Constraint-Based Online Transformation of Abstract Plans into Executable Robot Actions

Till Hofmann,[1] Victor Mataré,[2] Stefan Schiffer,[1,2]
Alexander Ferrein,[2] Gerhard Lakemeyer[1]
[1]Knowledge-Based Systems Group,
RWTH Aachen University, 52056 Aachen, Germany
[2]Mobile Autonomous Systems and Cognitive Robotics,
FH Aachen University of Applied Sciences, 52066 Aachen, Germany

Abstract

In this paper, we are concerned with making the execution of abstract action plans for robotic agents more robust. To this end, we propose to model the internals of a robot system and its ties to the actions that the robot can perform. Based on these models, we propose an online transformation of an abstract plan into executable actions conforming with system specifics. With our framework, we aim to achieve two goals. First, modeling the system internals is beneficial in its own right in order to achieve long term autonomy, system transparency, and comprehensibility. Second, separating the system details from determining the course of action on an abstract level leverages the use of planning for actual robotic systems.

Introduction

Despite promising advances in planning systems, they see surprisingly little use in actual robotics environments. We believe this is because solving a planning task by itself is not sufficient to accomplish high-level behavior control of a robotic system. For one, the robot's platform (i.e., its hardware and low-level software components) often requires additional constraints that are ignored during planning, e.g., a domestic service robot participating in RoboCup@Home (Wisspeintner et al. 2009) must calibrate its arm before performing any manipulation tasks. During planning, we do not want to plan for all the requirements of the underlying platform, as this would increase the problem size significantly and would make it infeasible in practice. However, ignoring those constraints at the behavior level and dealing with them at the lower levels is often impossible, because platform constraints may entail changes to the action plan.

Another reason for such a separation of high-level behavior and low-level platform is a design problem: When modelling the domain, an agent programmer usually does not want to deal with the robot platform. On the other hand, a platform designer should not need to consider and adapt the high-level behavior when modifying the platform. Also, a robot often has to deal with failed actions, unexpected changes, and exogenous events. Thus, a considerable amount of monitoring is required when executing a high-level plan on a robot.

For these reasons, we propose a framework that allows the modelling of the robot platform and its constraints independent of the behavioral component. While designing the platform, the user designs a self model of the robot and defines all the constraints of the platform. The world model of the agent can be designed without taking low-level constraints into account. During execution, the abstract action plan is transformed into a concrete executable plan that satisfies the constraints of the lower levels.

To actually achieve a separation between the problem domain and platform-related execution concerns, the platform needs a certain degree of "self-awareness" in terms of its components, their capabilities, their states and their interdependencies. Our goal in this paper is to sketch out requirements for a logically founded constraint language that can be used by platform experts to explicitly model component state transitions, dependencies among them, error conditions and possible recovery strategies, including the potential need for human assistance. The result is an agent system capable of self-maintenance by generating platform-specific monitoring and recovery strategies from the platform model and a platform-independent action plan. This eliminates much of the expert intervention that is required to keep robots running in dynamic domains, while providing a generic framework that helps in decoupling strategic decision-making from any platform details.

Foundations & Related Work

Especially the research into planning systems that is focused on temporal coordination of (concurrent) actions is of particular interest to our endeavour (Tsamardinos, Muscettola, and Morris 1998; Jónsson et al. 2000; Kim, Williams, and Abramson 2001; Lemai and Ingrand 2004). In theory, it would allow generalizing both the domain logic and the platform details as a temporal planning problem.

Temporal optimization and parallelization of platform-dependent operations is also being performed successfully at the task execution level. Keith et al. (2009) employ a temporal network that describes platform constraints to reorder and optimize the manipulator trajectories specified by a sequential plan. Konečnỳ et al. (2014) separate the strategic planning layer that only handles an abstract domain conceptualization from the detailed execution strategy that makes a plan executable on a real robot. However, the *Consistency Based Execution Monitoring* directly maps abstract, but fully grounded plan elements to a domain-specific execution strategy, without specifying an explicit platform model.

Kunze, Roehm, and Beetz (2011) introduce the Semantic Robot Description Language (SRDL) to bridge the gap between purely kinematic description languages and the more abstract level at which task specifications are usually formulated. They leverage the Web Ontology Language (Bechhofer et al. 2004) to model how domain-specific actions depend on platform-specific components that are required to realize them. Waibel et al. (2011) use SRDL to implement a shared knowledge base that allows robots to improve their search and execution strategies with previous observations possibly made by other robots. In this case, the knowledge base covers both platform-specific and domain-specific knowledge within a common deduction engine based on Description Logic (Baader 2003). The works based on SRDL are related to our work in their purpose, but differ significantly in that the SRDL model is purely a translation layer that sits between the abstract action plan and the executive layer. As such, SRDL specifications cannot be used to modify execution strategies at runtime, and thus cannot be used to dynamically deduce error recovery strategies. Mansouri and Pecora (2016) describe a constraint-based approach to hybrid reasoning with a meta-CSP that describes the different types of knowledge. The CSP is solved by a meta-solver that combines different kinds of reasoners. CHIMP (Stock et al. 2015) uses HTNs to solve such constraint-based hybrid reasoning tasks. HTN-based task decomposition approaches often model platform details as part of the planning problem. Dvorák et al. (2014) limit the problem size by delegating execution monitoring to a PRS subsystem with a simple success/failure interface.

Based on the Situation Calculus (McCarthy and Hayes 1969), the action language GOLOG allows a programmer to intermix imperative programming with planning on a logically formulated domain model (Levesque and Lakemeyer 2008). READYLOG (Ferrein and Lakemeyer 2008) extends the search functionality of GOLOG to allow for decision-theoretic planning. Finzi and Pirri (2005) provide a theoretical integration of the Situation Calculus with temporal constraints. De Giacomo, Reiter, and Soutchanski (1998) define an execution monitor in Golog that allows to react to unexpected changes during execution. Hofmann et al. (2016) interleave PDDL-based planning with Golog-based execution for monitoring purposes. Schiffer, Wortmann, and Lakemeyer (2010) describe an online transformation of a READYLOG program by inserting actions to satisfy qualitative temporal platform-specific constraints, under the assumption that agent domain and platform domain are disjunct.

Approach

Our goal is to design a framework that allows the user to formulate a platform constraint model that describes internal and external dependencies of component states, both in terms of hardware and software. An agent framework can then turn an abstract plan into a platform-specific execution and monitoring strategy that satisfies these constraints. This allows a separation of the high-level program from the specific platform properties while complying with the platform constraints. In the following, we present the different components of such a framework.

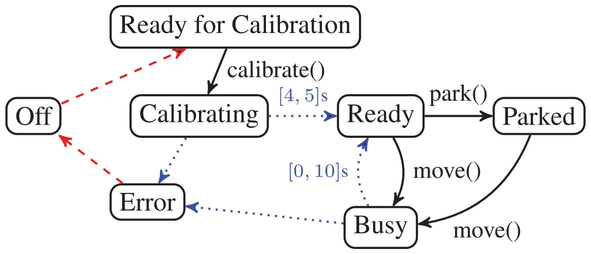

Figure 1: A finite state machine as a platform model for the Katana arm with three types of transitions: agent actions (black/solid), system events (blue/dotted), exogenous events (red/dashed). The edges are annotated with their action and expected time bounds.

Platform Models

Figure 1 shows an example for a model of a robotic manipulator arm, the Katana. Before the Katana arm can be used, it needs to be calibrated. Initially, the arm is turned off. It can only start its calibration process from a specific calibration position, so a human assistant must move the arm into the right position and then turn it on, which brings the arm into the state *Ready for Calibration*. From that state, the agent can decide to start the calibration. Note that this usually does not happen automatically, because the agent first has to make sure that it is in a location that allows an arm calibration, and second it may not need the arm at all. Since calibration is time-consuming, it should only be done if the arm is actually required. When the calibration is finished, the component driver triggers a transition to either the *Error* state or the *Ready* state. Similar to Schiffer, Wortmann, and Lakemeyer (2010), we model system components as state automata. But as the example in Fig. 1 shows, we need to differentiate between different kinds of transitions: 1. actions by the agent (black), 2. events triggered by the system (blue), 3. exogenous events (red).

Suitable Automata Models The platform model shown in Figure 1 is a finite state automaton with multiple edge types. However, more expressive automata models may be required to represent platform components. Consider a navigation stack that depends on the states of several low-level components, e.g., collision avoidance and localization. Each of these components will be modeled separately, but we might also want to formulate constraints on composite states covering multiple components. Hierarchical state machines as described in Girault, Lee, and Lee (1999) may be suitable to formulate such component-level abstractions. Timed transitions, such as the transition $\mathit{Calibrating} \rightarrow \mathit{Ready}$ may be modeled with timed automata (Alur and Dill 1994). While we will not change the foundation of our high-level reasoning, i.e., a situation calculus-based framework, we might consider a Petri-Net-based model such as the one described by Ziparo et al. (2011) for the component description as well.

Constraints

Platform constraints define properties that must always hold during the execution of the action plan. Based on Figure 1 and using Allen's interval relations (Allen 1983), we can define multiple constraints that must hold for the arm:

1. To calibrate the arm, the robot must be *at* a *free* location (i.e., a location without close objects).

$$\mathit{free}(\mathit{at}(x))\ \textbf{during}\ \mathit{state}(\mathit{arm}) = \mathit{Calibrating}$$

2. When starting to pick up an object, the arm must be ready or parked.

$$\mathit{state}(\mathit{arm}) = \mathit{Ready}\ \textbf{meets}\ \mathit{pickup}(x) \lor$$
$$\mathit{state}(\mathit{arm}) = \mathit{Parked}\ \textbf{meets}\ \mathit{pickup}(x)$$

3. Whenever the robot is moving, the arm must be parked.

$$\mathit{state}(\mathit{arm}) = \mathit{Parked}\ \textbf{during}$$
$$\mathit{state}(\mathit{navigation}) = \mathit{Moving}$$

Quantitative Temporal Constraints The examples above are qualitative temporal constraints. However, some components also require quantitative temporal constraints. Consider an RGBD camera that is used for perception. We can formulate the following constraints about the camera:

1. The camera needs some time to initialize, and therefore needs to be started one second before it can be used:

$$\mathit{state}(\mathit{camera}) = \mathit{Running}\ \textbf{before}_{\geq 1s}\ \mathit{detect}(x)$$

2. On the other hand, image processing is expensive, and thus should only be turned on if it is actually used within the next two seconds:

$$\mathit{state}(\mathit{camera}) \neq \mathit{Running}\ \textbf{unless}_{\leq 2s}\ \mathit{detect}(x)$$

The constraints above will be formulated in a temporal extension of the Situation Calculus and may refer to states of system components, fluents, and actions. While previous work only allowed qualititative temporal constraints (Schiffer, Wortmann, and Lakemeyer 2010), we want to allow for quantitative temporal constraints. In order to do so, we will extend the Situation Calculus based on Reiter (1996) and Gabaldon (2003) with qualitative and quantitative temporal aspects and embed the Metric Interval Temporal Logic (MITL) (Alur, Feder, and Henzinger 1996) into the Situation Calculus.

Events, Temporal Constraints, and Concurrency

The model of the Katana arm shown in Figure 1 has three kinds of edges: 1. Action edges that are directly triggered by the agent and are therefore under agent control, 2. Events that are triggered by the component itself, e.g. to end a durative action, 3. Exogenous events that are triggered by an external participant not under the agent's direct control, e.g., a human. Previously, both kinds of events were modeled as explicit exogenous actions with respective waiting actions. In our approach, we want to make use of concurrency in Golog with the *waitFor* construct (Grosskreutz and Lakemeyer 2003).

If we want to use the model of a system component to plan for a certain system configuration, e.g., a calibrated arm, we need to know about *expected* events. As an example, if the agent decides to start the calibration, it expects the calibration to finish successfully. If this was not the case, the agent could not cause state changes of system components in a meaningful way, as the outcome of any event transitions would be unknown. In addition to the information which transition is to be expected, we also annotate system events with expected time bounds. This allows the agent not only to reason about which event will occur, but also when it will occur. In the Katana example, we annotate the edge $\mathit{Calibrating} \rightarrow \mathit{Ready}$ with the expected time bounds $[4, 5]$, i.e., we expect the calibration to take at least four and at most five seconds. This way, the agent knows that it needs to start the calibration at least five seconds before it can use the arm.

Action Plan Transformation & Constraint Satisfaction

Given a platform-specific constraint model, an abstract action plan can be transformed into a platform-specific action plan that satisfies all constraints. To create such a plan, first the Golog interpreter determines an abstract action plan as usual. Next, the constraints are transformed into constraint networks (Dechter, Meiri, and Pearl 1991; Meiri 1996). In contrast to Finzi and Pirri (2005), we will not make use of *timelines*, but instead restrict our approach to interleaved and possibly true concurrency in order to allow a simpler formalization. Additionally, our approach will support quantitative constraints. The resulting constraint network will be evaluated with existing constraint solvers. A solution of the constraint network will determine the order of events with their interval limits. Platform constraints, e.g., $\mathit{state}(\mathit{arm}) = \mathit{Ready}$, must be transformed into actions by determining a suitable action sequence based on the platform model. The method of determining this action sequence depends on the underlying state machine model. For a simple state machine as shown in Figure 1, the actions can be determined by searching for a sequence of transitions that result in the desired state. For other, more expressive models, more complex methods may be necessary.

In some cases, such as the calibration of the Katana arm, inserting a single action may suffice. In other cases, the original action plan must be modified, e.g. to actively seek out localization features before some delicate manipulation task can be performed. Thus, a clear separation of the abstract agent and the plan transformation is not always possible and significant modifications of the original plan may be necessary. For this reason, the transformation of the abstract action plan into an executable plan will be part of the high-level agent and implemented within the Golog interpreter.

Conclusion

We presented a concept for an agent system with an explicit model of the robotic platform and its constraints. The robotic platform is modeled with state automata based on timed automata and hierarchical state machines and allows multiple transition types for agent actions, system events, and exogenous events. Based on these models, the user can formulate constraints in an extension of the Situation Calculus, which allows to define platform-specific, quantitative temporal constraints. During execution, the abstract action plan is modified to satisfy all constraints of the underlying platform. The proposed agent system allows the user to separate behavior control and platform management while taking into account that the constraints may require significant changes to the abstract action plan, which are handled by the agent system during execution.

Acknowledgments

This work was supported by the German National Science Foundation (DFG) under grant numbers GL-747/23-1 and FE 1077/4-1.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun ACM* 26(11):832–843.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183–235.

Alur, R.; Feder, T.; and Henzinger, T. A. 1996. The benefits of relaxing punctuality. *J ACM* 43(1):116–146.

Baader, F. 2003. *The description logic handbook: Theory, implementation and applications*.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2004. OWL Web Ontology Language Reference. Technical report, W3C.

De Giacomo, G.; Reiter, R.; and Soutchanski, M. 1998. Execution Monitoring of High-Level Robot Programs. *Proc. of the 6th Int'l Conf. on Knowledge Representation and Reasoning (KR)*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Dvorák, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014. Planning and acting with temporal and hierarchical decomposition models. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, 115–121. IEEE.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11):980–991.

Finzi, A., and Pirri, F. 2005. Representing flexible temporal behaviors in the situation calculus. In Kaelbling, L. P., and Saffiotti, A., eds., *Proc. of the 19th Int'l Joint Conf. on Artificial Intelligence (IJCAI-05)*, 436–441.

Gabaldon, A. 2003. Compiling control knowledge into preconditions for planning in the situation calculus. In Gottlob, G., and Walsh, T., eds., *Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence (IJCAI-03)*, 1061–1066.

Girault, A.; Lee, B.; and Lee, E. A. 1999. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18(6):742–760.

Grosskreutz, H., and Lakemeyer, G. 2003. ccGolog – a logical language dealing with continuous change. *Logic Journal of the IGPL* 11(2):179–221.

Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *Proc. of the 30th Conf. on Artificial Intelligence (AAAI)*.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in interplanetary space: Theory and practice. In Czhien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proc. of the 15th Int'l Conf. on Artificial Intelligence Planning Systems, (AIPS-00)*, 177–186.

Keith, F.; Mansard, N.; Miossec, S.; and Kheddar, A. 2009. From discrete mission schedule to continuous implicit trajectory using optimal time warping. In *Proc. of the 19th Int'l Conf. on Automated Planning and Scheduling*.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proc. of the 17th Int'l Joint Conf. on Artificial Intelligence (IJCAI-01)*, 487–493.

Konečný, Š.; Stock, S.; Pecora, F.; and Saffiotti, A. 2014. Planning domain+ execution semantics: A way towards robust execution? In *2014 AAAI Spring Symposium Series – Qualitative Representations for Robots*.

Kunze, L.; Roehm, T.; and Beetz, M. 2011. Towards semantic robot description languages. In *IEEE Int'l Conf. on Robotics and Automation (ICRA 2011)*, 5589–5595.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In McGuinness, D., and Ferguson, G., eds., *Proc. of the 19th Nat'l Conf. on Artificial Intelligence (AAAI-04) and 16th Conf. on Innovative Applications of Artificial Intelligence (IAAI-04)*, 617–622.

Levesque, H., and Lakemeyer, G. 2008. Chapter 23 Cognitive Robotics. In Frank van Harmelen, V. L., and Porter, B., eds., *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. 869–886.

Mansouri, M., and Pecora, F. 2016. A robot sets a table: a case for hybrid reasoning with different types of knowledge. *Journal of Experimental & Theoretical Artificial Intelligence* 28(5):801–821.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. 463–502.

Meiri, I. 1996. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence* 87(1-2):343–385.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Proc. of the 5th Int'l Conf. in Principles of Knowledge Representation and Reasoning (KR-96)*, 2–13.

Schiffer, S.; Wortmann, A.; and Lakemeyer, G. 2010. Self-Maintenance for Autonomous Robots controlled by ReadyLog. In Ingrand, F., and Guiochet, J., eds., *Proc. of the 7th*

IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE2010), 101–107.

Stock, S.; Mansouri, M.; Pecora, F.; and Hertzberg, J. 2015. Online task merging with a hierarchical hybrid task planner for mobile service robots. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems (IROS)*, 6459–6464.

Tsamardinos, I.; Muscettola, N.; and Morris, P. H. 1998. Fast transformation of temporal plans for efficient execution. In *Proc. of the 15th Nat'l Conf. on Artificial Intelligence (AAAI-98) and 10th Innovative Applications of Artificial Intelligence Conf. (IAAI-98)*, 254–261.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Haussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; Schiessle, B.; Tenorth, M.; Zweigle, O.; and van de Molengraft, R. 2011. RoboEarth - A World Wide Web for Robots. *IEEE Robotics Automation Magazine* 18(2):69–82.

Wisspeintner, T.; Van Der Zant, T.; Iocchi, L.; and Schiffer, S. 2009. Robocup@ home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies* 10(3):392–426.

Ziparo, V. A.; Iocchi, L.; Lima, P. U.; Nardi, D.; and Palamara, P. F. 2011. Petri net plans. *Autonomous Agents and Multi-Agent Systems* 23(3):344–383.