Application of MBSE to model Hierarchical AI Planning problems in HDDL

Jasmine Rimani, Charles Lesire, Stéphanie Lizy-Destrez, Nicole Viola
Politecnico di Torino, Torino, Italy
jasmine.rimani@polito.it, nicole.viola@polito.it
ONERA/DTIS, University of Toulouse, France
Charles.Lesire@onera.fr
ISAE-SUPAERO, Toulouse, France
Stephanie.lizy-destrez@isae-supaero.fr

Abstract

The recent improvements of hierarchical AI planning open the path to new and exciting applications in different areas of expertise. One domain with daring and complex planning and scheduling problems is the definition of operations for space exploration systems. For this specific application, the Hierarchical Definition Domain Language (HDDL) may be the most suitable AI planning language to be adopted, seeing its similarities to aerospace engineering functional analysis. The work proposed in this paper contributes to filling the gap between space operations engineers and the AI planning potentialities to solve planning and scheduling problems applied to space exploration systems. The problem and domain files typical of HDDL and PDDL can be built up from the formalism of SysML, a general-purpose architecture modelling language for System Engineering, and MBSE. The designers would be guided through a workflow that will aid them to simplify the translation from MBSE, or SySML, to HDDL. The workflow presented in this paper was applied and tested during an analogue space robotic mission, where a collaborative drone and a rover explore an unknown environment. The final aim of the method is to transfer the "human knowledge" in the planning problem and showing the capabilities of MBSE applied to Knowledge Engineering (KE) of AI planning problems.

Introduction

There are different studies and applications on the use of AI for complex scenarios like space missions (Chien et al. 2000) (Muscettola et al. 1998)(Coles et al. 2019) (Chi, Chien, and Agrawal 2020) (Gao et al. 2016). However, they usually focus on temporal planning, constrained based planning, probabilistic planning and, for simpler scenarios, on problem definition domain language (PDDL). Regardless of the strengths and capabilities of Hierarchical Definition Domain Language (HDDL) (Höller et al. 2020), it still fails to be used as a routine planner for complex scenarios, like space missions. Even if many problems can be well described using a hierarchy of tasks (Georgievski and Aiello 2014).

As evoked in (Strobel and Kirsch 2014), possible reasons can be found in the complexity of the domain and problem file redaction. The predicates, types, methods, tasks, and actions do not scale up easily as the considered problem becomes more and more complex. Moreover, complex problems to be analyzed by the AI planners need skilful engineers to capture and pass their knowledge of the problem. However, the complexity of a scenario can be easily handled by hierarchical modelling of Model-Based System Engineering (MBSE) and its base architecture SysML. SysML and MBSE are both domain-independent tools. Therefore, their use can be extended to any analysis beyond the subject of this paper. The formalism of HDDL is quite similar to the concept of functional analysis in system engineering, (Walden et al. 2015). The "functional layer" of MBSE, called logical architecture, has functions that describe the behaviours of the system or systems at different levels of granularity. Therefore, The main objective of the MBSE-HDDL translation analyzed in this paper is to facilitate the transfer of information from the designer of SysML or MBSE models to the HDDL file straightforwardly. The aim is to describe a workflow that would make the use of HDDL not only appealing but almost natural, starting from the logical layer of MBSE.

In this study, two MBSE standard schemes will be used to write the domain and problem files: the Enchanted Functional Flow Block Diagram (EFFBD) and the IDEF0 (Icam DEFinition for Function Modeling, where Icam stands for Integrated Computer-Aided Manufacturing). The first highlights the succession of events when designing methods of HDDL, and the second helps visualize the flow of the predicates, inputs and outputs of actions, methods and tasks.

Vitech Genesys (Corporation 2020), have been used as a modelling tool for the MBSE analysis. The benchmarks of the IPC2020 (Behnke et al. 2021) for hierarchical planning have been used as a starting point to define the predicates, tasks, actions and method in a consistent and correct formalism. The case study for this paper is an analogue mission organised by Space Innovation in Switzerland called IGLUNA[1]. The objective of the ISAE-SUPAERO team, CoRoDro[2], is to explore an unknown environment with a rover and a drone (Figure 1).

The systems may act as totally independent entities with

[1] https://space-innovation.ch/igluna/
[2] http://corodro.ae-isae-supaero.fr/

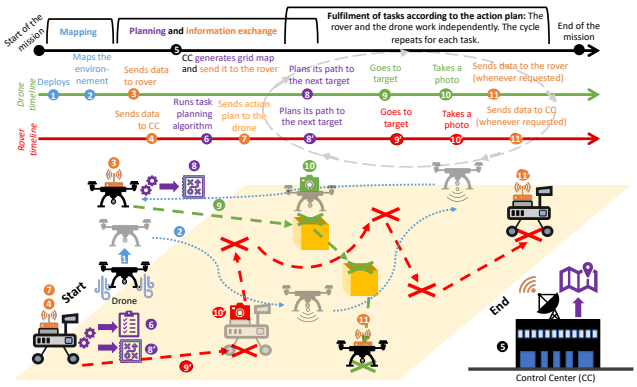

Figure 1: CoRoDro Design Reference Mission (DRM). In the first phase, the drone is in charge of mapping the environment autonomously. While in the second phase, both systems move autonomously, exploring the environment. Both systems should go to a target, read an arTag (Fiala 2005) and take a picture of it.

different capabilities or co-dependent systems where the drone extends the rover's capabilities. The translation from MBSE to HDDL is still manual for the domain file and partially automated for the problem file of the analysed aapplication. The following sections will focus on illustrating the MBSE-HDDL translation used following some examples from the case study. The conclusion will highlight the main outcomes of this work and the future work envisioned to automatize the overall workflow.

Related Work

To address these limitations exposed in the introduction and help the designers transfer their knowledge and correctly write PDDL files, different research teams analyzed and created tools that should assist the designer in creating the domain and problem files of AI planners. Domain files capture system behaviour using a set of actions and predicates, true or false sentences. Problem files indicate the goals to be accomplished and give some information on the environment and constraints that the system under study has to deal with. When solving a PDDL problem, or an HDDL problem, the files are parsed and analyzed to find the best plan that answers the problem, given the goals and initial conditions in the problem file. However, practical applications have many types, objects, predicates and actions (Strobel and Kirsch 2014). Therefore, as the project grows in size, designers need to be assisted with tools that help them keep track of changes and deal with the increased complexity of domain and problem files.

Focusing on the PDDL language, different tools have been created as to help the designer. However, most of them focus on providing a suite to check the syntax and constructions of predicates and actions. Therefore, even if relevant, they lack a pre-design phase where the engineer designs the problem before writing the files. Therefore tools, like *PDDL studio* (Plch et al. 2012) and the PDDL-mode of *Emacs* editor (Singhi 2005) focus more on ensuring the correctness of the PDDL syntax and semantic of the PDDL files. On the other hand, *myPDDL* (Strobel and Kirsch 2014) provides an intuitive IDE (Integrated Development Environment), code template to initialize PDDL constructs and diagrams of the domain file that show the connection between predicates, types and actions.

An interesting work that uses SysML for designing planning problems has been proposed in (Huckaby, Vassos, and Christensen 2013). The study applies the sequence diagram and SysML taxonomy to the study of manufacturing robots. However, the final planning language is still PDDL. Furthermore, the analysis is not backed up by a set of requirements, as it is the usual standard in system engineering to maintain traceability and justify design choices.

Among the most known tools, *itSimple* helps the designer model PDDL files starting from the Unified Modeling Language (UML) formalism (Silva and Silva 2019). In this work, the authors based their design process on UML Use Case diagrams, starting from the requirements definition. In the early version of *itSIMPLE*, the translation between the UML scheme and the PDDL files where manual (Vaquero et al. 2006). In the latest versions, the process has been automatized with an ad-hoc IDE integrating UML (Silva and Silva 2019). However, it is not possible to track back the changes done in the PDDL files to the starting UML schemes (Strobel and Kirsch 2014). On the other hand, the last version of the program integrated hierarchical task network (HTN) planning to model more complex scenarios (Silva and Silva 2019). In general, hierarchical methods permits a higher level of abstraction. To sustain HTN planning modelling, the *itSIMPLE* designer introduced Petri Nets to check the consistency of requirements and verify the decomposition of the plan. The latest version of *itSIMPLE* makes also use of more UML schemes like class diagrams and state machines. Therefore, introducing the complexity of hierarchy, the tool loses in simplicity. However, the integrated IDE helps the designer keep track of the changes, given previous knowledge in UML. The objective of HTN planners is to perform a set of tasks that can be of different levels of abstraction (Ghallab, Nau, and Traverso 2004), not directly achieve a goal state. To be clearer, the plan's end objectives are found by refining the initial tasks network (Georgievski and Aiello 2014), giving as output an executable sequence of actions. However, even if many tasks in real life are already built-in hierarchical structures, HTN planners need well-conceived and well-structured domain knowledge to be used correctly (Georgievski and Aiello 2014). Those requirements have slowed down a more ample use of the capabilities of HTN planners. However, there is a need to introduce hierarchies because domain experts may want to model their domain using the natural structure of many real-world problems, that is hierarchical (Bercher, Alford, and Höller 2019). Moreover, HTN planners have the primary advantage in terms of speed and scalability when applied to real-world problems in respect to other AI planners (Georgievski and Aiello 2014). What is unique of HTN planners is that they can reason about the effects of possible actions. That capability makes them incredibly expressive in how they describe behaviour

(Humphreys 2019). Nevertheless, HTN planning, in all its declination, is quite suitable to be modelled through MBSE. The objective of MBSE functional models is to be a hierarchical, well-defined and complete analysis of the behaviours of whichever system under study (Walden et al. 2015).

Similarly to the itSIMPLE tool, the work presented in this paper starts from requirements and a hierarchical planner. However, with SysML instead of UML, it is possible to simplify the number of schemes needed to frame the problem. For example, SysML introduced the *requirement scheme*, which can be directly linked to the functions to be performed by the systems. Moreover, using EFFBDs instead of Use Case diagrams introduces a hierarchy of tasks without the need for a Petri net. Furthermore, using EFFBD, it is possible to check the flow of predicates and how the plan evolves on the different levels of abstraction. The following section will give an overview of the methodology and its application to HDDL.

Modelling HDDL files from MBSE

Both HDDL problems and functional analysis of MBSE are based on task decomposition. Both methods start from an idea of *what* the system should do and go down to *how* the system can perform the *what*. However, before diving into the methodology and translation from MBSE to HDDL, a brief introduction on the most useful concepts is needed for the two.

HDDL modelling language

The HDDL language (Höller et al. 2020) is heavily based on PDDL. It starts from the same concepts of types, predicates and actions, and it extends them with the use of *tasks* and *methods* in the domain file (see Listing 1 for an example). The new entries in the domain file permit a higher level of abstraction: *tasks* and *methods* do not have a direct effect on predicates. They assemble actions to get a structured answer to a higher system functional need. Therefore, more complex scenarios can be modelled. More in detail, a *task* is an instance that should be accomplished. It indicate *what* the system should do. Usually, a *task* is defined by a unique name and some parameters that are needed to accomplish it. The *how* this task should be executed is usually defined thanks to a *method*. *methods* define how to achieve a task given a set of ordered *subtasks*. If a *subtask* can be further decomposed by a method, that *subtask* is called a *compound task* or simply *task*. If the *subtask* can be directly defined with a set of precondition and effects without a *method*, then the subtask is called *primitive task* or, more commonly, action (see Listing 1). For the same task, it is possible to define different methods that can satisfy it. Different methods have different preconditions, defined as predicates, that lead to a different organization of sub-tasks.

Listing 1: HDDL domain file example of task, method and action

```
(:task navigate_abs
        :parameters (?system - system ?to - waypoint)
        :precondition ()
        :effect ()
)

(:method m_navigate_abs_1_ordering_0
        :parameters (?from - waypoint ?system - system ?to
              - waypoint)
        :task (navigate_abs ?system ?to)
        :precondition (and
                (at ?system ?from)
        )
        :subtasks (and
         (task0 (visit ?from ?system))
         (task1 (navigate ?system ?from ?to))
         (task2 (unvisit ?from ?system))
        )
        :ordering (and
                (< task0 task1)
                (< task1 task2)
        )
)
(:action navigate
        :parameters (?x - system ?y - waypoint ?z -
             waypoint)
        :precondition
                (and
                        (can_traverse ?x ?y ?z)
                        (available ?x)
                        (at ?x ?y)
                )
        :effect
                (and
                        (not (at ?x ?y))
                        (at ?x ?z)
                )
)
```

Similarly to the domain file, the problem file of the HDDL domains changes a bit in respect to the PDDL language (Haslum et al. 2019). An initial hierarchical network should be laid out. It is based on which tasks should the system accomplish and in which order.

Listing 2: Problem file task definition.

```
(:htn
        :parameters ()
        :subtasks (and
         (task0 (release_second_system drone1 rover0))
         (task1 (get_image_data objective0 depth))
         (task2 (get_image_data objective1 depth))
         (task3 (get_image_data objective0 fisheye))
         (task4 (get_image_data objective2 fisheye))
         (task5 (call_back drone1 rover0))
         (task6 (evaluate_available_resources rover0))
         (task7 (evaluate_available_resources drone1))

        )
        :ordering (and
                (< task0  task1)
                (< task0  task2)
                (< task0  task3)
                (< task0  task4)
                (< task2  task5)
                (< task3  task5)
                (< task1  task5)
                (< task4  task5)
                (< task5  task6)
                (< task6  task7)

        )
)
```

Overall, the logical structure of the problem and domain files of HDDL describes the behavior of a system with some given formalism.

MBSE functional layer

This notion of *tasks* that represent *what* a system can do and its division in *subtasks* resembles the notion of *functions* in system engineering. Most operations in the space domain are engineered and designed by system engineers with little to no notion about AI planning, its applications and strength. Moreover, there is usually a reticence in adopting any new tool if it cannot be well documented or translated in a model-based database. However, leveraging on the function/tasks

similarity, it is possible to use system engineering methodologies to design and track changes in the HDDL domain and problem file. Exploiting the formalism of SysML and the capabilities of MBSE, a designer can study and simulate the behavioural layer of a system before exporting the modeling in the HDDL language and plan the system's operations. The backbone of the method relays in the functional analysis, where the the expected behaviours of the system are analysed. Starting from the goal "functions", it is possible to create a breakdown structure with all the *sub-functions* that effectively "answer" to the question *how to perform* the goal "function" (Figure 2). The goal "functions" are the tasks derived from the requirement analysis. They are the "objective behaviors" of the system. The designer identifies the sub-functions that will satisfy the main function. The breakdown goes down to the leaf functions, entities that can directly be performed by the system under study, like *move to a destination*. This hierarchy of functions is the fundamental process of any functional analysis of MBSE. The process has been detailed in both (Walden et al. 2015) and (Shishko and Aster 1995). The top-down process is usually represented as a functional tree or a functional block diagram. The first method represents a simple hierarchical decomposition that usually ends with the indication of a system, subsystem or component that can accomplish the task (Wertz, Everett, and Puschell 2011). The second one includes both a top-down decomposition and an information on the sequence of the functions to be performed (Wertz, Everett, and Puschell 2011).

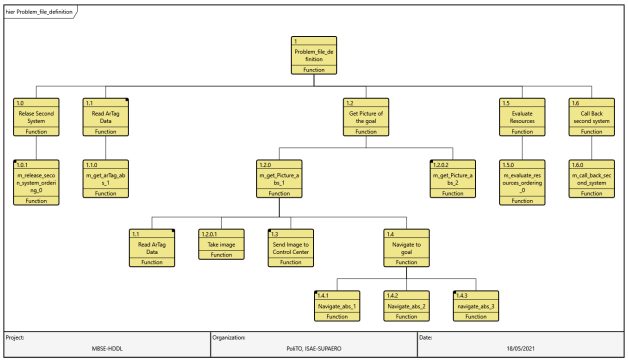

Figure 2: Hierarchical visualization of functions for the IGLUNA campaign. Only the functions related to the functions *Get Picture of the Goal* have been extended to show the hierarchical structure of the problem.

To visualize the logic flow of sub-functions, how they are related and their input/outputs, designers use two principal schemes: the activity diagram in SysML and the Enchanted Flow Functional Block Diagram (EFFBD)[3] in MBSE. In this study, we would use the latter. It is important to highlight that SysML is the foundation of MBSE, however, the latter is usually more expressive and facilitates the design of the system. The EFFBD shows the succession or parallelism of functions. It indicates if functions should be executed simultaneously, if the flow can take different branches or if a function or set of functions should be iterated or replicated. The main difference between a standard Flow Functional Block Diagram and the EFFBD is the possibility of visualizing inputs and outputs of a function. However, another helpful scheme to check the flow of inputs and outputs is the IDEF0[4]. The scheme does not give any information on the order of functions, just on the flow of the items. It can be used to check that the output of the leaf-functions is effectively the expected one of the high-level function.

MBSE to HDDL translation

The parallelism between HDDL and MBSE is relatively straightforward: tasks can be analyzed as high-level functions, methods can be modelled as second-level tasks that contain the other compound tasks and actions (Fig 3). On the other hand, actions can be compared to leaf functions.

The objects and their types can be modelled as components or items. The latter is preferred: it is possible to associate multiple items to a function but not multiple components.

Figure 3 shows this parallelism. The tasks, methods and actions of the HDDL problems can be compared to the functions of MBSE. At the same time, the predicates that advance the HDDL plan are related to the output and inputs of functions, usually modelled as *items* in MBSE. However, the designer choices of what to consider as a high-level function and how to define the items would affect the final form of the domain and problem file. Different designers may therefore obtain different tasks and method decomposition. Nevertheless, the final set of defined actions should be the same. The actions are "functions" directly executable by the system. They are directly interfaced with the components or main subsystems (mobility, power, sensing, etc.). Moreover, it is possible to cross-check the analysis coupling the functional analysis with a bottom-up approach. Starting from the known set of actions that the system can execute, it is possible to verify that all methods and tasks are considered.

Usually, the MBSE model is designed. Then, the parallelism between the HDDL entries and the functional analysis is used to easily translate the MBSE model to the HDDL domain and problem files, as shown in Figure 4. The red arrows represent the translation from MBSE instances to HDDL language entries. The blue arrows show the logical flow of MBSE functional analysis.

The logical steps of this MBSE-HDDL translation are:

1. Define the system functional requirements. This is done using the *Requirement Scheme*. Functional requirements are all the ones that define *what* the system should do (Wertz, Everett, and Puschell 2011), i.e., the actions it should perform.
2. Define the high-level functions that are generated from the functional requirements. Those functions are the translation of the requirement in the form of a verb followed by

[3] http://www.vitechcorp.com/resources/GENESYS/onlinehelp/desktop/Views/Enhanced_Function_Flow_Block_Diagram_(EFBD).htm

[4] https://www.vitechcorp.com/resources/core/onlinehelp/desktop/Views/IDEF0.htm

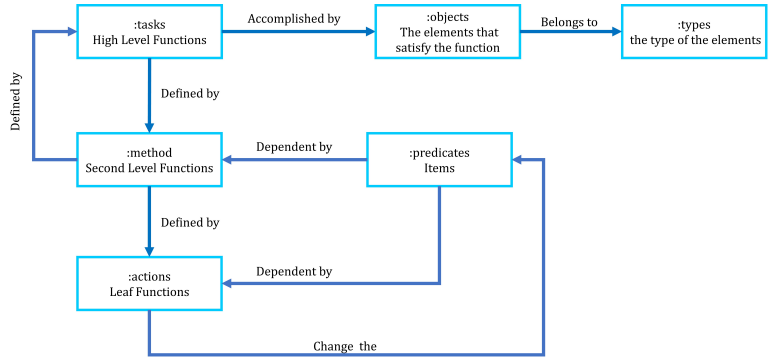

Figure 3: Parallellism between MBSE and HDDL files' entries.

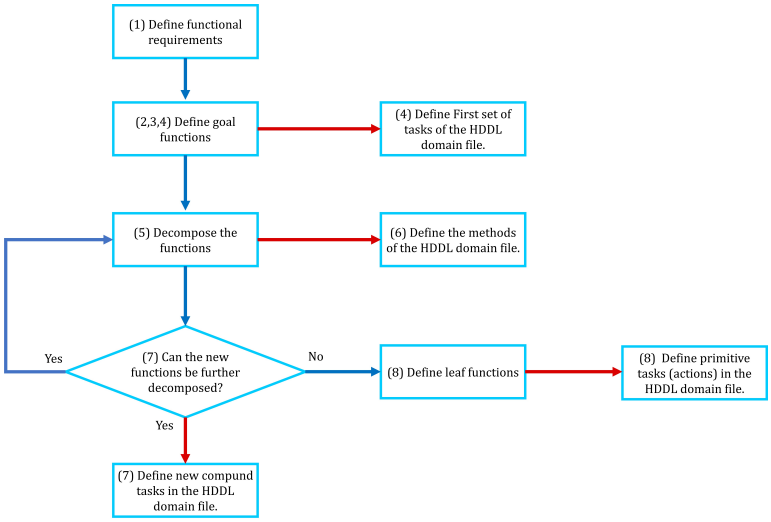

Figure 4: Logical flow of the methodology that exports MBSE model to HDDL files. The red arrows represent the translation from MBSE to HDDL, while the blue arrows show the logical flow of the MBSE functional analysis.

a complement. For example, the functional requirement *The involved systems shall be able to take photos of the point of interest.* can be translated in the high-level function *Get a picture of the goal*.

3. Define the first set of tasks in the HDDL domain file. The high-level functional requirements are the first set of tasks that the system should perform.

4. Assemble the high-level function in a EFFBD. The scheme should show if there is a hierarchy of functions. This analysis will be then translated in the *initial hierarchical network* in the HDDL problem file. It is even helpful to understand if a function can be incorporated into another one because, for example, it always precedes it. In the EFFBD, it is possible to start highlighting the inputs of each function and the expected outputs. That process helps understand the hierarchy of functions as well: a function may always precede another because its output is an essential input of the following one. To easily visualize this flow of inputs and outputs, the IDEF0 scheme can be used. The IDEF0 highlights which outputs of a function are the input of another. These inputs and outputs will then be translated as the predicates used in HDDL to advance a plan.

5. Decompose the high-level functions into sub-functions, answering the question *how* do we accomplish the function. This decomposition in ordered subtasks will define a method. If different decompositions are possible starting from different inputs, we will define different methods for the same function. Again, the prefered scheme to be used is the EFFBD to highlight the ordering of the subtasks. The input/output flow can be studied with an IDEF0 diagram. In the case of the methods, it is possible to verify that the expected output of the high-level function is effectively the one in output from the subtask.

6. Define the methods from the decomposition in ordered subtasks. The *:precondition()* of the method are the inputs of the high-level function (defined in step 4) and the specific inputs of each method. The predicates for each method can be easily visualized in the IDEF0 diagram.

7. Analyze the sub-functions. For each sub-function evalu-

ate if it can be further decomposed or if it can be identified as a leaf-level function (a function that the system can directly implement). If a function can be further decomposed, consider it as a *compound task* and go back to step five.

8. Translate the leaf functions in HDDL actions. If the subfunction is a leaf function, it is possible to write it as an HDDL action. In this case, only the IDEF0 diagram can be used for the translation to analyse the *:preconditions()* (input predicates) and the *:effects()* (output predicates) of the action.

At the end of the steps, the domain files' task, methods, and actions should be defined. The list of predicates will be written from the *:preconditions()* and *:effects()* of methods and actions. The problem file *initial hierarchical network* can be defined by step 4 using the EFFBD. The initial conditions of the problem file, *:init()*, can be derived from the preconditions of the methods. At this moment the translation from the MBSE model to the HDDL language is mostly manual. In fact, the MBSE model from Genesys can be exported as an Excel file where the task, its methods, the actions associated with each method and the items associated with both are listed. This excel file is then parsed, and a preliminary structure of the domain file is created. However, the designer still needs to cross-check the model. Moreover, the workflow still lacks direct feedback from the HDDL's files to the MBSE model: the modification in the domain files need to be manually reported in the MBSE model. The problem file is partially automatized similarly to the domain file regarding the initial set of true and false statements. In the specific case of the IGLUNA mission, the problem file objectives and "allowable" movement models are output directly from the mapping module of the drone. However, that is possible for this specific application.

The following section shows an example of applying the explained MBSE-HDDL translation to an analogue mission starting from the functional requirements.

Example of Application: The IGLUNA mission

Functional Requirement Definition

The design of every mission starts with a set of requirements that have to be satisfied. In system engineering, the requirements are divided into different categories: mission, configurations, operational, functional, interface, environmental and logistic support (Walden et al. 2015). In our specific case of designing the activities that systems should perform, we are interested in functional requirements. In the IGLUNA mission, the system has five principal functional requirements to accomplish: (i) recognize points of interest, (ii) take a picture of the recognized point of interest, (iii) evaluate remaining resources, (iv) release drone, (v) call back the drone. Therefore following step 1 and 2 of the workflow, we can define the *goal functions* that the systems have to accomplish from those functional requirements, as shown in Figure 5.

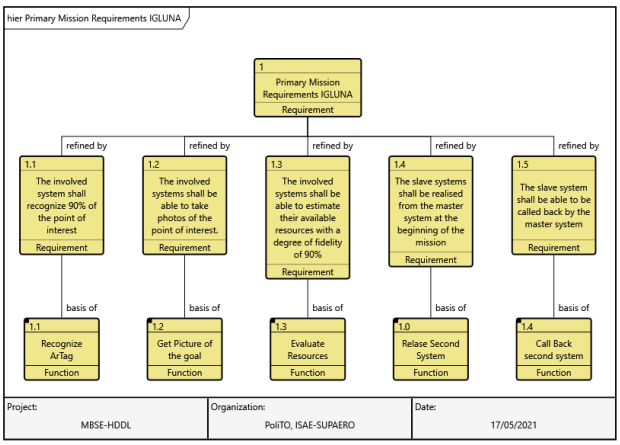

Figure 5: From the functional requirements analysis of IGLUNA to its high-level functions (step 1 and 2 of the workflow).

From Requirements to High-level Functions to HDDL Tasks

The defined high-level functions will be the first set of tasks in the domain files, Figure 4, Listing 3.

At the same time, those tasks are the ones that will appear on the problem file as *goal functions* to be executed. However, an ordering between the tasks may be needed to set up the initial task network definition of the HDDL problem file. Therefore, the notion of EFFBD becomes quite helpful to analyze the problem as explained in step 4 of the workflow (Figure 6, Listing 2).

Listing 3: Domain file task definition.

```
(:task get_image_data
        :parameters (?objective - objective ?mode - mode)
        :precondition ()
        :effect ()
)

(:task read_arTag_data
        :parameters (?objective - objective)
        :precondition ()
        :effect ()
)

(:task evaluate_available_resources
        :parameters (?system - system)
        :precondition ()
        :effect ()
)

(:task call_back
        :parameters (?system2 - system ?system1 - system)
        :precondition ()
        :effect ()
)

(:task release_second_system
        :parameters (?system2 - system ?system1 - system)
        :precondition ()
        :effect ()
)
```

Definition of the Hierarchy of Functions

In the EFFBD, the designer can already visualize the predicates linked to each task as the inputs and outputs of functions. However, as previously suggested in step 4 of the

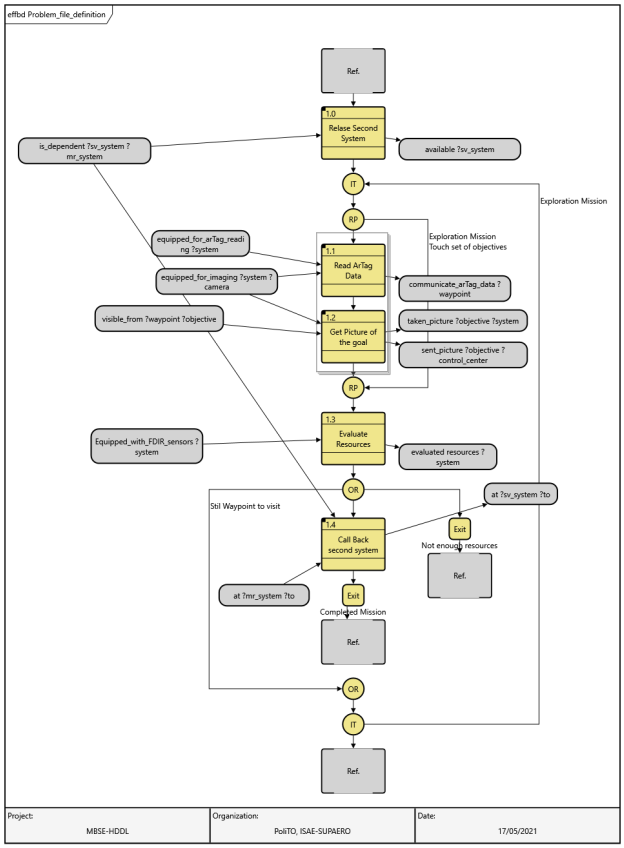

Figure 6: EFFBD of the problem file with high-level functions. The diagram shows the expected succession of tasks that should be translated in the *initial hierarchical network* of the problem file as explained in step 4 of the workflow.

workflow, the IDEF0 can help better visualize the predicate flow. Figure 7 shows the IDEF0 linked to the problem EFFBD, Figure 6. Moreover, from Figures 6 and 8 it is possible to visualize that the expected output of a high-level function becomes the effect of a leaf-function, therefore the effect of the action (steps 5 and 6 of the workflow).

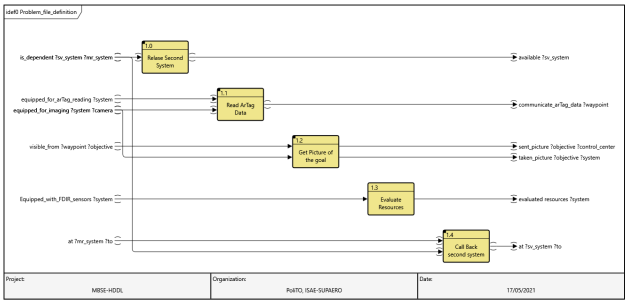

Figure 7: IDEF0 of the problem file. As outlined in step 4 and 5 of the workflow, from the high-level function is already possible to associate the expected output.

Furthermore, from the analysis of the problem EFFBD, it is possible to conclude that the function "Read ArTag Data" always come before the "Get Picture of the Target" function and that those tasks are replicated during the overall mission. Therefore as illustrated in step 4 of the workflow, to simplify the redaction of the problem file, the first function can be included in the second function using a *method*, Figures 8 and 9.

Definition of Methods and Primitive Tasks

In the example of the function "Get Picture of the Target", only one method was needed to satisfy it. However, it is possible to have different methods that may satisfy a task, as for the function "Navigate to goal", Figure 10. The plan solver may take one or the other branch from different predicates, as briefly laid out in step 5.

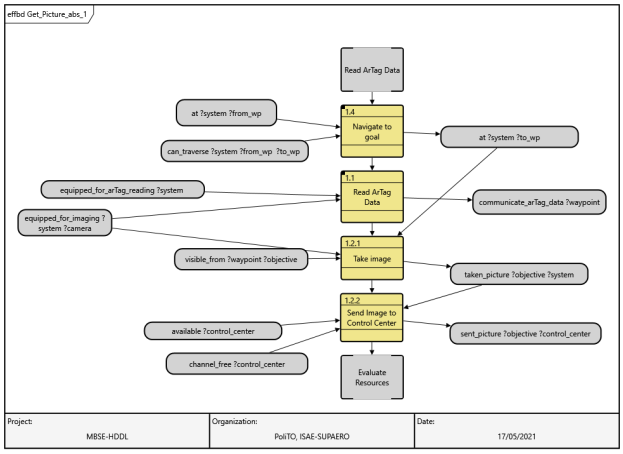

Figure 8: EFFBD of the "Get Picture of the Target" method (step 5 of the workflow). The sub-functions with a black square on the top-left corner are the ones that can be further decomposed. Those will be translated as *compound tasks*, while the others are *actions*.

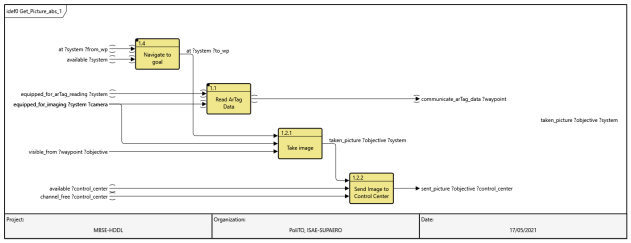

Figure 9: IDEF0 of the "Get Picture of the Target" method. From the IDEF0 of the problem file (Figure 7) it is possible to see that the expected output of the functions are *sent_picture ?objective ?control_center* and *take_picture ?objective ?system*. With this IDEF0, it is possible to verify that these outputs are effectively the end effects of the last two actions of this method as described in step 5 of the workflow.

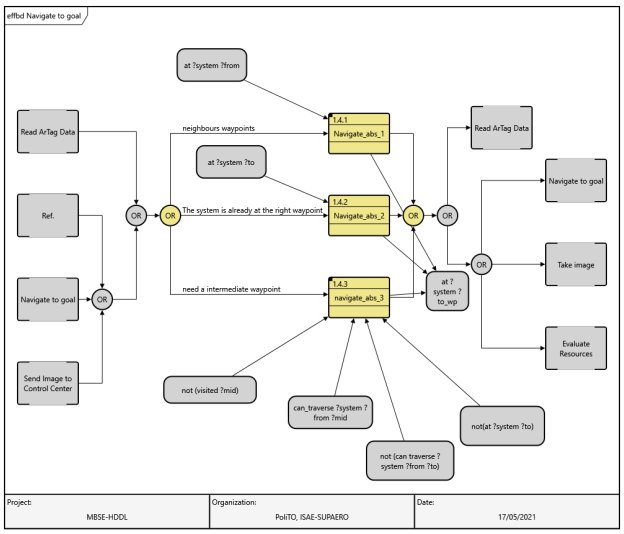

Figure 10: EFFBD of the ”navigate to goal” methods. The image visually summarizes step 6 of the worflow. Thanks to the functional analysis, defining different methods for a single task with an ”or” logical structure is possible. The chosen branch will depend on the ”active” predicates in the problem file.

The process of the top-down functional analysis is replicated for all the tasks and their subtasks, as described in step 7 of the workflow. In the end, it is possible to export the structure of the MBSE model to the HDDL file following the breakdown of the EFFBD and checking the predicate flow with the IDEF0. At the conclusion of the design process, the designer can check the consistency and correctness of the problem and domain files against the created MBSE model. In the case of the IGLUNA mission, the benchmark plan for the exploration of a terrain of $9m^2$ with three objectives is shown in Listing 4. HiPOP (Bechon, Lesire, and Barbier 2020; Lesire and Albore 2021) was used as solver for the HDDL problem. The problem file of IGLUNA was directly created from the SLAM (Simultaneous Localization and Mapping) of the two robotic systems. Therefore, during the real field camping, the planner had to deal with more than one hundred waypoints, two systems, two cameras per system, and ten objectives. The output plan has around eighty lines, and it was interfaced with the execution layer of the two systems through a state machine.

Listing 4: Output plan for a 3x3 map with 3 objetives

```
==>
0 (make_available drone1)
1 (visit waypoint0 drone1)
2 (navigate drone1 waypoint0 waypoint1)
3 (unvisit waypoint0 drone1)
4 (read_arTag rover0 waypoint0 objective2 camera0)
5 (communicate_arTag_data rover0 general objective2)
6 (take_image drone1 waypoint1 objective2 camera3 fisheye)
7 (communicate_image_data drone1 general objective2 fisheye)
8 (navigate drone1 waypoint1 waypoint4)
9 (visit waypoint4 drone1)
10 (navigate drone1 waypoint4 waypoint6)
11 (unvisit waypoint4 drone1)
12 (read_arTag drone1 waypoint6 objective0 camera2)
13 (communicate_arTag_data drone1 general objective0)
14 (take_image drone1 waypoint6 objective0 camera3 fisheye)
15 (communicate_image_data drone1 general objective0 fisheye)
16 (navigate drone1 waypoint6 waypoint4)
17 (visit waypoint4 drone1)
18 (navigate drone1 waypoint4 waypoint8)
19 (unvisit waypoint4 drone1)
20 (read_arTag drone1 waypoint8 objective1 camera2)
21 (communicate_arTag_data drone1 general objective1)
22 (take_image drone1 waypoint8 objective1 camera2 depth)
23 (communicate_image_data drone1 general objective1 depth)
24 (visit waypoint0 rover0)
25 (navigate rover0 waypoint0 waypoint5)
26 (unvisit waypoint0 rover0)
27 (read_arTag rover0 waypoint5 objective0 camera0)
28 (communicate_arTag_data rover0 general objective0)
29 (take_image rover0 waypoint5 objective0 camera0 depth)
30 (communicate_image_data rover0 general objective0 depth)
31 (navigate drone1 waypoint8 waypoint4)
32 (visit waypoint4 drone1)
33 (navigate drone1 waypoint4 waypoint5)
34 (unvisit waypoint4 drone1)
35 (get_data_from_sensors rover0)
36 (send_system_state rover0 general)
37 (get_data_from_sensors drone1)
38 (send_system_state drone1 general)
<==
```

Main Results and Conclusions

This paper presented a workflow to simplify the writing of HDDL problem and domain files starting from the functional layer of MBSE. The modelling starts from defining the goals of the mission as functional requirements. Then, it continues with the definition of the high-level functions linked to the requirements. Those first set of functions will define the first tasks of the domain file. Then the high-level functions are further broken down into leaf functions. The different sub-functions constitutes the other *compound and primitive tasks.* The EFFBD and IDEF0 schemes show this decomposition and the predicate flow of each function. The overall process permits fast prototyping and writing of HDDL files while verifying the consistency of the design. However, the work still lacks a comprehensive tool that can rapidly translate the functions and items into ready-to-run HDDL files. The translation from MBSE to HDDL is mostly manually executed, reporting the MBSE outputs to ”.hddl” files. Therefore, future work will focus on creating an integrated framework from the MBSE to the definition of ready to be used domain and problem files. In the specific case of the problem files for space robotic application, a partial knowledge of the environment is usually a must-know. However, this can be easily included if a map is available. In the specifics of the IGLUNA analogue mission, a C++ script that translates a given map in predicates that can be used for the navigation tasks has already been implemented and used for the problem file definition. At the same time, the team will be dedicated time to a more in-depth study of the formalism of MBSE and SysML and how to better used them as assets for Knowledge Engineering. The final aim is to define a logical and operational architecture usable in space-related scenarios, from rovers to satellites, to define operations with AI planning.

References

Bechon, P.; Lesire, C.; and Barbier, M. 2020. Hybrid planning and distributed iterative repair for multi-robot missions with communication losses. *Autonomous Robots* 44(3-4):505–531.

Behnke, G.; ; Höller, D.; and Bercher, P., eds. 2021. *Proceedings of 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*.

Bercher, P.; Alford, R.; and Höller, D. 2019. A survey on hierarchical planning-one abstract idea, many concrete realizations. In *IJCAI*, 6267–6275.

Chi, W.; Chien, S.; and Agrawal, J. 2020. Scheduling with complex consumptive resources for a planetary rover. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 348–356.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; et al. 2000. Aspen-automated planning and scheduling for space mission operations. In *Space Ops*, volume 82.

Coles, A. J.; Coles, A. I.; Munoz, M. M.; Savas, O. E.; Keller, T.; Pommerening, F.; and Helmert, M. 2019. Onboard planning for robotic space missions using temporal pddl. In *11th International Workshop on Planning and Scheduling for Space (IWPSS)*.

Corporation, V. 2020. Genesys: Enhancing systems engineering effectiveness.

Fiala, M. 2005. Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, 590–596 vol. 2.

Gao, Y.; Burroughes, G.; Ocón, J.; Fratini, S.; Policella, N.; and Donati, A. 2016. Mission operations and autonomy. *Contemporary Planetary Robotics: An Approach Toward Autonomous Systems* 321–401.

Georgievski, I., and Aiello, M. 2014. An overview of hierarchical task network planning. *arXiv preprint arXiv:1403.7426*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Morgan&Claypool.

Huckaby, J.; Vassos, S.; and Christensen, H. I. 2013. Planning with a task modeling framework in manufacturing robotics. In *International Conference on Intelligent Robots and Systems (IROS)*.

Humphreys, T. 2019. Exploring htn planners through example. In *Game AI Pro 360*. CRC Press. 103–122.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Lesire, C., and Albore, A. 2021. pyHiPOP – Hierarchical partial-order planner. In *Proceedings of 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial intelligence* 103(1-2):5–47.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. In *International Conference on Automated Planning and Scheduling (ICAPS), Demonstration Paper*.

Shishko, R., and Aster, R. 1995. Nasa systems engineering handbook. *NASA Special Publication* 6105.

Silva, J. M., and Silva, J. R. 2019. A new hierarchical approach to requirement analysis of problems in automated planning. *Engineering Applications of Artificial Intelligence* 81:373–386.

Singhi, S. 2005. Emacs mode for PDDL, `http://rakaposhi.eas.asu.edu/f04-cse574-mailarchive/msg00088.html`.

Strobel, V., and Kirsch, A. 2014. Planning in the Wild: Modeling Tools for PDDL. In *German Conference on Artificial Intelligence (KI)*.

Vaquero, T. S.; Tonidandel, F.; de Barros, L. N.; and Silva, J. R. 2006. On the Use of UML.P for Modeling a Real Application as a Planning Problem. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Walden, D. D.; Roedler, G. J.; Forsberg, K.; Hamelin, R. D.; and Shortell, T. M., eds. 2015. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Hoboken, NJ: Wiley, 4 edition.

Wertz, J. R.; Everett, D. F.; and Puschell, J. J. 2011. *Space mission engineering: the new SMAD*. Microcosm Press.