Formalization of Robot Skills with Descriptive and Operational Models

Charles Lesire, David Doose, Christophe Grand

Formalization of Robot Skills with Descriptive and Operational Models

Charles Lesire[1] and David Doose[1] and Christophe Grand[1]

***Abstract*— In this paper, we propose a formal language to specify robot *skills*, i.e. the elementary behaviours or functions provided by the robot platform in order to perform an autonomous mission. The advantage of the language we propose is that it integrates a wide range of elements that allows to define and provide automatic translation both to *operational* models, used online to control the skill execution, and *descriptive* models, allowing to reason about the expected skill execution, and then apply automated planning or model-checking taking skill models into account.**

I. Introduction

The use of intelligent and autonomous robotic systems in real conditions will inevitably need these systems to be *programmed* by an operator, to customize a service robot depending on current exhibitions in a museum, to specify which reaction the robot should adapt to face hazards in a surveillance mission in a defense context or to collaborate with other robots or humans for manufacturing tasks. Programming robots for achieving such tasks is known as *Task-level programming* [1]. It consists of assembling elementary behaviours, or *skills*, to design the complete task or mission of the robotic system. This programming approach is largely developed in manufacturing applications [2], where skill sequences are manually programmed or learned from gesture recognition. When considering more complex missions, e.g., search and rescue or surveillance in outdoor fields, it is necessary to integrate more decision-making capabilities into the robot controllers, and use of more advanced reasoning about these capabilities, for instance by performing some *task-level automated planning*, or some verification of properties by *model-checking*.

Applying such methods requires *skill models* in order to reason about the capabilities of robots. Such models must integrate different elements depending on the context on which they are used [3]: *Descriptive* skill models define *what* a skill is doing, i.e. predict its behaviour; such models are generally used for automated planning or model-checking; examples are PDDL [4] for planning, or Timed Automata, as used in UPPAAL [5]. *Operational* skill models define *how* a skill must be executed, e.g., defining its input/output interface; such models are generally used online to actually manage the execution of skills; examples are task programming languages [6], [7] or execution control formalisms (e.g., Petri nets [8] or Behavior Trees [9]). When designing a deliberative architecture, that integrates both long-term planning of robot activities, and online execution of behaviours, or when performing verification based on model-checking, the consistency between descriptive and operational models is of concern.

In this paper, we propose a formal definition of skills, from which we can derive both descriptive and operational models of these skills, then ensuring the consistency between these models by the use of this common skill definition. The models and tools we propose in this paper can typically be applied when one has developed the functional architecture of its robot, including control algorithms, sensor data processing, or other functions, typically in a ROS-based architecture. The work proposed in this paper then allows to define a general interface to the several *skills* implemented by this functional layer, and to provide models of these skills to allow automated reasoning.

After discussing some related works on skill modeling (Sec. II), we propose the definition of a skill (Sec. III), and we show which kind of model we can derive from this definition, along with some tools we have defined to automate the construction of such models (Sec. IV). We illustrate these tools and models in a simple case study.

II. Related Works

Existing works related to skill modeling and management can be categorized into (1) the ones that use descriptive skill models online (for execution), (2) the ones that define planning techniques on operational models or build descriptive models from operational models, and (3) the ones that model the functional layers of robots. These three kind of contributions are discussed here-after.

In the field of robotic manufacturing, [10] use descriptive models of skills, i.e. models that include input parameters, preconditions, and predictive effects, to plan skills sequences. They translate skills modelled using an OWL ontology into PDDL files. They do not consider possible failures or side-effects of skills (like consumption of resources). In [11], they also define skill monitors that check online the preconditions and effects of skills to detect failures. They do not provide an *operational model* for skill, then programming manually the interaction with the robot platform and limiting the possibilities to combine skills and manage failures through models. In [2], they extended the skill definition to distinguish predictive effects used in descriptive models and postconditions used online to determine the end of skill execution. For the T-REX architecture [12], which uses timeline-based descriptive models, [13] proposes a generic executive to interface with robot skills, based on a generic state machine, that can be seen as an operational model of the skill. Yet the link between the state machine and the

[1]Authors are with ONERA/DTIS, University of Toulouse, 31400 Toulouse, France firstname.lastname@onera.fr

timeline models of the T-REX architecture are not discussed.

A deliberative actor based on the refinement of operational models has been proposed in [14]. It uses operational models to describe decomposition and proposes a planner that uses these operations as planning primitives. However, the actual interface with robot functions are not modeled. Operational models such as Behavior Trees have also been used as a basic model for deliberation, using synthesis from a LTL formulation [15], or hierarchical planning using an HTN-like process [16]. The same LTL-based synthesis has been applied to Finite State Machines in [17]. The vSTL framework [18] also derives model-checking models from a DSL describing operational robot programs. However, the interaction with the robot functional layer is not really formalized, and generally left to the developer to implement. For instance, SMACH [19] or Behavior Trees [9] tools largely used in the ROS ecosystem just define an interface towards ROS primitives (topics, services, actions) without formalizing the skills of the robots.

Formalization of skills has been partly addressed in some existing works. A relational model representing the relations between skills (or elementary behaviours) and resources and data (provided by the robot or external) has been proposed in [20]. This model however neither represent the behaviour of resources, nor possible terminal modes of the skill executions. In [21] skills are represented with preconditions and device resources requirements in an OWL ontology, along with mechanisms to define synchronization of skill execution. Again, the skill model integrates a few elements, e.g., with no information about failure modes or rates.

Performance Level Profiles (PLP) have been proposed in [22], as a semi-formal language (based on XML schema) to represent modules available on a robot platform. The module's description is rich, including preconditions/effects, resources, modes, and rates. Moreover, some tools are provided to generate PDDL models or runtime monitors. The behaviour of the modules is however not defined, making impossible to integrate their behaviour in verification methods or to define sound control protocols.

In this paper, we propose a rich and formal language to represent skills, along with an internal execution model, that tends to address all the issues discussed above. The skill description in itself can be seen as an extension of PLP, as it takes most of the concepts from it. However, from this skill description, we also propose translations and tools to derive both operational and descriptive models from this language. In particular, we provide a direct interface towards skill implementation, which PLP does not.

III. Skill Definition

In this section, we define the content of the Domain Specific Language (DSL) we propose to specify robot skills. This definition takes elements from operational models, that describe *how* the execution of a skill can be controlled, and elements from descriptive models, that describe *what* the skills do. This definition takes then some inspiration from models like PLP [22] or PDDL [4] formalisms. The execution model corresponding to this DSL is given in the next section. The BNF of the language is detailed on the online documentation[1].

This DSL defines at the top a **SkillSet**, i.e. a collection of skills (and other related elements) declared in a common model. A *skillset* is defined with a *name* attribute, and contains the following elements:

1) Types: First, the types used in the model must be declared. Each **Type** will be further mapped to an actual type structure depending on the model generator used (see Sec. IV).

2) Data: In this part of the models are defined some **Data**, that can be actual data provided by the platform (robot position, battery level), or some data internal to the model (like a local variable). A data is defined by its *name* and its *type*.

3) Functions: In this model, we can declare some **Function**. A function is only defined by a *name* and its *parameters*, and will be further mapped to an implementation during the generation process (see Sec. IV).

4) Resources: A **Resource** is an element managed or provided by the robot that must be shared between skills, and on which skill execution will depend. A resource is defined as a state-machine, described by *states* and possible *transitions* between these states. These transitions can be either *extern*, i.e. only controlled by the functional layer of the robot, or labeled, in which case they can be controlled by skills. Skills will also condition their execution on the states of the several resources.

5) Skills: A **Skill** is then an elementary functionality or action provided by a robot. A skill is identified by a *name*, and:

- has *inputs*, i.e. typed parameters needed for execution,
- uses *resources*; each skill can define for each resource a required state (*pre*), and action taking place when the skill starts (*start*), and an invariant (*inv*) that must hold during execution;
- has terminal *modes* in which the skill can end its execution; they are identified by a *name*, and can check (*post*) resources states, change (*effect*) resources states, or declare modifications on data (*postcondition*); moreover, it is possible to define a *duration*, as the nominal time taken to reach this mode;
- may have a *precondition*, i.e. a boolean formula based on data/input values and possibly using functions;
- has a *progress* rate, indicating at which rate the skill will send some progress feedback (in terms of realization progress or the skill execution).

In this paper, we consider an aerial robot that must perform, on request by an operator, `go-and-track` tasks in order to go to the position of detected objects of interest,

[1] `http://oara-architecture.gitlab.io/robot-skills/`

and track these objects. To perform a `go-and-track` task, the provided skills are: a *takeoff* skill, a *goto* skill, a *track* skill, as well as two skills managing a safe return (*gohome*) and an emergency landing (*elanding*).

Listing 1 presents the model of the *m600* robot *skillset* describing and its *goto* skill. The robot has a *position* data representing its current position, and a *time_to* function representing a model of the time taken to go from position a to b at speed s. The robot has also a resource *authority*, with states (1) *MANUAL* representing that the pilot has authority over the robot, (2) *AVAILABLE* representing that the autonomous controller has authority, and (3) *USED* representing that a skill is already using the resource. The *goto* skill has two inputs, the target position and the motion speed. To execute, it needs the resource *authority* to be AVAILABLE, and changes it to *USED* during the skill execution. The skill has two final modes: (1) *ARRIVED*, whose time to arrive is computed from the *time_to* function; (2) *BLOCKED*, occurring when the robot encounters a problem when reaching the target (e.g., it would have to exit the authorized flying area, which is forbidden). This example only illustrates some of the language elements. More complete examples are available on the online documentation[1].

Listing 1. M600 *goto* skill model

```
type Position
type Speed

skillset m600 {
  data position: Position

  function time_to(a: Position, b: Position, s: Speed): Number

  resource authority {
    initial AVAILABLE
    extern AVAILABLE -> MANUAL
    extern USED -> MANUAL
    extern MANUAL -> AVAILABLE
    t_use: AVAILABLE -> USED
    t_free: USED -> AVAILABLE
  }

  skill goto {
    progress=1
    input {
      target: Position
      speed: Speed
    }
    use authority pre=AVAILABLE start=USED
    mode {
      ARRIVED {
        authority effect=AVAILABLE
        postcondition { position = target }
        duration=time_to(position, target, speed)
      }
      BLOCKED { authority effect=AVAILABLE }
    }
  }
}
```

IV. Skill Models and Tools

The Domain Specific Language described previously provides a means to specify the several skills of a robot, from which we can derive several operational and descriptive models. For operational models (Sec. IV-A), we have defined an execution model of the skills, and we provide tools that generate so-called *skill managers*, i.e., software components responsible of activating and managing skill on the functional layer of robots. For descriptive models (Sec. IV-B), we provide translations from skill models to standard languages or frameworks in order to use state-of-the-art algorithms and tools.

A. Operational Models

The management of skill execution follows a schema, depicted in Fig. 1, where some components are responsible for the correct execution of skills (called *Skill Managers*), in relation to resource status (managed by a so-called *Resource Manager*), while other components are responsible for the execution and control of several skills, in order to perform a mission or implement a behavior. These controllers will interact with skill managers through a *SkillSet Interface*.

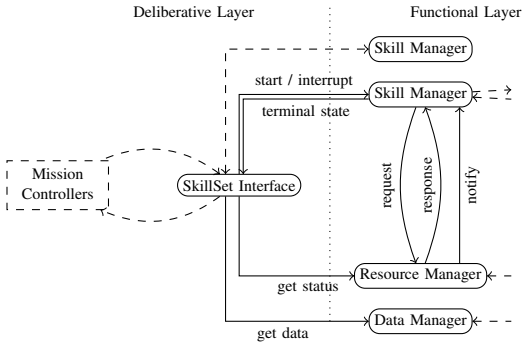

Fig. 1. Architecture principle using operational models

Mission Controllers (which are out of scope in this paper) will typically use the several skill interfaces provided in the *SkillSet Interface* library to select which skill to activate depending on the mission objective, the context, events, ...

In the case of the M600 robot and its *goto* skill, the resource manager would manage the *authority* resource, accepting transitions from *AVAILABLE* to *USED* and conversely from skill managers, and transitions to *MANUAL* from the functional layer when the pilot takes control. The *goto* skill manager would then interact with this resource manager, and possibly the *position* manager giving acess to the position data, and also manage the skill execution by interfacing with the navigation and guidance architecture of the M600 robot. On the controller side, a typical mission implemented in a FSM-based controller (described further in this paper) will use the *goto* interface to control the skill execution.

1) Data Manager: a *Data Manager* is a component that subscribes to the data coming from the functional layer, and stores this data so that controllers can access it through the skill interface library.

2) Resource Manager: the *Resource Manager* is the component in the architecture that manages the state and access to all the skillset resources. This component exposes the current state of the resources on one hand, and on the other hand, provides a service to change resource states by asking for transitions of the state-machines. This service is accessed by the skill managers (see below) to ensure the correct use of the resources, and from the functional layer to update the state of

the resources depending on events or processing performed by the robot.

3) Skill Managers: a *Skill Manager* is a component that manages the execution of one skill (i.e., one *skill* model). As one skill can be activated several times simultaneously, this execution is described by: (1) a finite state machine (FSM) defining the execution of one instance of the skill, and (2) a server that manages the several execution requests of the skill by coordinating the execution of several FSMs. The skill instance execution FSM is depicted in Fig. 2.

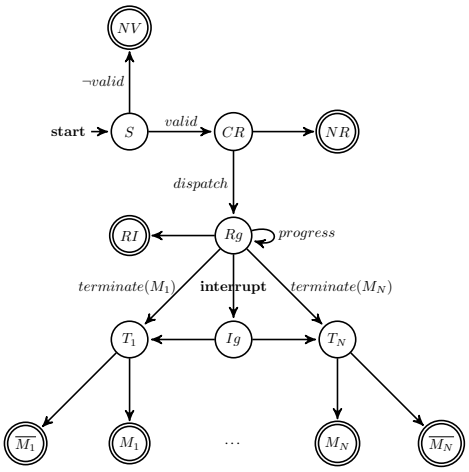

Fig. 2. FSM of a skill instance execution. Double circled states represent terminal states. Events in bold are incoming events (from the controller). Events in italic represent internal events of the manager generally linked to user-specified hooks.

When starting the execution of a skill (**start** event), the skill manager first *validates* the input parameters, by calling a user function that typically checks state variables of the functional layer. If the validation fails, the skill manager returns a validation failure (NV state). If the validation succeeds, the skill manager then checks the status of resources (CR state), by requesting the resource state transition to the resource manager. If the request fails, the skill manager returns a resource failure (NR state). Otherwise, the skill starting is fully validated, and the actual execution is started (*dispatch* event). While the skill is running (Rg state), it periodically reports a *progress*. The skill can then terminate in several ways:

- from an **interrupt** request, leading to the Ig state; the manager then waits for the completion of the interruption and then terminates (called by a user function) in one of the skill modes;
- if the resource state is no more consistent (state invariant evaluated to false); this situation is notified by the resource manager, and the skill manager then terminates with a resource interruption failure (RI);
- in a terminal mode of the skill, as defined in its skill model; transitions to these modes are triggered by the user-defined function, leading the skill state in one of the T_i states, where it checks the resource postcondition of this mode. If the postcondition holds, the final state of the skill instance execution is M_i, otherwise the final state is $\overline{M_i}$.

The skill manager instantiated for the *goto* skill of the M600 robot (see List. 1) has then a similar state-machine, where the terminal states corresponding to resource failures would mean that either the authority is not *AVAILABLE* when starting the skill (leading to state NR), or that the pilot took *MANUAL* control during the skill execution (leading to state RI). M_i final states are either *ARRIVED* or *BLOCKED*.

The skill manager server then manages the incoming execution requests and the corresponding FSM execution. If the skill accepts several concurrent requests (depending on its validation and resources), the server then manages the concurrent execution of several FSMs.

To implement the skill managers, we provide tools to generate from the skill models described earlier either a ROS actionlib server, or a ROS2 Action server. In both cases, the skill programmer has to fill callback functions corresponding to the internal events of the FSM of Fig. 2 (validation, dispatch, progress, interruption) to actually manage the execution of skills by orchestrating other components of the robot functional layer.

4) SkillSet Interface: Mission Controllers are components responsible for *controlling* the execution of skills, i.e. to select which skill to activate, and to manage the execution status of these skills (terminal modes) in order to perform a mission or implement a behavior. To do so, we provide a *SkillSet Interface*, implemented as a *SkillSet Client Library*. Therefore, from the skill models, we provide a tool to generate a python library giving access to the several skill, resource, and data managers through either direct function call, or several common frameworks used in robotic applications and based on discrete-event system models. In this paper, we illustrate skill controllers integrated within the SMACH framework.

Figure 3 shows the SMACH FSM of the general mission controller for the `go-and-track` task. In this FSM, the gray nodes are elements of the *SkillSet Interface* library: they are generated from skill models as SMACH action states (i.e., clients of the ROS Action lib), matching the skill model by providing as outcome all the possible terminal modes of the skill (as defined in Fig. 2). The link between these skill interface nodes and other internal states of the mission controller must then be designed by the mission programmer, by connecting outcomes of skills or internal events to other skills or states.

The mission FSM implements the following nominal sequence (when all skill succeed): takeoff (state `TO`), go to the waypoint (`GOTO`), track the object until identification (`TRACK`), then go home and land (`GOHOME`). If the authority is taken, the pilot may either perform a manual emergency landing, then *aborting* the mission; or (only if the authority is taken during the *goto* skill) perform a maneuver and give

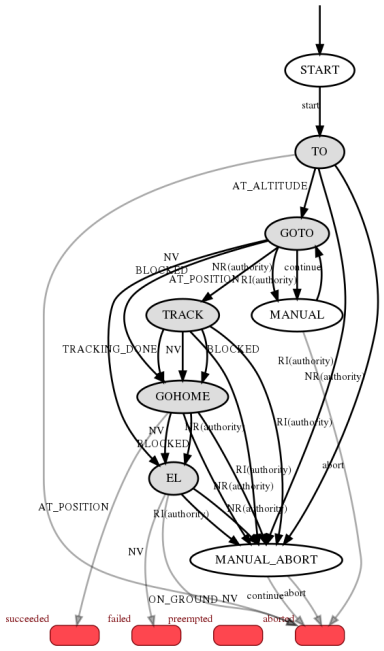

Fig. 3. SMACH FSM of the `go-and-track` controller

the authority back to the mission controller. If other failures occur, the FSM goes either to the `GOHOME` state or the emergency landing (`EL`) state depending on the mission step.

As a conclusion, we have described here the *operational* models that implement the execution of robot skills, made of a FSM-based *skill managers* on the functional layer, and on *skill interfaces* on the deliberative layer, that allow to integrate skill execution in the design of mission controllers. To ease the development of these components, we provide automated generation of resource and skill managers as ROS nodes, and generation of skill interfaces as a client library in several common frameworks, including SMACH.

B. Descriptive Models

From the skill models presented in Sec. III, we propose translation to descriptive models in order to perform on one hand automated planning, and on the other hand verification.

1) Automated Planning: Automated planning is a bunch of methods that compute action plans in order to achieve high-level objectives. Algorithms reasoning on state-action models mainly use the PDDL language [4] to model actions.

We then propose an automatic translation of skill models to PDDL domains. A PDDL domain defines some types, predicates or functions, and actions. To be as close as possible to the modeling level of skills, we consider a PDDL extension with durative actions (then integrating skill duration), and conditional effects. The translation of types and functions is straight-forward. Predicates correspond to an evaluation of the data of each robot as well as the state of its resources. Each robot skill is then translated into a durative action, integrating the management of resources in conditions and effects. As PDDL models are generally used to compute nominal plan, the PDDL generators accept to use only one terminal mode of the skill as its nominal mode. Listing 2 shows the PDDL description of the *goto* skill.

Listing 2. PDDL translation of the *goto* skill

```
(:durative-action M600_goto
    :parameters (
        ?r - M600Robot
        ?target - Position
        ?speed - Speed
        ?position - Position
    )
    :duration (= ?duration (time_to ?r ?position ?target ?speed))
    :condition (and
        (at start ( M600_position ?r ?position ))
        (at start ( M600_authority_AVAILABLE ?r ))
    )
    :effect (and
        (at start (not ( M600_authority_AVAILABLE ?r )))
        (at start ( M600_authority_USED ?r ))
        (at end (not ( M600_authority_USED ?r )))
        (at end ( M600_authority_AVAILABLE ?r ))
        (at start (not (M600_position ?r ?position )))
        (at end ( M600_position ?r ?target ))
    )
)
```

The PDDL problem to solve, i.e. the specific instance, cannot be generated from skill models, as it mainly relies on the environment in which the skills will be executed. In this example, such an environment model integrates the initial position of the M600 robot, the definition of the possible positions to reach, and the *time_to* function associated with them, as well as the final position to reach.

In our scenario, instead of considering that the `go-and-track` task would consist in only one *goto*, we have integrated automated planning in order to compute the sequence of *goto* to reach the position of the object of interest, based on a navigation graph whose edges are weighted according to the *time_to* function. We have then filled the PDDL problem from this graph, and used the Optic planner [23] to compute the sequence of *goto*.

2) Verification: Model-based description of skills allows using this model to verify that some desirable properties are satisfied by the model. From the models of robot skills, described using the DSL presented in Sec. III, we propose to generate state-machine models of skill execution, and translate these models so that they can be read by the NuSMV model-checker [24]. In order to perform some verification on the mission model, we also need to include in the NuSMV model: (1) a model of the mission controller, that represents the FSM of the mission, and (2) a model of the environment, also in the form of a FSM. In the case of the `go-and-track` task, we then model the controller FSM defined in SMACH (Fig. 3), as well as a model of the actions of the pilot on the authority resource. We then obtain a complete analyzable model including a detailed formalization of skill execution directly generated from the skill specification.

For the `go-and-track` task controller we have then been able to check the following LTL/CTL formulas:

a) The final success state is reachable:

$$\mathrm{EF}\ succeeded \tag{1}$$

This property holds in the analysed model.

b) If there is no skill failure and the pilot does not take control, then the mission succeeds:

$$\mathrm{G}[\neg skill_{failure} \wedge \neg manual] \rightarrow \mathrm{F}\ succeeded \tag{2}$$

where $skill\_failure$ is the disjunction of the failure modes of every skill, and $manual$ the *MANUAL* state of the authority resource. This property holds.

c) If the pilot does not take control, the mission does not fail:

$$\mathrm{G}[\neg manual] \rightarrow \mathrm{F}[succeeded \vee aborted] \tag{3}$$

This property does not hold, and NuSMV returns a counter-example where the emergency landing fails, leading to a failure of the mission. We can check that the failure of the emergency landing is then the only case leading to failure with the LTL formula:

$$(\mathrm{G}[\neg manual] \wedge \mathrm{G}[\neg EL_{failure}]) \rightarrow \mathrm{F}[succeeded \vee aborted] \tag{4}$$

which holds.

d) If there is no other failure than the authority taken by the pilot, then either the pilot ends the mission, or the mission succeeds:

$$\mathrm{G}[\neg skill_{failure} \wedge \neg skill_{NV}] \rightarrow \mathrm{F}[succeeded \vee aborted_M] \tag{5}$$

where $aborted_M$ is aborted via the `MANUAL_ABORT` state. This property does not hold, and the provided counter-example consists in an infinite trace where the pilot takes control during the goto and returns control to the mission controller infinitely. We remove such situation with:

$$\begin{aligned} \mathrm{G}[\neg skill_{failure} \wedge \neg skill_{NV} \wedge \neg \mathrm{G}[\mathrm{F}\ manual]] \\ \rightarrow \mathrm{F}[succeeded \vee aborted_M] \end{aligned} \tag{6}$$

which holds.

V. Conclusion

In this paper, we have proposed a formal specific language to specify robot skills. This language integrates a wide set of elements, that allow to derive several domain-specific models including *operational* models used for online execution, and *descriptive* models used for reasoning.

Regarding operational models, we have presented the manager architecture principle we propose, and the formal operational models built on a FSM on the manager side, and on an interface client library on the controller side.

Regarding descriptive models, we have presented on one hand the PDDL translation that allows to compute skill composition using temporal planning algorithms, and the use of skill models as NuSMV state machines in order to perform model-checking of mission properties that integrates models of skills and resources.

The clear advantage of the approach we have proposed is to automatically derive, from a common and formal specification of skills both the models used for online execution of these skills and the models used for automated planning or model-checking. This ensures that the descriptive models are consistent with what is actually executed. The definition of the language and some of the generation tools are available on:

`https://oara-architecture.gitlab.io/robot_skills`

References

[1] T. Lozano-Pérez, "Robot programming," *Proceedings of the IEEE*, vol. 71, pp. 821–841, 1983.

[2] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bogh, V. Krüger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282 – 291, 2016.

[3] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[4] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *JAIR*, vol. 20, pp. 61–124, 2003.

[5] A. David, G. Behrmann, K. G. Larsen, and W. Yi, "Unification & Sharing in Timed Automata Verification," in *SPIN*, Portland, OR, USA, 2003.

[6] V. Verma, T. Estlin, A. Jonsson, C. Pasareanu, R. Simmons, and K. Tso, "Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences," in *i-SAIRAS*, Munich, Germany, 2005.

[7] R. Simmons and D. Apfelbaum, "A Task Description Language for Robot Control," in *IROS*, Victoria, BC, Canada, 1998.

[8] C. Lesire and F. Pommereau, "ASPiC: An Acting System Based on Skill Petri Net Composition," in *IROS*, Madrid, Spain, 2018.

[9] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a Unified Behavior Trees Framework for Robot Control," in *ICRA*, Hong Kong, China, 2014.

[10] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *ICIT*, Singapore, 2015.

[11] M. R. Pedersen and V. Krüger, "Automated Planning of Industrial Logistics on a Skill-equipped Robot," in *IROS Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*, Hamburg, Germany, 2015.

[12] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen, "A deliberative architecture for AUV control," in *ICRA*, Pasadena, CA, USA, 2008.

[13] F. Ropero, P. Muñoz, and M. Moreno, "A Versatile Executive Based on T-REX for Any Robotic Domain," in *SGAI*, Cambridge, UK, 2018.

[14] S. Patra, M. Ghallab, D. Nau, and P. Traverso, "Acting and Planning Using Operational Models," in *AAAI*, Honolulu, HI, USA, 2019.

[15] M. Colledanchise, R. M. Murray, and P. Ögren, "Synthesis of correct-by-construction behavior trees," in *IROS*, Vancouver, Canada, 2017.

[16] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *IROS*, Vancouver, Canada, 2017.

[17] S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit, "Reactive high-level behavior synthesis for an Atlas humanoid robot," in *ICRA*, Stockholm, Sweden, 2016.

[18] C. Heinzemann and R. Lange, "vTSL - A Formally Verifiable DSL for Specifying Robot Tasks," in *IROS*, Madrid, Spain, 2018.

[19] J. Bohren and S. Cousins, "The SMACH High-Level Executive," *IEEE RA-M*, vol. 17, no. 4, pp. 18–20, 2010.

[20] L. Pitonakova, R. Crowder, and S. Bullock, "Behaviour-data relations modelling language for multi-robot control algorithms," in *IROS*, Vancouver, BC, Canada, 2017.

[21] E. A. Topp, M. Stenmark, A. Ganslandt, A. Svensson, M. Haage, and J. Malec, "Ontology-based knowledge representation for increased skill reusability in industrial robots," in *IROS*, Madrid, Spain, 2018.

[22] R. I. Brafman, M. Bar-Sinai, and M. Ashkenazi, "Performance level profiles: A formal language for describing the expected performance of functional modules," in *IROS*, Daejeon, South Korea, 2016.

[23] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *ICAPS*, Atibaia, São Paulo, Brazil, 2012.

[24] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A New Symbolic Model Verifier," in *CAV*, Trento, Italy, 1999.