Automated PDDL Domain File Generation for Enhancing Production System Development based on SysML Models

Hamied Nabizada[1,*], Tom Jeleniewski[1], Lasse Beers[1], Felix Gehlhoff[1] and Alexander Fay[2]

[1]Institute of Automation Technology, Helmut Schmidt University Hamburg, Holstenhofweg 85, 22043, Hamburg, Germany
[2]Chair of Automation, Ruhr University Bochum, Universitätsstrasse 150, 44801, Bochum, Germany

Abstract

As modern production systems become increasingly complex, traditional document-based systems engineering methods struggle to meet contemporary challenges, such as maintaining consistent knowledge bases across disciplines and ensuring early validation and verification processes. Model-Based Systems Engineering (MBSE) using Systems Modeling Language (SysML) provides a structured, model-centric approach for managing this complexity, yet the manual generation of AI planning descriptions in the Planning Domain Definition Language (PDDL) remains labor-intensive and error-prone. This paper presents an automated approach to generating PDDL domain files from SysML-based system models enriched with a specific SysML profile for PDDL, as part of a broader workflow aimed at enhancing planning processes in complex systems engineering. The proposed algorithm systematically extracts types, predicates, and actions from the enriched models, converting them into comprehensive PDDL descriptions suitable for automated planning systems. An Implementation using the Velocity Template Language (VTL) integrated into MBSE tools like Magic Systems of Systems Architect (MSoSA) demonstrates the approach's efficacy, as shown in an application case in aircraft fuselage assembly. This automation enhances the accuracy and efficiency of planning processes, facilitating rapid exploration and optimization of system configurations. The generated PDDL files were validated by integrating them into a PDDL solver, confirming the utility of the approach in real-world scenarios.

Keywords

Model-Based Systems Engineering, Systems Modeling Language, AI Planning, Planning Domain Definition Language

CEUR-WS.org/Vol-3883/paper3_IPS2.pdf

1. Introduction

The increasing complexity of modern production systems has rendered traditional document-based systems engineering methods inadequate for addressing today's challenges [2]. This includes the need for a unified knowledge base as system complexity increases, allowing data exchange across different engineering phases and disciplines [3] and ensuring that validation and verification processes can already be carried out during the early engineering phases [4]. To effectively manage this complexity, Model-Based Systems Engineering (MBSE) has emerged as a powerful methodology, providing a structured and model-centric approach to system development. MBSE facilitates a detailed, consistent, and comprehensive description of production systems, significantly improving collaboration across the various disciplines involved in the development process [5, 6].

Among the languages available for MBSE, the Systems Modeling Language (SysML) is particularly well-suited. SysML provides a broad range of modeling elements that can represent the structure, behavior, and requirements of a system [7]. By leveraging workflows such as those described in [8], SysML

AI4CC-IPS-RCRA-SPIRIT 2024: International Workshop on Artificial Intelligence for Climate Change, Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy. November 25-28th, 2024, Bolzano, Italy [1].
*Corresponding author.
hamied.nabizada@hsu-hh.de (H. Nabizada); tom.jeleniewski@hsu-hh.de (T. Jeleniewski); lasse.beers@hsu-hh.de (L. Beers); felix.gehlhoff@hsu-hh.de (F. Gehlhoff); alexander.fay@rub.de (A. Fay)
https://www.hsu-hh.de/aut/team/nabizada (H. Nabizada); https://www.hsu-hh.de/aut/team/jeleniewski (T. Jeleniewski); https://www.hsu-hh.de/aut/team/beers (L. Beers); https://www.hsu-hh.de/aut/team/gehlhoff (F. Gehlhoff); https://www.aut.ruhr-uni-bochum.de (A. Fay)
0000-0001-8251-837X (H. Nabizada); 0009-0007-0360-4108 (T. Jeleniewski); 0000-0003-0946-7742 (L. Beers); 0000-0002-8383-5323 (F. Gehlhoff); 0000-0002-1922-654X (A. Fay)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

can support a structured approach to system modeling that not only ensures consistency and early error detection but also enables exploration and comparison of various system configurations [9]. However, evaluating these alternatives requires both static and dynamic analysis of the system configurations [10].

One of the critical challenges in dynamic analysis is the allocation of individual process steps to the appropriate technical resources capable of executing them, enabling a detailed view of setup and cycle times [9]. As production systems become more complex, this manual process becomes increasingly time-consuming and labor-intensive. Moreover, process planning often must be repeated for every possible system configuration, further amplifying the effort required. This repetitive nature not only demands a high level of expertise to maintain accuracy but also increases the risk of human errors. Given these challenges, automating process planning during the engineering phase is highly desirable [11], as it can significantly reduce the time and effort needed to evaluate and select the optimal system configuration while minimizing errors and improving overall efficiency.

Artificial Intelligence (AI) planning could be a valuable approach for tackling dynamic analysis challenges in the engineering of modern production systems, as it can automatically generate the necessary sequences of steps and assign the appropriate resources to each step to achieve specific goals [12]. The Planning Domain Definition Language (PDDL) has emerged as the de facto standard for formalizing planning descriptions for AI planning solvers [13]. However, despite its widespread use, manually modeling PDDL descriptions remains a complex and time-consuming task that requires substantial domain knowledge and expertise [14]. Modeling these planning descriptions involves defining a domain file, which specifies the available actions, objects, and predicates for planning. The following listing (Listing 1) is a simple example of a PDDL domain file that defines actions, objects, and predicates for an assembly task:

Listing 1: Example of a simplified PDDL domain file for an assembly task

```
(define (domain assembly)

    (:requirements :strips :typing)

    (:types
        tool part location)

    (:predicates
        (at ?p - part ?l - location)
        (available ?t - tool)
        (in-assembly ?p - part)
        (assembled ?p - part))

    (:action move
        :parameters (?p - part ?from - location ?to - location)
        :precondition (and (at ?p ?from))
        :effect (and (not (at ?p ?from)) (at ?p ?to)))

    (:action start-assembly
        :parameters (?p - part ?t - tool)
        :precondition (and (available ?t) (not (in-assembly ?p)) (not (assembled ?p)))
        :effect (and (in-assembly ?p) (not (available ?t))))

    (:action finish-assembly
        :parameters (?p - part ?t - tool)
        :precondition (and (in-assembly ?p))
        :effect (and (assembled ?p) (not (in-assembly ?p)) (available ?t)))
)
```

This PDDL domain file provides a simple example of how **actions**, **objects**, and **predicates** are structured in an assembly task. The **requirements** section specifies that basic PDDL planning functions

and typed objects are used. In the **types** section, three object categories are defined: `tool`, `part`, and `location`. These types form the foundation for modeling the planning process.

The **predicates** describe the states of the system, such as a part's position (`at`), the availability of a tool (`available`), whether a part is being assembled (`in-assembly`), and when it is fully assembled (`assembled`).

The **actions** defines the operations within the domain. The `move` action represents moving a part from one location to another, with the precondition that the part must be at its current location, and the effect is updating the part's position. The `start-assembly` action initiates the assembly process, ensuring that the tool is available and the part is neither assembled nor in assembly. Once initiated, the part is marked as `in-assembly`, and the tool is made unavailable. The `finish-assembly` action completes the assembly process, transitioning the part from `in-assembly` to fully `assembled` and making the tool available again for other tasks.

This example illustrates how domain files formalize the necessary elements for automated planning, which will be critical for generating PDDL descriptions from SysML models, as discussed later in the paper.

Much of the information needed to construct at least the domain file is already embedded within SysML-based system models. These models are used to represent the structure, behavior, and constraints of production systems. However, extracting and organizing this information manually is a time-consuming process.

This paper introduces an approach for automating the generation of PDDL domain files by extracting information from SysML-based system models. It is part of the broader workflow introduced in [15], which focuses on enhancing the modeling and planning processes in complex systems engineering. The system models are enriched with PDDL aspects using the SysML profile for PDDL[1], ensuring that the generated PDDL files are both syntactically correct and semantically meaningful. By enabling the rapid and reliable generation of PDDL domain files, this method facilitates more effective decision-making in the engineering phase, supporting the exploration of multiple system configurations and improving overall system design.

The remainder of this paper is structured as follows. Section 2 provides an overview of related work. Section 3 presents the proposed algorithm for the automated PDDL description generation as a pseudocode. Section 4 demonstrates a possible implementation using the Velocity Template Language (VTL). Section 5 applies the algorithm to an application case in aircraft assembly. Finally, Section 6 concludes the paper and outlines future research.

2. Related Work

In this section, various approaches to generating PDDL descriptions are reviewed. These methods range from manual modeling techniques to automated solutions.

A SysML taxonomy for describing assembly tasks is proposed by Huckaby et al. [16], which provides a basis for deriving PDDL actions to represent system capabilities. However, this method involves manually creating PDDL descriptions, restricting its use to the predefined taxonomy. On the other hand, an ontology-based method to automatically generate PDDL descriptions is introduced by Vieira da Silva et al. [17], which aligns required and offered capabilities. Although this approach automates the generation process, it relies on a specific capability model and does not incorporate existing models.

To simplify planning in Hierarchical Domain Definition Language (HDDL), a PDDL dialect, Rimani et al. [18] compare elements of HDDL and SysML and propose a conceptual workflow for modeling in HDDL. While this method facilitates the organization of complex planning tasks, the conversion from SysML to HDDL still requires considerable manual effort. Their work, demonstrated in a space exploration context, points to the potential for greater automation in the future to enhance the efficiency and applicability of model-based planning.

[1] https://github.com/hsu-aut/SysML-Profile-PDDL

In a different domain, Wally et al. [19] present a method that translates ISA-95 manufacturing system models into PDDL descriptions for production planning. However, the method is tailored to ISA-95-compliant systems, which limits its applicability across other modeling standards and systems that are not based on ISA-95.

Focusing on robotics, Konidaris et al. [20] describe a method to create abstract symbolic representations that support high-level planning by learning from direct sensorimotor interactions rather than using existing models. Stoev et al. [21] aim to automate the creation of domain-specific symbolic models from text by extracting knowledge from instructional materials, such as cooking recipes, to generate PDDL descriptions automatically. Both approaches [20, 21], however, do not utilize existing models as a basis, which limits their ability to integrate with established frameworks and leverage pre-existing knowledge.

Although the approach of Konidaris et al. is data-driven rather than model-based, it merits special consideration because it use a probabilistic PDDL representation, which could be beneficial in contexts where uncertainty plays a significant role. However, in our approach, the need for structured, parametric PDDL representations generated directly from PDDL models is crucial for integration into MBSE workflows. The set-theoretic, non-parametric format of Konidaris et al. limits flexibility and reusability within MBSE, where parametrization is essential for describing complex system configurations. While the probabilistic aspect of PDDL could offer value under high uncertainty, our application context prioritizes deterministic planning, where consistency with the structured system model is critical.

Nabizada et al. [22] introduce a SysML profile specifically designed to enhance the modeling and planning of complex production systems by integrating planning logic through PDDL. The profile enables an PDDL-specific extension of existing SysML-based system models, resulting in consistent and synchronized updates of planning and system information. As a result, any changes made to the system model automatically lead to corresponding modifications in PDDL components. The applicability of the profile was validated in a real-world use case within the aerospace industry. However, utilizing this profile requires a structured methodology that outlines necessary steps to systematically extend system models with the profile.

In a related contribution, Nabizada et al. [15] provide the necessary structured methodology which is illustrated in Figure 1. It is a model-based workflow for generating PDDL descriptions that integrates both system and product models and uses the SysML Profile for PDDL. The proposed workflow comprises four phases, each detailing the steps required to extend existing system and product models and automate the generation of PDDL files.

In Phase IV, the focus is on automatically generating a domain description and a problem description. This phase is essential as it requires an algorithm capable of generating PDDL descriptions effectively. The present paper focuses specifically on this phase, introducing and detailing the algorithm developed for generating the PDDL domain descriptions.

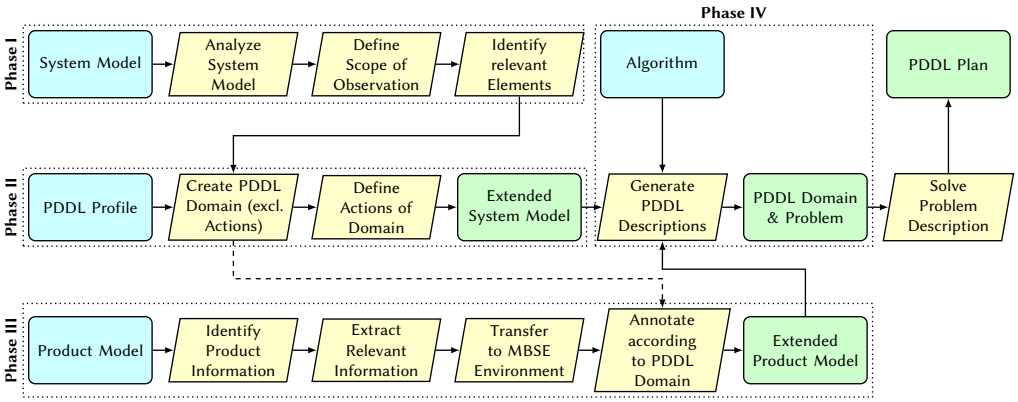

Figure 1: Workflow model for automated generation of PDDL descriptions by Nabizada et al. [15]

3. Algorithm for Generating PDDL Domain Files

The proposed algorithm automates the generation of a PDDL domain file from a SysML-based system model that is enriched with a specific SysML profile for PDDL, as described in [22]. This enrichment process integrates all necessary annotations and extensions into the SysML model to facilitate an accurate translation of its components into PDDL constructs.

The algorithm systematically extracts types, predicates, and actions from the enriched SysML model and converts them into a comprehensive PDDL domain file. Each step of the algorithm is designed to ensure that the resulting PDDL file is both syntactically correct and semantically meaningful, ready for use in automated planning systems.

The pseudocode for the algorithm, presented in Algorithm 1, illustrates the direct correspondence between the SysML model elements and their PDDL representations. The variables in the pseudocode align with the elements defined by the PDDL profile, providing a clear mapping and flow of data from the SysML model to the PDDL domain file. This structured approach enhances the development and optimization of production systems by ensuring a precise and efficient conversion process. This approach supports the engineering of production systems by enabling the generation of process sequences for various system configurations, which can then be evaluated afterwards.

Following this overview, the specific steps outlined in the pseudocode (Algorithm 1) will now be explained in detail.

Algorithm 1 Generation of the PDDL domain file

Input: SysML-based system model *sm*
Output: PDDL domain file *df*

```
function CreatePDDLDomain(sm)
    df ← InitializePDDLDomain(sm.PDDL_Domain)
    T ← ExtractTypes(sm.PDDL_Type)
    for each type in T do
        ProcessTypeToPDDL(df, type)
    end for
    P ← ExtractPredicates(sm.PDDL_Predicate)
    for each predicate in P do
        ProcessPredicateToPDDL(df, predicate)
    end for
    A ← ExtractActions(sm.PDDL_Action)
    for each action in A do
        ProcessActionToPDDL(df, action)
    end for
    return df
end function
```

1. Initialization of the PDDL Domain File The algorithm begins by initializing the PDDL domain file, denoted as *df*. This step involves setting up the base structure of the domain file based on the predefined domain specifications within the SysML model (*sm.PDDL_Domain*). By using the «PDDL_Domain» stereotype from the SysML profile, the model is annotated with all necessary planning-specific attributes, ensuring compatibility with PDDL syntax and semantics. This initialization phase prepares the domain file to correctly incorporate additional elements, such as types, predicates, and actions, which are defined in subsequent steps.

2. Extraction and Processing of Types Following the initialization, the algorithm extracts and processes the types defined in the SysML model, which are crucial for structuring the PDDL domain.

- **Extraction of Types**: The algorithm identifies all elements in the SysML model that have been annotated with the «PDDL_Type» stereotype. These elements represent the object classes needed for the PDDL domain, such as machines, products, resources, tools, and more. The extracted types are gathered into a set T. This extraction process takes advantage of the SysML profile's ability to clearly define type hierarchies and relationships, providing a structured and reusable framework for the PDDL domain.
- **Processing of Each Type**: For each type in the set T, the algorithm converts the SysML-based type information into the PDDL format. This includes translating any hierarchical relationships (e.g., supertype and subtype structures) defined within the SysML model into corresponding PDDL representations. The resulting types are then incorporated into the PDDL domain file (df), ensuring that all object categories are properly defined and structured according to the requirements of the planning task.

3. Extraction and Processing of Predicates The algorithm then focuses on predicates, which are essential for defining the logical structure of the planning domain.

- **Extraction of Predicates**: Predicates in the PDDL domain are defined using elements in the SysML model that have been annotated with the «PDDL_Predicate» stereotype. These predicates describe the conditions or states of the system that must be true or false during the planning process. The algorithm extracts these elements and compiles them into a set P. The use of derived properties in the SysML profile allows for a dynamic representation of predicate relationships and dependencies, ensuring that all logical conditions are accurately captured.
- **Processing of Each Predicate**: Each predicate in set P is transformed into its corresponding PDDL format. This involves defining the predicate's parameters and logical statements as required by PDDL. The predicates are then added to the domain file (df), providing a foundation for specifying preconditions and effects of actions in the planning domain.

4. Extraction and Processing of Actions Next, the algorithm processes the actions defined in the SysML model, which represent the operations that can be performed within the planning domain.

- **Extraction of Actions**: The algorithm extracts all elements in the SysML model annotated with the «PDDL_Action» stereotype. These elements define the various actions available in the domain, including their parameters, preconditions, and effects. These actions are collected into a set A. The SysML profile's customization elements, such as derived properties and possible owners, are leveraged to ensure that the extracted actions are contextually appropriate and syntactically valid.
- **Processing of Each Action**: Each action in set A is converted into PDDL syntax, specifying its preconditions and effects in terms of the predicates defined earlier. This step ensures that the actions are fully compatible with the PDDL requirements and that they reflect the intended operations and workflows defined in the SysML model. The processed actions are then added to the domain file (df), finalizing the dynamic aspects of the planning domain.

5. Finalization and Output of the PDDL Domain File Once all types, predicates, and actions have been added to the PDDL domain file (df), the algorithm completes the generation of a well-structured PDDL domain file. This file accurately represents the components and relationships defined in the enriched SysML model. With the domain file fully constructed, it is now ready for deployment in automated planning systems to generate and analyze production sequences for specific system configurations. The use of the SysML profile for PDDL simplifies the conversion of system models into planning-ready formats, enhancing the efficiency and effectiveness of production system engineering.

4. Implementation

The VTL was selected to implement the algorithm for automated PDDL description generation primarily due to its integration with MBSE tools like Magic Systems of Systems Architect (MSoSA). The Velocity Engine, which utilizes VTL, is embedded in these tools, enabling efficient and seamless automation within existing modeling environments. While VTL is well-suited for this task, the approach is flexible and could be implemented using other templating languages or scripting methods, depending on the specific requirements and tools available.

VTL is a templating language that facilitates dynamic text generation, making it particularly useful for converting SysML-based system models into PDDL domain files. Its straightforward syntax and powerful constructs, such as loops, conditionals, and macros, are ideal for generating structured documents like PDDL files.

Listing 2 shows a core component of the VTL script, focusing on the automated generation of actions in the PDDL domain file.[2] The script iterates over the actions defined in the SysML model and dynamically generates their PDDL representations, including parameters, preconditions, and effects. Each element is checked for completeness and validity to ensure that the generated PDDL file is both syntactically correct and semantically meaningful.

Listing 2: Snippet of the VTL Template for defining PDDL actions

```
#foreach ($actionElement in $pddlDomainElement.action)
    #set($actionName = $actionElement.name)
    ## Check if action has a name and defined parameters, preconditions, and effects
    #if($actionName && $actionElement.parameters && $actionElement.precondition && $actionElement.effect)
    (:action $actionName
        ## Generate parameters with type checking
        :parameters (#foreach($parameterElement in $actionElement.parameters)
                        #set($parameterName = $parameterElement.variablename.variable)
                        #set($parameterTypeName = $parameterElement.variable_is_of_type.name)
                        #if($parameterName && $parameterTypeName)
                            $parameterName - $parameterTypeName
                        #else
                            ERROR: MISSING PARAMETER NAME OR TYPE FOR ACTION $actionName #end
                    #end)
        ## Generate preconditions
        #set($preconditions = "")
        #set($preconditionCount = 0)
        #foreach($preconditionElement in $actionElement.precondition.predicates_have_to_be_true.
             predicates_are_switched)
            #set($preconditionPredicateName = $preconditionElement.predicatename.predicatename)
            #if($preconditionPredicateName)
                #if($preconditionElement.inverse)
                    #set($preconditions = "$preconditions (not ($preconditionPredicateName #foreach(
                         $parameter in $preconditionElement.parameter) $parameter.variable #end))")
                    #else
                    #set($preconditions = "$preconditions ($preconditionPredicateName #foreach($parameter
                         in $preconditionElement.parameter) $parameter.variable #end)")#end
                #set($preconditionCount = $preconditionCount + 1)
            #else
                ERROR: MISSING PREDICATE NAME FOR PRECONDITION IN ACTION $actionName #end #end
        #if($preconditionCount > 1)
            :precondition (and $preconditions)
        #elseif($preconditionCount == 1)
            :precondition $preconditions #end
        ## Generate effects
        #set($effects = "") #set($effectCount = 0)
        #foreach($effectElement in $actionElement.effect.logical_expression.predicates_are_switched)
            #set($effectPredicateName = $effectElement.predicatename.predicatename)
            #if($effectPredicateName)
```

[2]The complete template is available on GitHub: https://github.com/hsu-aut/VTL-PDDL_Domain.

```
                #if($effectElement.inverse)
                    #set($effects = "$effects (not ($effectPredicateName #foreach($effectParameter in
                        $effectElement.parameter) $effectParameter.variable #end))")
                #else
                    #set($effects = "$effects ($effectPredicateName #foreach($effectParameter in
                        $effectElement.parameter) $effectParameter.variable #end)") #end
                #set($effectCount = $effectCount + 1)
            #else
                ERROR: MISSING PREDICATE NAME FOR EFFECT IN ACTION $actionName #end #end
        #if($effectCount > 1)
            :effect (and $effects)
        #elseif($effectCount == 1)
            :effect $effects #end
    )
    #else
        ERROR: ACTION $actionElement IS MISSING NAME, PARAMETERS, PRECONDITIONS, OR EFFECTS
#end #end
```

Action Iteration and Name Assignment: The script begins by iterating over each action element within the `$pddlDomainElement`. For each action, the script assigns the action's name to the variable `$actionName`.

Parameter Generation: Parameters for each action are generated by iterating over the `parameters` attribute of the action element. The script checks whether each parameter and its type are defined. If any are missing, an error message is generated. This process ensures that every action has a complete and valid parameter list, which is essential for the correct functioning of the planning domain.

Precondition Generation: The script constructs the preconditions for each action by iterating over the `precondition` attribute. It dynamically builds the logical expressions required by PDDL, handling both normal and negated predicates. If multiple preconditions are defined, they are grouped using the operator `and`. This part of the script is crucial for defining the conditions under which each action can be executed, ensuring that all necessary logical constraints are accurately represented.

Effect Generation: Similarly, the script handles the effects of each action. It iterates over the `effect` attribute and constructs the corresponding PDDL expressions. The script also accounts for negated effects where necessary. Correctly handling action effects is vital for defining the outcome of executing each action in the planning domain, ensuring that the resulting state changes are accurately modeled.

Error Handling: Throughout the code, various checks are included to ensure the integrity of the generated PDDL file. If any action is missing critical information, such as a name, parameters, preconditions, or effects, the script generates an error message. This robust error-checking mechanism helps prevent incomplete or invalid PDDL files from being produced, thereby enhancing the reliability and correctness of the automated planning process.

5. Application in Aircraft Fuselage Assembly

To evaluate the proposed algorithm, an aircraft manufacturing use case was selected. Initially, holes are drilled from the outside of the fuselage, after which fasteners are installed. Inside the fuselage, collars are then screwed onto these fasteners. This process, which is repetitive and ergonomically challenging, is typically performed by a robotic system.

The robotic system utilized for the tasks inside the fuselage comprises a UR10 manipulator equipped with various end-effectors, each capable of screwing different types of collars onto the rivets. Identifying an optimal plan for this task poses a significant challenge. The robotic system must follow the shortest

possible path while minimizing the number of end-effector reconfigurations needed during the assembly process.

For the development of this robotic system, a detailed system model was created using SysML in the software MSoSA. A comprehensive description of the use case and the system model is presented in [9]. This system model was subsequently extended using the PDDL profile. The PDDL profile facilitates the annotation of SysML elements that contain relevant information necessary for constructing the corresponding PDDL file.

Figure 2 shows an excerpt from the activity diagram of the enriched system model, showcasing the integration between the system's modeled behavior and the PDDL profile. It includes three «PDDL_Action» elements, each accompanied by its respective preconditions and effects. The «PDDL_Action» elements "ScrewCollarTypeA" and "ScrewCollarTypeB" describe the process steps for screwing different types of collars. After completing these steps, the system proceeds to the next rivet through the «PDDL_Action» "MoveToNextRivet". This sequence is repeated cyclically throughout the assembly process.

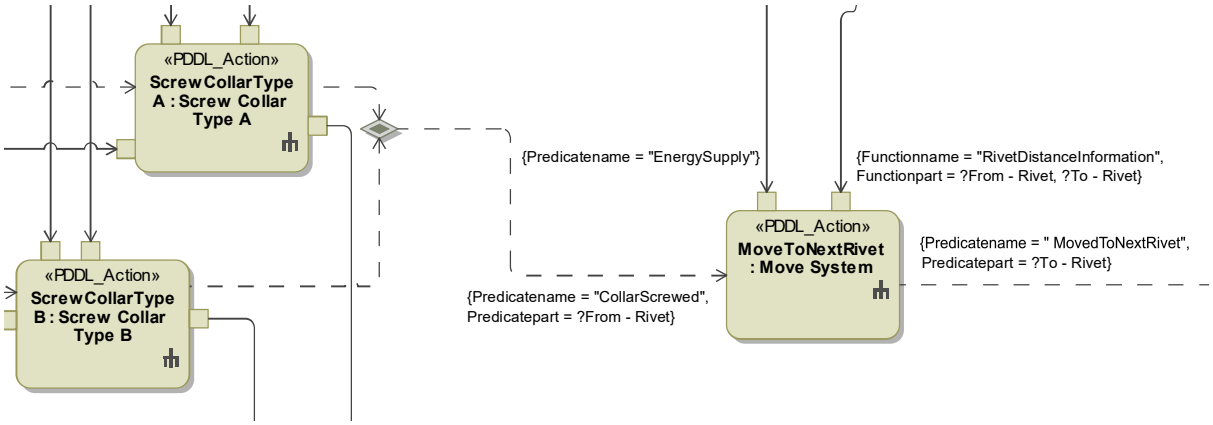

Figure 2: Excerpt from the enriched system model [22]

To execute the action "MoveToNextRivet," which moves the end-effector from one rivet to the next, the following preconditions must be satisfied:

1. The system must be powered, which is ensured by the predicate "EnergySupply" included in the action.
2. The collar at the current end-effector position must already be screwed. This is managed by the actions "ScrewCollarTypeA" and "ScrewCollarTypeB," which set the predicate "CollarScrewed" to "True" once their respective actions are completed.
3. The coordinates of the rivets and the distances between them are defined in the *Function* "RivetDistanceInformation" and are used in the "MoveToNextRivet" action. A PDDL solver can utilize these distances and rivet types to determine the optimal sequence for processing rivets.

Once the action is executed, its effect is that the end-effector is positioned at a new working location. The corresponding predicate "MovedToNextRivet" is set to "True" with the new rivet position.

The automated generation of the PDDL domain files using the VTL template allows for efficient and accurate conversion of the system model into planning descriptions. Listing 3 presents the PDDL action "MoveToNextRivet," which has been automatically generated using the VTL template for PDDL domain file generation. This example illustrates how the algorithm captures the necessary parameters, preconditions, and effects directly from the enriched SysML model.

The generated PDDL domain files, produced through the automated process using the VTL template, were subsequently provided to a PDDL solver for creating the sequence of process steps. By maintaining a strict linkage between the PDDL descriptions and the SysML system model, this approach allows for a

Listing 3: PDDL action definition for ”MoveToNextRivet” with preconditions and effects

```
(:action MoveToNextRivet
    :parameters (?From - Rivet ?To - Rivet)
    :precondition (and
                    (CollarScrewed ?From)
                    (EnergySupply)
                   )
    :effect (and
                (MovedToNextRivet ?To )
                (not (CollarScrewed ?From ))
                (increase (total-cost) (RivetDistanceInformation ?From ?To))
            )
)
```

seamless exploration of different system configurations and their behaviors. This integration facilitates more efficient planning and decision-making, enabling engineers to quickly assess the impact of various configurations and optimize the production process accordingly. The close alignment between the model and the planning descriptions ensures consistency and accuracy, making it easier to adapt and refine strategies based on specific requirements and constraints.

6. Conclusion and Outlook

This paper presented an automated approach for generating PDDL domain files directly from SysML-based system models, leveraging the SysML profile for PDDL. The automation is achieved through a newly developed algorithm, which systematically extracts the PDDL annotated elements from the enriched system models, thereby reducing the manual effort typically required for creating such files. The integration of this automation into MBSE tools like MSoSA, utilizing the VTL, demonstrates the feasibility of embedding the approach into existing engineering workflows.

The application of this method to an aircraft assembly use case highlights the potential of automating process planning in complex production environments. By maintaining a strict linkage between the system model and the generated PDDL domain, engineers can easily explore and evaluate different system configurations. This capability improves the overall efficiency of the system design and decision-making processes, enabling faster and more accurate planning.

However, one notable drawback is the initial complexity of setting up the system model. While the proposed approach reduces manual work in the long term, it requires a significant amount of effort to initially extend the SysML models with the PDDL profile. Engineers must ensure that the system model is correctly annotated and that all relevant elements are configured properly for automation. This setup process can be time-consuming and presents a steep learning curve for teams unfamiliar with SysML or PDDL, potentially delaying the benefits of automation in the early stages of implementation.

To address this drawback, the standardization of system models could play a crucial role in reducing the complexity of the annotation process. By adhering to well-defined modeling standards and conventions, it becomes easier to identify the relevant elements in SysML models and apply PDDL annotations systematically. Standardized system models would ensure consistency across different projects, enabling automated tools to more easily detect, annotate, and extract relevant information. In turn, this would reduce the manual effort required during the initial setup and streamline the overall process.

Additionally, the challenge of annotation complexity could be further addressed by developing an intelligent assistance tool to support engineers during the modeling process. Such a tool would provide suggestions for annotations based on the user’s modeling intent, leveraging the context of the current model elements and their relations. By incorporating advanced techniques, potentially including large language models (LLMs), the tool could predict suitable PDDL annotations dynamically, streamlining the setup process even further. This interactive assistance would not only minimize manual annotation

efforts and reduce errors but also enhance accessibility for teams with varying levels of familiarity with SysML and PDDL.

Future work will focus on extending the current approach by automating the generation of PDDL problem files from product models. This step is essential for fully automating the planning process, as the problem file defines the initial state and goals of the planning task. Additionally, the method will be applied to even more complex production systems, involving multiple process steps and a wider variety of technical components, to further evaluate its scalability and robustness in different industrial contexts.

Furthermore, the developed algorithm is designed to be broadly applicable and can be implemented in other environments, such as the Unified Planning Framework (UPF)[3]. By leveraging the UPF, the approach could offer direct access to various PDDL solvers, streamlining the process of selecting and using the most appropriate solver for each use case. This flexibility will enhance the adaptability of the algorithm across different tools and platforms, making it more accessible for diverse planning challenges in industry.

Acknowledgments

This research paper [project iMOD and LaiLa] is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr. dtec.bw is funded by the European Union – NextGenerationEU.

References

[1] D. Aineto, R. De Benedictis, M. Maratea, M. Mittelmann, G. Monaco, E. Scala, L. Serafini, I. Serina, F. Spegni, E. Tosello, A. Umbrico, M. Vallati (Eds.), Proceedings of the International Workshop on Artificial Intelligence for Climate Change, the Italian workshop on Planning and Scheduling, the RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and the Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (AI4CC-IPS-RCRA-SPIRIT 2024), co-located with 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), CEUR Workshop Proceedings, CEUR-WS.org, 2024.

[2] S. Friedenthal, A. Moore, R. Steiner, A practical guide to SysML: The systems modeling language, Morgan Kaufmann, 2015. doi:10.1016/C2013-0-14457-1.

[3] J. D'Ambrosio, G. Soremekun, Systems engineering challenges and MBSE opportunities for automotive system design, in: 2017 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, 2017, pp. 2075–2080. doi:10.1109/SMC.2017.8122925.

[4] A. M. Madni, M. Sievers, Model-based systems engineering: Motivation, current status, and research opportunities, Systems Engineering 21 (2018) 172–190. doi:10.1002/sys.21438.

[5] K. Henderson, A. Salado, Value and benefits of model–based systems engineering (MBSE): Evidence from the literature, Systems Engineering 24 (2021) 51–66. doi:10.1002/sys.21566.

[6] M. M. Schmidt, R. Stark, Model-Based Systems Engineering (MBSE) as computer-supported approach for cooperative systems development, in: Proceedings of 18th European Conference on Computer-Supported Cooperative Work, European Society for Socially Embedded Technologies (EUSSET), 2020. doi:10.18420/ecscw2020_ep04.

[7] OMG, Systems Modeling Language (SysML™ 1.6), 2019. URL: https://www.omg.org/spec/SysML/1.6/PDF.

[8] T. Weilkiens, SYSMOD - The Systems Modeling Toolbox: Pragmatic MBSE with SysML, Lulu.com, 2016.

[9] L. Beers, M. Weigand, H. Nabizada, A. Fay, MBSE Modeling Workflow for the Development of Automated Aircraft Production Systems, in: 28th International Conference on Emerging Technologies and Factory Automation (ETFA), 2023, pp. 1–8. doi:10.1109/ETFA54631.2023.10275444.

[3] https://github.com/aiplan4eu/unified-planning

[10] K. Pohl, H. Hönninger, R. Achatz, M. Broy, Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-34614-9.

[11] B. Alkan, R. Harrison, A virtual engineering based approach to verify structural complexity of component-based automation systems in early design phase, Journal of Manufacturing Systems 53 (2019) 18–31. doi:10.1016/j.jmsy.2019.09.001.

[12] C. Mayr-Dorn, A. Egyed, M. Winterer, C. Salomon, H. Fürschuß, Evaluating PDDL for programming production cells: a case study, in: IEEE/ACM 4th International Workshop on Robotics Software Engineering (RoSE), 2022, pp. 17–24. doi:10.1145/3526071.3527519.

[13] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, Cambridge University Press, 2016. doi:10.1017/CBO9781139583923.

[14] A. Lindsay, On Using Action Inheritance and Modularity in PDDL Domain Modelling, Proceedings of the International Conference on Automated Planning and Scheduling 33 (2023) 259–267. doi:10.1609/icaps.v33i1.27203.

[15] H. Nabizada, T. Jeleniewski, F. Gehlhoff, A. Fay, Model-based Workflow for the Automated Generation of PDDL Descriptions, in: 29th International Conference on Emerging Technologies and Factory Automation (ETFA), 2024, pp. 1–4. doi:10.1109/ETFA61755.2024.10710982.

[16] J. Huckaby, S. Vassos, H. I. Christensen, Planning with a task modeling framework in manufacturing robotics, in: International Conference on Intelligent Robots and Systems, 2013, pp. 5787–5794. doi:10.1109/IROS.2013.6697194.

[17] L. M. Vieira da Silva, R. Heesch, A. Köcher, A. Fay, Transformation eines Fähigkeitsmodells in einen PDDL-Planungsansatz, at-Automatisierungstechnik 71 (2023) 105–115. doi:10.1515/auto-2022-0112.

[18] J. Rimani, C. Lesire, S. Lizy-Destrez, N. Viola, Application of MBSE to model Hierarchical AI Planning problems in HDDL, in: International Conference on Automated Planning and Scheduling (ICAPS), 2021.

[19] B. Wally, J. Vyskočil, P. Novák, C. Huemer, R. Šindelár, P. Kadera, A. Mazak, M. Wimmer, Flexible production systems: Automated generation of operations plans based on ISA-95 and PDDL, IEEE Robotics and Automation Letters 4 (2019) 4062–4069. doi:10.1109/LRA.2019.2929991.

[20] G. Konidaris, L. P. Kaelbling, T. Lozano-Perez, From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning, Journal of Artificial Intelligence Research 61 (2018) 215–289. doi:10.1613/jair.5575.

[21] T. Stoev, T. Sosnowski, K. Yordanova, A Tool for Automated Generation of Domain Specific Symbolic Models From Texts, in: IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), 2023, pp. 276–278. doi:10.1109/PerComWorkshops56833.2023.10150252.

[22] H. Nabizada, L. Beers, M. Weigand, F. Gehlhoff, A. Fay, SysML-Profil zur automatisierten Generierung von PDDL-Beschreibungen aus Systemmodellen, in: Tagung Entwurf komplexer Automatisierungssysteme (EKA), 2024. doi:10.25673/116041.