Bridging Engineering and AI Planning through Model-Based Knowledge Transformation for the Validation of Automated Production System Variants

Hamied Nabizada[1,*], Lasse Beers[1], Alain Chahine[2], Felix Gehlhoff[1], Oliver Niggemann[1], Alexander Fay[3]

[1]Institute of Automation Technology, Helmut-Schmidt-University Hamburg, Hamburg, Germany
[2]Airbus Operations GmbH, Hamburg, Germany
[3]Chair of Automation, Ruhr University Bochum, Bochum, Germany

Abstract

Engineering models created in Model-Based Systems Engineering (MBSE) environments contain detailed information about system structure and behavior. However, they typically lack symbolic planning semantics such as preconditions, effects, and constraints related to resource availability and timing. This limits their ability to evaluate whether a given system variant can fulfill specific tasks and how efficiently it performs compared to alternatives.

To address this gap, this paper presents a model-driven method that enables the specification and automated generation of symbolic planning artifacts within SysML-based engineering models. A dedicated SysML profile introduces reusable stereotypes for core planning constructs. These are integrated into existing model structures and processed by an algorithm that generates a valid domain file and a corresponding problem file in Planning Domain Definition Language (PDDL). In contrast to previous approaches that rely on manual transformations or external capability models, the method supports native integration and maintains consistency between engineering and planning artifacts.

The applicability of the method is demonstrated through a case study from aircraft assembly. The example illustrates how existing engineering models are enriched with planning semantics and how the proposed workflow is applied to generate consistent planning artifacts from these models. The generated planning artifacts enable the validation of system variants through AI planning.

1 Introduction

Engineering automated production systems involves complex decisions that affect not only the physical structure of a system but also its operational feasibility and efficiency. *Model-Based Systems Engineering* (MBSE) has become a widely adopted approach to support this process by enabling consistent, tool-supported modeling of system structure and behavior (Henderson and Salado 2021). Through integrated models, engineers can represent cross-disciplinary aspects of a system and maintain consistency throughout the development lifecycle (Schmidt and Stark 2020).

Systems Modeling Language (SysML) provides a standardized notation for creating such models and supports the representation of structure, interfaces, and system functions (Weilkiens 2020). While SysML models capture rich technical detail, they typically do not include planning semantics such as action preconditions, resource constraints, or temporal dependencies. These aspects are essential for task-oriented planning and decision support.

As a result, engineers are often unable to use existing system models to answer critical planning questions. Examples include whether a specific system variant can fulfill a given task, or how well it performs compared to alternative system variants. These questions are essential for validating functional feasibility and for evaluating key performance indicators such as cycle time, idle time, or reconfiguration effort (Törmänen et al. 2017).

Symbolic *AI Planning* provides formal mechanisms to generate action sequences that satisfy goal conditions under defined constraints (Ghallab, Nau, and Traverso 2016). *Planning Domain Definition Language* (PDDL) is the established standard for describing such planning problems (Haslum et al. 2019). However, manually creating PDDL descriptions is time-consuming and error-prone (Lindsay 2023; Sleath and Bercher 2024). Translating engineering models into planning representations typically requires additional manual effort and introduces a disconnect between the system model and the planning logic (Köcher et al. 2022). While planning tools such as domain editors or syntax checkers exist (Strobel and Kirsch 2020), they are external to engineering workflows and require expertise not typically found among system engineers.

A central challenge is that system models and planning descriptions follow fundamentally different modeling logics. System models focus on structural and behavioral aspects of technical systems, emphasizing physical consistency and implementation detail. Planning descriptions, in contrast, rely on an abstract, symbolic representation of actions, including preconditions and effects, to evaluate goal satisfaction under given constraints. Translating between them requires not only syntactic conversion, but also a consistent way to represent planning semantics within the structure of engineering models.

To address this limitation, a structured workflow was developed to connect system and product models with planning semantics and to enable automated generation of PDDL domain and problem files from engineering artifacts (Nabizada et al. 2024b). As part of this workflow, a dedicated SysML profile was introduced to allow planning

constructs such as types, predicates, and actions to be embedded directly into system models (Nabizada et al. 2025). A transformation algorithm complements the approach by extracting the annotated model content and generating syntactically valid PDDL files (Nabizada et al. 2024a).

This paper integrates the previously introduced workflow, SysML profile, and transformation algorithm into a complete method for generating planning descriptions from engineering models. The focus lies on demonstrating the practical application of this approach in a case study from aircraft assembly, including the enrichment of engineering models with planning semantics and the automated generation of PDDL files.

The remainder of this paper is structured as follows. Section 2 discusses related work on planning model generation and MBSE integration. Section 3 outlines the four-phase workflow for embedding planning semantics into system and product models. Section 4 introduces the SysML profile used to represent planning constructs. Section 5 describes the transformation algorithm for generating PDDL files. Section 6 presents a case study from aircraft assembly and evaluates the resulting plans. Finally, Section 7 summarizes the contributions and discusses directions for future research.

2 Related Work

A range of approaches exists for generating planning descriptions from structured representations. These differ in modeling formalism, degree of automation, and the handling of planning semantics.

Huckaby, Vassos, and Christensen (2013) introduced a SysML-based taxonomy for assembly tasks that provides the basis for deriving PDDL actions manually. The approach supports the representation of system capabilities but does not automate the creation of planning files and is limited to predefined taxonomic structures.

Vieira da Silva et al. (2023) proposed an ontology-based transformation of capability descriptions into PDDL by matching required and offered functions. While this enables automated generation of planning descriptions, it relies on a separate capability model and does not leverage existing engineering models.

Rimani et al. (2021) propose a conceptual mapping between SysML-based functional architectures and the *Hierarchical Domain Definition Language* (HDDL). The approach enables the structuring of hierarchical planning domains from engineering models but remains largely manual. It lacks a dedicated profile, formal validation mechanisms, and support for traceability between planning artifacts and system models.

Wally et al. (2019) present a model-driven approach for generating PDDL representations from ISA-95-compliant manufacturing system models. The method defines metamodel-level mappings and supports automated generation of domain and problem files. While effective in industrial automation contexts, it is limited to ISA-95-compliant models, making it unsuitable for integration with general-purpose modeling frameworks such as SysML.

Konidaris, Kaelbling, and Lozano-Perez (2018) present a data-driven approach for learning symbolic AI planning models from sensorimotor trajectories. The method derives abstract state transitions and operator models by analyzing the effects of high-level robotic skills in continuous environments. The resulting models are tailored to specific tasks and environments and are not parameterized or linked to reusable engineering artifacts. As such, they are not suitable for integration with structured system modeling environments where consistency, traceability, and reuse are required.

Stoev, Sosnowski, and Yordanova (2023) present a tool that automatically extracts symbolic planning models in PDDL from instructional texts. While effective in knowledge extraction from unstructured sources, the approach is not integrated into model-based engineering workflows and does not leverage formal system representations.

Recent work has explored the use of large language models (LLMs) for generating symbolic AI planning models from natural language input. Tantakoun, Zhu, and Muise (2025) provide a comprehensive survey and taxonomy of this emerging line of research, framing LLMs as modelers rather than solvers. They distinguish between model generation, model editing, and model benchmarking, and highlight the potential of LLMs to create domain and problem descriptions in PDDL from textual input. These methods demonstrate strong syntactic capabilities and can generate plausible planning structures without dedicated modeling tools. However, they often suffer from limited semantic accuracy and inconsistencies in domain logic. While hybrid approaches combining LLM output with validation tools can mitigate some issues, the generated models typically lack integration with engineering data sources and are not traceable to structured system descriptions.

In contrast, the approach presented in this paper builds on formally defined system and product models created in MBSE environments. Rather than inferring planning knowledge from text, it uses stereotype annotations to embed planning constructs directly into SysML artifacts. This ensures consistency with system architecture and enables traceable and automated generation of domain and problem files from verified engineering models.

3 Workflow for Automated Generation of PDDL Descriptions

In symbolic planning, a plan consists of a sequence of actions that transform an initial state into a goal state, based on a formally defined planning domain (Ghallab, Nau, and Traverso 2016). Each action has parameters, preconditions, and effects, and may depend on resource availability, spatial relations, or system constraints such as energy or tool compatibility. To generate a valid plan, the domain must define object types, predicates, and actions, while the problem instance specifies the concrete objects involved, the initial conditions, and the goal (Haslum et al. 2019). This structure imposes specific requirements on the modeling of technical systems: symbolic relations between system elements must be made explicit, and behavioral aspects must be translated

into symbolic actions with parameters, preconditions, and effects.

Integrating planning capabilities into engineering models enables consistent evaluation of system variants and task feasibility. In practice, however, system and product data are often maintained in separate tools, making it difficult to generate and maintain consistent planning artifacts. To address this, a structured workflow was developed that connects SysML-based system models with product data and formal planning constructs. The workflow supports the automated generation of a PDDL domain and a PDDL problem file and was introduced in earlier work (Nabizada et al. 2024b). An overview is shown in Figure 1.

Phase I addresses the challenge of working with existing system models that were not originally created with planning tasks in mind. These models often contain heterogeneous information, including structural, behavioral, and functional aspects of technical systems (Henderson and Salado 2021; Schmidt and Stark 2020). Users applying the workflow are typically not involved in the creation of the model and must therefore analyze its content to understand the system architecture and the terminology used (Nabizada et al. 2024b).

Since such models may describe complex and large-scale systems, for example entire production facilities (Törmänen et al. 2017), it is necessary to restrict the scope of analysis. In many cases, the planning problem concerns only a specific machine, an individual production cell, or a functional module. A scoping step is therefore required to isolate the parts of the model that are relevant for planning and to exclude unrelated information. Within this defined scope, relevant components, resources, and their relationships are identified. This step ensures that the subsequent enrichment of the model focuses only on information needed for the planning domain (Nabizada et al. 2024b). To gain this understanding, engineers must review the model structure, analyze relevant block and activity diagrams, and identify consistent terminology and modeling patterns that describe system elements and their behavior.

In **Phase II**, the system model is extended with planning semantics using a dedicated SysML profile (Nabizada et al. 2025). This profile introduces stereotypes for core PDDL constructs and allows the integration of domain knowledge directly into the model. The enrichment begins with the specification of object types and predicates, which define structural elements and relationships relevant for planning.

Based on these definitions, actions are defined that describe system behavior in terms of parameters, preconditions, and effects. All planning elements are modeled consistently within the system architecture and linked to existing structures where applicable. The result is an extended system model that contains the complete domain information required for generating a valid PDDL domain file. The structure and function of the SysML profile used in this phase are described in more detail in Section 4.

In **Phase III**, information from the product model is incorporated to define the specific planning task. While the system model provides the available structure and capabilities, the product model contains instance-level data such as the positions and types of components required for problem definition (Beers et al. 2024b). Relevant information is extracted from the product model and transferred into the MBSE environment, where it is annotated using the planning concepts previously defined in the system model. By referencing the same types and predicates, consistency between domain and problem description is ensured. The result is an extended product model, integrated into the MBSE environment, that serves as the basis for generating the PDDL problem file.

In **Phase IV**, the information embedded in the system and product models is used to automatically generate the corresponding PDDL domain and problem file. Since all relevant planning elements have been defined in the previous steps, this generation can be carried out without additional manual effort. A transformation algorithm processes the annotated model content and converts it into syntactically correct PDDL descriptions using a template-based approach. The details of this algorithm are described in Section 5.

The resulting domain file contains all types, predicates, and actions as specified in the system model, while the problem file reflects the initial state and goal conditions derived from the product data. These files can be passed to a standard PDDL solver to compute a valid plan. Because the planning artifacts are derived directly from the models, changes to the system or product can be propagated automatically, ensuring that the planning logic remains aligned with the current system variant.

4 SysML Profile for PDDL Integration

Standard SysML[1] does not provide native constructs for expressing symbolic planning semantics such as typed action parameters, logical preconditions, or effects. To address this, a dedicated SysML profile was developed and introduced in Nabizada et al. (2025). It is based on the *Backus-Naur-Form* (BNF) specification of PDDL 3.1 (Kovacs 2011) and enables core planning concepts to be represented as stereotypes within SysML models. The profile supports model-based specification of planning descriptions and enables the consistent generation of PDDL files from annotated engineering models. An overview of the mapping between PDDL constructs and SysML elements is provided in Table 1.

Each planning construct defined in the PDDL specification is represented in the profile by a dedicated stereotype. These stereotypes extend selected metaclasses from the SysML or UML metamodel and serve to embed planning-specific semantics into existing modeling constructs.

The stereotype `<<PDDL_Domain>>` is applied to elements of type `Model` or `Package`. It provides the structural container for a planning domain and groups all relevant model elements, including types, predicates, and actions, under a consistent planning context. This approach allows domain definitions to be integrated into larger system models while remaining encapsulated.

[1]For a detailed introduction to SysML concepts including stereotypes, metaclasses, and UML-based metamodeling, see (Friedenthal, Moore, and Steiner 2014).

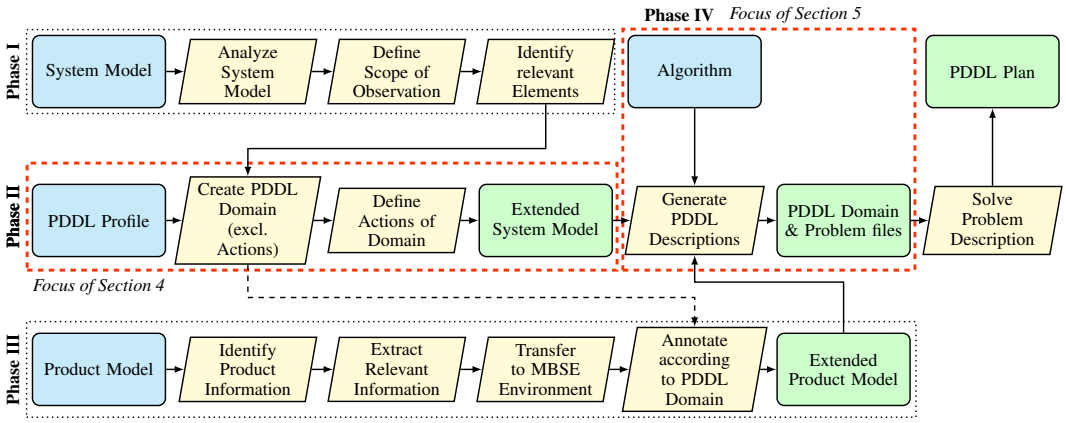

Figure 1: Workflow model for automated generation of PDDL descriptions (adapted from Nabizada et al. (2024b)).

PDDL Concept	Applied Stereotype	Extended SysML Metaclass
Domain Definition	`<<PDDL_Domain>>`	`Model`, `Package`
Object Type	`<<PDDL_Type>>`	`Class`
Logical Condition	`<<PDDL_Predicate>>`	`ObjectFlow`, `ControlFlow`
Numeric Expression	`<<PDDL_Function>>`	`ObjectFlow`, `ControlFlow`
Action Specification	`<<PDDL_Action>>`	`CallBehaviorAction`

Table 1: Mapping of PDDL Concepts to SysML Elements (Nabizada et al. 2025)

Object types are defined using `<<PDDL_Type>>`, which extends the metaclass `Class`. These types are used to categorize domain objects and to define the types of symbolic parameters used in planning actions. Such parameters represent abstract references to objects manipulated by an action, for example the part to assemble or the tool to use. Inheritance relations between types can be modeled using generalization mechanisms available in *Unified Modeling Language* (UML) and SysML.

Logical conditions and numerical values are described using `<<PDDL_Predicate>>` and `<<PDDL_Function>>`. These stereotypes are applied to `ObjectFlow` and `ControlFlow` elements in activity diagrams. Predicates represent logical conditions relevant to the planning logic, such as whether a resource is available. Functions allow the use of numeric data in planning, for example cost, duration, or distance.

Actions are modeled using `<<PDDL_Action>>`, which extends the metaclass `CallBehaviorAction`. This integration allows planning behavior to be described within standard SysML activity diagrams. Action parameters are defined based on previously declared types, while preconditions and effects reference predicates and functions. The use of `CallBehaviorAction` aligns planning actions with commonly used behavioral modeling elements (Beers et al. 2024a).

To clarify the role of the SysML profile within Phase II of the workflow (see Figure 1), Listing 1 shows an example of a PDDL action that can be automatically generated from a suitably annotated system model. In a typical system model, elements such as *tool* and *part* are already defined as structural types, and behavioral steps like *assemble-part* are modeled as actions with incoming and outgoing flows. During Phase II, these elements are enriched with planning semantics by applying the profile's stereotypes. This annotation enables the extraction of planning-relevant logic and the generation of consistent PDDL actions.

Listing 1: Example PDDL action: *assemble-part*

```
(:action assemble-part
 :parameters (?p - part ?t - tool)
 :precondition (available ?t)
 :effect (assembled ?p))
```

In this example, the object types `part` and `tool` correspond to classes stereotyped as `<<PDDL_Type>>`. The action node representing the assembly process is annotated as `<<PDDL_Action>>`, while the flows that express the required and resulting state are marked with `<<PDDL_Predicate>>`. These annotations define the structure and semantics needed to derive the PDDL action shown above through model transformation. Since the correctness of the generated planning artifacts relies on the internal consistency of the annotated model, syntactic and structural validation is required, particularly in large or evolving system descriptions.

To support this, the profile integrates formal validation mechanisms based on the *Object Constraint Language* (OCL). Constraints are defined for each stereotype and can be evaluated during model editing. This allows for early detection of modeling errors and promotes consistent use of planning semantics across the system model.

Listing 2 shows a constraint applied to `<<PDDL_Domain>>` elements. It checks that each contained `<<PDDL_Type>>` element has a unique name. This prevents ambiguous type references and avoids errors during domain interpretation (Sleath and Bercher 2024).

Listing 2: OCL constraint for enforcing unique type names within a domain (Nabizada et al. 2025)

```
context PDDL_Domain inv UniqueTypeNames:
  self.ownedElement
    ->select(e | e.oclIsKindOf(PDDL_Type))
    ->isUnique(t | t.name)
```

The validation logic is integrated directly into the profile and can be processed by tools that support OCL evaluation. In complex planning domains, modeling errors such as redundancies or inconsistencies can occur even in well-structured models, particularly when tool support for validation and feedback is limited (Shah et al. 2013).

The profile is published as an open resource[2] and can be applied in modeling environments to support planning integration. It provides a general mechanism for embedding planning constructs into system models and supports both domain and problem description, depending on the available model content.

5 Algorithm for PDDL Domain File Generation

The transformation algorithm applied in Phase IV of the workflow (see Figure 1) automatically derives a complete PDDL domain description from a SysML system model enriched with planning semantics. Its input consists of model elements annotated with stereotypes defined in Section 4, including types, predicates, functions, and actions. The result is a syntactically valid and semantically consistent domain file that can be submitted to any standard PDDL solver.

The overall process follows a sequential logic that extracts, interprets, and formats the planning-relevant elements. Its core workflow is illustrated as pseudocode in Algorithm 1. The procedure begins by initializing the domain context and continues by collecting all model elements annotated with the relevant stereotypes. Each planning concept is then translated into its corresponding structure in the PDDL domain description.

The algorithm starts by initializing the PDDL domain structure based on the model element annotated with <<PDDL_Domain>> (line 2), before extracting planning-relevant elements using stereotype-based model queries. All classes annotated with <<PDDL_Type>> are processed recursively to reflect inheritance hierarchies (lines 3–6). Logical predicates and numeric functions are extracted from control and object flows that are annotated with <<PDDL_Predicate>> or <<PDDL_Function>> (lines 7–10). For each element, variable names and types are resolved and checked for consistency. Missing definitions are reported to the user to prevent the generation of an invalid domain file.

Behavioral actions are derived from elements annotated with <<PDDL_Action>> (lines 11–14). These actions are modeled as `CallBehaviorAction` nodes and define parameters, preconditions, and effects. The algorithm analyzes incoming and outgoing flows to determine the associated logical conditions. The algorithm examines the flows connected to each action to identify its preconditions and effects. It supports both positive and negative predicates, as well as combinations of multiple conditions using logical conjunctions like `and`. Numerical effects, such as cost increments, are expressed using PDDL functions that update numeric values during planning. The resulting PDDL action definitions reflect the logic embedded in the system model and remain directly traceable to their origin.

Algorithm 1: Generation of the PDDL domain file (Nabizada et al. 2024a)

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

```
 1: function CreatePDDLDomain(sm)
 2:     df ← InitializePDDLDomain(sm.PDDL_Domain)
 3:     T ← ExtractTypes(sm.PDDL_Type)
 4:     for each type in T do
 5:         ProcessTypeToPDDL(df, type)
 6:     end for
 7:     P ← ExtractPredicates(sm.PDDL_Predicate)
 8:     for each predicate in P do
 9:         ProcessPredicateToPDDL(df, predicate)
10:     end for
11:     A ← ExtractActions(sm.PDDL_Action)
12:     for each action in A do
13:         ProcessActionToPDDL(df, action)
14:     end for
15:     return df
16: end function
```

The transformation of annotated SysML models into PDDL descriptions is implemented using the *Apache Velocity Engine*. Templates written in the *Velocity Template Language* (VTL)[3] define the syntactic structure of the output and inject model content into predefined sections. Planning-relevant elements such as types, predicates, functions, and actions are identified through stereotype-specific model queries and converted into PDDL syntax.

Listing 3 shows an excerpt of the generated `:predicates` section, based on stereotyped flows within the system model.

Listing 3: Excerpt from a generated PDDL predicate section

```
(:predicates
    (is-available ?r - Resource)
    (connected ?a - Location ?b - Location)
)
```

Each parameter and type is extracted from the annotated model and checked for completeness. If required names or references are missing, the transformation inserts explicit error messages into the generated PDDL code. These embedded checks help identify modeling issues early and prevent incomplete or inconsistent planning constructs from being generated without notice. While not a substitute for formal validation, these checks complement the OCL constraints

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

[3] Available at https://github.com/hsu-aut/VTL-PDDL_Domain

defined in the SysML profile by covering additional technical aspects required for code generation. They provide immediate feedback during transformation and help ensure the completeness of the generated PDDL files.

A key advantage of the transformation approach is its alignment with the modeling perspective of system engineers. All planning constructs are embedded directly into the SysML model, avoiding the need to switch between external tools or learn additional planning languages. This integration ensures that the system model remains the single source of truth, reducing inconsistencies and supporting traceable generation of planning artifacts.

The use of reusable templates enables flexible adaptation to domain-specific constructs, solver-specific syntax, or alternative planning dialects. Because the transformation is fully embedded in the modeling environment, it supports repeatable workflows and consistent integration of planning logic across engineering projects.

6 Application in Aircraft Assembly

To evaluate the approach in a realistic engineering context, a case study from aircraft structure assembly was conducted. The use case forms part of the iMOD research project (Gehlhoff et al. 2022) and addresses the automated planning of collar screwing operations inside the fuselage. This process involves a UR10 robotic arm equipped with interchangeable end effectors, each capable of processing a specific collar type. Rivets are fasteners used to join structural components, while collars are threaded sleeves that are screwed onto the rivet shafts from the inside to secure the connection. The robot operates in a constrained environment within the fuselage and must reach multiple rivet positions while performing tool changes as required by the collar types. The objective is to generate an optimized plan that minimizes cycle time and reconfiguration effort while ensuring all rivets are processed correctly.

The underlying system model used in this work was developed following the structured modeling workflow presented in (Beers et al. 2023). It provides a detailed representation of the *Collar Screwing System* (CSS), including stakeholder requirements, functional decomposition, and technical subsystem structures. The model was created and extended using *Magic Systems of Systems Architect* (MSoSA), an MBSE environment with SysML support. It served as the basis for applying the model-based workflow described in Section 3.

Phase I

In **Phase I**, the existing SysML model of the CSS (Beers et al. 2023) was analyzed to identify model content relevant for the planning task. Since the system model covers multiple engineering aspects, including control logic, vision systems, and user interaction, it was necessary to restrict the observation scope to elements directly related to the physical execution of screwing operations.

This included, for example:

- the UR10 robotic manipulator as execution unit,
- interchangeable screwing tools as subsystems,
- the structural description of rivets as domain-relevant types,
- and actions for motion and assembly behavior.

While instance-level rivet positions are part of the product model considered in Phase III, the abstract description of rivet types and their functional role is embedded in the system model and relevant for domain specification.

Based on this scoping, a subset of the model was defined as the foundation for introducing planning semantics in Phase II. The selected model elements reflect the key resources, object types, and behavioral steps required for defining symbolic actions and constraints. This focused selection ensures that the planning logic remains aligned with the system architecture while avoiding unnecessary complexity by excluding subsystems that are out of scope.

Phase II

In **Phase II**, the relevant parts of the system model were enriched with planning semantics using the SysML profile introduced in Section 4. Stereotypes were applied to define PDDL types (e.g., *Rivet*, *ScrewingTool*), predicates (e.g., *CollarScrewed*, *EnergySupply*), and functions (e.g., *RivetDistanceInformation*).

Figure 2 shows how structural components of the system, such as end effectors, were annotated with the stereotype `<<PDDL_Type>>` to define object types used in planning. The subsystems *ScrewingToolA* and *ScrewingToolB* are modeled as tool variants and form the basis for typing action parameters in the generated PDDL domain. Interfaces such as mechanical, electrical, and communication ports remain part of the system model but are not explicitly represented in the planning description.

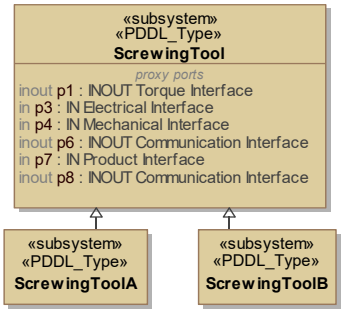

Figure 2: PDDL type annotation for screwing tool variants

Based on the defined types and predicates, planning-relevant system behavior was formalized through annotated actions.

Figure 3 shows an excerpt from the activity diagram in the MBSE environment. The actions *ScrewCollarTypeA*, *ScrewCollarTypeB*, and *MoveToNextRivet* are annotated with the stereotype `<<PDDL_Action>>` and define planning-relevant logic in terms of parameters, preconditions, and effects. Logical conditions such as energy availability or the completion of screwing operations are modeled using stereotyped predicates, while numerical values such as

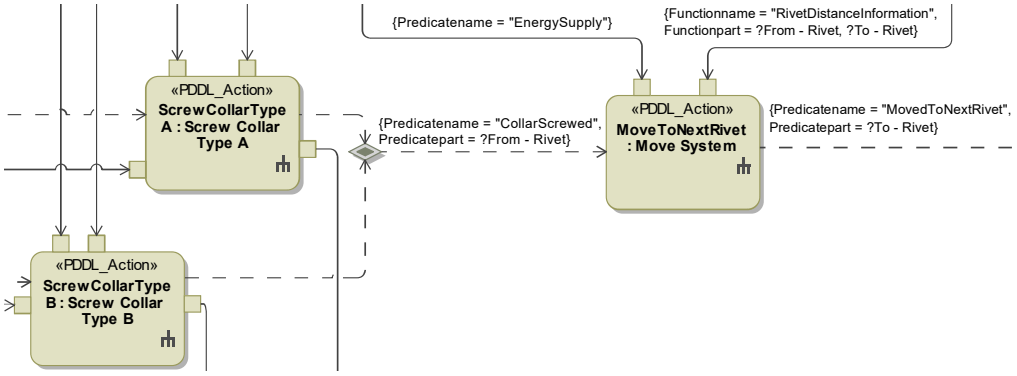

Figure 3: Excerpt from the enriched system model of the collar screwing system (Nabizada et al. 2025)

travel cost are defined via functions. These annotations allow the behavioral logic of the system to be translated directly into structured PDDL actions.

The result is an extended system model in which planning constructs are embedded alongside the existing structural and behavioral descriptions. This model provides a consistent and traceable basis for generating the PDDL domain file during Phase IV of the workflow.

Phase III

In **Phase III**, instance-level product data was incorporated to define the specific planning problem. While the system model describes general capabilities and structural elements of the CSS, the product model provides configuration-specific information required to instantiate a concrete planning task. In this case, the product model was maintained in *Dassault Systèmes 3DExperience* and contained the geometry, type, and position of rivets to be processed in a particular aircraft section.

Relevant product data was extracted from the product structure model and transferred into MSoSA. This information includes the unique identifiers and spatial coordinates of the rivets as well as their assigned collar types. To align the product data with the previously defined planning domain, each instance was annotated using the same PDDL types and predicates introduced in Phase II. For example, each rivet instance was typed as *Rivet* and linked to its location and assembly status via appropriate predicates such as *CollarScrewed*.

This integration results in an extended product model that complements the enriched system model. It defines the initial state of the planning problem and specifies the goal conditions, such as all collars being screwed at their designated positions. By referencing domain-level types and predicates, consistency between domain and problem descriptions is ensured.

Phase IV

In **Phase IV**, the enriched system and product models were used to automatically generate the corresponding PDDL domain and problem file. Since all relevant planning semantics were defined through model annotations in the previous phases, no additional manual modeling steps were required. Instead, the transformation algorithm (see Section 5) was applied that systematically extracts model content based on the applied stereotypes and converts it into syntactically correct PDDL code.

In the applied modeling environment, the transformation is executed using the integrated *Report Wizard*. It provides direct access to the annotated model and invokes the *Apache Velocity Engine* to process VTL-based templates. These templates retrieve stereotype-specific content from the model and assemble it into structured domain and problem file. The transformation is embedded in the modeling tool and does not require external scripts or additional software components. This setup enables reproducible file generation directly within the MBSE environment.

Figure 2 and Figure 3 shows how object types and actions were defined within the SysML model using the profile introduced in Section 4. The corresponding PDDL code fragments are generated automatically and reflect these structures in symbolic form.

Listing 4 shows an excerpt of the generated `:types` section, capturing the type hierarchy defined in the system model through stereotyped class elements.

Listing 4: Excerpt from the generated PDDL types

```
(:types
 ; ... (omitted types)
  ScrewingTool - CSS
  ScrewingToolA ScrewingToolB - ScrewingTool
)
```

The action logic is derived from the behavioral annotations within the activity diagram shown in Figure 3. The model element *MoveToNextRivet* is annotated with the stereotype `<<PDDL_Action>>` and includes planning-relevant constructs such as preconditions (*CollarScrewed*, *EnergySupply*) and numerical effects (*RivetDistanceInformation*). These annotations are automatically translated into structured PDDL syntax.

Listing 5 illustrates the generated representation of this action, including typed parameters, logical conditions, and a cost-based effect modeled using a numeric function. In total,

the generated domain contained 27 types, 21 predicates, 3 functions, and 12 actions.

Listing 5: Generated PDDL action for *MoveToNextRivet*

```
(:action MoveToNextRivet
 :parameters (?from - Rivet ?to - Rivet)
 :precondition
   (and
     (CollarScrewed ?from)
     (EnergySupply)
   )
 :effect
   (and
     (MovedToNextRivet ?to)
     (increase
       (total-cost)
       (RivetDistanceInformation ?from ?to))
   )
)
```

Application Results

Based on the automatically generated PDDL domain and problem files, the analysis tool *VAL* (Howey, Fox, and Long 2004) was used to check their syntactic and semantic correctness. *VAL* reported no errors or warnings, confirming that all types, predicates, and actions were consistently defined and that no undefined symbols or unreachable actions were present.

After validation, the *Delfi* planner (Katz et al. 2018) was used to compute a valid plan for the evaluated system variant. *Delfi* is a heuristic-based planner for cost-optimal planning and was accessed via *Planutils* (Muise et al. 2022), which provides a standardized interface for invoking a wide range of planning tools. The plan was validated using *VAL* (Howey, Fox, and Long 2004), confirming that it satisfies the goal conditions and is consistent with the domain specification.

The resulting plan defines the sequence of screwing and movement operations required to assemble all rivets, taking into account tool constraints and spatial dependencies. In the evaluated scenario, two different rivet types were placed within the robot's workspace, each requiring a dedicated end effector. The planner minimized overall process time by considering travel distances between rivet positions and the overhead of tool changes. As a result, the plan grouped operations by tool type to avoid unnecessary reconfigurations, thereby reducing motion overhead and idle time. This execution strategy reflects a potentially optimized cycle time and can be used as a reference for evaluating alternative system variants.

The case study confirms the practical applicability of the proposed approach. It demonstrates how the integration of planning semantics into engineering models enables the automated generation of consistent planning artifacts and supports traceable variant analysis. Although applied here in the context of aircraft assembly, the method is domain-independent and applicable to other engineering workflows involving symbolic reasoning and decision support.

However, the results also highlight certain limitations that merit further investigation. Although the automated transformation reduces the need for manual effort, the quality of the resulting plans still depends on the precision and completeness of the model, as well as the annotation process. In particular, the usability of the modeling environment and the clarity of the profile elements can affect modeling speed and accuracy. Future improvements should focus on simplifying the modeling process itself, for example through guided annotation workflows or semi-automated suggestions for stereotype application.

7 Conclusion and Future Work

This paper presents a model-driven approach for generating planning descriptions in PDDL from engineering models. The method combines a structured workflow, a dedicated SysML profile for embedding planning semantics, and a transformation algorithm for generating consistent PDDL files.

The approach was applied in a case study from aircraft structure assembly, demonstrating how system behavior and product data can be enriched with symbolic planning constructs and used to automatically derive planning artifacts. The resulting domain file and corresponding problem file reflect the modeled system architecture and enable symbolic reasoning on configuration feasibility and task fulfillment. This confirms the practical relevance of the method for model-based variant evaluation and task-oriented system planning.

A key benefit lies in the traceable integration of planning semantics into engineering models. The approach supports the repeated generation of planning artifacts when system or product configurations change, thereby reducing manual effort and enabling consistent scenario analysis. The reuse of annotated model elements across variants further enhances modeling efficiency and planning reliability.

Future work will focus on lowering the entry barrier for planning integration. First, an interactive assistant is envisioned to support the annotation of planning constructs. Such a tool could guide users in applying stereotypes, validate structural consistency, and provide context-aware suggestions. Techniques based on rule inference or LLM-based prompting may assist in semi-automated annotation.

Second, further automation is needed in linking engineering data to planning tasks. In particular, the extraction and mapping of instance-level product information remains partially manual. Establishing an interface between tools such as *3DExperience* and MSoSA could enable direct reuse of product data within the modeling environment. This would reduce manual effort and support consistent integration of planning information across tool boundaries.

Additionally, compatibility with the *Unified Planning Library* (Micheli et al. 2025), a Python-based API for modeling planning problems and interfacing with a wide range of planning engines, will be investigated to enable integration with standardized planning tools and workflows. Migration to SysML v2 (Object Management Group 2024) is also planned to ensure continued applicability as modeling languages and tool support evolve.

Acknowledgments

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

References

Beers, L.; Nabizada, H.; Weigand, M.; Gehlhoff, F.; and Fay, A. 2024a. A SysML Profile for the Standardized Description of Processes during System Development. In *Proc. IEEE SysCon*.

Beers, L.; Nabizada, H.; Weigand, M.; Gehlhoff, F.; and Fay, A. 2024b. Towards an MBSE Approach for Modeling Complex Production Systems Based on Industrial Standards. In *Proc. IEEE ETFA*.

Beers, L.; Weigand, M.; Nabizada, H.; and Fay, A. 2023. MBSE Modeling Workflow for the Development of Automated Aircraft Production Systems. In *Proc. IEEE ETFA*.

Friedenthal, S.; Moore, A.; and Steiner, R. 2014. *A Practical Guide to SysML: The Systems Modeling Language*. Waltham, USA: Morgan Kaufmann, third edition edition.

Gehlhoff, F.; Nabizada, H.; Weigand, M.; Beers, L.; Ismail, O.; Wenzel, A.; Fay, A.; Nyhuis, P.; Lagutin, W.; and Röhrig, M. 2022. Challenges in Automated Commercial Aircraft Production. In *IFAC PapersOnLine*, volume 55.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge: Cambridge University Press.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Cham: Springer International Publishing.

Henderson, K.; and Salado, A. 2021. Value and benefits of model–based systems engineering (MBSE): Evidence from the literature. *Systems Engineering*, 24(1).

Howey, R.; Fox, M.; and Long, D. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *Proc. IEEE ICTAI*.

Huckaby, J.; Vassos, S.; and Christensen, H. I. 2013. Planning with a task modeling framework in manufacturing robotics. In *Proc. IEEE/RSJ IROS*.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online Planner Selection for Cost-Optimal Planning. In *Proc. IPC*.

Köcher, A.; Heesch, R.; Widulle, N.; Nordhausen, A.; Putzke, J.; Windmann, A.; and Niggemann, O. 2022. A Research Agenda for AI Planning in the Field of Flexible Production Systems. In *Proc. IEEE ICPS*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61(1): 215–289.

Kovacs, D. L. 2011. Complete BNF description of PDDL 3.1. *Language Specification, Department of Measurement and Information Systems, Budapest University of Technology and Economics*.

Lindsay, A. 2023. On Using Action Inheritance and Modularity in PDDL Domain Modelling. In *Proc. ICAPS*.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29.

Muise, C.; Pommerening, F.; Seipp, J.; and Katz, M. 2022. PLANUTILS: Bringing Planning to the Masses. In *Proc. ICAPS*.

Nabizada, H.; Jeleniewski, T.; Beers, L.; Gehlhoff, F.; and Fay, A. 2024a. Automated PDDL Domain File Generation for Enhancing Production System Development based on SysML Models. In *Proc. AI4CC-IPS-RCRA-SPIRIT*.

Nabizada, H.; Jeleniewski, T.; Beers, L.; Weigand, M.; Gehlhoff, F.; and Fay, A. 2025. Integrating AI Planning Semantics into SysML System Models for Automated PDDL File Generation. (Accepted for presentation at ETFA 2025), arXiv:2506.06714.

Nabizada, H.; Jeleniewski, T.; Gehlhoff, F.; and Fay, A. 2024b. Model-Based Workflow for the Automated Generation of PDDL Descriptions. In *Proc. IEEE ETFA*.

Object Management Group. 2024. *Systems Modeling Language (SysML), Version 2.0 beta 2*.

Rimani, J.; Lesire, C.; Lizy-Destrez, S.; and Viola, N. 2021. Application of MBSE to model Hierarchical AI Planning problems in HDDL. In *KEPS Workshop at ICAPS*.

Schmidt, M. M.; and Stark, R. 2020. Model-Based Systems Engineering (MBSE) as computer-supported approach for cooperative systems development. In *Proc. ECSCW*.

Shah, M.; Chrpa, L.; Kitchin, D.; McCluskey, T.; and Vallati, M. 2013. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In *Proc. IJCAI*.

Sleath, K.; and Bercher, P. 2024. Detecting AI Planning Modelling Mistakes – Potential Errors and Benchmark Domains. In Liu, F.; Shen, L.; Lau, H. C.; Lukowicz, P.; Chen, F.; Yan, H.; and An, B., eds., *PRICAI 2023: Trends in Artificial Intelligence*, volume 14326 of *Lecture Notes in Computer Science*. Springer.

Stoev, T.; Sosnowski, T.; and Yordanova, K. 2023. A Tool for Automated Generation of Domain Specific Symbolic Models From Texts. In *Proc. IEEE PerCom Workshops*, 276–278.

Strobel, V.; and Kirsch, A. 2020. MyPDDL: Tools for Efficiently Creating PDDL Domains and Problems. In Vallati, M.; and Kitchin, D., eds., *Knowledge Engineering Tools and Techniques for AI Planning*. Cham: Springer International Publishing.

Tantakoun, M.; Zhu, X.; and Muise, C. 2025. LLMs as Planning Modelers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models. arXiv:2503.18971.

Törmänen, M.; Hägglund, A.; Rocha, T.; and Drenth, E. 2017. Integrating Multi-Disciplinary Optimization into the Product Development Process using Model-Based Systems Engineering (MBSE). In *NAFEMS World Congress*.

Vieira da Silva, L. M.; Heesch, R.; Köcher, A.; and Fay, A. 2023. Transformation eines Fähigkeitsmodells in einen PDDL-Planungsansatz. *at-Automatisierungstechnik*, 71(2).

Wally, B.; Vyskočil, J.; Novák, P.; Huemer, C.; Šindelár, R.; Kadera, P.; Mazak, A.; and Wimmer, M. 2019. Flexible production systems: Automated generation of operations plans based on ISA-95 and PDDL. *IEEE Robotics and Automation Letters*, 4(4).

Weilkiens, T. 2020. *SYSMOD - The Systems Modeling Toolbox: Pragmatic MBSE with SysML*. Fredesdorf: MBSE4U, 3rd edition edition.