICAPS), AAAI, 2011.
[112] T. Zimmerman, S. Kambhampati, Learning-assisted automated planning: looking back, taking stock, going forward, AI Mag. 24 (2003) 73.