humantech

D5.1 – Development of the ontology for demolition task planning

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement n° 101058236. This document reflects only the author's view, and the EU Commission is not responsible for any use that may be made of the information it contains.

Project Title	Human-Centred Technologies for a Safer and Greener European Construction Industry.
Project Acronym	HumanTech
Grant Agreement No	101058236
Instrument	Research & Innovation Action
Topic	HORIZON-CL4-2021-TWIN-TRANSITION-01-12
Start Date of Project	June 1, 2022
Duration of Project	36 months

Name of the Deliverable	Development of the ontology for demolition task planning
Number of the Deliverable	D5.1 (D20)
Related WP Number and Name	WP5 - Construction Robotics and Human-Robot Collaboration
Related Task Number and Name	T5.1 - Robotic demolition task planning
Deliverable Dissemination Level	PU - Public
Deliverable Due Date	31/01/2024
Deliverable Submission Date	31/01/2024
Task Leader/Main Author	Francesca Canale (STAM)
Contributing	Morten Lind (SINTEF Manufacturing)

Partners	
Reviewer(s)	Dr. Gabor Sziebig, Dr. Jason Rambach

Keywords

Robotic demolition, ontology, knowledge base, task planning

Revisions

Version	Submission date	Comments	Author
V1.0	12/01/2024	For submission	Francesca Canale

Disclaimer

Abstract

This report comprehensively outlines the efforts undertaken during Task 5.1, focusing on the development of a task planner for the automatic execution of demolition activities. At its core, it leverages a demolition ontology, an extension of ifcOWL, to establish a cognitive foundation. This ontology meticulously delineates the demolition environment, encompassing representations of walls, openings, and robots. Demolition task planning has a detailed focus on marking, drilling, and cutting operations. These tasks seamlessly unfold through the task planner, which meticulously assesses feasibility based on available resources.

The HumanTech project

The European construction industry faces three major challenges: increase the safety and wellbeing of its workforce, improve its productivity, and become greener, making efficient use of resources.

To address these challenges, HumanTech proposes to develop **human-centred cutting-edge technologies** such as wearables for workers' safety and support and robots that can harmoniously coexist with human workers while contributing to the ecological transition of the sector.

HumanTech aims to achieve major advances in cutting-edge technologies that will enable a safe, rewarding, and digital work environment for a new generation of highly skilled construction workers and engineers.

These advances will include:

- **Robotic devices equipped with vision and intelligence** that allow them to navigate autonomously and safely in highly unstructured environments, collaborate with humans and dynamically update a semantic digital twin of the construction site in which they are.
- **Smart, unobtrusive workers protection and support equipment**. From exoskeletons activated by body sensors for posture and strain to wearable cameras and XR glasses that provide real-time workers' location and guidance for them to perform their tasks efficiently and accurately.
- An entirely new breed of **Dynamic Semantic Digital Twins (DSDTs) of construction sites** that simulate in detail the current state of a construction site at the geometric and semantic level, based on an extended Building Information

Modelling (BIM) formulation that contains all relevant structural and semantic dimensions (BIMxD). BIMxDs will act as a common reference for all human workers, engineers, and autonomous machines.

The **HumanTech consortium** is formed by 22 organisations — leading research institutes and universities, innovative hi-tech SMEs, and large enterprises, construction groups and a construction SME representative — from 10 countries, bringing expertise in 11 different disciplines. The consortium is led by the German Research Center for Artificial Intelligence's Augmented Vision department.

Contents

1. Introduction 7
2. Demolition task in HumanTech 8
3. Demolition ontology for HumanTech 10
3.1. ifcOWL 12
3.2. What is missing? 14
3.2.1. IfcWall class 15
3.2.2. Opening class 17
3.2.3. Robot class 24
3.3. Ontology architecture 26
4. Demolition task planning for HumanTech 29
4.1. Demolition task definition 30
4.1.1. Marking 31
4.1.2. Drilling 37
4.1.3. Cutting 40
4.2. Task planning and scheduling 42
5. ROS-based integration 45
6. Towards an actual deployment: Pilot III 54
7. Conclusions 56

1. Introduction

In HumanTech a task planner for the automatic execution of demolition activities has been developed. The concept behind the task planner is to empower HumanTech demolition robots to autonomously perform some of the critical demolition operations, thereby mitigating the risks associated with manual labour. In this innovative approach, human operators assume a supervisory role, remotely monitoring and ensuring the task's successful execution.

Automatic task planning in the HumanTech system relies on the HumanTech demolition ontology, a structured knowledge framework containing information about building demolition and the demolition environment. This ontology equips the robot with an understanding of its surroundings and the potential consequences of its actions.

This ontology is an extension of ifcOWL, a Web Ontology Language (OWL) representation of the Industry Foundation Classes (IFC) schema, made available by Building Smart. To enhance its applicability to demolition activities, ifcOWL has been expanded with additional concepts, properties, and relationships. Specifically, new classes have been defined, like "Robot", "Tool", and "Opening", while the already existing “IfcWall” class has been extended with supplementary properties.

For instance, within the ontology:

- The "Robot" class has sub-classes "MobileRobot" and "StationaryRobot" and includes properties such as "hasPayload", "hasTool", and "isAvailable".
- The "hasTool" property establishes a connection between a “Robot” and the specific “Tool” it employs, which could be a "Marker", a "Driller", or a "Sawblade".
- The "isAvailable" property is represented by a boolean value indicating whether a particular “Robot” is ready to perform a task.

Reasoning on all this information, it is possible to find out if a robot equipped with the correct tool is at the current moment available to perform one of the demolition tasks. If it is available, the task planner can then assign the robot the task and trigger all the requisite sub-tasks seamlessly.

This document presents a comprehensive overview of the efforts invested in creating the HumanTech demolition ontology and the associated demolition task planner. It offers intricate details about the ontology's concepts and properties, providing a thorough understanding of the demolition environment and the robotic systems

involved. The document also delves into the automatic generation of high-level robotic tasks, pivotal for the successful completion of demolition activities.

2. Demolition task in HumanTech

The demolition activity taken into consideration in the HumanTech project specifically focuses on cutting openings into existing walls, a common task during renovation projects. These openings serve various purposes, such as conduits for essential elements such as MEP (mechanical, electrical, and plumbing) ducts and pipes, as well as facilitating the addition of doors or windows. Traditionally, these activities are labour-intensive and potentially hazardous, involving human-operated tools like grinders and rudimentary guidance systems, leaving the human workers exposed to harmful environmental factors like dust and falling debris.

Typically, the manual process commences by marking the intended opening on the wall using tools such as chalk, paint, or markers. This involves identifying and outlining the specific area of the wall that needs to be removed, emphasizing distinct points such as corners and key locations along the sides. These marks should be visible and well-defined to guide the demolition crew in cutting or breaking the wall along the edges delineated by the marked points. This process is crucial to ensure precision and accuracy during the demolition work.

Subsequently, the marked area undergoes a survey to ensure the absence of rebar or pipes inside the concrete (Figure 1). This survey involves the use of a specialized concrete detector designed for locating rebar, with a typical detection depth of up to 200 mm.

Figure 1: Rebar surveying

Following the survey, a machine is utilized to systematically drill the marked opening (Figure 2). Depending on the wall material (concrete, masonry, drywall, or wood), appropriate drilling equipment such as a hammer drill or rotary hammer is used. The drilling process is performed by drilling holes at the corners and critical points which have been previously marked. These holes serve as starting points for subsequent cutting or demolition.

Figure 2: Concrete drilling

Once all holes have been successfully cored, the edges of the designated area are then cut using a circular saw blade (Figure 3). The drilled holes are used as entry points for the saw blade to initiate the cut, and then the opening is cut along its sides. The saw is periodically stopped to clear away debris from the cutting area. This helps maintain visibility and prevents the accumulation of material that could interfere with the saw's performance.

Ultimately, the concrete at the center of the cut area is isolated and subsequently removed.

Figure 3: Wall cutting

This conventional manual process not only consumes significant labor, but also poses potential risks to workers due to the operation of heavy machinery and exposure to environmental hazards. Thanks to the HumanTech project, there is the opportunity to explore and implement advanced technologies that can enhance the efficiency and safety of this demolition activity.

Within the project scope, three primary demolition tasks - marking, drilling, and cutting - have been identified as particularly compelling for robotic automation. This strategic focus arises from their intricacies and the potential for robotic systems to streamline and elevate the precision of these operations, making a significant leap towards a safer and more efficient demolition process.

The exclusion of the rebar surveying task from the robotic scenario was a deliberate decision, driven by a careful evaluation of its robotic feasibility and value addition. The perceived value added by automating the rebar surveying task was comparatively lower in contrast to the marking, drilling, and cutting operations. While the rebar survey is crucial in the manual workflow for ensuring structural integrity and safety, its exclusion in the robotic domain was justified by the inherent complexities and the limited enhancement it would bring to the overall efficiency of the demolition process. The HumanTech project strategically prioritized tasks with higher potential for robotic optimization, aiming to maximize the impact of automation on both safety and efficiency aspects of demolition activities. It is essential to note that, although the rebar surveying step is not automated in HumanTech, it will still be diligently executed by a qualified professional before commencing the robotic demolition process, ensuring the necessary safety measures and structural integrity are upheld.

3. Demolition ontology for HumanTech

In the realm of computer science, an ontology is a formal representation of knowledge, delineating concepts, objects, properties, and the relationships among them within a specific data structure known as a Knowledge Base (KB). This structured representation provides a shared understanding of a particular domain, allowing the organization and classification of knowledge. Ontologies play a pivotal role in modelling and structuring information, making it comprehensible and machine-readable.

The construction of ontologies often entails the identification of core concepts, establishing hierarchies and taxonomic relationships between them, and defining attributes or properties that characterize each concept. These relationships can be expressed using various logical operators, such as subsumption, disjointness, and equivalence, to capture the intricate structure of the domain.

The development of ontology languages, such as Web Ontology Language (OWL), has provided a standardized framework for representing and exchanging ontology-based knowledge. OWL is part of the Semantic Web technology and allows for the definition of classes, properties, and individuals, as well as the specification of relationships between them.

Different kinds of ontologies can be defined according to specific application needs:

- **Top-level ontologies.** They describe very general concepts like e.g., space, time, object, event or action that are independent from a particular problem or domain.
- **Domain ontologies** and **task ontologies.** They describe, respectively, the vocabulary related to a generic domain or a generic task or activity, by specializing the terms introduced in the top-level ontology.
- **Application ontologies.** They are strictly related to a specific application and used to describe concepts whose relevance is limited to a specific domain and task.

Their versatility has led to their widespread adoption in numerous fields, particularly in the context of knowledge management, semantic search, and intelligent systems.

In the context of HumanTech, the goal is to formulate a novel domain ontology that defines general concepts, properties, and relationships applicable to robotic demolition environments. Drawing upon existing literature, various ontologies have been proposed, and therefore there is a number of publications that can be taken as references to define the HumanTech ontology. One specific work that is particularly pertinent with respect to the objectives of the project is the ifcOWL[1] ontology.

[1] https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/

3.1. ifcOWL

ifcOWL is an ontology that represents the Industry Foundation Classes (IFC) data model using the Web Ontology Language (OWL), contributing to the interoperability and semantic richness of building information models in the construction industry.

The Industry Foundation Classes[2], developed by buildingSMART, are an open international standard for Building Information Model (BIM) data that are exchanged and shared among software applications used by the various participants in the construction or facility management industry sector. The standard includes definitions that cover data required for buildings over their life cycle, including entities that define building elements (walls, beams, doors, etc), geometry features (extruded solid area, swept area solid, etc) and basic constructs (cartesian points, etc). Each IFC data model is represented as a schema in the EXPRESS data specification language defined in the 10303-11:1994 ISO standard.

In the literature, there are several works that describe attempts to obtain a usable ontology from IFC EXPRESS schemas. Schevers and Drogemuller[3] developed a primitive version of a unidirectional conversion of an IFC schema to an OWL ontology for research purposes. Later, Pauwels and Terkaj, based on several previous studies[4], proposed a conversion procedure from EXPRESS schema to OWL ontology[5]. That work can be regarded as the basis of the ifcOWL ontology.

The ifcOWL ontology is generated directly from the IFC EXPRESS schema. The recommended conversion procedure is entirely open and documented, and an opensource set of reusable Java components that allows to parse IFC-SPF files and convert them into directed labelled graphs (RDF) is provided in the "pipauwel/IFCtoRDF" GitHub repository[6]. The conversion procedure follows the principles displayed in the below schema (Table 1).

[2] https://standards.buildingsmart.org/IFC/DEV/IFC4_2/FINAL/HTML/

[3] Schevers, Hans, and Robin Drogemuller. "Converting the industry foundation classes to the web ontology language." 2005 First International Conference on Semantics, Knowledge and Grid. IEEE, 2005.

[4] Beetz, Jakob, Josef P. van Leeuwen, and Bauke de Vries. "An ontology web language notation of the industry foundation classes." Proceedings of the 22nd CIB W78 Conference on Information Technology in Construction. Technische Universität Dresden, 2005.

[5] Pauwels, Pieter, and Walter Terkaj. "EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology." Automation in construction 63 (2016): 100-133.

[6] https://github.com/pipauwel/IFCtoRDF

Table 1: Summary of the adopted conversion pattern

EXPRESS	OWL
Schema	Ontology
Simple data type	OWL class with a restriction on an owl:DatatypeProperty
Defined data type	OWL class
Constructed SELECT data type	OWL class with equivalence to a union of collection of OWL classes
Constructed ENUMERATION data type	OWL class with equivalence to a one of collection of owl:NamedIndividual items
Entity data type	OWL class
Attribute of entity data type	Functional object property with specified domain and range; owl:AllValuesFrom restriction; owl:qualifiedCardinality or owl:maxQualifiedCardinality restriction
Attribute of entity data type as a SET	Non-functional object property with specified domain and range; owl:AllValuesFrom restriction; owl:minQualifiedCardinality and/or owl:maxQualifiedCardinality restriction or owl:qualifiedCardinality restriction
INVERSE attribute	object property with a specified owl:inverseOf
DERIVE attribute	N/A
WHERE rule	N/A
FUNCTION	N/A
RULE	N/A

Using the ifcOWL ontology, it is possible to represent building data using state of the art web technologies. This graph model and the underlying web technology stack allows building data to be easily linked to material data, GIS data, product manufacturer data, sensor data, classification schemas, social data, and so forth. The result is a web of linked building data that brings major opportunities for data management and exchange in the construction industry and beyond.

3.2. What is missing?

While ifcOWL is a well-structured ontology, defining concepts and properties that are relevant for HumanTech, it does not cover all the knowledge needed to deal with the demolition activities.

Notably, ifcOWL excels in encapsulating information about architectural constructs, spatial relationships, and environmental elements intrinsic to the building domain. However, it lacks provisions for representing the aspects related to robotics, an indispensable factor for the robotic demolition activities considered in HumanTech. For this reason, the "Robot" class was added to the ontology, along with necessary information related to it.

Moreover, to align the ontology with the specific requirements of demolition, the addition of a new class became necessary. Introducing the "Opening" class was essential, as it serves as the central subject in the context of demolition, a concept that ifcOWL did not explicitly encompass. This addition ensures a more complete representation, filling the gap in knowledge coverage and making the ontology better suited for the targeted domain of robotic demolition.

Finally, to refine the ontology for HumanTech, modifications were made to the existing IfcWall class within the ifcOWL ontology. This adjustment involved a meticulous review to streamline the organization of its properties. The goal was to enhance accessibility to information crucial for HumanTech's objectives, such as the dimensions and position of the wall. By optimizing the structure of the IfcWall class, we aimed to ensure a more user-friendly and efficient framework for capturing relevant data in the context of robotic demolition.

In the next subsections of this document (3.2.1, 3.2.2, and 3.2.3), a comprehensive description of all the additions and modifications made to the ifcOWL ontology are provided. Together, these changes collectively shape the HumanTech demolition ontology.

Table 2 shows the list of namespaces which have been used in the ontology.

Table 2: Prefixes and namespaces for the HumanTech ontology

Prefix	Namespace	Comment

ifc	<http://standards.buildingsmart.org/IFC/DEV/IFC4/ADD1/OWL#>	IFC namespace (v 4)
owl	<htthp://www.w3.org/2002/07/owl#>	OWL namespace
rdf	<http://www.w3.org/1999/02/22-rdf-syntax-ns#>	RDF namespace
rdfs	<http://www.w3.org/2000/01/rdf-schema#>	RDF Schema namespace
ht	HT:	Namespace created ad-hoc for the HumanTech project

3.2.1. IfcWall class

The IfcWall class is a part of the ifcOWL ontology and represents walls in a building. It is used to describe various types of walls, such as exterior walls, interior walls, and partition walls. The IFC standard defines different properties and attributes associated with the IfcWall class to capture relevant information about walls in a building model.

Some key aspects typically associated with the IfcWall class are its name, geometric representation, information about the materials that make it up and its composition, and relationships with other elements of the model, such as doors and windows. In the context of demolition, the parameters which emerge as pivotal factors are the ones related to the geometric representation of the wall, which include the shape, dimensions, and placement of the wall within the building.

However, the inclusion of these parameters in the ifcOWL ontology is intricated and not always performed in the same way. For this reason, in the HumanTech ontology, the IfcWall class has been enhanced by incorporating new standardized properties that pertain to the dimensions and location of the wall. Table 3 outlines the RDF triples used to represent these ontological statements, with the "Subject" column denoting the entity, the "Predicate" column indicating the property or relationship, and the "Object" column representing the value or another entity.

Three new datatype properties have been introduced to capture the dimensions of the wall: hasHeight, hasLength, and hasWidth. These properties respectively attribute height, length, and width values to the IfcWall entity.

To express the location of the wall, a new class named Pose has been introduced (as detailed in Table 4). The Pose class is linked to the IfcWall class through the hasPose

object property. The Pose class expresses the position and orientation of an object - in this case an IfcWall - in a three dimensions space. It is defined by six datatype properties: three related the translational component (i.e. hasX_pos, hasY_pos, and hasZ_pos), which defines X, Y, and Z cartesian coordinates of the object with respect to a predefined frame, and three related to the rotational component (i.e. hasYaw_rot, hasPitch_rot, and hasRoll_rot), defining yaw, pitch, and roll rotations of the object's frame with respect to the predefined frame. In the case of the IfcWall, the Pose class precisely expresses the position and orientation of the wall with respect to the origin frame of the IFC file.

Table 3: IfcWall class extension

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
ifc:IfcWall	hasHeight	Height value Datatype = Float
ifc:IfcWall	hasLength	Length value Datatype = Float
ifc:IfcWall	hasWidth	Width value Datatype = Float
ifc:IfcWall	hasPose	Pose

Table 4: Pose class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Pose	hasX_pos	X position value Datatype = Float
Pose	hasY_pos	Y position value Datatype = Float
Pose	hasZ_pos	Z position value Datatype = Float
Pose	hasYaw_rot	Yaw rotation value Datatype = Float
Pose	hasPitch_rot	Pitch rotation value Datatype = Float
Pose	hasRoll_rot	Roll rotation value

		Datatype = Float

Figure 4 presents a graphical representation of the relations between the IfcWall class, its properties, and the Pose class.

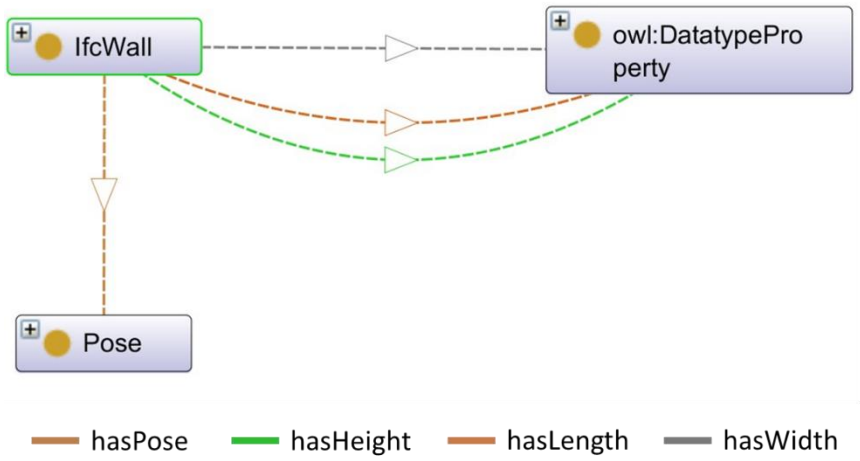

Figure 4: Graphical representation of relationships among classes related to IfcWall

3.2.2. **Opening class**

The Introduction of the Opening class serves to specifically represent the focal point of demolition activities.

Its definition revolves around parameters crucial for accurately describing the opening, encompassing dimensions and position. These parameters are defined by a qualified professional, typically an architect or a structural engineer, deciding to create an opening in a wall. These parameters are translated into the demolition input described by the variables listed in Table 5.

The demolition input encapsulates key information, including the names of both the opening and the wall where the opening is intended to be made. These names serve to uniquely identify each entity. The dimensions of the opening are described by height, width, and length parameters, defined as shown in Figure 5. The position of the opening in relation to the wall's origin is specified by X and Y coordinates, as depicted in Figure 5, along with an angle in degrees that denotes the offset between the two horizontal lines of the opening and the wall. Finally, the last parameter describing the opening is the direction which clarifies the side of the wall, front or back, on which the opening has to

be made. This parameter is essential when the opening does not span the entire width of the wall (i.e. the width of the opening is smaller than the width of the wall). This direction input is denoted to be equal to 1 if the opening is on the front side, and to -1 if it is on the back side, as illustrated in Figure 6.

Table 5: Demolition input

Input	Description
Opening name	Name of the opening Datatype = String
Wall name	Name of the wall where the opening has to be made Datatype = String
Height	Height of the opening Datatype = Float
Width	Width of the opening Datatype = Float
Length	Length of the opening Datatype = Float
X	X coordinate of the corner of the opening with respect to the origin of the wall Datatype = Float
Y	Y coordinate of the corner of the opening with respect to the origin of the wall Datatype = Float
Angle	Angle (in degrees) between the horizontal directions of the wall and of the opening Datatype = Float
Direction	Whether the opening is on the front (1) or on the back (-1) side of the wall Datatype = Int

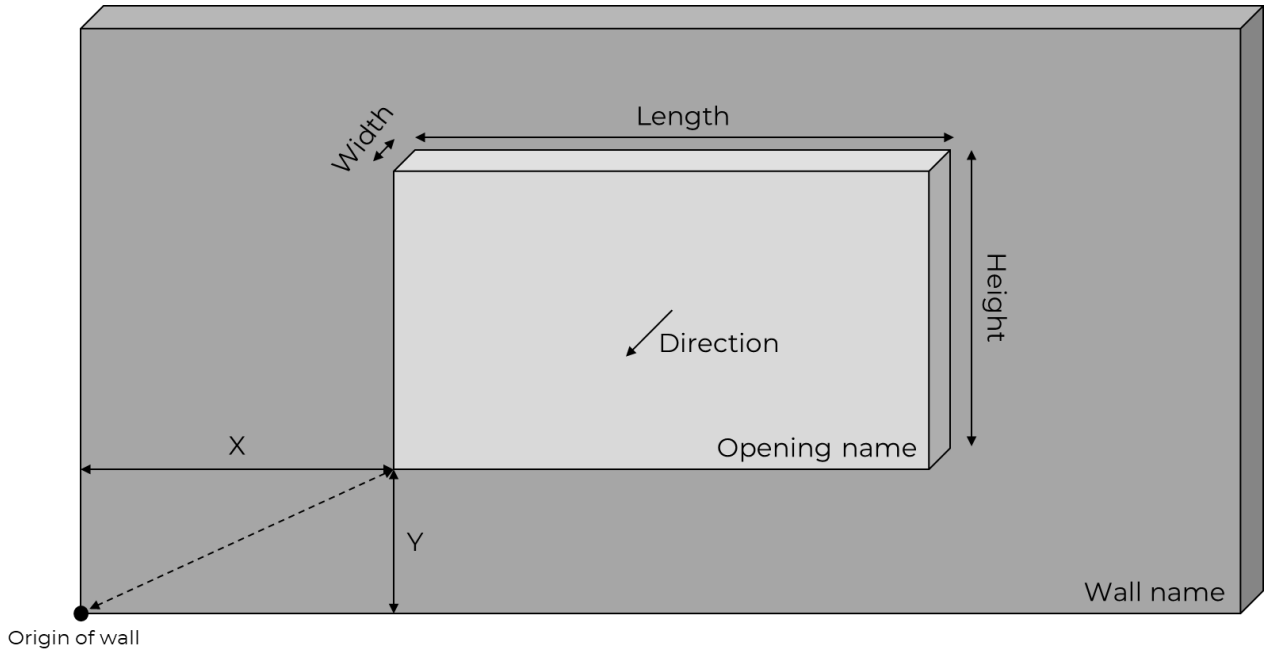

Figure 5: Definition of demolition inputs

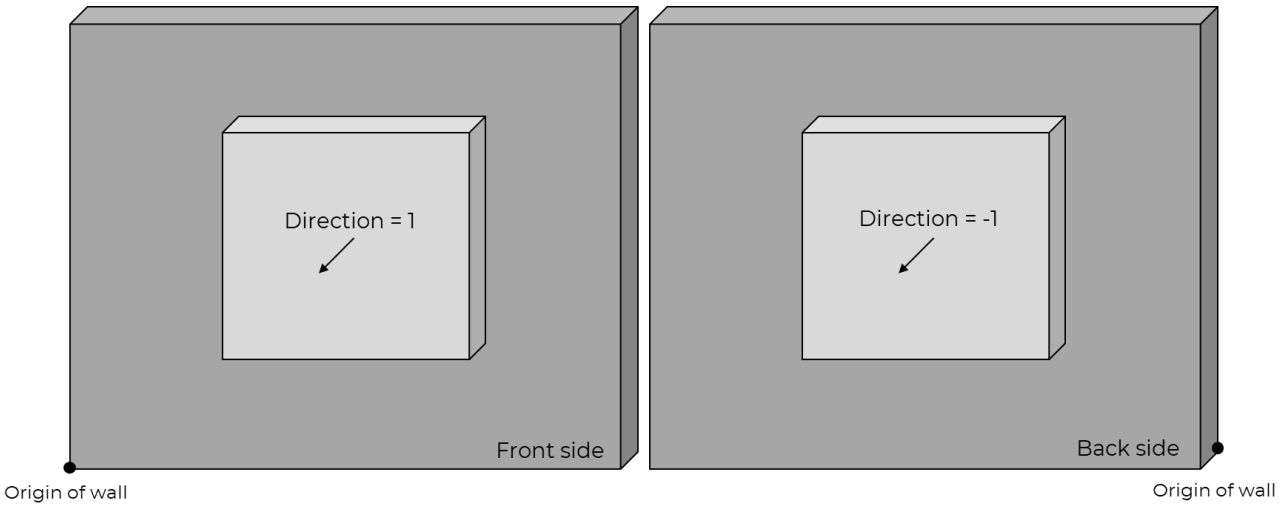

Figure 6: Definition of direction of opening input

The demolition input has been translated in the HumanTech ontology into the Opening class defined by the properties and relationships described in Table 6.

The name is assigned to the opening through the hasOpeningName datatype property, and the opening is linked to its corresponding wall through the isLocatedOnWall object property. In a parallel manner, the same datatype properties previously employed for defining the dimensions of the wall - hasHeight, hasLength, and hasWidth - are now repurposed to describe the dimensions of the opening. Additionally, the Pose class, initially utilized for conveying the position of the wall, is now linked to the Opening class

through the hasPoseOnWall object property. The hasDirection datatype property defines the direction of the opening.

Table 6: Opening class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Opening	hasOpeningName	Name of the opening Datatype = String
Opening	isLocatedOnWall	ifc:IfcWall
Opening	hasHeight	Height value Datatype = Float
Opening	hasLength	Length value Datatype = Float
Opening	hasWidth	Width value Datatype = Float
Opening	hasPoseOnWall	Pose (defined in Table 4)
Opening	hasDirection	Direction of opening Datatype = Int

Other than the information coming from the demolition input, there are other data that it is useful that the Opening class describes.

One first information is related to the demolition status of the opening, indicating whether the opening has undergone marking, drilling, or cutting, which are the three fundamental tasks involved in creating an opening in a wall. To this purpose, the openingStatus class has been created, which is linked to the Opening class through the hasOpeningStatus object property. The openingStatus class, described in Table 7, has three boolean datatype properties: openingMarked, openingDrilled, and openingCut. When set to True, these properties signify that the corresponding demolition task has been successfully executed.

Table 7: openingStatus class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)

Opening	hasOpeningStatus	openingStatus
openingStatus	openingMarked	Marking status of opening Datatype = Boolean
openingStatus	openingDrilled	Drilling status of opening Datatype = Boolean
openingStatus	openingCut	Cutting status of opening Datatype = Boolean

Moreover, an Opening is characterized by two key spatial attributes: points and sides. The opening points are, for example, the ones showed in red in Figure 7, indicating the four corners and the four mid-points of the edges of the opening. These points hold significant importance in the context of the demolition activity, serving as precise locations where the wall will be firstly marked and then drilled.

Within the HumanTech ontology each point is described by the openingPoint class, linked to the Opening entity through the hasOpeningPoint object property. As outlined in Table 8, an openingPoint is defined by a global position with respect to the origin frame of the IFC file, expressed through the Pose class and the hasGlobalPosition object property. Each openingPoint is also characterized by the isCorner boolean datatype property, indicating whether it represents one of the four corners of the opening. Finally, two additional boolean datatype properties, pointMarked and pointDrilled, convey whether a specific point has already been marked or drilled, similarly to the properties within the openingSatus class. These serve as a valuable indicator within the ontology to track the completion status of the marking and drilling operations for each individual point.

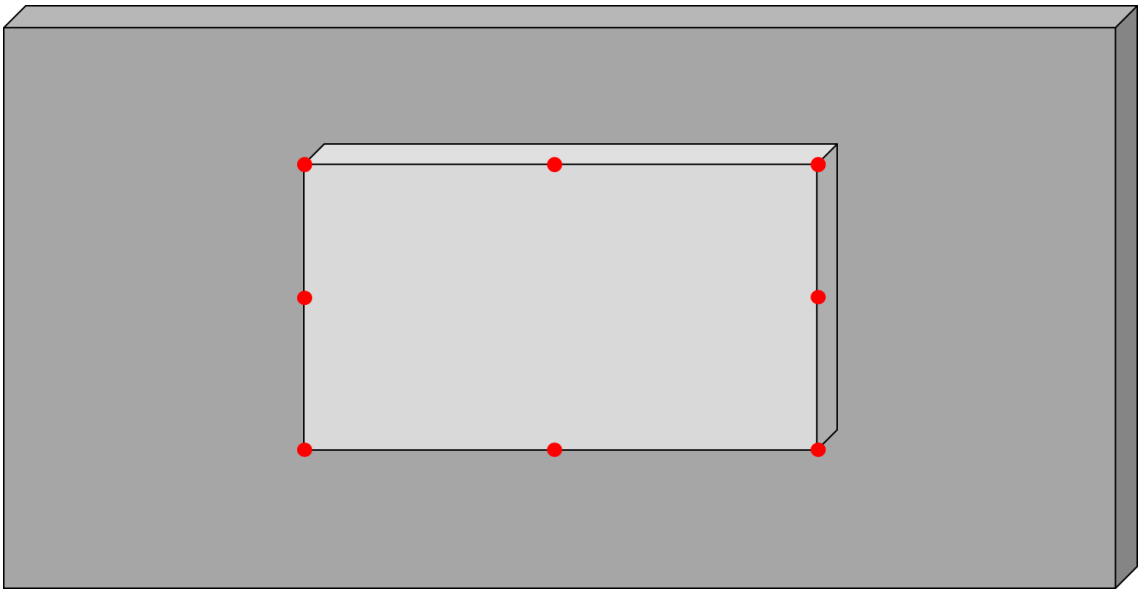

Figure 7: Opening points

Table 8: openingPoint class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Opening	hasOpeningPoint	openingPoint
openingPoint	hasGlobalPosition	Pose
openingPoint	isCorner	Corner of opening Datatype = Boolean
openingPoint	pointMarked	Marking status of point Datatype = Boolean
openingPoint	pointDrilled	Drilling status of point Datatype = Boolean

The opening sides are the ones represented in red in Figure 8, indicating the four edges of the opening, where the cutting task is set to take place. Similar to the representation of opening points, each edge is described within the ontology by the openingSide class, linked to the Opening entity through the hasOpeningSide object property. As outlined in Table 9, an openingSide is characterized by two global positions relative to the origin frame of the IFC file, specifying the location of the side as the two points connected by the edge. These positions are expressed through the Pose class and the hasStartingPosition and hasEndingPosition object properties. Mirroring the approach

taken with opening points, an additional boolean datatype property, sideCut, signifies whether a specific side has undergone the cutting task.

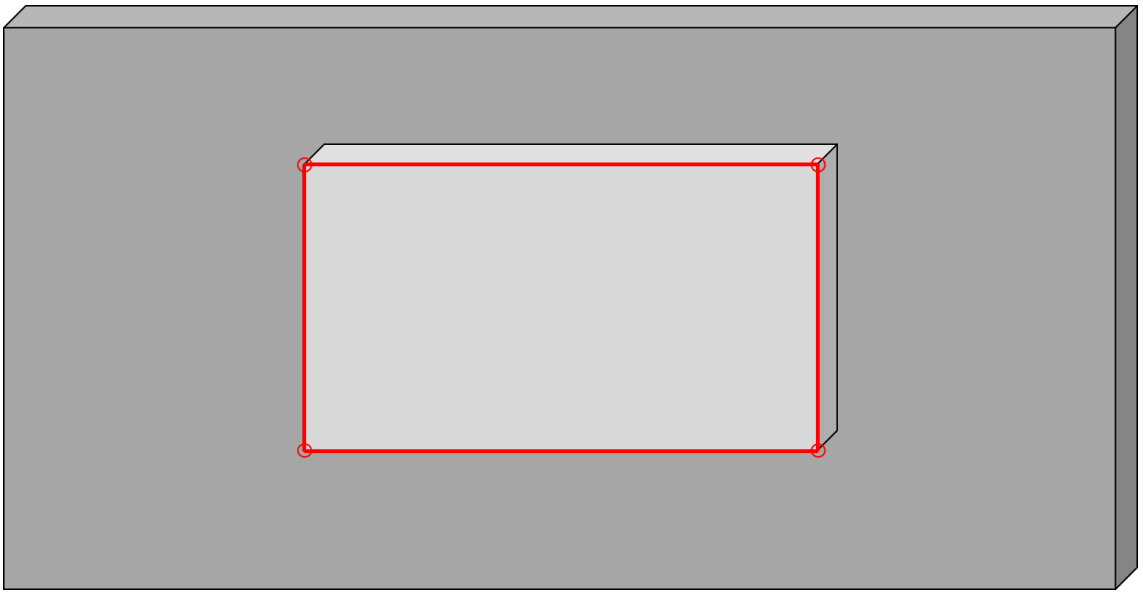

Figure 8: Opening sides

Table 9: openingSide class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Opening	hasOpeningSide	openingSide
openingSide	hasStartingPosition	Pose
openingSide	hasEndingPosition	Pose
openingSide	sideCut	Cutting status of side Datatype = Boolean

Figure 9 provides a graphical representation of the relations between the Opening class, and the IfcWall, Pose, openingStatus, openingPoint, and openingSide classes.

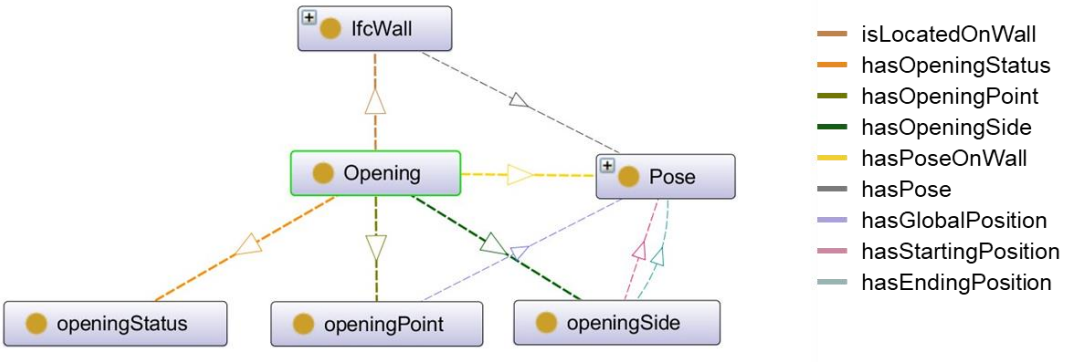

Figure 9: Graphical representation of relationships among classes related to Opening

3.2.3. **Robot class**

The Robot class within the HumanTech ontology is a pivotal entity designed to encapsulate attributes and functionalities about the robotic systems considered in the project involved in the demolition activities.

The class is defined by the properties and relationships listed in Table 10. The hasRobotName datatype property assigns a unique identifier or name to each robot, facilitating clear recognition and task assignment within the dynamic context of the demolition workflow. The isAvailable datatype property, represented as a boolean value, provides real-time insight into the availability status of a robot. This information becomes a cornerstone for the task planner, allowing it to make informed decisions based on the readiness of the robotic resources. Two other important properties of the Robot class are the payload of the robot [kg] and its reach [cm], expressed by the three datatype properties hasPayload, hasHReach, and hasVReach.

To provide information about the demolition tool employed by a robot, the Tool class defined in Table 11 has been created. Knowing a robot's tool equipment is fundamental for effective task planning, ensuring that each demolition task assignment is directed to a robot equipped with the appropriate tool to carry out the task. The object properties hasTool and isToolOf are one the inverse of the other and represent the link between the Robot and Tool classes, facilitating a seamless representation of the tool-robot relationship.

Within both the Robot and Tool classes, additional subclasses contribute to a hierarchical structure, offering a more refined categorization within the broader top-level classes. In particular, the MobileRobot and StationaryRobot classes distinguish

between mobile platforms, that have the ability to navigate to specific locations, and stationary manipulators, which lack mobility. Regarding the Tool subclasses, the introduction of Marker, Driller, and SawBlade represents specific functions these tools fulfill within the demolition process.

Table 10: Robot class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Robot	hasRobotName	Name/ID of the robot Datatype = String
Robot	isAvailable	Availability of robot Datatype = Boolean
Robot	hasPayload	Payload of robot Datatype = Int
Robot	hasHReach	Horizontal reach of robot Datatype = Int
Robot	hasVReach	Vertical reach of robot Datatype = Int
Robot	hasTool	Tool
MobileRobot	rdfs.subClassOf	Robot
StationaryRobot	rdfs.subClassOf	Robot

Table 11: Tool class definition

Subject (Entity)	Predicate (Property/Relationship)	Object (Value/Entity)
Tool	hasToolName	Name/ID of the tool Datatype = String
Tool	isToolOf	Robot
Marker	rdfs.subClassOf	Tool
Driller	rdfs.subClassOf	Tool
SawBlade	rdfs.subClassOf	Tool

Figure 10 provides a graphical representation of the relations among the Robot and Tool classes and the definition of their respective subclasses.

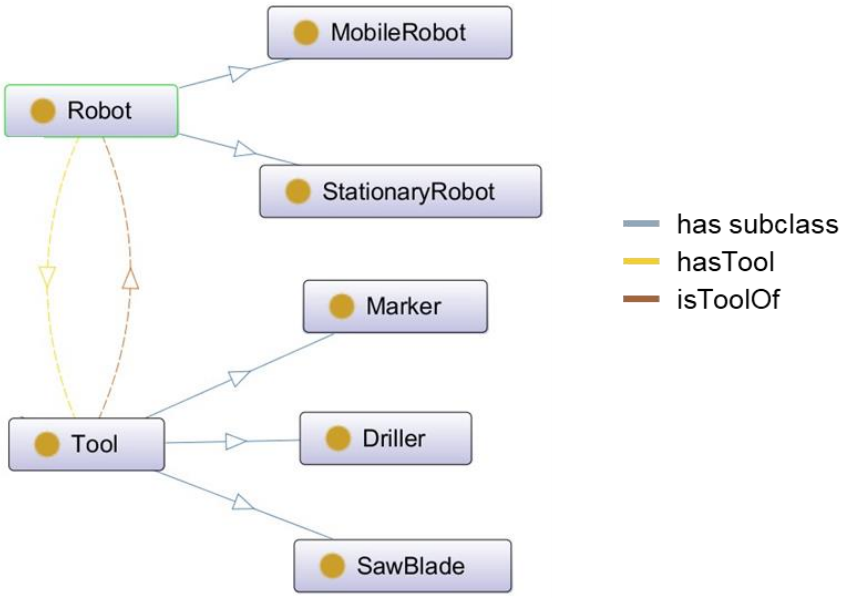

Figure 10: Graphical representation of relationships among Robot and Tool classes

3.3. Ontology architecture

Figure 11 provides an insightful overview of the ontology architecture, where the "Final RDF" element represents the core version of the ontology, including all necessary information to conduct efficient demolition task planning.

The starting point of the architecture is the IFC file which describes the specific environment where the demolition task is going to take place. The information within the IFC file is automatically converted into the RDF format using the Java components provided by "pipauwel/IFCtoRDF" GitHub repository[6]. This process yields an RDF file providing a description of all individual elements, including walls, within the specific environment.

Subsequently, the newly created RDF file is merged with the knowledge description included in the HumanTech ontology and described in the above sections of this document. This integration enriches the RDF file with essential knowledge pertaining to Robots, Tools, and Openings.

Moreover, the description of the individual IfcWalls, already included in the automatically generated RDF file, is enriched adding values to the new dimensions and locations properties introduced by the HumanTech ontology. These values are automatically

extracted from the IFC file using a Python script based on the IfcOpenShell[7] opensource library. Integration into the ontology is automatically achieved using another script based on the RDFLib[8] Python package.

Furthermore, the Final RDF file incorporates real-time information concerning the individual robot availabilities and their respective tool equipment. Additionally, this file encapsulates the opening description derived from the demolition input specified by qualified professionals.

To conclude the overview, the Final RDF file encompasses organized descriptions of all individual building elements at the demolition site, information about available robots and tools, and detailed descriptions of planned openings.

The HumanTech demolition ontology, described in the "Final RDF" file, serves as the backbone for perception and reasoning within the HumanTech system. Through logical inference on information such as robot availability and tool compatibility, the system can make informed decisions, enabling seamless task assignment and execution.

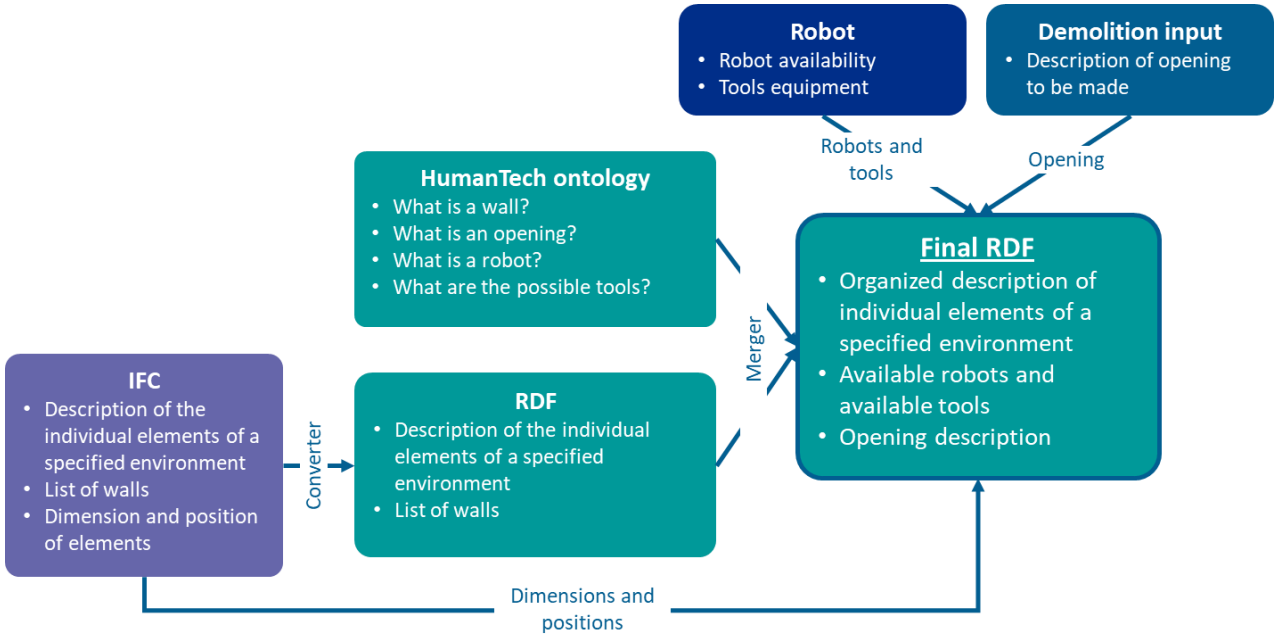

Figure 11: Ontology architecture

To provide a concrete example, Figure 12 shows the top view of the environment described in the "HT_hospital_Weingarten_C_surgery#4.ifc" file created within the

[7] https://ifcopenshell.org/

[8] https://rdflib.readthedocs.io/en/stable/

HumanTech project, visualized through the Xbim Xplorer[9] software application. This environment includes sixteen IfcWall elements.

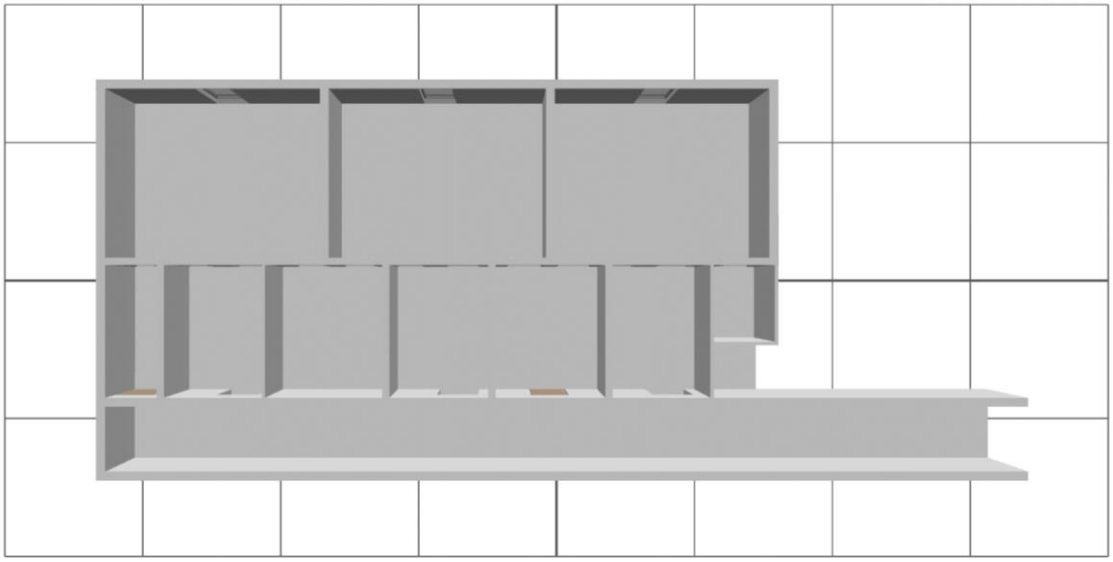

Figure 12: "HT_hospital_Weingarten_C_surgery#4" IFC file

Figure 13 shows a graphical representation of the same environment, for what concerns the IfcWall class, in the Final RDF file. This representation encapsulates sixteen distinct IfcWall individuals. Furthermore, the example incorporates a demolition input, with the corresponding Opening individual thoughtfully represented in the graph.

[9] https://docs.xbim.net/downloads/xbimxplorer.html

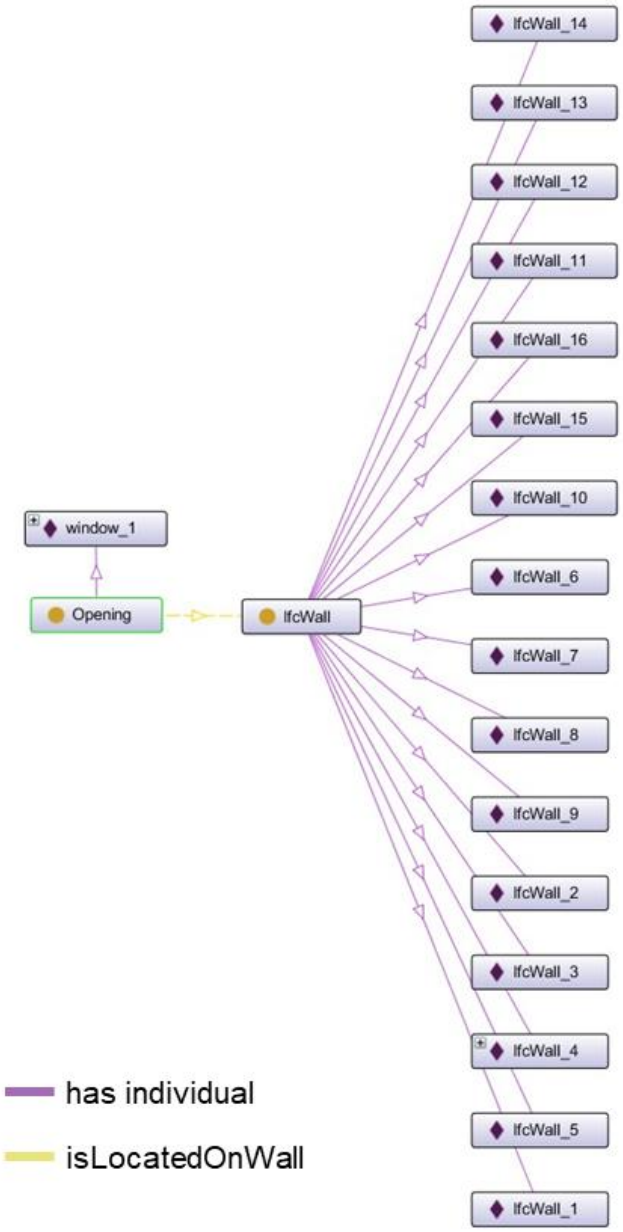

Figure 13: Graphical representation of IfcWall and Opening individuals for the "HT_hospital_Weingarten_C_surgery#4" IFC file

4. Demolition task planning for HumanTech

In the context of robotics, a task planner plays a pivotal role in orchestrating and coordinating various activities within a robotic system to achieve specific goals efficiently. It serves as the brain behind the machine, enabling it to make informed decisions, adapt to changing environments, and execute tasks with precision. The task

planner acts as the bridge between high-level objectives and low-level control, translating abstract goals into a sequence of actions that the robot can perform.

In essence, a task planner in robotics is responsible for generating plans or action sequences that guide the robot through a series of steps to accomplish a desired outcome. This process involves considering factors such as the robot's capabilities, environmental constraints, and the overall mission or task at hand. The planner must be capable of handling uncertainties, dynamic environments, and unexpected events, making it an integral component in creating versatile and adaptive robotic systems.

The sophistication of task planners varies depending on the complexity of the robotic application. Simple tasks may involve straightforward planning, while more complex scenarios, such as those encountered in industrial automation, autonomous vehicles, or search and rescue missions, demand advanced planning algorithms and decision-making processes.

In the context of HumanTech, the task planner is responsible for generating a plan for the robotic execution of the demolition activity considered in the project: cutting an opening into a wall. Once the plan, consisting in a series of high-level robotic tasks is generated, the task planner should check its feasibility with the available resources. This step is made possible thanks to the communication with the HumanTech ontology, storing an always updated description of the available robots and tools equipment. If a robot with the appropriate abilities is available to perform a demolition task, then the task planner can proceed triggering the specified task.

4.1. Demolition task definition

The task planner is responsible of delineating a series of high-level robotic activities that must be executed to accomplish the precise cutting of an opening into a wall.

In the HumanTech demolition planning, four primary robotics actions are considered: navigation, marking, drilling, and cutting. The meticulous order of these activities is paramount, requiring a specific sequence to ensure the successful creation of the wall opening. The task planner's objective is to generate this precise order, referred to as a plan, and to identify the exact locations where these activities must be executed.

In the subsequent three sub-sections of this document (4.1.1, 4.1.2, and 4.1.3), comprehensive descriptions of the specific plans required for accomplishing the marking, drilling, and cutting operations are provided.

4.1.1. Marking

The robotic marking task consists in drawing some marks on the wall in correspondence of some distinctive points of the opening, such as its corners. The exact locations where the markers should be made are the ones specified by the openingPoints stored in the HumanTech ontology, visually highlighted in red in Figure 7.

In the HumanTech project, to perform the marking task, the manipulator installed on the HumanTech robotic platform (see deliverable D5.3 for details) will be equipped with a marking tool. As a mobile platform, the robot undergoes a two-step process: first, navigating to the correct position in front of the wall, and second, employing the manipulator and its marking tool to draw the marker on the wall.

The navigation process involves several sub-tasks, including localization, path-planning, and motion control. Precise coordination is crucial to ensure the robot's arrival at the correct location. For marking, accurate localization is vital to ensure the marking tool aligns perfectly with the desired location.

To determine the robot's navigation positions, calculations are made to position it directly in front of where the mark should be, maintaining a consistent distance of 50 cm from the wall. The robot's orientation is crucial, requiring the front side of the platform to face the wall. These computations are made considering the direction value of the opening (either 1 or -1) and the yaw rotation of the specific wall, information both included in the ontology.

In scenarios where multiple openingPoints align on the same vertical line, the robot's navigation positions remain constant. Thus, the operational logic involves moving the robot to a specific position and executing all necessary marks situated in front of that designated location.

To illustrate visually, consider the scenario with eight openingPoints as represented in Figure 14. In this case, the mobile platform should navigate to the three distinct positions highlighted in the figure by an arrow. For instance, when the robot aligns with the red arrow, the manipulator is tasked with marking the three points highlighted in red. This

navigation-to-marking correlation persists for the orange and yellow arrows and their respective points.

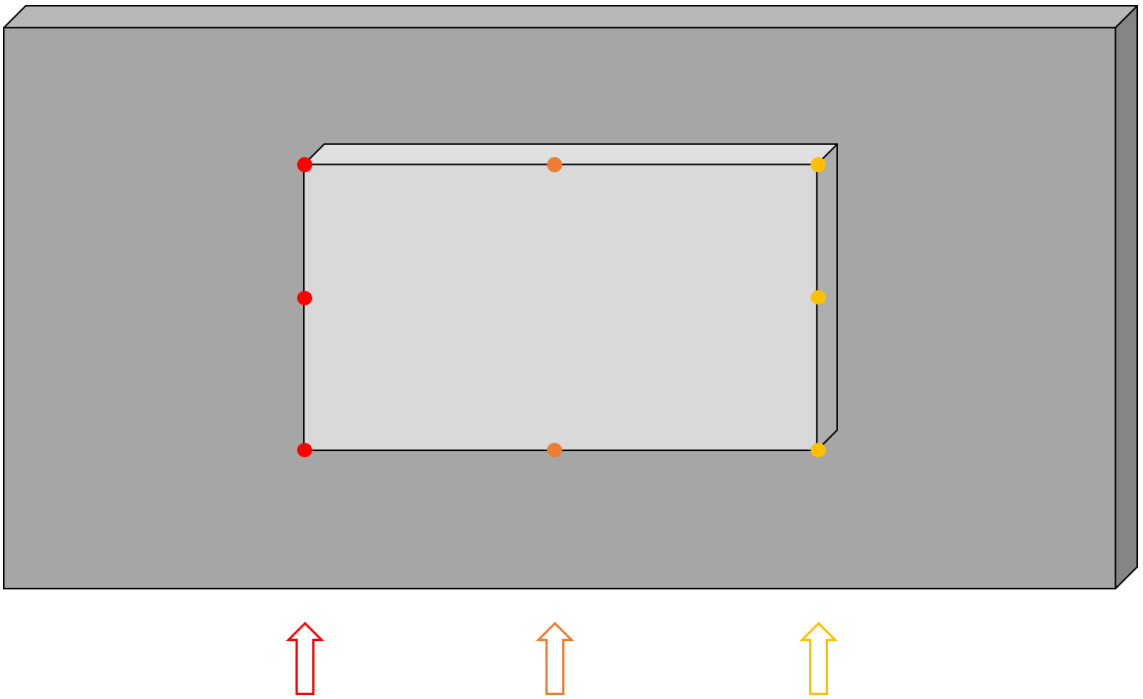

Figure 14: Marking task

The precise workflow of the marking plan generation is illustrated in Figure 15. The locations of the openingPoints are extracted from the ontology, and for each of these points, a corresponding robot navigation position is meticulously calculated. Subsequently, the computed navigation positions are analysed to eliminate duplicates arising from openingPoints aligned on the same vertical line. This process ensures the establishment of a direct and unambiguous correlation between the robot navigation positions and the specific openingPoints that must be marked when the robot occupies that specific position.

At this point, the complete lists of sub-tasks necessary to complete the marking operation can be defined.

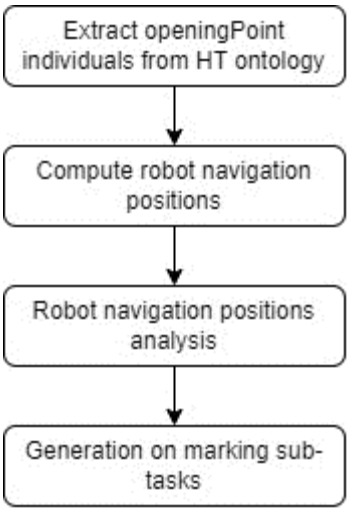

Figure 15: Flow of the marking plan generation

The content of the marking plan is reported in Figure 16, and it encompasses navigation to a specific position and marking all openingPoints in front of that position, continuing until all positions have been visited and all points have been marked.

The tasks are triggered by directly communicating with the specific robot assigned to carry them out. Concerning navigation, the information given to the mobile platform controller consists of the exact pose (including position and orientation) that it has to reach. For marking, the manipulator controller is provided with the 3D position of the openingPoint and the current pose of the mobile platform. All actors involved are expected to provide feedback to the task planner upon completing the corresponding task. This feedback loop enables the task planner to know precisely when to proceed with triggering the subsequent tasks in the sequence.

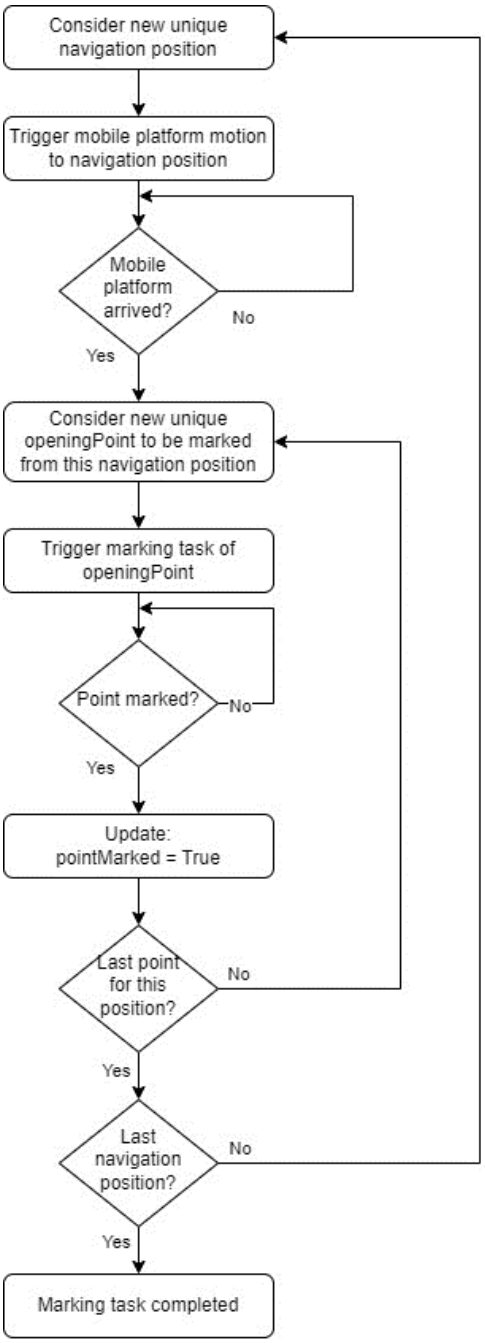

Figure 16: Marking plan

To offer a more concrete example, let us take into account the "HT_hospital_Weingarten_C_surgery#4.ifc" IFC file, whose content is visually shown in Figure 12, and a demolition input described by the parameters listed in Table 12.

Table 12: Example of demolition input values

Parameter	Value

Opening name	window_1
Wall name	Basiswand:KS 115:2502143
Height	1.5
Width	0.3
Length	1.0
X	0.35
Y	0.5
Angle	0.0
Direction	1

For a complete overview, Table 13 provides some attributes of the "Basiswand:KS 115:2502143" wall.

Table 13: Description of wall "Basiswand:KS 115:2502143"

Attribute	Value
Wall name	Basiswand:KS 115:2502143
(X, Y, Z) coordinates	(0.46, 1.01, -1.0)
(yaw, pitch, roll) rotations	(-180.0, 0.0, 0.0)
Height	3.0
Width	0.11
Length	2.02

Based on this information, the marking plan generated by the task planner is the one described in Figure 17.

```
At position (-0.89, 1.52) with yaw orientation (-90.0) make the following marks:
(-0.89, 1.02, -0.5)
(-0.89, 1.02, 0.25)
(-0.89, 1.02, 1.0)
At position (-0.39, 1.52) with yaw orientation (-90.0) make the following marks:
(-0.39, 1.02, 1.0)
(-0.39, 1.02 -0.5)
At position (0.11, 1.52) with yaw orientation (-90.0) make the following marks:
(0.11, 1.02, 1.0)
(0.11, 1.02, 0.25)
(0.11, 1.02 -0.5)
```

Figure 17: Example of marking plan for wall "Basiswand:KS 115:2502143"

To illustrate this plan visually, Figure 18 provides a drawing of the top view of the walls of the environment, with the "Basiswand:KS 115:2502143" wall highlighted in orange. The three red dots in the figure denote the positions where the mobile robot should navigate according to the marking plan, with Cartesian coordinates (X, Y) as follows: (-0.89, 1.52), (-0.39, 1.52), (0.11, 1.52). For completeness, the origin of the wall, represented by (0.46, 1.01) coordinates, has been illustarted with a red circle. In addition, Figure 19 offers front and back views of the 'Basiswand:KS 115:2502143' wall, with the opening highlighted in yellow and the eight marking points from the plan in red. In this example, the direction of the opening was equal to 1, meaning it was to be made on the front side of the wall, where the front side is the one that has the origin of the wall at the bottom left corner.

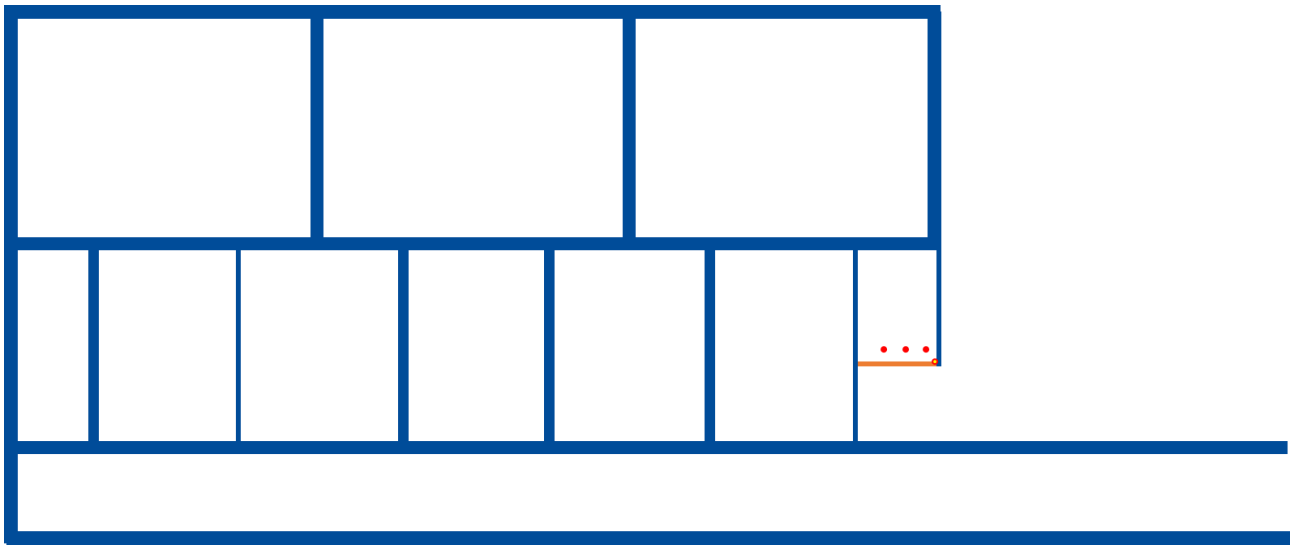

Figure 18: Navigation positions for marking wall "Basiswand:KS 115:2502143"

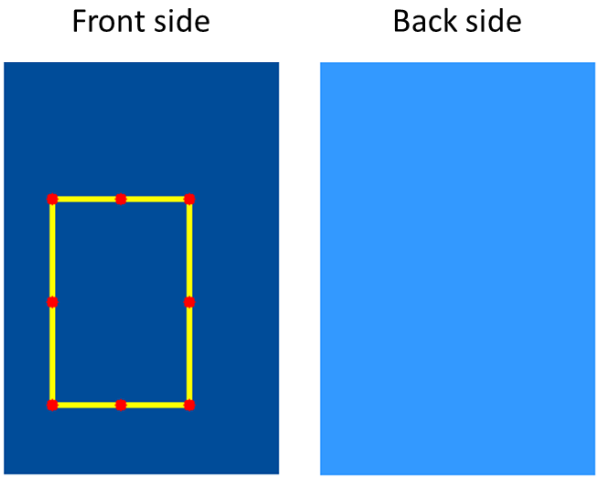

Figure 19: Marking points for wall "Basiswand:KS 115:2502143"

Figure 20 and Figure 21 provide again a visual representation of the navigation positions within the environment and the marking points on the wall, for the case in which the direction of the opening would be equal to -1, signifying that the opening was to be created on the back side of the wall.

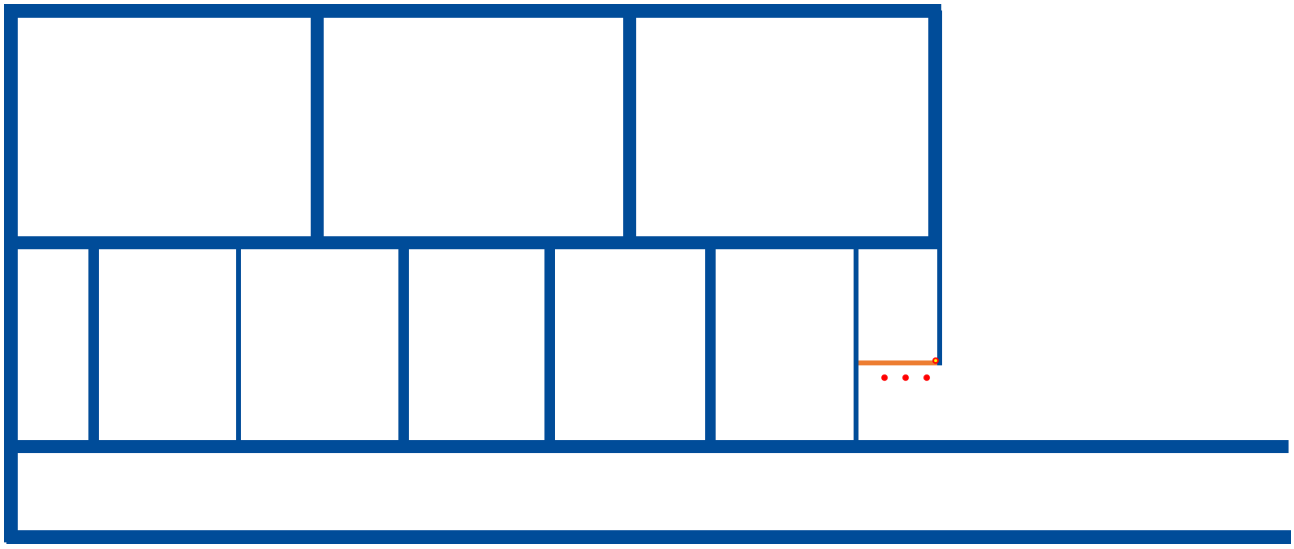

Figure 20: Navigation positions for marking wall "Basiswand:KS 115:2502143" – Direction = -1

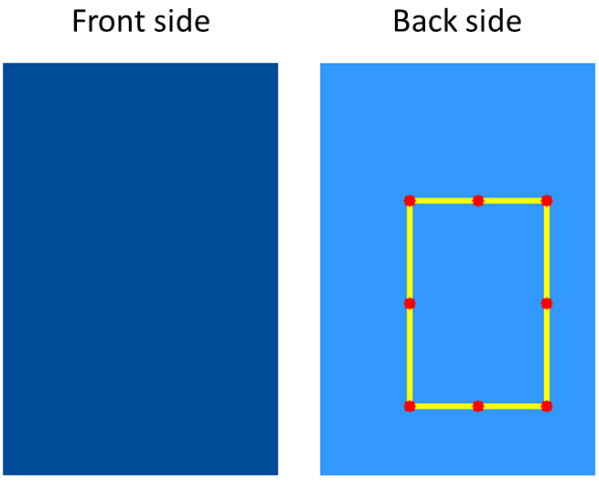

Figure 21: Marking points for wall "Basiswand:KS 115:2502143" – Direction = -1

4.1.2. **Drilling**

The robotic drilling task is structured in the same exact way of the marking task, and it consists in drilling holes on the wall in correspondence of the specific openingPoints stored in the HumanTech ontology.

This task would be typically assigned to a mobile platform with a manipulator equipped with a drilling tool, similarly to the HumanTech platform with the marker. However, in

the HumanTech project, a drilling tool was not directly available. Consequently, this specific task is not addressed within the project, but the demolition task planner takes it into consideration for the comprehensive application scope.

Since, once again, a mobile platform is involved, similar to the marking task, the robot undergoes a two-step process. First, it navigates to the correct position in front of the wall. Second, it employs the manipulator along with the drilling tool to create holes in the wall. The drilling process particularly demands tight control over both the tool and the robot to guarantee precision and safety.

Figure 22 illustrates the navigation positions and hole locations for the scenario with eight openingPoints. The mobile platform should navigate to the three distinct positions highlighted by an arrow, and in each position it should drill all the holes of the same colour as the arrow.

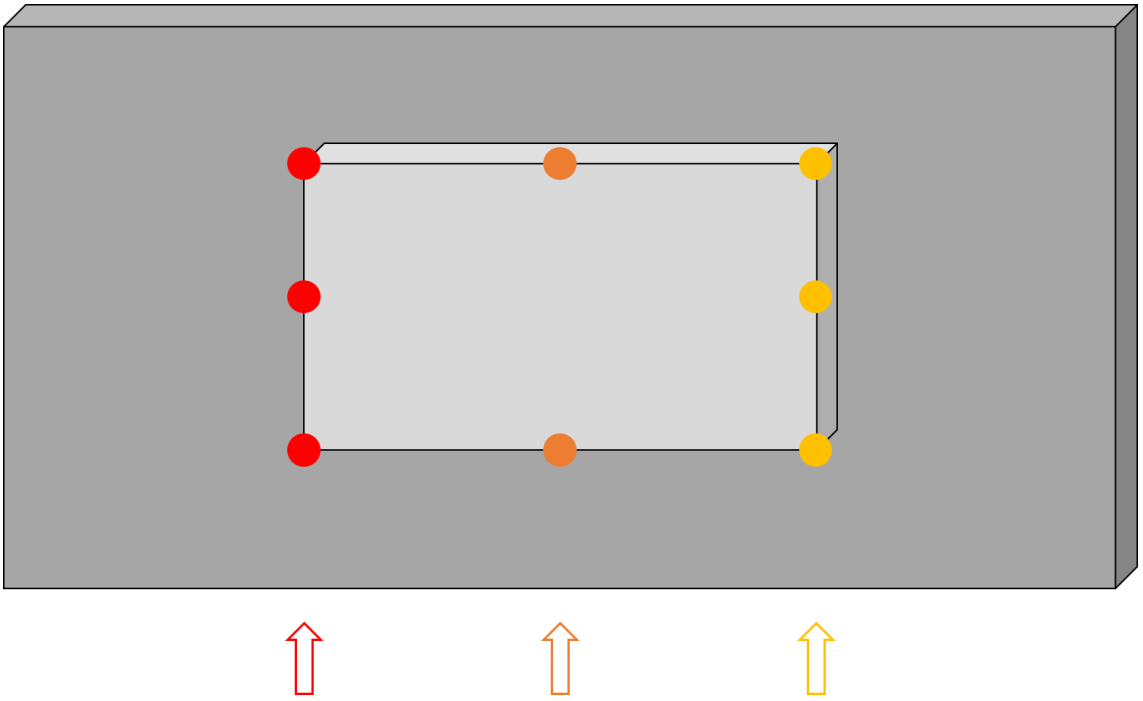

Figure 22: Drilling task

The precise workflow of the drilling plan generation is illustrated in Figure 23, and is exactly equivalent to the one explained for the marking task.

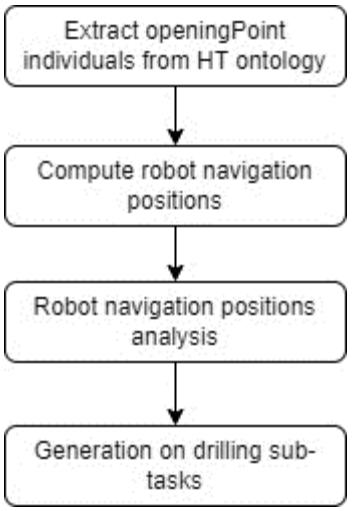

Figure 23: Flow of the drilling plan generation

The details of the drilling plan are shown in Figure 24, including navigation to specific positions and drilling of all openingPoints situated in front of those designated locations. This sequence persists until all positions have been visited, and every corresponding point has been successfully drilled.

The tasks are triggered by directly communicating with the specific robot assigned to carry them out. Concerning drilling, the manipulator controller is provided with the 3D position of the openingPoint and the current pose of the mobile platform.

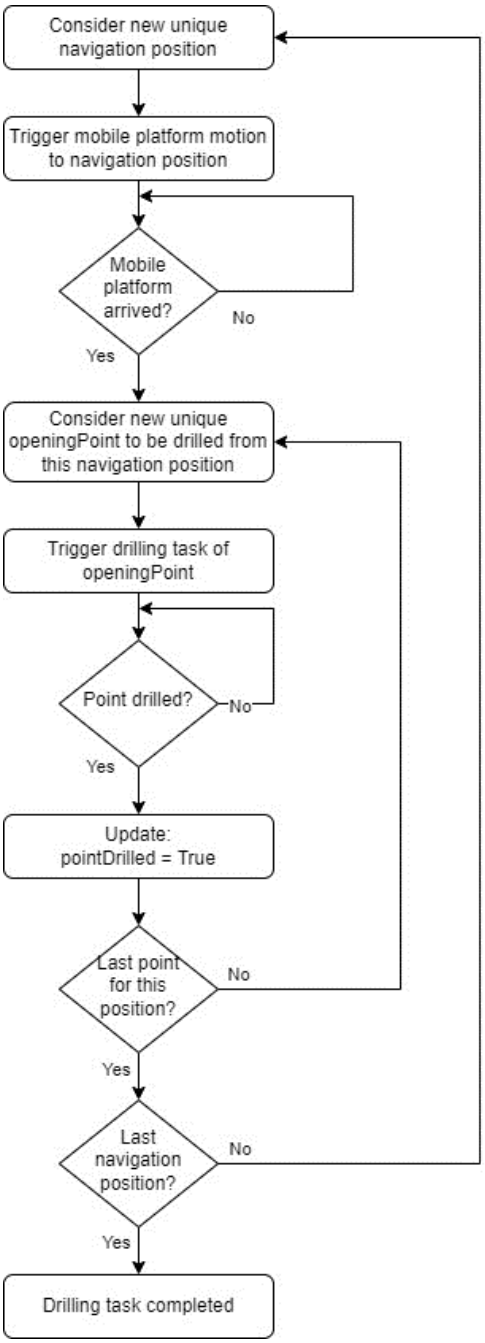

Figure 24: Drilling plan

4.1.3. **Cutting**

The robotic cutting task consists in cutting the wall in correspondence of the edges of the opening. The exact locations where the cuts should be performed are the ones specified by the openingSides stored in the HumanTech ontology, visually highlighted in red in Figure 8.

In the HumanTech project, a stationary robot equipped with a circular sawblade is designated for the cutting task. This selection is grounded in the necessity for heavy

equipment to carry out the cutting task, typically unsuitable for mobile platforms' manipulators due to payload limitations. A stationary robot can have instead an enhanced load capacity, ideal for handling heavier tools. Additionally, it also has a wider reach, so, if positioned in a suitable position, it is able to cover larger distances without the need to relocate its base.

Given the stationary robot context, the need for navigation is eliminated, and the cutting process focuses solely on utilizing the manipulator and its blade to cut the sides of the opening. This demands meticulous control over both the tool and the robot.

To illustrate visually, consider the opening represented in Figure 25. Here, the stationary robot is positioned centrally concerning the length of the opening, as indicated by the colored arrow in the figure. From this central position, the robot systematically cuts the various sides of the opening, delineated by the yellow, orange, red, and burgundy colors.

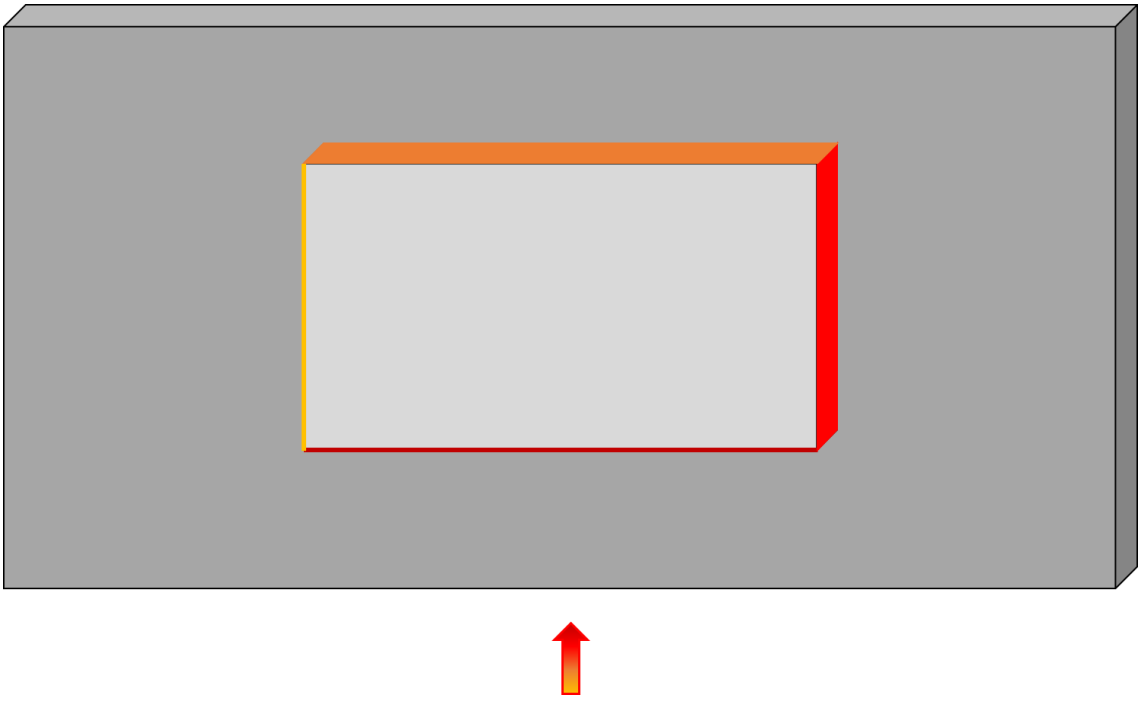

Figure 25: Cutting task

The specific workflow for generating the cutting plan is illustrated in Figure 26. It is relatively less intricate compared to the marking and drilling tasks, as in this case, the computation of navigation positions is unnecessary. Following the extraction of openingSides locations from the ontology, it becomes straightforward to define a comprehensive list of sub-tasks essential for executing the cutting operation.

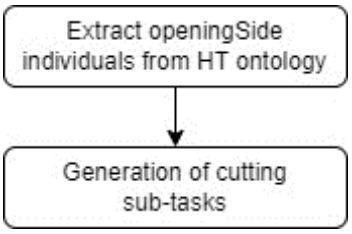

Figure 26: Flow of the cutting plan generation

The content of the cutting plan is reported in Figure 27, and it encompasses performing the cutting operation for all openingSides.

The tasks are triggered by directly communicating with the specific robot assigned to carry them out. For cutting, the manipulator controller is provided with the 3D position of the two points located at the extremity of the specific openingSide, and the current global pose of the robot base.

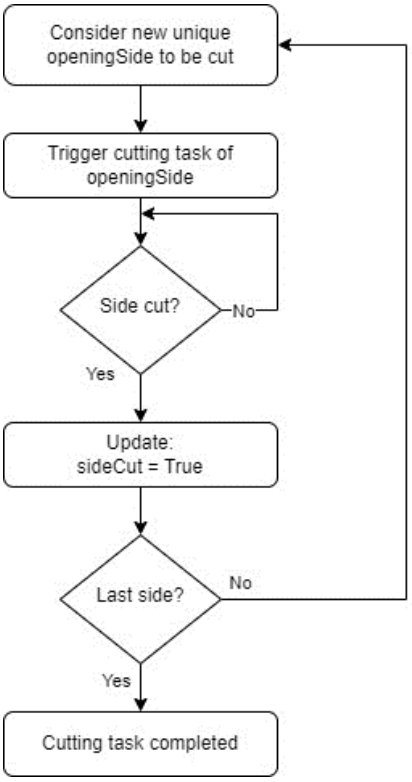

Figure 27: Cutting plan

4.2. Task planning and scheduling

After generating the plan, which comprises a series of high-level robotic tasks, the task planner must assess its feasibility based on the available resources. This evaluation is facilitated through communication with the HumanTech ontology, storing an always

updated description of the available robots and tools equipment. If a robot possessing the requisite capabilities is available for a demolition task, the task planner proceeds to initiate the specified task.

Going into more details, the precise flow is represented in Figure 28. It commences with the reception of a new demolition input detailing the desired opening, following the information described in Table 5. These input parameters are then automatically translated into the concepts and properties describing the Opening individual within the HumanTech ontology.

Initially, all the properties of the openingStatus - openingMarked, openingDrilled, and openingCut – are all set to all False, signifying that the Opening has not undergone marking, drilling, or cutting. It is by checking the values of these key properties that the planning of the demolition activities starts. The first property checked is openingMarked, as marking is the initial robotic task to be carried out. If the value of this datatype property is equal to False, indicating that the opening has not been marked yet, the process can continue with planning marking. Otherwise, the next property, openingDrilled, is examined, as drilling is the subsequent demolition activity. Following the same logic, if its value is False, the drilling task is planned; otherwise, the openingCut property is checked. If its value is False, it is possible to proceed with planning the cutting activity, otherwise, it means that the opening has already been marked, drilled and cut and thus the demolition task is completed.

Once the specific robotic task to be performed has been selected - whether marking, drilling, or cutting - the next step involves checking the ontology to confirm the availability of a robotic system for that particular task. In the case of marking, the ontology is queried for MobileRobot equipped with a Marker, with the isAvailable property currently set to True. Similarly, for drilling, the search involves an available MobileRobot equipped with a Driller tool. For cutting, a currently available StationaryRobot equipped with a SawBlade is searched. Additionally, the robot's reach should be compatible with the opening dimensions since it lacks mobility. Therefore, the hasHReach property of the robot should be bigger than the length of the opening, and the hasVReach value should exceed the opening's height. These dimensional properties of the opening are stored under the hasLength and hasHeight datatype properties of the Opening individual.

If a suitable robot instance meeting these criteria is found, it is selected for the specific demolition task, and its availability is set to False. Otherwise, if no suitable robot is found, it indicates that the plan is currently unfeasible.

Upon confirming the availability of the correct robot, the plan for marking, drilling, or cutting is generated, comprising the sub-tasks outlined in Figure 16, Figure 24, and Figure 27. These sub-tasks are subsequently triggered by direct communication with the available robot instance. Once the entire marking, drilling, or cutting task is completed, the openingStatus is updated accordingly, and the process restarts determining if the opening requires additional demolition tasks.

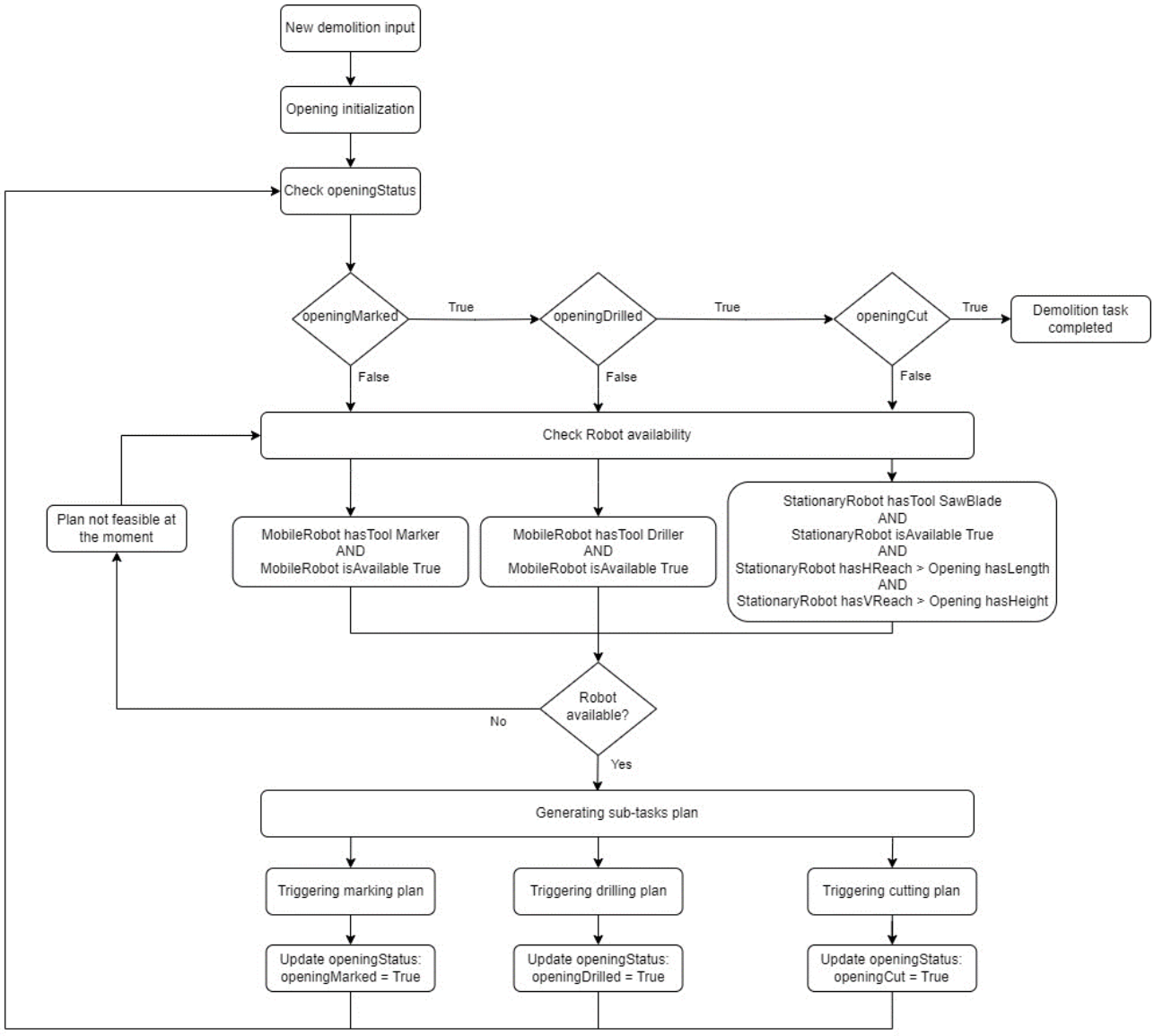

Figure 28: Task planning flow

In real-world scenarios, the complexity of the task assignment process can escalate, particularly when contemplating the potential deployment of a fleet of robots within the environment. In such dynamic settings, considerations extend beyond the mere availability of a single robot for a specific task. The task planner would need to consider the spatial distribution of the robotic fleet in relation to the opening's location. Proximity becomes a crucial criterion, as it influences the efficiency and speed of task execution. Additionally, the task planner would need to evaluate the capabilities of each robot within the fleet. This involves assessing not only the general availability but also the proficiency of the robots in performing specific tasks. Some tasks might be commonplace, with numerous robots equipped to handle them, while others could be more specialized and require a meticulous selection process. Consequently, the task assignment process should be adaptive, taking into account the diverse capabilities and locations of the robotic fleet, ensuring an optimized and efficient execution of demolition activities in real-world scenarios.

While the HumanTech project's task planner operates within a simplified framework, recognizing this potential complexity offers a pathway for future enhancements.

5. ROS-based integration

The integration of the Robot Operating System (ROS) plays a pivotal role in the operationalization of HumanTech's demolition system. ROS facilitates communication and coordination between the various components, ensuring a cohesive and synchronized execution of demolition tasks.

Figure 29 shows an overview of the ROS architecture. The three main actors here are the robots, the HumanTech ontology, and the task planner. Different ROS services have been implemented to allow the communication between these actors. In the graph, the services have been divided into three groups, depending on who is the service provider and who is the client of the specific service. In particular, the clients are the elements from which the arrows start, and the service providers are the elements where the arrows end.

Regarding the communication between the robots and the ontology, the created services are related to update the ontology regarding the robots' conditions. This

includes adding and removing robots, changing their availability status and changing their tools equipment.

For the communication between the task planner and the ontology, the available services are related to:

- initialize a new demolition input, this includes adding to the ontology a new opening and its opening points and sides;
- check and update the status (i.e. related to marking, drilling, and cutting operations) of an opening and of opening points and sides;
- get from the ontology the complete list of opening points and sides, which are the key locations for the demolition activities;
- check the availability of robots equipped with correct tools to perform some specific demolition activities;
- get information about IfcWalls.

For what concerns the communication between the task planner and the robots, the ROS services pertain the triggering of the navigation, marking, drilling and cutting tasks.

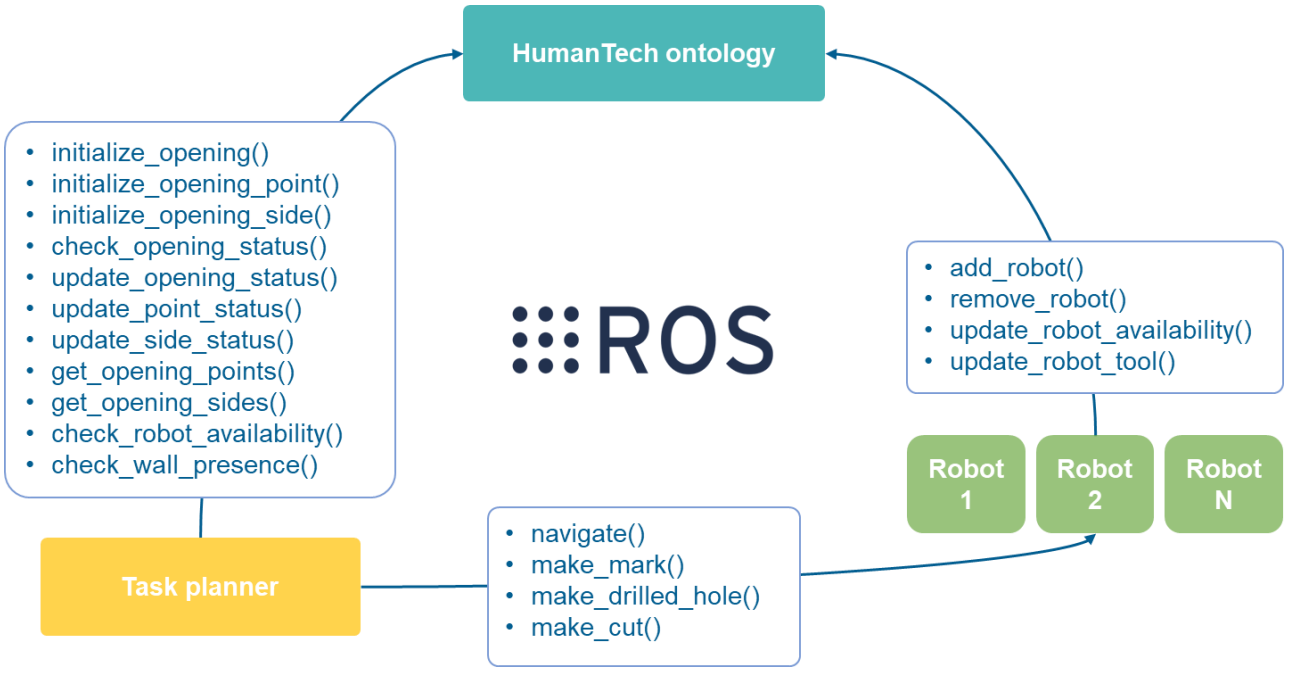

Figure 29: ROS Architecture

Below, for each ROS service a table detailing its requests and reply parameters is provided.

addRobot.srv

Table 14: addRobot.srv

Type	Name
string	robot_name
string	robot_type
string[]	tool_names
string[]	tool_types
int16	payload
int16	vertical_reach
int16	horizontal_reach
---	
bool	done

removeRobot.srv

Table 15: removeRobot.srv

Type	Name
string	robot_name
string[]	tool_names
---	
bool	done

updateRobotAvailability.srv

Table 16: updateRobotAvailability.srv

Type	Name
string	robot_name
bool	availability
---	
bool	done

updateRobotTool.srv

Table 17: updateRobotTool.srv

Type	Name
string	robot_name
string[]	tool_names_to_remove
string[]	tool_names_to_add
string[]	tool_types_to_add
---	
bool	done

checkRobotAvailability.srv

Table 18: checkRobotAvailability.srv

Type	Name
string	robot_type
string	tool_type
int16	vertical_reach
int16	horizontal_reach
---	
bool	availability
string	robot_name
string	tool_name

checkWallPresence.srv

Table 19: checkWallPresence.srv

Type	Name
string	wall_name
---	
bool	found
float32	x
float32	y
float32	z
float32	yaw
float32	pitch

float32	roll
float32	height
float32	width
float32	length

initializeOpening.srv

Table 20: initializeOpening.srv

Type	Name
string	opening_name
string	wall_name
float32	height
float32	width
float32	length
float32	x
float32	y
float32	yaw
int16	direction
---	
bool	done

initializeOpeningPoint.srv

Table 21: initializeOpeningPoint.srv

Type	Name
string	opening_name
int16	n_point
float32	x
float32	y
float32	z
bool	corner
---	
bool	done

initializeOpeningSide.srv

Table 22: initializeOpeningSide.srv

Type	Name
string	opening_name
int16	n_side
float32	x_start
float32	y_start
float32	z_start
float32	x_end
float32	y_end
float32	z_end
---	
bool	done

checkOpeningStatus.srv

Table 23: checkOpeningStatus.srv

Type	Name
string	opening_name
---	
bool	marking_status
bool	drilling_status
bool	cutting_status

updateOpeningStatus.srv

Table 24: updateOpeningStatus.srv

Type	Name
string	opening_name
string	what_status
string	new_value
---	
bool	done

updatePointStatus.srv

Table 25: updatePointStatus.srv

Type	Name
string	opening_name
float32	x
float32	y
float32	z
float32	what_status
float32	new_value
---	
bool	done

updateSideStatus.srv

Table 26: updateSideStatus.srv

Type	Name
string	opening_name
float32	x_start
float32	y_start
float32	z_start
float32	x_end
float32	y_end
float32	z_end
float32	new_value
---	
bool	done

getOpeningPoints.srv

Table 27: getOpeningPoints.srv

Type	Name
string	opening_name
bool	corner
---	

float32[]	x
float32[]	y
float32[]	z

getOpeningSides.srv

Table 28: getOpeningSides.srv

Type	Name
string	opening_name
---	
float32[]	x_start
float32[]	y_start
float32[]	z_start
float32[]	x_end
float32[]	y_end
float32[]	z_end

navigate.srv

Table 29: goToWall.srv

Type	Name
string	robot_name
string	wall_name
float32	x
float32	y
float32	yaw
---	
bool	done

makeMark.srv

Table 30: makeMark.srv

Type	Name
string	robot_name

string	opening_name
float32	x
float32	y
float32	z
float32	x_mobile_platform
float32	y_mobile_platform
float32	yaw_mobile_platform
---	
bool	done

makeDrilledHole.srv

Table 31: makeDrilledHole.srv

Type	Name
string	robot_name
string	opening_name
float32	x
float32	y
float32	z
float32	x_mobile_platform
float32	y_mobile_platform
float32	yaw_mobile_platform
---	
bool	done

makeCut.srv

Table 32: makeDrilledHole.srv

Type	Name
string	robot_name
string	opening_name
float32[]	x_start
float32[]	y_start
float32[]	z_start

float32[]	x_end
float32[]	y_end
float32[]	z_end
float32[]	width
---	
bool	done

6. Towards an actual deployment: Pilot III

In the context of the HumanTech project, the ontology and demolition task planner described in this document will be demonstrated in Pilot III: Remote controlled demolition. In particular, the HumanTech robotic platform (see deliverable D5.3 for details) will be used to demonstrate the marking task application.

Some development effort has therefore been made towards a concrete architecture. Figure 30 gives an overview of the current consensus of this architecture together with the fundamental control system architecture, in a combined UML component and deployment diagram.

At the top level a human operator, the Demolition Operator, initiates a session with the singleton component DemolitionTaskManager for establishing a demolition task. Reference information for establishing a demolition task will have been retrieved from an interaction with the BIMModel component.

The DemolitionTaskManager creates a new DemolitionTaskPlanner for planning and executing the sub-tasks. The task planner queries the OntologyService, first for getting feature information and analysing the given demolition task into sub-tasks, and then for seeking available and capable robots to execute the sub-tasks.

The DemolitionTaskManager, DemolitionTaskPlanner, and OntologyService components are all reflected as deployed on the same PC node, Offboard Application. The components DemolitionTaskManager and OntologyService are shown as singletons, which suffices for the sake of the pilot and demonstration.

For the physical HumanTech robotic system to be able to be available and capable to solve a sub-task, a task-specific application must be deployed on board. In Figure 30, the Onboard Applications is a PC for hosting the onboard application specific to solving

marking tasks, which are sub-tasks under a demolition task. In addition to an onboard application PC there is one further customization to the physical HumanTech robotic system: the addition of a marking tool sensor, which may publish or be queried about the current state of contact of the marker in the tool.

At the top level of the onboard application is the MarkingTaskOrchestrator component. It is responsible for providing the availability and capability of the pertinent robot system under its command, which is required by the OntologyService component. It is also responsible for the execution of a set of marking sub-tasks assigned to it from a DemolitionTaskPlanner that have obtained reservation of service.

Once the MarkingTaskOrchestrator has received a list of marking tasks, it will iterate over them, while orchestrating the onboard system's Navigation component and the onboard application's MarkingTaskExecutor. For each task, the mobile base is commanded to a suitable target pose near the structure on which to set a mark, chosen such that the position and direction of marking is within reach of the robot arm. This mobile base motion is executed by the Navigation component. Once in pose, the most accurate estimate of the final pose of the mobile base is acquired from the Navigation component. The mobile base pose and the mark position and direction are now all in common building or construction site reference. Thus, the marking task can be formulated with reference to the robot arm base and executed by the MarkingTaskExecutor component.

For the physical tool for marking, an instrumented marker pen holder for mounting at the robot arm flange has been developed. Mechanically it maintains the direction of the pen, while allowing translational compliance along the pen axis with spring return to resting position. A magnetic sensor can detect if the travel of the pen into the holder is above some threshold, signalling contact. For real-time sensing into the controlling application, the node Marking Sensor System hosts the MarkingToolSensor component, which continually monitors the magnetic sensor and publishes the contact state over UDP. This is picked up and integrated into ROS by the MarkingToolAdapter component.

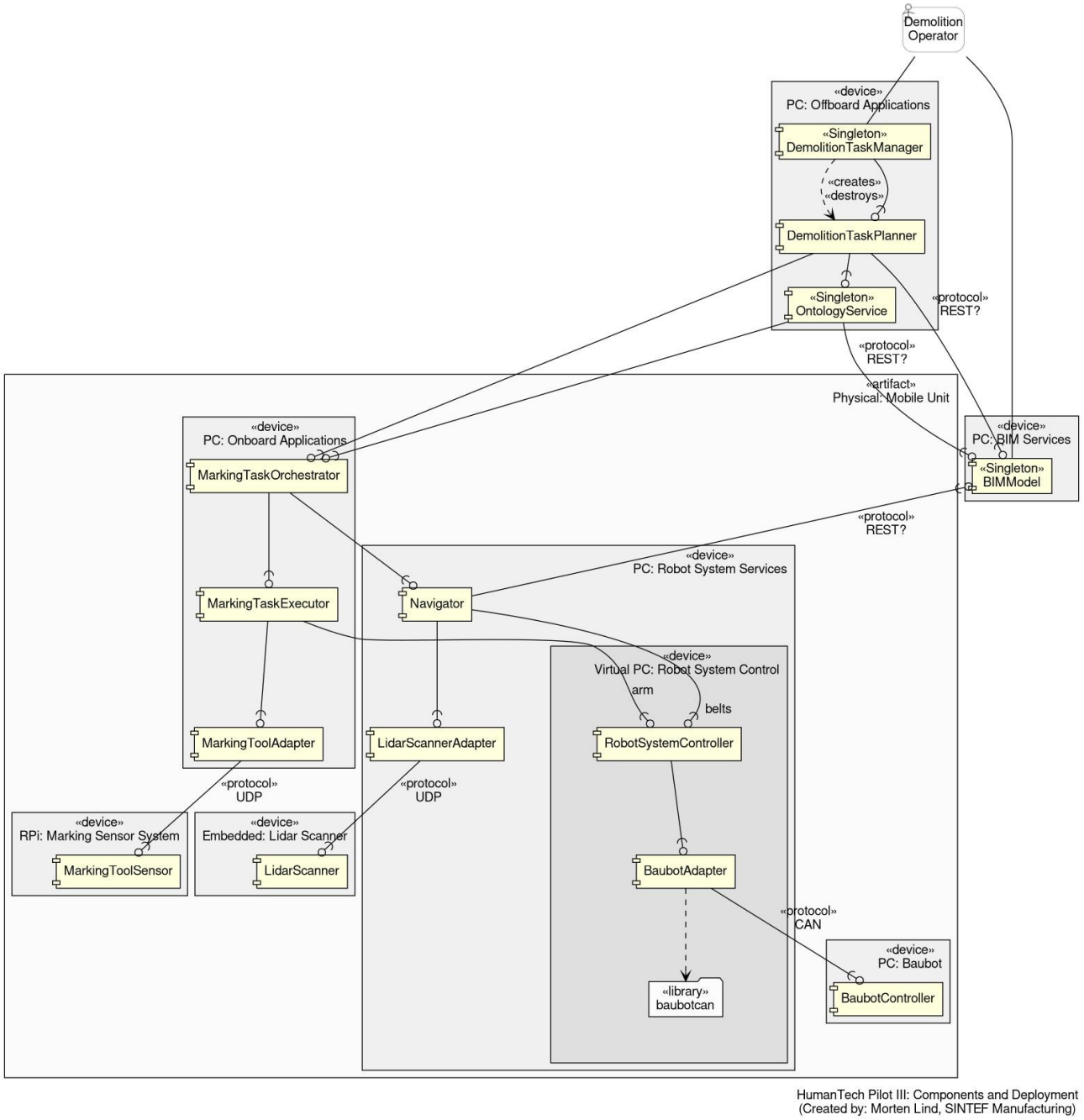

Figure 30: Marking task application architecture

7. Conclusions

In the field of demolition, the HumanTech project seeks to redefine conventional practices by introducing cutting-edge automation. The central focus is on developing a task planner, a strategic tool empowering HumanTech demolition robots to perform critical tasks autonomously. This paradigm shift aims to diminish the risks associated

with manual labor, with human operators playing supervisory roles to ensure the successful execution of tasks in this new paradigm.

This report presents a comprehensive description of the work carried out during Task 5.1 “Robotic demolition task planning”, dedicated to the development of the task planner.

The demolition ontology, an extension of ifcOWL, serves as the project's cognitive foundation, offering a detailed representation of the demolition environment, encompassing walls, openings, and robots. The ontology's architecture establishes a structured knowledge framework vital for automated task planning.

Demolition task planning is intricately detailed, focusing on marking, drilling, and cutting operations. These tasks are executed seamlessly through the task planner, which assesses feasibility with available resources. The integration with the Robot Operating System (ROS) facilitates efficient communication, allowing the generated plans to be executed by robotic systems.

As the project advances towards deployment, Pilot III emerges as a pivotal scenario for validating and refining the task planner theoretical framework into a practical setting. This phase will serve as a crucial testing ground, offering insights into the application's real-world efficacy.

In conclusion, the synergic collaboration between the demolition ontology, the task planner and the robotic systems allows to plan and execute the cutting of openings into existing walls efficiently and accurately.