FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Space-based Task Allocation in Robotic Fleets with Adaptive Autonomy

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

M.Sc. Domagoj Drenjanac
Matrikelnummer 0929086

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. DI eva Kühn

Diese Dissertation haben begutachtet:

eva Kühn | Hans-Peter Schwefel

Wien, 15. Juni 2016

Domagoj Drenjanac

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Space-based Task Allocation in Robotic Fleets with Adaptive Autonomy

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

M.Sc. Domagoj Drenjanac
Registration Number 0929086

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ. Prof. Dr. DI eva Kühn

The dissertation has been reviewed by:

eva Kühn	Hans-Peter Schwefel

Vienna, 15th June, 2016

Domagoj Drenjanac

Erklärung zur Verfassung der Arbeit

M.Sc. Domagoj Drenjanac
Siebenbrunnengasse 66/27, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Juni 2016

Domagoj Drenjanac

Acknowledgements

First of all, I would like to thank my supervisors, who provided me with the motivation and inspiration when it was needed the most. Dana Tomić who spent countless hours in reviewing our joint papers and giving them the final touch that always increased the quality. She was always eager to invest additional time in our endless discussions which were steering and motivating my work. My university supervisor, eva Kühn, always steered my research in the right direction when it faced a crossroad. I would like to thank her for being supportive and helpful, especially during the writing phase of this thesis where her contribution put the thesis in good shape. My second university supervisor, Hans-Peter Schwefel, who agreed to co-supervise me and also provided me with valuable comments which significantly increased the readability of thesis and thus made it understandable to the experts in other technical fields. I would also like to thank the FTW for giving me an opportunity to work with incredible people and on amazing projects.

Finally, I would like to thank my family, my girlfriend who in the meanwhile became my wife and who shared innumerable good and bad time with me during this work. A special thanks goes to my friends who were always understandable when I had to refuse to go out with them. Without all of you, I have no doubt that this thesis would not have been completed.

Kurzfassung

Eine vernetzte Roboterflotte besteht aus Robotern, die eine gemeinsame Mission durchführen können. Da die einzelnen Roboter sich in ihrer Umgebung koordinieren können, um ihre besonderen, oft heterogenen Kapazitäten bestens in der gemeinsamen Mission einzusetzen, ist der Einsatz einer Roboterflotte effizienter als der Einsatz eines einzigen multifunktionalen Roboters. Allerdings ist der Einsatz einer Roboterflotte heutzutage meistens auf strukturierte und kontrollierte Umgebungen begrenzt, wo die Mission der Flotte durch ein zentralisiertes System gesteuert wird.

In dieser Doktorarbeit wurden verteilte Steuerungsansätze erforscht und Methoden und Algorithmen für die Anwendung von Roboterflotten in heterogenen und unstrukturierten Umgebungen entwickelt, in denen die Mission in einem Wechselspiel zwischen Robotern und Menschen ausgeführt wird. In der Arbeit wird ein bereits etabliertes zentralisiertes Koordinationsverfahren mit mehreren neuartigen verteilten Algorithmen verglichen. Die Arbeit adressiert neue Anforderungen an die Interkommunikation, die Aufgabenverteilung, die Koordination und die Zusammenarbeit von Roboterflotten, die sich aus den Besonderheiten von Flotteneinsätzen ergeben. Insbesondere ist die zuverlässige Kommunikation zwischen den Robotern der Flotte für eine effiziente Koordination erforderlich.

Dem in der Arbeit entwickelten neuartigen Koordinationsmodell liegt eine effiziente Aufgabenverteilung zugrunde, die eine komplexe Mission in einzelne Aufgaben unterteilt, und diese einzelnen Robotern entsprechend ihrer Kapazitäten optimal zuweist. In diesem verteilten Verfahren unterstützen die Menschen, die an der Mission teilnehmen, die Flotte bei der Aufgabenverteilung, wenn das formalisierte Wissen über die Aufgaben und Roboterkapazitäten für den Entscheidungsprozess nicht ausreicht.

So wird der Mensch, der in traditionellen Systemen die Rolle eines Teleoperators hatte, in diesem System, das Kooperation und Koordination in den Vordergrund stellt, zu einem Flottenmitglied, das die Steuerungsaufgabe nach Bedarf dem System überlässt oder sie selbst übernimmt. Dieses Konzept wird als adaptive (Entscheidungs-) Autonomie bezeichnet.

In dieser Arbeit wurde ein allgemeines Koordinationsframework und verteiltes Entscheidungssystem entwickelt, das auf jedem Roboter der Flotte vorhanden ist und so zum Einsatz kommt. Das Koordinationsframework wird basierend auf dem Space-Based Computing (SBC) Middleware Mozartspaces entwickelt, und mittels semantischen Modellierungs- und Schlussfolgerungseinsetzens erweitert. Das Wissen, das zwischen

Mensch und Flotte geteilt wird, und für die Entscheidungen bezüglich der Aufgabenverteilung genutzt wird, ist semantisch beschrieben. So vereint dieses Koordinationsframework die Vorteile der "loose coupling Architektur des SBC Middlewares, die Entscheidungsautonomie der Roboter, sowie Flexibilität in der Datenmodellierung. Für die Evaluierung des Frameworks wurde eine umfassende Evaluierungsstudie durchgeführt. Die Performance verschiedener Kooperationsmodelle wurde mittels konkreten Leistungsindikatoren quantifiziert. Infolge wird ermittelt, in welchen Szenarien des Flotteneinsatzes sich verschiedene Koordinationsmodelle am besten eignen. Die Ergebnisse zeigen, dass adaptive Autonomie und geteiltes Wissen die Performance der Aufgabenverteilung verbessern.

Abstract

Networked robotic fleet consists of multiple robots with heterogeneous capabilities that jointly perform a mission. Due to the ability to distribute heterogeneous robots in a working environment and exploit their cooperation and collaboration capacities, it has been shown that utilizing a robotic fleet for accomplishing a given mission is faster and more efficient than using a single robot. However, nowadays, robotic fleets are mostly limited to execute missions in structured and controlled environments where a centralized system manages and supervises mission execution.

With the goal to shift a focus from centralized to distributed control systems, this thesis explores the potential to use robotic fleets in heterogeneous and unstructured environments in scenarios requiring collaboration in mixed human-robot teams. It addresses new challenges related to communication, task allocation, coordination, collaboration, cooperation, and adaptive autonomy, and compares the well-established centralized coordination approach with several new distributed approaches. Reliable communication between distributed heterogeneous team members is a necessary condition for efficient coordination. An efficient coordination model is a critical enabler for task allocation which is a fundamental problem in multi-robot systems where the core requirement is to find an optimal set of heterogeneous robots that have to cooperate to execute a complex mission. To work together on complex tasks, humans and robots have to conform to efficient collaboration and cooperation models. The transition to cooperation and collaboration mode implies the change of role a human has as a remote operator in traditional systems to the peer who jointly performs tasks with the fleet members.

The thesis focuses on a development of a general coordination framework with a distributed decision making system deployed on each robot in a fleet. The coordination framework is based on the Space-Based Computing (SBC) paradigm extended with the Semantic Web Technologies resulting in semantic shared data space which allows annotating shared data with machine-understandable metadata. While the SBC paradigm provides for strongly decoupled architecture and preserves autonomy of the interacting team members, processing the metadata introduces flexibility in modelling collaboration processes in the fleet. For the evaluation of the developed coordination framework, established is a comprehensive benchmark setting utilized to quantify concrete performance indicators of different coordination approaches. Moreover, it is analyzed which scenarios are best suited for the developed coordination approaches. The results indicate that adaptive autonomy and shared knowledge improve performance in task allocation in complex missions.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

CHAPTER 1

Introduction

A networked robotic fleet consists of multiple robots with heterogeneous capabilities that jointly perform a mission [78]. Utilizing a heterogeneous robotic fleet for accomplishing a given mission has been shown faster and more effective than using a single robot because of the ability to distribute robots in a working environment and utilize their cooperation and collaboration capabilities [126], [123], [89]. However, nowadays, robotic fleets are mostly limited to operate in controlled and structured environments where they are managed from one central place where a human operator supervises mission execution [19].

This thesis explores the potential to use robotic fleets in heterogeneous and unstructured environments in scenarios requiring collaboration in mixed human-robot teams. It addresses new challenges related to communication, task allocation, coordination, collaboration, cooperation, and adaptive autonomy, and compares the well-established centralized coordination approach with several new distributed approaches. Reliable communication between distributed heterogeneous team members is a necessary condition for efficient coordination [128]. An efficient coordination model is a critical enabler for task allocation which is a fundamental problem in multi-robot systems where the core requirement is to find an optimal set of heterogeneous robots that have to cooperate to execute a complex mission [109]. To work together on complex tasks, humans and robots have to conform to efficient collaboration and cooperation models [41].

This thesis deals in particular with coordination, collaboration, and adaptive autonomy within heterogeneous robotic fleets. The transition to cooperation and collaboration mode implies the change of role a human has as a remote operator in traditional systems to the peer who jointly performs tasks with the fleet members [47], [55]. The level of human involvement in decision-making processes depends on the level of autonomy, i.e., adaptive autonomy, a robot has in a certain moment [124]. Human capabilities are captured in the concept of adaptive autonomy which defines up to what extent a human can interfere with the robots and thus increase the robustness of the system and minimize the impacts of unstructured environments.

Space-based computing (SBC)[1] paradigm has been established as a technology to provide agile coordination capabilities that can meet requirements in distributed multi-actor systems. The introduced paradigm provides communication and coordination mechanisms upon the blackboard model, i.e., communication and coordination between distributed entities is implemented by reading and writing data to a shared space. To support data exchange between heterogeneous actors, the SBC paradigm is extended with the Semantic Web Technologies [161] resulting in semantic shared data spaces [10] which allow annotating the shared data with machine-understandable metadata. While the blackboard model provides for a strongly decoupled architecture and preserves autonomy of the interacting team members, processing the metadata promotes the extraction of additional information from the existing data in the space. Consequently, semantic spaces combine the concepts of knowledge description languages and inference mechanisms with the space-based interaction mechanisms to address many of the aforementioned problems.

Having recognized the potential of integrating semantics with the SBC paradigm, several semantic space systems have been developed [42], [134], [107], [67], [10], [103]. While these implementations served as a proof-of-concept for the semantic spaces approach, they also revealed that there still exist unaddressed conceptual problems to solve. In particular, the following has not yet been investigated: (1) how the semantically enriched coordination framework benefits from the semantic support to overcome heterogeneous system resources, (2) to what extent does the semantic extension of the SBC paradigm facilitate transition from a centralized to a distributed coordination approach, and (3) what is the impact of shared knowledge on adaptive autonomy.

Providing solutions to these problems defines the scope of this thesis. Firstly, the proposed design abstracts the underlying heterogeneity issues using an ontology-based Model-Driven Architecture approach which formalizes the addressed heterogeneous resources by means of ontologies. Secondly, it utilizes an implementation of semantic spaces [69] to implement three different coordination approaches that reflect the transition from centralized to distributed coordination while addressing the aforementioned challenges. And finally, implemented is a framework which demonstrates the use of shared knowledge as a basis for adaptive autonomy in mixed teams.

The remainder of this section is structured as follows. Section 1.1 introduces precision farming as a reference use case. Section 1.2 describes in detail each of the problems that arise with the transition from centralized to distributed coordination approach in order to support different levels of robots' autonomy, i.e., adaptive autonomy. These are presented in a form of research questions addressed in this thesis. The approach and contributions are outlined in Section 1.3 and the method for evaluating the proposed approach is described in Section 1.4. Finally, an overview of the thesis structure is given in Section 1.5.

[1] www.spacebasedcomputing.org

1.1 Precision Farming as a Use Case

An inspiring use case which requires collaboration of humans and robots in unstructured environments is a precise management of agricultural land. The aim of the precise management is to diminish the use of chemical inputs and improve crop quality and humans' safety, as well as to reduce production costs, by using a fleet of heterogeneous robots equipped with advanced sensors and actuators. In particular, the precision farming scenario introduced in the RHEA [2] project provides motivation and requirements for this thesis.

The RHEA scenario starts with a field inspection performed by two aerial mobile units equipped with high-resolution cameras taking the field photos to elicit the growth stage of crop and the diffusion of weed. After the inspection, the centralized fleet management system (base station) assists a human acting as a system operator in choosing a suitable robot configuration for field treatment. The selection of a configuration takes into account multiple parameters such as the type of tasks to be performed, e.g., spraying, flaming, or tilling tasks, the number of available robots (tractors) and skills they provide, i.e., implements they have, as well as field information. Additionally to the task type, each task requires some amount of resources to be executed, e.g., spraying resources, flaming, or tilling resources. Therefore, selected configuration denotes a set of robots where each robot provides some skills and resources to execute more of these tasks.

In the precision farming the complexity of agricultural lands stems from applying more treatment strategies in order to increase crop quality, e.g., instead of utilizing chemical inputs on a crop filed, the crop field can be tilled and flamed. In contrast to traditional farming where each task requires only one skill for complete execution, i.e., a simple task, in the precision farming tasks require more than one skill for a complete execution, e.g., a task that requires a combination of tilling and flaming skills. In a specific scenario a task can require even more than two skills for a complete execution. However, agricultural robots are still rather simple and usually provide only one skill. Thus, in most cases two or more robots would have to collaborate to execute such a task where each robot provides at least one matching skill, i.e., each robot provides either a tilling or a flaming skill. If there would exist a robot which provides multiple skills, it could completely execute such a task. In the thesis these tasks are named complex tasks. Independently of task complexity, all tasks are assembled in a mission which is successfully finished only when all tasks are executed.

In addition to the heterogeneous robots, there are two main roles assigned to human operators in the referenced agriculture scenario, i.e., a fleet owner who leases out his/her heterogeneous robotic fleet (a configuration) consisting of multiple tractors with different implements able to perform different tasks, and a farmer who owns a field which has to be cultivated. The fleet owner can also be a fleet operator responsible for selecting a suitable strategy for a field treatment. When during a mission execution a robot tries to execute a task that requires a human intervention, it has to consult the operator who decides whether the robot has necessary skills and resources to execute the task. The

[2] http://www.rhea-project.eu

robot can also consult the fleet operator for assistance if there are ambiguities during the execution of the task. In that case the operator decides whether the robot has compliant skills and needed amount of resources to execute the task. Upon receiving confirmation from the human, the robot is allowed to proceed with regular task execution.

However, the intention of this thesis is to develop a general coordination framework which will improve the existing RHEA scenario. The components to be improved are addressed with the research questions listed in the next section.

1.2 Problem Description and Research Questions

The scope of this thesis is defined by the research questions derived from the subsequently described problem domains.

1.2.1 Heterogeneity and Complexity in Unstructured Environments

Advances in the perception and locomotion technology enable a robotic fleet to extend its operating field to heterogeneous and unstructured environments and scenarios requiring collaboration in mixed human-robot teams. Unstructured operating environments, like agricultural fields, are characterised by dynamically emerging tasks that require collaboration of multiple heterogeneous robots, and potentially a human involvement, to solve a task. Due to the lack of semantically modeled resources, many coordination frameworks are limited to a specified set of skills applied in a specific domain [110], [1], [65], [127], [99], [53]. In particular, they are constrained to utilize only a predefined set of available skills, e.g., spraying robots, and thus require extensive adaptation in code to support new skills and dynamically emerging tasks. Consequently, their operation is highly limited to structured domains.

Due to the operations in unstructured environments, a fleet owner who rents robots would benefit from a robotic framework which is not coupled to a specific operating environment, rather which is able to operate in various agricultural fields without adapting the underlying implementation. To address different challenges related to each operating environment, e.g., different number of tasks, different tasks' complexity, different coordination policies, the robotic framework should support operation of different robotic fleets, i.e., configurations. Moreover, the fleet owner should be able to operate different robotic fleets within the robotic framework without changing the underlying framework implementation. Consequently, it is necessary that the fleet owner has various robots which can be combined in a fleet and thus enable operation in different environments. This approach contrasts to the traditional frameworks, referenced in the previous paragraph, where a single robotic fleet is designated to operate only in one environment.

The problem extends to having a single coordination framework that is neither coupled to a specific configuration of robotic fleet nor the working environment. Due to the agility of the space-based middleware XVSM[3] (eXtensible Virtual Shared Memory) [21], [22], in

[3] www.xvsm.org

particular its Java-based implementation MozartSpaces[4], supported by means of different components such as a shared space that enables a communication between distributed and heterogeneous entities, different coordination capabilities, as well as notification mechanisms, and its efficiency in robots coordination demonstrated in [74], it is decided that the general coordination framework will extend those components and combine them with the Semantic Web Technologies [161] to account for heterogeneous resources. On the one hand, XVSM introduces the concept of containers which enable communication between distributed entities by reading and writing data to a container. XVSM containers implement various coordination patterns and notification mechanisms to support coordination capabilities between distributed entities. On the other hand, semantic extension of XVSM facilitates annotation of shared data with machine-understandable metadata and thus enables distributed robots to overcome heterogeneity issues and have a common understanding of exchanged data. Moreover, the general framework should be independent of operating environment and also support mixed human-robot teams as well as different coordination policies. Therefore, the following research questions summarize the problem statement.

RQ.1 **Extending Semantic XVSM to prevail heterogeneity issues** How to semantically enrich the XVSM coordination framework to support heterogeneous resources and changing coordination policies in robotic fleets? How to promote the semantics for dynamic resource discovery and to utilize it for sharing knowledge within a robotic fleet?

1.2.2 Distributed Task Allocation

The centralized architecture approach addressed in RHEA scenario where the system operator is responsible for selecting a suitable strategy for a field treatment, is practised in traditional agricultural missions where a base station is a central place which manages, coordinates, makes decision, collects data, instructs, and monitors all robots in a fleet. Consequently, this limits robots' autonomy to a very few basic functionalities, e.g., small adjustments related to path correction in a field. Thus, to perform any operation that is not in the original mission plan, e.g., if a robot wants to autonomously select a task to execute, it has to contact the base station where a human operator, e.g., a fleet owner, defines new mission parameters, e.g., reschedules a task. Another limitation pertains to the inability of multiple robots to collaborate on the execution of complex agricultural tasks. Additionally, since all data are stored at one central place, the robots are unable to retrieve data from neighbours at the time data are generated. This may be a big disadvantage because the mission is running in real-time and having data at the right moment may improve the overall efficiency of a performed mission.

Thus, the challenge is to design a distributed decision making system that ensures reliable data distribution among robots and provides efficient distributed coordination and collaboration of robots to secure successful mission execution. A critical enabler for successful mission execution is an efficient task allocation that influences coordination

[4]www.mozartspaces.org

between heterogeneous team members. In particular, the problem is lack of mechanisms that enrich distributed robots with additional information that increases their autonomy and enables them to carry out autonomous decisions crucial for building collaboration patterns for the execution of complex tasks. This, however, implies mechanisms that support robot-robot coordination activities. Moreover, the distributed decision making system has to address the problem of spatial interference when multiple robots collaborate on the same task. Furthermore, it is often unclear, with respect to which key performance indicators, does a distributed coordination approach outperform the centralized one. Accordingly, the following research questions summarize the problem statement.

RQ.2 **How to utilize Semantic XVSM for distributed task allocation** How the Semantic XVSM supports transition from a centralized to a distributed coordination approach by increasing robots' autonomy in a fleet and how it models mechanisms to distribute critical data that drive robots' autonomous decisions?

1.2.3 Shared Knowledge for Adaptive Autonomy

The undergoing transition is perceived as a decentralization process where the robot control, previously limited only to a centralized fleet management system, is spread to all participants with adaptive levels of autonomy. This implies the change of the role a human operator, i.e., a fleet owner, had before as a remote operator at the base station. Consequently, the fleet owner will jointly perform tasks with robots in a fleet where its involvement depends on the level of autonomy a robot exhibits in a certain moment.

This entails that a fleet owner acts as a peer in a robotic fleet who shares knowledge with peers to increase mission efficiency. A critical enabler for successful mission execution is an efficient task allocation that influences coordination between a human and robots. To work together, a human and robots have to collaborate and coordinate effectively. Therefore, the problem is to model the shared knowledge and human-robot interaction mechanisms to be able to evaluate different coordination aspects as well as adaptive autonomy. The adaptive autonomy should take into account the requirements of human-robot interactions which are important in many types of missions. When a complex task emerges during the robot's autonomous operation in an agricultural field, the human operator can either assign the task to robot or, if the task is automatically assigned to the robot, the robot may need to acquire additional help from the human who interacts directly with the robot during a mission. In that way robots extend their decision-making mechanisms to a human operator, i.e., the fleet owner.

Therefore, the research strives to develop new coordination protocols between robots and humans by utilizing advantages, such as scalability, high-performance, distributed object exchange, and coordination mechanisms, of a space-based architecture complemented with ontology and reasoning capabilities provided by semantic technologies.

RQ.3 **Modeling of shared knowledge for adaptive autonomy** How to model the shared knowledge and how to distribute it between heterogeneous robots and a human operator in order to support adaptive autonomy? How to utilize distributed

shared knowledge to implement human-robot interaction mechanisms and to evaluate different aspects of adaptive autonomy?

1.3 Approach and Contribution

This thesis designs a novel coordination approach applicable to different configurations of a robotic fleet that utilizes human interaction and operates in unstructured and heterogeneous environments. First two chapters (Chapter 2 and Chapter 3) introduce background technologies and review existing coordination and task allocation approaches used as a basis for the design of our coordination approach. Our approach is structured and presented in three main parts:

The first part (Chapter 4) attempts to give a comprehensive answer to the first research question, the question that addresses heterogeneity problems in multi-robot systems. Multi-robot systems heavily rely on dynamic and intensive collaboration to provide interoperability between heterogeneous autonomous robots. In this thesis, the notion of interoperability is narrowed down to the model of task allocation. Achieving an efficient task allocation, and thus interoperability, is a challenging task which requires to address the following issues: (1) heterogeneity of resources, i.e., robots and tasks to execute, (2) coordination which manages the possible interactions between involved parties, (3) dynamic resource discovery and utilization, (4) sharing knowledge within a robotic fleet, and (5) domain independent and flexible collaboration approach.

To overcome the semantic discrepancies stemming from the integration of heterogeneous components, ontologies are utilized, i.e., formal descriptions to specify and conceptualize knowledge. They are machine-readable, shareable, and enable reasoning to infer new information. Utilizing ontologies formalizes modeling of domain knowledge and simplifies the design of coordination patterns between robots which otherwise is a complex and cumbersome task [73]. Here, the contribution of the thesis is a Model-Driven Architecture (MDA) approach that separates among the specification of system functionality and their implementations.

The second part (Chapter 5 and Chapter 6) introduces design and implementation details of a general coordination framework named Shared Knowledge Interaction Modeling framework (SKIM) which is the main contribution of this thesis. These two chapters present three specific coordination approaches referred to as centralized SKIM (cSKIM), distributed SKIM (dSKIM), and hybrid SKIM (hSKIM). In particular, it is described how each of these coordination approaches addresses the following: (1) task allocation problem, (2) task transfer model from a central task repository to a robot and between robots, (3) knowledge-based resource matching algorithms, (4) robustness support, and (5) decision-making mechanisms. Moreover, hSKIM models human interactions in the system as well.

The objectives of SKIM are: (1) to model shared knowledge as a basis for adaptive autonomy in mixed teams and to investigate its impact on task allocation, and (2) to

evaluate the performance of finding a set of robots to execute a certain task. To address these challenges, SKIM utilizes the semantic approach and describes the collaboration activities by means of ontologies: SKIM Resource Ontology (SKIM-RO) and SKIM Coordination Ontology (SKIM-CO). With respect to the three coordination approaches, hSKIM is the most advanced because robots have their local decision-making systems which reason on SKIM Coordination Ontology to determine the level of autonomy. Finally, these three coordination approaches are evaluated with respect to the defined parameters, i.e., a task allocation rate, communication overhead, a utilization rate, a load balancing, and mission duration. Moreover, a setup is characterized, in which the hSKIM framework outperforms the other two.

The third part (Chapter 7) describes various test cases that evaluate the impact of shared knowledge between a human and the robots in terms of the several different performance measures. To work together, humans and robots have to collaborate and coordinate effectively and thus one of the main challenges in evaluating coordination performance in mixed teams is how to model which knowledge is shared among team members. Shared knowledge enables various autonomy levels, i.e., adaptive autonomy. The adaptive autonomy allows autonomous robots to acquire additional help from a human who directly interacts with them during a mission. The most demanding application of an autonomous robotic fleet with a human-robot interaction is in catastrophic scenarios such as fire, earthquake, floods, and humans search and rescue missions.

hSKIM supports the model of shared knowledge between robots and a human using the inference-based task allocation algorithm which performs reasoning on SKIM-CO. The result of reasoning is a set of tasks mapped to a set of matching robots. The inference-based task allocation algorithm in hSKIM influences two coordination models: (1) robot-robot, and (2) robot-human. Robot-robot coordination model is based on reasoning on the SKIM-CO ontology which enables a robot to determine own level of autonomy in decision-making, e.g., whether to involve a human in the task allocation process. Robot-human coordination model enables the human to interact with a fleet in two different ways. Firstly, observing the task allocation process, the human can apply shared knowledge to allocate a task which was not allocated in the inference-based task allocation process. Alternatively, a robot can ask a human for assistance while executing a complex task.

1.4 Framework Evolution Methodology

The two following sections introduce methods and the evaluation of the proposed coordination framework. The former describes how the prototype framework was evolving by sequentially implementing three coordination approaches, i.e., cSKIM, hSKIM, and dSKIM. The latter introduces evaluation setup used for comparing implemented coordination approach.

1.4.1 Methods

Reliable data distribution among robots and efficient coordination of robots directly influence the level of autonomy a robot has. This thesis addresses adaptive autonomy concept by utilizing the space-based paradigm and semantic technologies. In order to semantically describe resources and processes, an extended MozartSpaces is used what provides semantic support [69].

Firstly, the SKIM-RO was developed to model heterogeneous resources in a system. Developed ontology was imported in Semantic XVSM where it served as a basis for modeling robots' capabilities and task requirements. On the other hand, a simple coordination mechanism based on Semantic XVSM was designed as a proof-of-concept. This proof-of-concept had a centralized task repository and distributed robots which were able to interact only with the repository. This implementation resembled cSKIM coordination approach. Moreover, it integrated the area decomposition algorithm and the space-based paradigm with underlying semantics and thus provided a robust and scalable middleware for a task allocation in multi-robot systems. Two important features in cSKIM were: (1) the area decomposition which ensured that each robot operated in own cell and therefore decreased spatial interference between robots which left the robots more time to focus on a domain work, and (2) Semantic extension for MozartSpaces that provided a data model which automatically translated heterogeneous resources into the common objects understandable by all team members. Although the cSKIM framework integrated two different technologies, MozartSpaces and semantics, it retained complete set of functionalities from both, i.e., coordination mechanisms and transactions support from MozartSpaces and query and reasoning capabilities endowed by semantic technologies.

Since a critical enabler for task allocation is an efficient coordination between heterogeneous team members, human and robots have to collaborate and coordinate effectively. Consequently, SKIM framework utilized the semantic approach to describe the collaboration activities by means of ontologies: SKIM Resource Ontology (SKIM-RO) and SKIM Coordination Ontology (SKIM-CO). SKIM-RO described resources, including robot capabilities and task requirements, and SKIM-CO described coordination constraints for robot-robot and robot-human interactions. Hence, these ontologies were used as the model of shared knowledge and the decisions were results of automated reasoning on them. The SKIM framework, as a simulation environment for performance evaluation of task allocation algorithms, was built on space-based middleware which was already demonstrated as an efficient platform for developing robotic functionalities [74]. Moreover, in SKIM approach, the ontology-based shared knowledge description was used not only to describe robot resources and task requirements, but also to represent capabilities of the human in the decision-making process. Thus, by reasoning on this ontology, each robot was able to decide if further interaction with the human is necessary, i.e., whether to involve him/her in task allocation.

The extension of the existing cSKIM coordination approach, which modeled a shared knowledge and thus enabled human interaction, is called hybrid SKIM (hSKIM). However, since hSKIM partially utilized the central component, it was not completely distributed

system. Therefore, developed is distributed SKIM (dSKIM) which utilized the central component only at the beginning to distribute task to robots. To conclude, there were three different coordination approaches referred to as cSKIM [26], dSKIM, and hSKIM.

1.4.2 Evaluation

The first part of the evaluation was performed to acquire initial performance and scalability metrics of Semantic XVSM. Conducted experiments were focused on finding an optimal fleet size with respect to the coordination complexity which implicity influences mission duration. To emphasize SKIM generality with respect to the application domains, all three coordination approaches were evaluated in three different scenario classes using five different configurations of a robotic fleet. The same coordination approaches were evaluated with respect to the various performance metrics, i.e., a task allocation rate, communication overhead, a utilization rate, a load balancing, and mission duration. These metrics have been designed specifically for the benchmarking of various SKIM based coordination approaches and can therefore also be applied to measure performance of other coordination frameworks which have a set of robots required to execute a set of tasks.

After that, the focus is shifted to the evaluation of hSKIM coordination approach with respect to the different levels of adaptive autonomy. Then, it is evaluated how the ontology-based Model-Driven Architecture approach facilitates the addressing of changing requirements in a robotic fleet, i.e., in a configuration, as well as in an operating environment, i.e., in a scenario. The evaluation concludes with an overview and recommendations on selecting an appropriate coordination approach based on an available scenario and a configuration considering evaluated parameters.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2. Background Technologies provides an overview of the technologies utilized in the scope of this theses and discusses their advantages and disadvantages.

Chapter 3. Related Work reviews existing frameworks for task allocation and coordination in robotic fleets as well as adaptive autonomy in mixed human-robot teams.

Chapter 4. Ontology-based Approach for Task Allocation introduces ontologies as a semantic concept to overcome the semantic discrepancies inherited by integrating heterogeneous components. Introduced ontologies specify and conceptualize knowledge using a formal description that is machine-readable, shareable, and

enables reasoning to infer new information.

Chapter 5. SKIM - Shared Knowledge Interaction Modeling Framework proposes the design of main building blocks of the Shared Knowledge Interaction Modelling (SKIM) framework. SKIM framework is designed to investigate model of shared knowledge as a basis for adaptive autonomy in mixed teams and to evaluate its impact on task allocation.

Chapter 6. Implementation of SKIM Framework describes implementation details of the components addressed in the previous section. Described components are enablers to build a generic architecture that the developed SKIM framework complies with. Moreover, implemented are three different coordination approaches based on the semantic extension of the Space-Based Computing architectural style.

Chapter 7. Evaluation and Discussion presents the results of the evaluation of the general SKIM coordination framework which is based on the semantic extension of the Space-Based Computing architectural style, i.e., Semantic XVSM, and discusses these results with regard to the specified research issues. To emphasize SKIM generality in the sense of application domains, introduced are three scenario classes and different configurations for the framework evaluation.

Chapter 8. Conclusion summarizes the developed approaches and results of this thesis, presents findings and conclusions, and describes future research.

CHAPTER 2

Background Technologies

This chapter summarizes the technologies utilized in the scope of the thesis. The first section introduces the Semantic Web Technologies[1] that enable the development of platform-independent domain models and knowledge. The benefit of using semantics for task allocation is twofold: (1) an ontology that models domain knowledge provides uniform description of heterogeneous and distributed resources, i.e., tasks and robots, and (2) semantically annotated resources enable accurate resource matching. The second section presents the Space-Based Computing (SBC)[2] architectural style. Due to the agility and adaptability with respect to the different coordination patterns in multi-robot systems operating in distributed and heterogeneous environment, XVSM[3] (eXtensible Virtual Shared Memory) [21], [22] is selected as the reference architecture for the realization of the SBC paradigm. After that introduced is the notion of Semantic Spaces that combines Semantic Web Technologies with the SBC paradigm. The focus is on the existing implementations of Semantic Spaces as well as on the Semantic XVSM [69] which is used in SKIM framework. Finally, the section is concluded with the comparison of Semantic Spaces implementation.

2.1 Semantic Web Technologies

As envisaged by Tim Berners-Lee [161], the main idea of the Semantic Web is to enable data publishing in such a way that allows autonomous computer systems to retrieve and process the data without requiring human interaction; in particular, to produce and publish machine-understandable information. An example of such a task is planning a journey which requires searching, collecting, and processing data from different and heterogeneous online resources. Apart from the automation of daily tasks, it is expected

[1] www.w3.org/standards/semanticweb/
[2] www.spacebasedcomputing.org
[3] www.xvsm.org

that the processing of such vast amount of information on the Web will result in the emerge of completely new information which was not recognized so far.

However, to realize the idea of Semantic Web, several issues have to be tackled: (1) define mechanisms for a knowledge representation, (2) define and develop algorithms for a knowledge evaluation, and (3) define mechanisms for automated annotation of already existing information on the Web. Moreover, standards and tools for these concepts have to be defined and established.

The following sections describe the most important and most used Semantic Web technologies and standards.

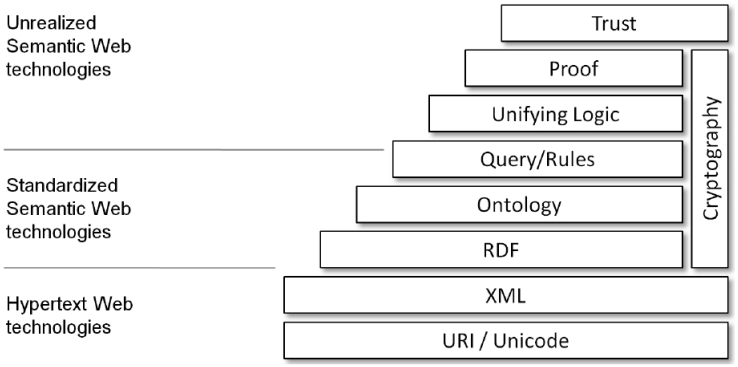

Figure 2.1: The Semantic Web Stack [161]

2.1.1 The Semantic Web Stack

The Semantic Web Stack [161] describes the layered architecture of the Semantic Web. Figure 2.1 reflects the representation of this architecture where some layers are still under research and thus subject to changes. According to [161], the single layers of the Semantic Web Stack are often grouped into the following three categories: (1) Hypertext Web technologies, (2) standardized Semantic Web technologies, and (3) unrealized Semantic Web technologies.

Hypertext Web Technologies

This presents the bottom layer of the Semantic Web Stack which encompasses already established Web technologies. IRIs [4] are used for the unique identification of resources and Unicode [5] is utilized for referencing and describing resources. The use of IRIs in the classic Web is limited to referencing documents and hypermedia. In contrast, the use of IRIs in the Semantic Web is used to describe any kind of artefact.

[4] IRI (Internationalized Resource Identifier): a generalization of URI (Uniform Resource Identifier)
[5] http://www.unicode.org

The most widely used data formats for describing structured data are based on XML. However, the Semantic Web Stack does not prescribe any particular data format, rather it provides a basis for other formats supporting the overlying RDF layer.

Standardized Semantic Web Technologies

The middle layer of the Semantic Web Stack encompasses technologies which have, during the time, reached a certain level of maturity, have been standardized, and are widely adopted.

The most established technology in this part of the Semantic Web Stack is Resource Description Framework (RDF) [169] which utilizes graphs for information description. RDF is designed with the purpose to describe data in the Web and thus uses IRIs to define nodes and edges in an RDF graph (see Section 2.1.2).

The concept of RDF graphs serves as a basis for the overlying layer. The next layer, i.e., ontology, processes the provided RDF graph to infer additional data from it by means of ontologies. Ontologies conceptualize a particular knowledge domain by extracting objects and their relations to infer new information. A language for defining ontologies is RDF Schema [168] which is suitable for defining rather simple ontologies with basic hierarchical structures, e.g., *subClassOf*. If more complex relationships are required, more powerful ontology language, like Web Ontology Language (OWL) [166], is needed. Reasoning engine is a tool necessary to process the information modeled in an ontology and, consequently, represent inferred information in an RDF graph.

Once when the provided and inferred data are present in the system, i.e., RDF graph, there should be a mechanism which will allow to query the data to extract particular information. Thus, the next layer on the Semantic Web Stack introduces a special query language named SPARQL [170]. SPARQL query is built of query statements which define graph patterns and are executed against very large RDF graphs. If the underlying RDF graph has reasoning capabilities, the result may not only contain explicitly defined information, but also the information inferred by using the ontology.

Unrealized Semantic Web Technologies

The upper layers of the Semantic Web Stack are still subject to research and have not yet been applied in a wider industrial setting. Technologies like Rule Interchange Format (RIF) [167] and Semantic Web Rule Engine (SWRL) [164] pertain to the extension of the Semantic Web Stack with support for rule based approaches with the sole purpose to infer new knowledge. These technologies support defining relations between resources which cannot be modeled with OWL.

In the Unifying Logic layer, the above introduced rules are evaluated to conclude whether certain statements are true or not. This layer utilizes information collected in the Proof layer to evaluate the statements and the evaluation process itself. The top layer in the Semantic Web Stack, i.e., Trust layer, checks whether the statements utilized in the evaluation process can be trusted or not. The Trust layer employs diverse security and trust mechanisms with the purpose to guarantee the authenticity of information.

Next section describes RDF as the most established technology in the middle layer of the Semantic Web Stack.

2.1.2 RDF

The Resource Description Framework (RDF) [169] defines a series of W3C [6] standards for the representation of both, data and meta-data. RDF describes all kinds of data using directed labeled graphs, i.e., vertices and labels. Described data are defined with a triple *<subject, predicate, object>* where the subjects and objects are vertices, and the predicates are directed edges of the directed graph. Usually, a triple is called *RDF statement* and the predicate of a triple is *RDF property*. Each field in a triple is represented by an IRI which points to a real Web resource that usually describes the resource. The object field can be represented by a literal value.

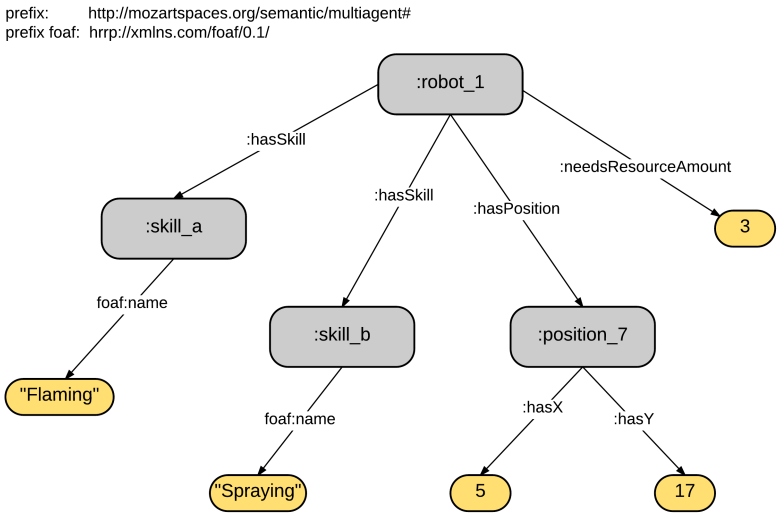

Figure 2.2: RDF graph example

Figure 2.2 depicts an example of RDF graph consisting of 8 triples describing a robot (: *robot_1*) which is a main actor in the use case presented within the thesis. Gray entities in Figure 2.2 are resources and yellow ones are literal values assigned to these resources. The robot is described with three RDF properties (*:hasSkill, :hasPosition*, and *:needsResourceAmount*) creating four triples. The RDF property *:hasSkill* is utilized twice meaning that the robot (: $robot_1$) has two skills, each with a different name. To increase readability, IRIs describing fields in a triple are extended by the defined prefix value. This is similar to XML Namespaces [163] in a XML document. Resources with a default prefix, i.e., defined by a standalone colon, point to an ontology that defines general concepts in the use case domain and their relations. Those IRIs with the prefix *:foaf* refer to the FOAF Vocabulary Specification for describing people and their relations

[6] http://www.w3.org/

to other. Although the FOAF specification does not perfectly fit in the agricultural use case, it is the best practice to reuse existing RDF properties instead of creating new ones when building an ontology.

There are several common serialization formats defined for the exchange of RDF data: XML/RDF [159], Turtle [160], N3 [162], and N-Triples [165].

2.1.3 RDF Schema

RDF Schema [168] is a W3C Recommendation that defines how to specify RDF Vocabulary for describing a specific knowledge domain. Similar to FOAF Specification used in Figure 2.2, RDF Schema describes types and relationships between resources, e.g., classes and properties. The main advantage of RDF Schema is the ability to infer additional information from a provided set of RDF triples. For example, if a class *Tractor* is defined as a subclass of a class *Robot*, i.e., *(:Tractor rdfs:subClassOf :Robot)*, and the (: *robot_*1) is of type *Tractor*, i.e., *(:robot_1 rdf:type :Tractor)*, then it can be inferred that this resource is also of type *Robot*, i.e., *(:robot_1 rdf:type :Robot)*. Hence, RDF Schema is not only used to describe resources and their relations, but also to derive new information based on the defined concepts. Therefore, RDF Schema is a simplified and limited form of an ontology language.

The following are most important and widely used concepts defined in RDF Schema:

- Classes: every resource can be defined as a class where the class is either a predefined RDF Schema class, e.g., *rdfs:Resource, rdfs:Literal*, or it is self-defined with RDF Schema concepts [100].
- Properties: RDF Schema supports definition of hierarchical properties, e.g., *rdfs:subPropertyOf*, as well as the domain and range of a property, i.e., ensuring that the property is only applicable to specific resource types.

However, if more complex relationships are required in a working knowledge domain, a more powerful ontology language, like OWL, is desirable.

2.1.4 OWL

Web Ontology Language (OWL) [166] is a knowledge representation language for the definition of ontologies. Same as RDF Schema, ontologies are represented with RDF graphs and thus are suitable to infer additional information from the explicitly provided information. The inference in OWL is based on Description Logic [2] and applies the *Open World Assumption*, i.e., if a statement cannot be proven to be true, it does not mean that the statement is false.

Besides support for the features of RDF Schema, OWL additionally provides the following concepts which can be utilized to facilitate ontology design:

- Property characteristics: properties can describe functional, transitive, symmetric, or inverse relations between resources, e.g., *owl:FunctionalProperty, owl:SymmetricProperty, owl:TransitiveProperty, owl:InverseOf.*

- Property restrictions: limit a property to have a particular value, e.g., *owl:hasValue, owl:someValuesFrom, owl:allValuesFrom*, or to define cardinality of a property, e.g., *owl:cardinality*.
- Ontology alignment: supports mapping between resources originating from different ontologies, e.g., *owl:equivalentClass, owl:equivalentProperty*, and defining that two individuals, i.e., instances of a class, are either same, e.g., *owl:sameAs*, or distinct, e.g., *owl:differentFrom, owl:AllDifferent*.
- Class expressions: complex classes in an ontology can be defined using the set operators, e.g., *owl:intersectionOf, owl:unionOf, owl:complementOf*.

Above listed OWL language constructs provide a very expressive mechanism for modeling domain knowledge. Due to the large expressivity, the modeling of domain knowledge can become extremely complex. Therefore, OWL defines three sub-languages where each language supports reasoning on different OWL language constructs.

- OWL Lite: supports a subset of the OWL language constructs to define classification hierarchies and simple constraints.
- OWL DL: supports all OWL language constructs, but introduces several limitations on the usage of these to ensure that OWL DL ontologies are always complete and decidable, i.e., all inferences are guaranteed to be computed and all computations will finish in finite time [100].
- OWL Full: similar to OWL DL, it supports all OWL language constructs, but does not introduce any limitations.

OWL could be used to define an ontology describing individuals and their relations on the RDF graph represented in Figure 2.2. The ontology can define a class which can be described in a following way: *Something that has a position, one or more skills, and a resource amount, is a Tractor*. A reasoning engine could then infer that the individual (: *robot_1*) is of type *Tractor*.

2.1.5 SPARQL

SPARQL [170] is a query language for data represented as RDF graphs. A SPARQL query is designated as a graph pattern consisting of a set of triple patterns where each triple pattern describes an RDF triple. In each RDF triple one or multiple fields can be substituted with variables. All possible combinations of those variables, with a valid binding, represent the result of a SPARQL query.

Listing 2.1: SPARQL query

```
PREFIX: <http://mozartspaces.org/semantic/multiagent#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?r ?s ?n
WHERE {
    ?r rdf:type :Tractor .
    ?r :hasSkill ?s .
    ?s foaf:name ?n .
}
```

The Listing 2.1 illustrates an example query with a graph pattern consisting of three triple patterns. The query selects all skills that resources of a type *Tractor* have. Upon executing the query on the RDF graph from Figure 2.2, the result set will contain variable bindings shown in Table 2.1.

Table 2.1: SPARQL query results

?r	**?s**	**?n**
:*robot_1*	:*skill_a*	"Flaming"
:*robot_1*	:*skill_b*	"Spraying"

SELECT query form was used in the query above and therefore the results mirrored in a table structure. However, SPARQL also supports other types of result presentation which are defined by the CONSTRUCT and ASK query forms [100]. The CONSTRUCT query form returns a graph data structure reflecting variable bindings, while the ASK form can be used to find out whether any results are available in a graph for a specific query. Moreover, the most important part of a query is WHERE clause where triple patterns are specified. Triple patterns encompass variables which are later, during the query evaluation, bounded to the calculated values. Although not used in the above query, the FILTER statement is commonly used in queries to filter out statements based on the bounded values. More detailed description of various SPARQL constructs can be found in [170].

Due to the graph-based data model, SPARQL provides at the same time a powerful, yet, easy to learn and use query language.

2.1.6 Triplestore

What a relational database is in the traditional software systems, that is a triplestore in Semantic Web. The sole purpose of a triplestore is to manage and persist data structured in RDF graphs. Persisted data can then be fetched utilizing the capabilities of SPARQL query language. Moreover, some triplestores integrate a reasoner enabling an executed SPARQL query to return inferred data as well.

Most popular Java implementations of a triplestore with an open source licences are Apache Jena [155] and Sesame [156]. Both implementations support SPARQL and can manage in-memory persistence. Moreover, Jena includes an open source reasoner for OWL 1, but for more complex ontologies it is advised to use an external reasoner, e.g., Pellet [158].

2.2 Space-based Computing

Since the XVSM is already demonstrated as an efficient platform for developing robotic functionalities [74], it is chosen as the reference implementation of the SBC paradigm. XVSM supports communication and coordination between heterogeneous and distributed entities by means of reading and writing data to a shared space which facilitates coordination in a mixed human-robot teams. The shared space enables communication between entities decoupled both in time and space.

Moreover, there exist frameworks that seek to combine Semantic Web Technologies with the SBC paradigm to provide a middleware for coordinating heterogeneous entities in unstructured environments. In particular, the focus is on the framework that integrates Semantic Web Technologies with XVSM in the framework named Semantic XVSM [69].

2.2.1 eXtensible Virtual Shared Memory (XVSM)

Similar to the Linda [52] coordination model which introduces the logically shared memory with a set of handful operations as a communication mechanism for parallel and distributed processes, SBC is a data-driven coordination model where heterogeneous application components running on different physical platforms communicate by means of reading, writing, and taking structured entries from/to a shared space. The space provides communication and coordination mechanisms based on the blackboard model. Figure 2.3 illustrates a space shared between multiple clients, i.e., tractors (robots), which communicate by means of writing and reading entries from the shared space.

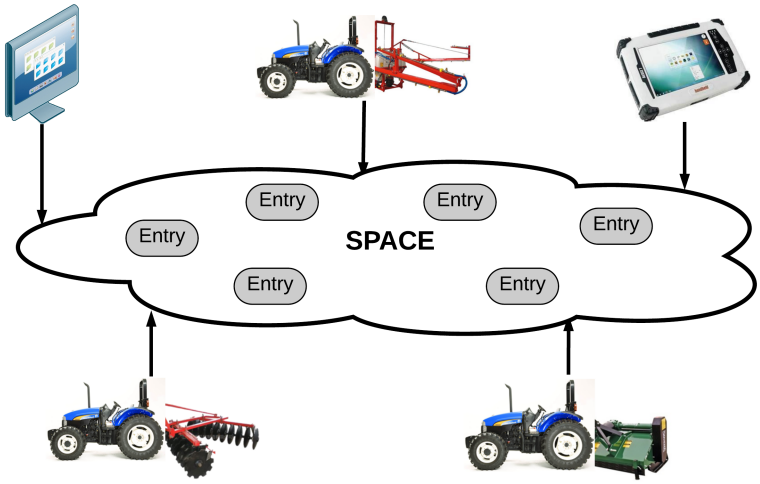

Figure 2.3: Space-Based Computing Paradigm

As a representative of SBC, XVSM [21] is used in this thesis which has been developed at the Institute of Computer Languages of the Vienna University of Technology. The main objective of the XVSM middleware is to provide a modular platform which facilitates

dynamically joining and leaving of distributed and heterogeneous processes. In terms of a coordination platform, a shared medium is required where the processes can write and read information, and be informed about events as well. To achieve a high degree of decoupling between distributed processes, it is advisable to utilize space-based architectural style because of its shared data space which acts as a mediator where the processes write and read data, and are informed about changes.

[22] relates a space to a database since databases support data storage and data access. Moreover, both, a database and a space, have in common that they manage structured data and support transactions. However, the purpose of a database is to administer large amount of static data and to support complex queries on them, whereas a space serves to coordinate distributed and heterogeneous processes.

Although XVSM supports multiple concepts, e.g., transactions, replications, logging, authentication, etc., the following sections introduces only those that are utilized in the implementation of SKIM framework.

Entries, Containers and Coordinators

An XVSM space is composed of multiple XVSM Cores. A Core is a fundamental component that hosts containers which store data, i.e., entries. All Cores are connected by a peer-to-peer based communication infrastructure [75] which enables communication and cooperation between distributed Cores. The objectives of the Core [4] are manifold, from the data coordination, over transactional isolation to asynchronous and blocking operations. One of the main responsibilities is to abstract the access to a Core in a way that for the client (tractor) it does not make any difference whether the user is accessing a local or remote Core.

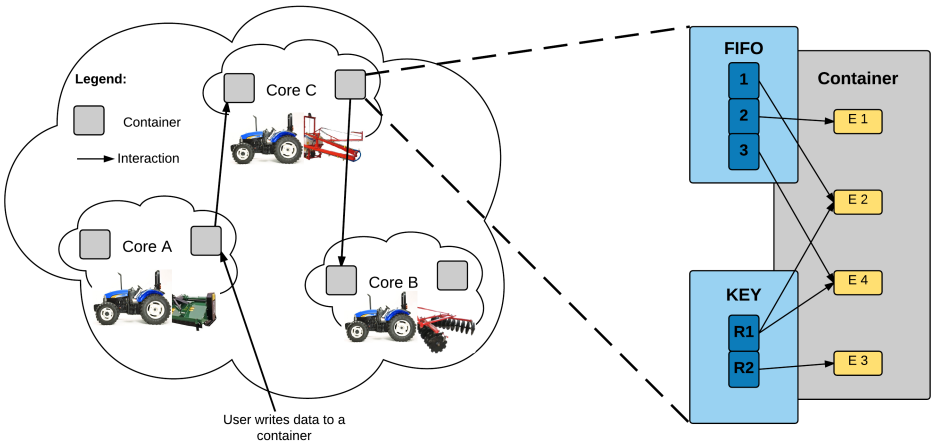

Figure 2.4: XVSM Space with Core instances and containers

Entries

The user defined objects stored in the space are called entries, Figure 2.3. The formal XVSM model handles only entries in a special format [22], however, the XVSM implementation named MozartSpaces [122] allows any form of Java-Serializable objects. Entries are not directly stored in the space, rather, in containers that structure a space and can be seen as subspaces. Moreover, the entries in a container are managed by one or more coordinators [25] where each coordinator can be of a different type, e.g., FIFO coordinator that ensures FIFO order of entries in a container, or a Linda coordinator that supports Linda template matching. Figure 2.4 shows an example of 2 coordinators and 4 entries bounded to a container hosted in the Core C. All entries are managed by the coordinator *ANY*, an explicit coordinator. In the example also the coordinators FIFO and KEY are used. As opposed to Figure 2.3, now the tractors are distributed in a global space where each tractor owns an embedded Core instance that hosts two containers. Moreover, distributed tractors communicate by writing data to distributed containers.

Containers

As in Linda, in XVSM application components coordinate themselves by means of writing and reading entries, i.e., user defined data, into/from a shared space. In XVSM data is stored in so called container that can be interpreted as a collection containing entries [94]. In the XVSM space multiple containers may reside at the same time. The Core residing in a space provides four operations to manage containers: (1) create, (2) destroy, (3) lookup, and (4) lock.

XVSM containers are Internet addressable using an URI of the form *xvsm://mycomputer.mydomain.cin:1234/ContainerName*. Depending on the application domain, or the underlying network infrastructure, the *xvsm* protocol may be translated to, e.g., TCP + Java, specifying that communication takes place via a tcp-connection using Java objects [95]. The default communication is based on an XML based protocol which is platform independent. Moreover, XVSM has a lookup mechanism which resolves published container names to URLs.

In its basic form, a container is similar to a tuple space, i.e., it is a collection of entries. The main difference to a tuple space is that a container [94]: (1) structures the space, (2) extends the original Linda API with the destroy method, (3) introduces coordinators which extend the coordination law, and (3) may be bounded to a maximum number of entries.

Coordinators

Coordinators that manage entries in a container are extendable components responsible for managing container's view on the stored entries. The aim of a coordinator is to represent a coordination model and to structure and organize the entries in the container for efficient access [95]. Having domain knowledge, i.e., a type and a structure of business data, a programmer can implement a domain-specific coordinator which operates more efficient for domain-specific tasks.

Coordinators are independent of each other and each coordinator can be one of the following types: (1) implicit coordinator, or (2) explicit coordinator. Representatives of the implicit coordinator usually maintain an order of the entries in the container, e.g., FIFO, LIFO, RANDOM. Explicit coordinators require additional meta information,

provided by the user, for managing the view on the container. A representative of this type of coordinators is a Map Coordinator which denotes *key:value* data structure, e.g., the meta information to be provided as the key.

For each coordinator in the space, there is a selector acting as a counterpart to the coordinator. Selectors contain parameters, like a counter for the minimum number of entries to be retrieved from a container, for queries in a case of *read*, *take*, and *destroy* access [95].

In case a container deploys several coordinators, operations may use multiple selectors as well. The number of specified selectors depends on the business requirements, i.e., a coordination model, and thus is not bounded to the number of deployed coordinators in the container. However, if more than one selector is utilized in querying the container, the outcome of the execution of the first selector is piped to the second selector, and so on. It means that selectors are chained. For example, if a FIFO Selector with count 5 is followed by a Key Selector with value k, the container will first ask the FIFO coordinator to select first five entries and afterwards it will ask the Key selector to look whether any of those five entries has a key with the value k.

Aspects and Notifications

Aspects and, in particular, notifications are extensively used in the implementation of SKIM coordination framework.

Aspects

Due to the focus on extensibility as a major part of the middleware, Aspect Oriented Programming (AOP)[38] was introduced in XVSM [75] in a sense of so called *Space Aspects* placed at different points of a container. The join points of AOP are called interception points (IPoints in XVSM). Aspects can be triggered by executing operations on the container, e.g., a space aspect could be triggered to execute an operation before a container is created (*pre create container*). IPoints can be located before or after the execution of an operation indicating two categories: *pre* and *post*. Pre-Aspects are triggered before the operation is executed, e.g., on a container to check credentials, and post-Aspects are triggered after the operation, e.g., to write to a log file.

Along every operation issued from the client application, the client is able to parameterize the deployed aspects to influence the operation of deployed aspects. This is performed by passing a so called Aspect Context with its parameters along the invoked operation. In case multiple aspects are installed on the same container, they are executed in the same order they were added. Adding and removing aspects can be performed dynamically during the runtime.

Figure 2.5 shows a container with three local pre- and post- Aspects along with their various return values. The XVSM Runtime layer accepts incoming requests and passes them immediately to the first pre-Aspect of the targeted container. The request passed to the aspect, i.e., operation, contains the parameters of the operation, like transaction, selectors, timeout, and the Aspect Context. The called aspect may contain any business logic, e.g., call third-party services.

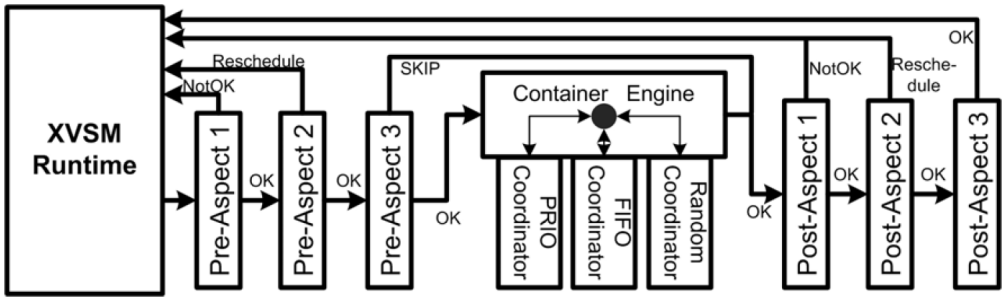

Figure 2.5: Execution sequence and return values of Aspects in a container with three deployed pre- and post-Aspects [75]

In the container's focus is the implementation of the container's business logic which handles the storage of the entries and manages coordinators. The operation is processed on the container only after the all deployed pre-Aspects are successfully passed. Only if all aspects return OK, the container interprets the selectors of the operation and executes the operation [74]. The received operation is successfully completed if it passed all pre-Aspects, the container engine, and all post-Aspects without any errors.

The return value of an aspect can manipulate the execution of the operation which triggered the aspect. This is achieved with the following return values an aspect can throw:

- OK: the execution of the operation may proceed regularly.
- NotOK: the execution of the operation is stopped and the subtransaction is rolled back, e.g., when a user passed inadequate credentials to access a system.
- SKIP: the operation is not performed (neither on the container, nor on the space), nor any of the following pre-Aspects. The post-Aspects are executed immediately afterwards. This return value is only supported in pre-Aspects.
- Reschedule: the execution of the operation is stopped and will be rescheduled for a later processing. Subtransaction is rolled back as well.

Notifications

Notifications are implemented by means of aspects and serve for notifying subscribed applications on occurred changes in a container. For example, a notification could fire when a certain amount of entries have been written in the container. If the defined condition is fulfilled and the example notification is triggered, the aspect writes the required information into the notification container from where an application can take the information for further processing. The notification container is a regular container described in Section 2.2.1 where the generated notifications are written and where from the subscribed application receives them via a callback method for processing.

XVSM supports multiple notification. Figure 2.6 illustrates the components and the workflow for processing one example notification in XVSM. There is a container C, a container D, and an application component, i.e., notified component, that wants to be notified whenever there is a change in a container C, e.g., a new entry is written to the container. To register the write notification on the container C, the application component invokes the method on XVSM Runtime which registers an aspect on the container C and creates a notification container. The registered post-Aspect intercepts the write operations on the container C and writes data into the notification container. Information forwarded to the notification container is customizable and thus can be tailored for a specific use case. The notification container is an ordinary container and thus is capable of deploying additional pre- and post-Aspects.

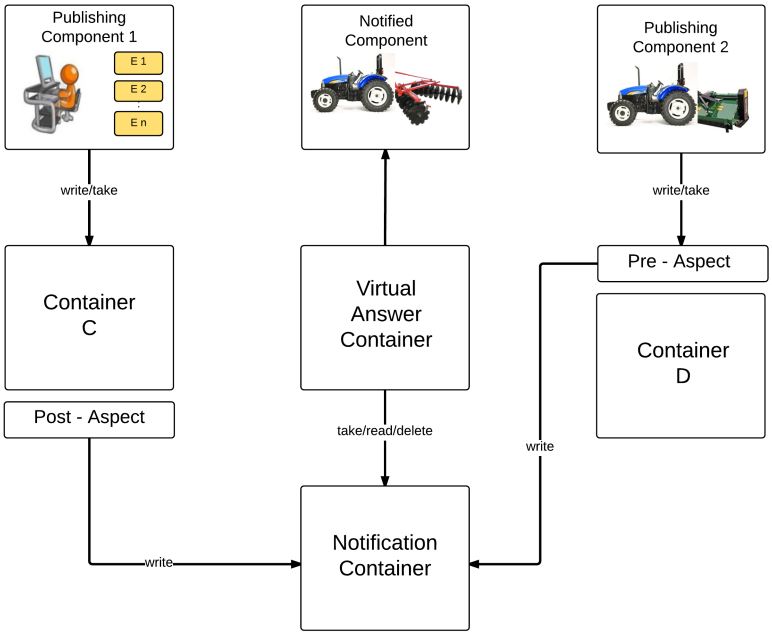

Figure 2.6: Example for XVSM Notifications

In addition to the notification aspect and the notification container, XVSM Runtime performs *take* operations on the notification container and specifies a virtual answer container where the result for that *take* operation has to be placed [94]. The application component bounded a callback method to the virtual answer container and thus the application component is notified whenever there is a change in the container C. Moreover, the notification container is not limited to receive notifications from one aspect only. In addition, if the application component is interested in notification from the container D, it can register pre-Aspect which can trigger a notification before the publishing component writes an entry in container D. Thus, XVSM supports several notification aspects writing into a single notification container.

Figure 2.6 does not specify where the containers are hosted. It is possible to host

containers either on the same node or on different ones. The latter approach supports the creation of durable [39] subscriptions by placing the notification container on a node which is always reachable [75]. The notification container does not depend on the subscribed clients, rather, it aggregates notification events while the interested clients are unreachable. If the subscribed clients are again online, aggregated notifications are pushed via the specified callback method to the subscribed clients.

Customizable notifications allow software developer to create domain and application specific notification mechanisms which can increase the efficiency of a business process.

2.2.2 Semantic Spaces

This section focuses on frameworks striving to combine Semantic Web Technologies with a tuple space framework to provide a middleware for coordinating heterogeneous entities in unstructured environments. Common to the presented semantic spaces is that all of them adhere to Linda's space-based interaction paradigm where clients do not interact directly with each other. Rather a shared space takes a role of a communication medium. Benefits this paradigm introduces are information dissemination and decoupling, both in time and space, between clients.

sTuples

sTuples [67] was one of the first initiatives extending the tuple space with inference mechanisms to solve interoperability issues between heterogeneous and dynamic agents in pervasive environments. Semantic Web Technology addresses the interoperability problem in heterogeneous environments, while tuple space provides data persistence, as well as temporal and spatial decoupling. The sTuples builds upon the JavaSpaces [50] standard and adapts the Linda coordination model.

Semantic tuples in sTuples consist of RDF graphs stored in a reasoner's knowledge base where for each tuple an own named graph is created [107]. A semantic tuple can be declared either as a *data tuple* or a *service tuple*. The declared type depends on whether a semantic tuple contains semantic information provided by a service or an agent, or it advertises an available service, e.g., a service controlling a light. Due to the advertising capabilities, service tuples allow agents to find a particular service. Data tuples are utilized to communicate with the found service.

Communication between agents is performed by writing and reading/taking semantic tuples from a space. After each write and take operation, the consistency of a knowledge base is checked. The operation is aborted if the new state of the knowledge base violates constraints defined in the associated ontology. On the other hand, the reasoning process is carried out for finding matching tuples for read and take operations.

There are two performance drawbacks in sTuples: (1) consistency check and (2) blocking read and take operations. The former relates to the consistency check triggered upon each change in the knowledge base which results in certain delays. The latter refers to the blocking operations where sTuples searches for results of read and take operations

every time new information is added to the sTuples space [100]. The repeated processing of blocked operations may impose critical load on the sTuples system.

TSC

The TSC [7] (Triple Space Computing) [44] project developed a platform for the management of distributed RDF data where the central data unit is represented by a named graph. Named graphs store multiple RDF statements and thus enable to read, update, or add multiple statements to the Triple Space in one atomic step. A Triple Space can manage an arbitrary number of named graphs. A prototype of the framework has been developed within the TSC project.

When a new tuple is written to the space, a new ID is generated and assigned to the tuple. Therefore, different tuples with the same data are allowed. To read a tuple from the space, the user has to pass either the URI of the tuple, or a template which describes the target named graph [44]. The template is transformed in the SPARQL query. In addition to utilizing templates for reading data, TSC supports queries as well. A query facilitates the templates to match RDF statements stored in named graphs and as a result, a new graph with the matching triples is returned.

In addition to the basic read, write, and take operations, TSC supports a simple form of publish/subscribe mechanism. It allows interested parties, i.e., content publishers and content consumers, to use template path expressions to either advertise or subscribe for particular information. Independently of the utilized operation, either read, update, or publish/subscribe, interaction between clients is always implicit and thus ensures high flexibility and strong decoupling.

The prototype implementation of TSC is based on the YARS [59] storage framework for persisting and querying RDF graphs. YARS is a highly scalable RDF store that supports queries suitable to manage semantic templates. Coordination and communication middleware utilized in TSC is CORSO [72]. CORSO provides a virtual shared memory space of objects and it supports transaction management and data replication mechanisms [76]. Named graphs are mapped to CORSO objects with distinct OID (Object Identifier) which facilitates their sharing among participating nodes.

The application areas addressed within TSC project are primarily related to the domain of semantic web services [100]. TSC was employed to integrate services in WSMX (Web Service Execution Environment) [43] for the integration of business applications

TripCom

The TripCom [8] (Triple Space Communication) [42] project was focused on developing a distributed knowledge base and scalable coordination platform facilitating capabilities of the tuple space technology. To achieve the goal, the project focused on the integration of three building blocks [100]: (1) tuple space technology, (2) Semantic Web Technology, and (3) Web Service technology. Tuple space technology was aimed at realizing efficient

[7] http://tsc.deri.at

[8] http://www.tripcom.sti2.at

client-to-client communication models as well as flexible coordination mechanisms for distributed clients. Moreover, Semantic Web Technology strived to provide knowledge representation and knowledge inference mechanisms, while Web Services were intended to enable agents to autonomously execute various tasks.

A space is modeled using named graphs which contain tuples. Tuples stored in a named graph are modeled utilizing triple data structure. Due to the fact that a named graph cannot contain two triples with the same data, two tuples with the same content are not allowed to reside in the same space. Moreover, reading tuples from a space is performed by means of defining a SPARQL SELECT form query which, as a result, returns tuples. The same template used for read operations can be applied to register a subscription as well [42]. As soon as the predefined template, i.e., SPARQL query, matches the data in repository, a user is notified.

The architecture of the TripCom is designed following the client-server paradigm. Similar to the modern client-server systems, TripCom supports operation of multiple distributed server nodes that use Distributed Hash Table (DHT)[130] for distributing tuples. Each server node hosts the space infrastructure and the data. On the other hand, clients communicate with any server node using the TripCom API and do not have to care about the data distribution [42] while the contacted server delegates write and read operations to the responsible server node.

With respect to the reasoning capabilities, the reasoning process is triggered upon each write or delete operation resulting in slow entry updates and fast queries. Consequently, a read operation returns both, user inserted triples, as well as inferred triples. However, a user cannot distinguish between these two.

The developed Triple Space has twofold purpose: (1) to publish and find semantically described Web Services, and (2) to realize the communication between clients and services. Similar to the Triple Space nodes of TSC, a TripCom Triple Space also handles a set of RDF statements. On the other hand, TripCom neither synchronizes nor replicates data stored on the nodes [100].

Semantic Web Spaces (SWS)

Semantic Web Spaces [134], [136] has been conceived as a middleware for the Semantic Web based on the extension of the existing Linda coordination model called XMLSpaces [135]. The main notion of Semantic Web Spaces is to enable clients using Semantic Web data to access and process knowledge to coordinate their interdependent activities [107]. Consequently, a client can utilize knowledge inserted into the space by inferring new information to facilitate its future decisions. At the same, another client that was waiting on the availability of these new, inferred, information is able to utilize it to draw some conclusions. Therefore, the space as a shared medium enables both, sequential knowledge processing, as well as the parallel coordination between concurrently active clients.

Similar to other semantic-enabled coordination frameworks, Semantic Web Spaces models semantic tuples by means of RDF triples. A modeled tuple consists of four fields which take URIs as values. The first three fields correspond to the subject, predicate, and object of an RDF statement, while the fourth field is a tuple identifier generated with the

help of a tuple space ontology [107]. Furthermore, querying a tuple space is envisaged as a triple pattern which is transformed to a SPARQL query which is further applied to the appropriate tuple space. Moreover, a basic tuple matching is extended with an RDF-specific reasoner which, in combination with a provided ontology, can be utilized to check ontological conformance. As a result, querying does not only have to be performed against existing RDF statements, rather, it can also include inferred statements.

Semantic Web Spaces define an ontology for describing the space itself which explicitly reflects the hierarchical structure of the space. The tuple space ontology is Jena RDF model with the meta-model of the space [107]. To mitigate performance degradation, the ontology update is not triggered on every operation on the space, rather, the models log changes and the update process is triggered only after a certain number of operations.

The prototype implementation revealed the feasibility of an approach based on Linda and a tuple space which supports coordination and communication between agents operating on the Semantic Web. However, a scalability challenge still remains an open issue.

Semantic Tuple Centres (STC)

Semantic Tuple Centres [102] extends the TuCSon [108] tuple space implementation with Semantic Web Technologies. Semantic Tuple Centres defines a semantic tuple as an instance of a user-defined OWL class called concept. For each tuple, a user has to declare an individual name which is translated to the URI for storing a tuple as an RDF resource into a triple store [69]. A tuple can contain properties which either describe a value, i.e., a literal value, or model a relationship with another tuple, i.e., an individual name. Due to a lack of support for blank nodes, complex properties are not allowed. An RDF blank node is an RDF node that itself does not contain any data, but serves as a parent node to a grouping of data.

In Semantic Tuple Centres, querying and reasoning capabilities, are realized combining Jena [155] and Pellet [158] reasoner. Therefore, upon inserting a tuple in a space, first, an OWL reasoner checks consistency of the inserted tuple against the space ontology, and then, the Pellet reasoner is utilized to generate inferred data. Generally, a tuple selection is performed by means of a semantic template which is converted to a SPARQL query.

To demonstrate coordination capabilities of Semantic Tuple Centres, a coordination infrastructure for electronic health care records (EHR) in the e-health domain [104] is showed. The demonstration was focused on presenting distribution, interoperability, as well as security features of Semantic Tuple Centres.

2.2.3 Semantic XVSM

This section introduces and describes the main concepts of the Semantic XVSM framework and its reference Java-based implementation, Semantic Mozartspaces, developed in [69]. The coordination model of XVSM is complemented and enriched with the Semantic Web Technology to produce a new coordination framework with a reasoning mechanism and

which can, due to ontology support, uniformly describe and communicate with distributed heterogeneous resources.

The section starts with the description of the semantic data model explained through the example. After that, basic query capabilities and improvements are introduced, as well as the ontologies and the reasoning support. Finally, the section is concluded with the description of notifications.

Semantic Data Model

The core part of the architecture is the data model that exposes a mapping process between MozartSpaces entries and semantic entries. The basic concept of XVSM is a container which hosts different entries where the value of an entry is an object with several properties, which themselves can be either literals or objects.

The data structure of an object used as MozartSpaces entry value is a recursive X-tree [21], which is either a sequence or a multiset of labeled X-trees, or a literal. In order to create a semantic MozartSpaces entry representation, an X-tree is modeled as RDF nested unlabeled blank node. Similar to the X-tree that exclusively owns all its nodes, a nested blank node has a single reference to its sub-nodes, and a complete nested blank node can be read out of a triple store including all its sub-nodes, which are blank nodes. Accordingly, an X-tree multiset is mapped to an RDF collection. Since nested blank nodes are distinguishable from each other inside a named graph, the counterpart of a MozartSpaces container is the named graph.

One of the most important components in Semantic XVSM is the resource mapper. This component maps Java objects to nested blank nodes. To accomplish this, Java classes have to be respectively annotated (@RDFType, @RDFField).

```
1  @RDFType (
2      defaultNamespace =
3    "http://mozartspaces.org/ma#", name = "Task " )
4
5  public class Task {
6
7      @RDFField (name = "hasName")
8      String name;
9
10     @RDFField (name = "needsSkill")
11     Set<URI> neededSkills;
12 }
```

```
1  @prefix :<http://mozartspaces.org/ma#>
2
3  [
4      rdf:type :TaskEnhanced ;
5
6      :hasName "spraying" ;
7
8      :needsSkill :SprayingSkill ;
9  ]
```

Figure 2.7: Semantic data model [32]

Figure 2.7 shows an example of an annotated Java class on the left hand side and a nested blank node on the right hand side. The example uses Turtle syntax to provide a suitable representation for a nested blank node. Figure 2.7 illustrates the structure of the Java class *Task* which is described later in the thesis as a part of a use case. The *Set* property *needsSkill* refers to the type of the skill required on the robot, e.g., a spraying skill.

Querying

One of the crucial factors for using Semantic XVSM is the capability to select entries with SPARQL queries. This feature is realized by implementing a new, customized, semantic coordinator with accompanying selector, which can still be combined with existing XVSM coordinators/selectors. The result of a SPARQL query is a list of entry IDs which is converted to a list of entries loaded on demand from a triplestore [26]. Same as with default selectors, SPARQL query supports the user parameter denoting the number of triples to be retieved, e.g., *count_ALL*.

The basic structure of a SPARQL query for selecting entries from a space is illustrated in the listing below.

Listing 2.2: Example for a SPARQL query in Semantic XVSM for entry selection [69]

```
SELECT ?entryId
FROM <container:1>
WHERE {
    [ sxvsm:id ?entryId ;
      rdf:type sxvsm:Entry ;
      sxvsm:hasValue [
        :hasName "spraying" ;
        :requiresSkill :SprayingSkill
      ]
    ]
}
```

The template for selecting entries is a nested blank node, same as the entry itself. The FROM query form specifies the named graph of the container *container:1* as a dataset on which the query is executed. For the sake of brevity, prefixes are omitted from the listing.

A semantic selector in Semantic XVSM can be extended to enable the use of external context entries in the query. The context can be added at the client side, or this can be achieved via a pre-aspect of the container (server-side). Context entries can be used as a parameter for SPARQL queries, so that more general and flexible queries are supported, e.g., in the use case presented in this thesis a context entry can describe the state of a robot. Additionally to the basic SPARQL capabilities, a semantic selector in Semantic XVSM supports also optional SPARQL constructs like FILTER, ORDER BY, and GROUP BY. For detailed description of the supported constructs, please refer to [69].

Ontologies and Reasoning

After the querying capabilities, reasoning is the most prominent feature of Semantic XVSM. Reasoning enriches entries with inferred data which can be queried and thus utilized by a user. However, entries classified as inconsistent, are blocked from getting

inserted into a container. Therefore, the container state, from the ontological point of view, is always consistent.

Supporting ontologies in the Semantic XVSM has manifold advantages: (1) describing a domain model to infer implicit knowledge, (2) classifying entries for a coordination purpose, and (3) preventing insertion of inconsistent data. In Semantic XVSM, an ontology is associated with a container using an *ontology entry* as a special type of entry. The ontology stored in ontology entries is then used for reasoning on newly inserted entries. Due to the simplicity, there is always only one ontology at the time available for reasoning. Since the reasoning is done for each entry on insert operation, ontology changes in a runtime do not affect the existing entries. This solution has several advantages [69]: (1) on entry insertion, a consistency check has to be performed anyway, (2) reasoning over a small dataset scales better for big containers, (3) reasoning can be limited to special type of entries instead of doing it for each entry. However, there are disadvantages as well: (1) an ontology update does not impact existing entries, and (2) not all inferred data are allowed in the container's graph. The former drawback can be overcome by triggering a reasoning process on existing entries after each ontology update. However, such a process is computational expensive and thus has to be used carefully. The latter disadvantage allows only enhancing the existing nodes of an entry with inferred data such as new properties and values.

Since the inferred data are generated during the write phase, i.e., when an entry is about to be written in a container, they have to be stored in order to be able to query them later. Although the new data are generated, it could still sometimes be beneficial to have the original data as well. Therefore, the original data must not be overwritten, rather stored with a possibility to distinguish between original and the inferred data. Semantic XVSM tackles this challenge by storing inferred data in an extra named graph which enables them to be used in a query for selecting entries by attaching the named graph in an additional FROM clause in SPARQL query. Same as for the ontology update when the reasoning has to be redone, the same process is required in case of deleting the *ontology entry*. In case inconsistency occurs during the management of *ontology entry*, transaction is aborted.

Type-based Notifications

Semantic XVSM utilizes the advantage of having aspects, especially post-aspects, to generate a notification as a result of an executed operation on an entry. In particular, Semantic XVSM enables semantically-supported notifications to be triggered when the special type of entry data, i.e., a class defined in the ontology, becomes present in a system.

Notifications are realized by means of subscriptions which observe a container of interest. In Semantic XVSM, subscriptions are written as special entries, i.e., *subscription entries* in the observed container to enable an easy management, i.e., a user is revealed from creating new aspects or extending an API. Every time the post-aspect of an entry operation is executed, a semantic read operation looks for subscription entries which conform with the processed entry [69]. Therefore, a subscription entry has following

information: (1) the entry type, i.e., an OWL class defined in an ontology, (2) action type, e.g., read, take, or write, and (3) a subscription id.

A notification is generated after the execution of an entry operation. This results in writing entries in a notification container. In the case of an aborted transaction, there will be no notification. Upon a successful commit of the entry operation, the notification entry becomes visible and ready to be processed by a subscriber. The notification entry encompasses the following data: (1) the action type, (2) the processed entry, and (3) the subscription id. To immediately receive the generated notification, the subscriber should issue a blocking take on the notification container. Due to the frequent notification check (for every action the system checks whether a notification should be generated) a notification service decreases the performance of space.

2.2.4 Comparison of Semantic Spaces

Table 2.2 compares the reviewed semantic tuple space coordination frameworks with respect to the following criteria: (1) **Data Model** describes how the data are organized in the space, (2) **Comm/Coord model**, i.e., communication/coordination, reflects the way agents (robots) interact with a tuple space, (3) **Querying** capabilities for retrieving the data from a tuple space, (4) **Subscription** mechanism for supporting agents' dynamic reactions, (5) **Reasoning** enables generation of new information by performing inference on existing data, (6) **Consistency check** supports for validity and consistency check upon inserting or deleting a semantic data from a space, and (7) **Transaction** mechanisms to support multiple operations as one atomic operation.

Table 2.2: Comparison of semantic tuple spaces

Features	**sTuples**	**TSC**	**TripCom**	**SWS**	**STC**	**Semantic XVSM**
Data Model	DAML + OIL	RDF graph	RDF triple	RDF triple	RDF graph	RDF graph
Comm Coord	publish/ subscribe	Linda publish/ subscribe	Linda	Linda	Linda	XVSM
Querying	own template	N3QL template	SPARQL	own template	own template	SPARQL
Subscription	entry level	entry level	space level	entry level	entry level	entry level
Reasoning	✓	-	✓	✓	✓	✓
Consistency Check	✓	-	✓	✓	✓	✓
Transaction	✓	✓	✓	✓	-	✓

With respect to the underlying data model, common to all analyzed semantic tuple space frameworks is the support to manage data using Semantic Web languages. Except the sTuples which utilizes DAML + OIL to represent semantically described data, the

remaining approaches enable a certain level of support for RDF data. In all approaches, tuples can be referenced using publicly accessible IDs. The communication/coordination model underlying the presented approaches is based on Linda or the publish/subscribe paradigm. Although XVSM is based on Linda as well, it is explicitly stated that Semantic XVSM, described in Section 2.2.3, is based on XVSM since it provide a broader set of coordination features than a basic Linda model, e.g., various coordinators. Moreover, all reviewed frameworks take into account the advantage of Semantic Web Technologies when querying the data from a space. However, most of them are limited to use own templates, like sTuples, SWS, and STC, which are then internally translated in SPARQL queries. TSC utilizes N3QL syntax [151] for querying a space which is a predecessor of SPARQL. Although, both TripCom and Semantic XVSM support SPARQL, the main difference is that Semantic XVSM supports the latest SPARQL version, i.e, SPARQL 1.1. Consequently, Semantic XVSM can be utilized to model heterogeneous resources as well as distributed task allocation process and thus can address RQ. 1 and RQ. 2.

Furthermore, all analyzed semantic tuple space frameworks support notifications on an entry level, except TripCom which uses notifications on a space level. Thus, they all support coordination of multiple agents. Reasoning capabilities can be utilized to facilitate a query performed on a space, as well as to trigger a notification when new data is inferred. Due to the lack of a reasoning mechanism, TSC has shrunken capabilities compared to other frameworks. The STC framework does not reveal inferred data to the outside world, i.e., a user. A reasoning capability infers consistency checks as well. Consistency checks ensure that a newly inserted, or deleted triple, does not violate ontology consistence. Due to the lack of a reasoning mechanism, TSC does not support consistency check. With respect to the consistency check, there are two modes: (1) consistency check is triggered automatically upon each change in a space, i.e., write/update/delete operations, or (2) consistency check is triggered periodically. The consistency check mechanism in sTuples, TripCom, STC, and Semantic XVSM adheres to the former mode, and in SWS to the latter. Except TSC and Semantic XVSM, there is no framework which supports long-lived concurrent transactions with timeouts. Since long-lived transactions with timeouts are important in distributed systems where robots operate in unstructured environments, TSC is perceived as a competitor to Semantic XVSM.

To conclude, the comparative advantage of Semantic XVSM, based on the MozartSpaces [74] which is s Java implementation of XVSM [22], over the other semantic tuple space frameworks is that it structures the space into containers that store entries using different coordination laws, e.g., FIFO, LIFO, KEY [26]. An entry includes both, coordination and entry data. Semantic XVSM borrows some concepts from the STC in which it treats a semantic tuple as an object of an application domain. In contrast to TSC which uses its own data format for semantic tuples, Semantic XVSM uses Turtle. Moreover, the objects in STC must belong to exactly one OWL class, while in Semantic XVSM a tuple can be a type of any numbers of OWL classes. This property facilitates robot-robot and robot-human coordination activities addressed in RQ. 2 and RQ. 3. Furthermore, Semantic XVSM offers a powerful capability to select entries by introducing minor re-

strictions on SPARQL query language [26]. Due to the described advantages of Semantic XVSM over TSC, Semantic XVSM is selected as a framework for modeling behaviour of distributed robots and a user in the scope of this thesis.

2.3 Summary

Since there are two different logical sections in the background technologies: (1) Semantic Web Technologies and (2) Space-based Computing paradigm, this summary is structured accordingly.

XVSM, as the representative of SBC paradigm, can take the advantage of extending the existing coordinators and combine them with the programmable aspects to enrich the existing coordination capabilities of XVSM with Semantic Web Technologies; in particular with ontologies support and SPARQL query language. Having integrated these two technologies ensures a solid basis for the development of general SKIM coordination framework. On the one hand, the benefit of using existing XVSM concepts can be reflected in seamless coordination between distributed and heterogeneous robots in a fleet that are decoupled in time and space. On the other hand, the benefit of using semantics is twofold: (1) support for having an ontology provides uniform description of heterogeneous and distributed resources, and (2) it models shared knowledge as a basis for adaptive autonomy in mixed teams.

Since this chapter covered the technologies utilized for the development of SKIM coordination framework, next chapter reviews existing frameworks for task allocation and coordination in robotic fleets as well as the notion of adaptive autonomy in mixed human-robot teams.

CHAPTER 3

Related Work

In contrast to Chapter 2, which reviews the technologies utilized in the scope of this thesis and discusses their advantages and disadvantages, this chapter gives a detailed overview of the existing task allocation models.

In the past couple of decades, research in multi-agent field has focused on designing systems containing multiple, autonomous agents (robots) that work together to accomplish a common objective. Having a system which encompasses multiple cooperating robots is inspired by natural systems where the groups of animals can solve problems that could not be solved by solitary individuals [14]. Due to the advances in the perception and locomotion technology there is a great potential to use robotic fleets in heterogeneous and unstructured environments and scenarios requiring collaboration in mixed human-robot teams [56], [111], [28]. This, however, imposes new demanding requirements on communication, data availability, and coordination of actions in such teams, and the well-established centralized coordination approach will be replaced with a new, distributed, approach.

The chapter begins with a general overview of the coordination middlewares where an autonomous robotic fleet executes a mission similar to the precision agriculture scenario introduced in Section 1.1. The outcome of the overview is a set of challenges that should be addressed when designing the multi-robot systems. According to [54] [88] [149] [109] [14], due to the increased complexity of the multi-robot systems, task allocation has risen to a prominent research challenge in this field. Therefore, the chapter proceeds with an overview of the existing task allocation models with a focus on the environment-based task allocation approach. Since [47], [55], [56], [111] envision the adaptive autonomy as the next evolution phase in multi-robot systems, the chapter is concluded with an analysis of the adaptive autonomy in mixed human-robot teams.

3.1 Coordination Middlewares for Robotic Fleets

The past decade has witnessed a huge increase in a number of proposed middleware solutions for robotic fleets operating in unstructured environments. As a result, it has become difficult to decide which middleware is the most appropriate one for a specific application or application domain. This section is limited to review coordination middlewares where an autonomous robotic fleet executes a mission similar to the precision agriculture scenario introduced in Section 1.1. Firstly, extracted is a set of common and specific challenges that middlewares address, and then these are grouped according to the source domain they originate from. The challenges are derived from the precision agriculture scenario introduced in Section 1.1. After that, the section provides an analysis of different middlewares. The aim of this analysis is to assist the process of finding an adequate middleware for a specific application domain.

3.1.1 Middelware Challenges in Robotic Fleets

When a new application is being developed, understanding the challenges of a specific application domain is the first important step towards making the decision which middleware to use. Analysis of challenges on a general level has been already undertaken in [111] and [78], where the focus is on the challenges imposed by unpredictable environments. This work has been extended in [93] and [61] where authors discuss some more specific issues and compare several middleware implementations against them. Our work [30] further extends this effort with a focus on extracting additional, less covered issues imposed on the middleware for robotic fleets. Proposed is a classification of middleware challenges according to their specific domains of concern: (1) general issues as related to a fleet realization, (2) issues imposed by operating in unstructured environments, and (3) challenges resulting from the task complexity. This analysis can be used to identify a suitable middleware for a specific robotic fleet application.

General Fleet Realization Challenges

Following are five identified problem statements common for the middlewares reviewed in Section 3.1.2 as well as for the introduced agricultural use case. **(1) How to distribute control?** Control can be either centralized or distributed. The control organization is a concern critical in the early stages of the system design ([111] and [61]). It influences other system decisions that are to be made in later phases, e.g., the autonomy level and collaboration patterns. **(2) How to distribute functionalities within a robotic fleet?** There are two types of fleets: homogeneous and heterogeneous. The robotic fleet application directly drives a decision on the type of robots in a fleet ([111] and [78]). The systems exploiting parallel, and in time and space distributed tasks, often use large scale fleets of interchangeable homogeneous robots. On the other hand, more demanding applications may require teams of individuals with specific sensors or actuators. **(3) How do communication, collaboration, and coordination influence the level of autonomy?** Information exchange is vital for collaboration and coordination, which

are complementary processes running over a communication backbone of multi-robot systems. The information that supports robots in achieving their goals can be obtained in different ways, e.g., by sensing the environment in which robots are operating, by observing actions of the peers (other robots), or by an explicit message exchange with the peers. Based on the acquired information, robots can decide on their collaboration and coordination patterns. Furthermore, autonomy is a significant feature of robotic systems [93]. Autonomous robots utilize communication infrastructure and sensors for automated local or group decisions and actions, with no human intervention. Current research in autonomy is focused on developing different levels of autonomy and thus providing the robots with adaptive autonomy functions. Adaptive autonomy enables human-robot fleets to incorporate advanced coordination and collaboration mechanisms with different levels of robots and humans involvement. **(4) How to specify a mission?** A robot task can be decomposed into independent subtasks, hierarchical task trees or roles [111]. The mission tasks can be designed either by an autonomous planning function, or by the human designer. In general, a common way of defining a mission is by defining a set of tasks that have to be completed within a specified time. Each task can be processed by a variety of different robots, and each robot can work on different tasks. The mapping between available robots and generated tasks is the solution to the task allocation problem. **(5) How to semantically represent resources?** Semantics can be used to model resources provided by heterogeneous devices distributed in the environment. Describing functions of cameras, sensors, actuators, etc., with a common ontology eases the process of finding appropriate resources in the environment. Semantic technology is underutilized in robotic fleets but has a large potential due to the flexibility it offers. [30] identified these core challenges as both imposed by the introduced agricultural use case and imminent to the design of every middleware. They have to be continuously addressed in different phases of the middleware design, and build a basis for extensions according to special needs of robotic application.

Environmentally Imposed Challenges

This class comprises of three problem statements characteristic for robots which have to cope with unstructured and volatile environment that introduces additional complexity in system implementation. **(1) How to deal with uncertainty in communication?** A robotic fleet can switch between disconnected and connected operation mode. This may be a crucial requirement for a robotic middleware applied in unstructured environments. In most scenarios with intermittent communication robots have a connection with others, but only for a limited, unknown time. **(2) How to deal with dynamics due to faults?** Robustness and fault tolerance are relevant features of every multi-robot system that executes time critical tasks [111] and [61]. In general, robustness and fault tolerance guarantee operation in the presence of malfunctioning components, which requires that the system has autonomy capabilities to continue the work with reduced resources. Adaptability is a design feature that enables robots to change behavior according to the dynamically changing requirements posed by the environment, e.g., as triggered by mission customization, changing resources of teammates, or the need to prevent performance

degradation [78], [61], [110]. **(3) Can a new behavior emerge?** Behavior-based system design enables robots to perform tasks without having explicit set of instructions for their execution in advance [111]. In general, robots may use knowledge of the current state of the robot mission, robot team member capabilities, and robot actions, to decide, in a distributed fashion, which robot should perform which task. Dynamic team formation may occur either when one or more robots move away and lose connectivity to others, or when a robotic fleet is split in groups according to assigned tasks. The former is the physical, and the latter is the logical separation. Due to the volatile environment where the agricultural use case takes place, the assumption is that support for operation in disconnected mode is a comparative advantage in a robotic middleware. Ability to operate in this mode, complemented with an adaptive behavior and dynamic team formation, has the potential to increase system robustness.

Task Dependent Challenges

Task specific challenges are related to mission or task requirements that may be different in each application domain. Here are identified four common problem statements: **(1) How big the fleet may be?** The scalability support is an essential feature of a robotic fleet operating in unstructured environments [61]. In general, scalable, open systems, e.g., systems comprising different types and numbers of components, such as agents, computers, humans, have to support dynamic joining and leaving of components. Scalability relates to the ability of the system to accept new components without significant change in architecture and design. **(2) How much knowledge shall be shared?** Shared knowledge is a driver for successful coordination between robots [61]. To attain knowledge about other robots in a fleet, a single robot does not have to contact a centralized knowledge repository. Instead, the local (distributed) knowledge can be maintained and used in tasks where robots have to combine their services in order to successfully accomplish a given task. In this context, discovery mechanisms are essential components of dynamic computing environments [78]. During environment exploration, mobile robots discover external resources, like cameras, sensor networks, and configure themselves to interact with them. **(3) What is the human role?** Robotic fleets are designed for limited autonomous operation in the field and thus require interaction with humans. The requirements on the human-robot interaction capabilities depend on the challenges of specific applications. A control concept exploiting different levels of autonomy is an important trend in human-robot interaction research [27]. **(4) Shall fleet resources be dynamically allocated based on context and costs awareness?** The context and costs awareness are important task-specific challenges [150] and [53]. To support cost-based decision making each task has to be assigned with an objective function to minimize the cost of resources and maximize the benefit gained by performing that task. In this way a fleet operates as a system where actions are driven by business objectives. It is hard to identify one specific task dependent challenge as the most relevant because the requirements depend on the application domain. Thus a middleware designer needs to decide which should receive more attention, based on requirements of the specific use-case.

3.1.2 Comparative Study of Selected Middlewares

This section reviews the number of prominent existing middleware solutions and compares them against challenges introduced in the section above. The selection criteria was that a reviewed coordination middlewares supports an autonomous robotic fleet which executes a mission similar to the precision agriculture scenario introduced in Section 1.1. Moreover, considered are only solutions that deal with higher layer functionalities (software-based functionalities) instead of middlewares focused on the development of hardware controllers, like the middlewares ALLIANCE [110] and MARTHA [1], Collaborative tasking middleware [85], Physically Embedded Intelligent Systems (PEIS) [11], [117], [58], a market based approach [150] which is referred to as MarketE, and Human-Agent-Robot Teamwork (HART) [65]. Solutions controlling hardware components on robots are out of the scope of this review. Each reviewed middleware addresses the larger number of challenges identified as important in previous sections. This is not the case for the majority of other existing implementations which address more restricted number of challenges. Excluded are solutions which are limited to a specific application domain, e.g., Distributed Robot Architecture DIRA [127], Linda in Mobile Environment (LIME) [99], middleware referred to as SOLD [53] and Autonomous Robot Architecture (AuRA) [3].

ALLIANCE [110] defines a framework that allows robot teams, where each robot is equipped with a variety of high-level functionalities, to individually select appropriate actions based on the mission requirements, an environment, activities of other robots, and an internal state. The middleware framework is distributed and implements behavior-based architecture which enables robots to act based upon their current state. MARTHA [1] focuses on the control and management of autonomous fleets for transhipment tasks in harbors, airports and marshaling yards. The focus is on the increase of robots' autonomy as a key solution for decentralization, which allows robots to efficiently cope with unexpected environmental issues, e.g., obstacles and other robots. The central station does not intervene in the robot coordination tasks, nor does it calculate precise trajectories robots have to take. Thus, the required communication bandwidth between robots and the central station is very low. Collaborative Tasking [85] middleware supports market-based task allocation through the implementation of the standard Contract Net Protocol called Collaborative Tasking Protocol (CTP) for a group of heterogeneous unmanned vehicles. When a task is injected, each vehicle estimates its cost to perform the task, taking into account remaining consumables, required effort, its other pending tasks, and user preferences. The concept of Ecology of Physically Embedded Intelligent Systems or PEIS-Ecology, aims at building intelligent robots in the service of people [11]. In general, PEIS is defined as a set of connected PEIS components that reside in the same physical place. PEIS-kernel enables each PEIS component to communicate and participate in a PEIS-Ecology by implementing distributed tuplespace. PEIS components use cooperation model based on the linking of functionalities: each PEIS component is able to use services provided by other PEIS components complementing its own functionalities. [150] exploits market architecture to maximize information gain while minimizing incurred costs. It uses the concept of market economies, which are distributed

systems where individuals exchange goods and services by establishing contracts. Here, multiple robots interact in a distributed fashion to attain global goals in the efficient way by maximizing their profits. Finally, KAoS HART [65] supports coordination in a mixed-team with adaptive autonomy where human-robot interaction takes place. Due to its hierarchical organizational structure, control is centralized and performed by the team leaders. A team leader defines a common goal and monitors its execution. Other team members follow the leader. Team members register at centralized directory service where they publish a description of capabilities they provide. This enables them to perform a lookup for desired services and match them against their own requirements. Coordination among team members is based on a set of policies that manage the organizational structure among the agents.

Conducted is a comparative study to detect gaps which existing middlewares do not address, to indicate caveats in a design of a distributed middleware, and to help developers to avoid them. Three tables presented in this section summarize mapping between challenges (columns) from the Section 3.1.1 and selected middleware implementations (rows). Fleet realization challenges discussed in the Section 3.1.1 are reviewed in Table 3.1. Environment specific are included in Table 3.2 and task entailed are shown in Table 3.3.

Table 3.1 compares the above coordination frameworks with respect to the following general fleet realization challenges: (1) Control structure referring to centralized (C) or distributed control (D), (2) Robot diversity referring to heterogeneous (Het) or homogeneous robots, (3) 3C - Communication, Collaboration and Coordination, (4) Robots' autonomy, (5) MD and TA - Mission Definition and Task Allocation, and (6) Semantics.

ALLIANCE is a distributed software architecture that facilitates fault tolerant cooperative control (Table 3.2, Robustness) of heterogeneous mobile robots. The control is distributed and supported via control mechanisms deployed on all robots. Robots have different abilities, e.g., different sensors and actuators. Information sharing occurs when each robot broadcasts a state of its current actions on which other robots are listening. Collaboration is attained through the common work on same tasks. If one robot cannot finish a task it is in charge of, within the required time, or is not able to finish it at all, other robots will be notified. Due to the implemented features, especially support for distributed and heterogeneous robots, ALLIANCE partially satisfies the presented challenges. However it lacks support for fine-granular mapping of the mission into a set of tasks and for task allocation.

Due to the centralized control in MARTHA, the framework is able to control both heterogeneous and homogeneous robot. Whenever a robot produces a plan, which uses some shared resources, cells or trajectories, it advertises it, and collects from other robots their resource usage plans. Then it produces a coordinated plan and informs the other robots of events like cells exit or particular point traversal on trajectory. Furthermore, different strategies for plan coordination are proposed, with the local scope of planning actions, instead of a global. This means that robots plan their actions locally instead of receiving them from a centralized place. The communication between robots has higher

Table 3.1: Middlewares compared against fleet realization challenges

Framework	Control structure	Robots diversity	3C	Autonomy	MD and TA	Semantics
ALLIANCE	D	Het	Broadcast	✓	-	-
MARTHA	C/D	Gen	Plan merging	✓	Local task planning	-
Collaborative tasking	C	Het	Broadcast, bidding	-	Market-based	-
PEIS	D	Het	Tuplespaces	✓	Tuple collection	✓
MarketE	D	Gen	Price-map exchange	✓	Tasks trading	-
KAoS HART	C	Het	Policies	✓	Utterances	✓

priority than the communication with a Central Station, which itself requires a high bandwidth and reliable communication link.

Collaborative Tasking automatically designs and re-designs tasks for a group of heterogeneous unmanned vehicles. It uses a central agent which runs the bidding processes. The agent runs the Collaborative Tasking Module which decomposes a high-level mission tasks into executable tasks and broadcasts these to the robots in charge for performing them. Robots are heterogeneous with complementary services that are combined together towards successfully achieving mission goals. Each robot knows a set of tasks it is capable to perform, as well as the cost of performing a specific task under current circumstances. The middleware supports market-based task allocation through the implementation of the extended version of the standard Contract Net Protocol called Collaborative Tasking Protocol (CTP).

PEIS addresses all challenges summarized in Table 3.1. Decentralized control is identified as a main requirement in the peer-to-peer PEIS-Ecology. The solution to decentralization is based on advertisements by each PEIS-component in the same space where this component runs. Distributed tuplespaces enable each component to make a decision locally, within its own decision space rather than having a central decision making system. Furthermore, to establish information sharing between heterogeneous components, each PEIS component displays its services and functionalities by sending XML based messages. Components use a cooperation model based on the linking of functionalities: each PEIS component is able to use services provided by other PEIS component complementing its own functionalities. The semantics is introduced to overcome the issue of heterogeneous distributed components. Semantic resource

description is a comparative advantage over the other frameworks which makes the entire system extendable.

MarketE distributes control mechanisms over the robots assigned to explore a certain area. Each robot, which implements negotiation algorithms can participate on a market and compete for announced resources. Thus, the framework supports both heterogeneous and homogeneous robots. The robots make decisions by communicating price information and continuously negotiating with others to improve their task execution plains. In addition, robots explicitly share their maps of visited, explored, areas in exchange for revenue. The framework exposes results that show how the collaboration between robots increases task and mission efficiency.

KAoS HART is a middleware with centralized control that supports coordination in a mixed-team where human-robot interaction takes place. Mixed human-robot teams introduce heterogeneity in the framework. The framework uses policies implemented in OWL as rules for dynamically regulating behaviors imposed by different components. Policies are used for a mapping between natural language and commands which are basis for a successful coordination between team leaders and team members. The commander uses the same policies to decompose a mission into tasks, and by utilizing utterances it delegates those tasks to specified team members.

Table 3.2 compares the above coordination frameworks with respect to the following environment specific challenges: (1) Disconnected mode, (2) Robustness, (3) Adaptability, (4) Emerging behavior, and (5) Dynamic team formation. A tick denotes that a framework supports a certain feature, while "-" denotes that it doesn't.

Table 3.2: Middlewares compared against environment specific challenges

Framework	Disconnected mode	Robustness	Adaptability	Emerging behaviour	Dynamic teams
ALLIANCE	-	✓	✓	✓	-
MARTHA	-	✓	✓	-	-
Collaborative tasking	-	✓	-	-	-
PEIS	-	-	-	-	-
MarketE	✓	✓	-	-	-
KAoS HART	-	✓	✓	-	✓

ALLIANCE has control mechanisms that rely on different sets of behaviors where the activation of a certain behavior depends on: (1) the efficiency of performing a local task, and (2) how efficiently the teammates are performing their tasks. The framework tackles

adaptability and emerging behavior as well. Since robots in MARTHA plan the mission locally, instead of communicating it with the Central Station, they expose certain level of robustness and fault tolerance. Furthermore, the local mission planning system increases adaptability and autonomy levels. Collaborative Tasking Module integrates mechanisms for handling unsuccessful task allocation processes by means of robustness. The PEIS framework does not directly address any of the challenges related to the open environment because it is designed to operate in the structured and controlled environments. However, it supports semantic resource modeling that makes it easy adaptable to various use-cases.

Due to the communication uncertainties in MarketE, the robots are equipped with mechanisms to retain system's functionalities with zero communication. Robot's actions are triggered by arrival of messages that contain goals a robot is going to execute. If for some reason the robot did not receive a message it expected, either due to a communication problem or due to its peer failure, it has to be able to proceed with a task rather than indefinitely wait on a message. Hence, unreliable wireless communication does not disable a team to perform tasks, but reduces its efficiency. Retaining operation capabilities in case of reduced or even broken communication is a desired feature of robotic fleet systems. KAoS HART supports dynamically changing policies and therefore can accommodate changes imposed by volatile environment. Furthermore, using utterances that are mapped to commands utilizing policies, the system can easily adapt to changing environments and dynamically form teams.

Table 3.3 compares the above coordination frameworks with respect to the following task specific challenges: (1) Scalability, (2) Aggregated knowledge, (3) Resource discovery, (4) HRI - Human-Robot Interaction, and (5) Cost awareness.

Table 3.3: Middlewares compared against task specific challenges

Framework	**Scalability**	**Aggregated knowledge**	**Resource discovery**	**HRI**	**Cost awareness**
ALLIANCE	-	✓	-	✓	-
MARTHA	-	-	✓	✓	-
Collaborative tasking	-	-	-	✓	✓
PEIS	✓	-	✓	-	-
MarketE	✓	-	-	-	-
KAoS HART	✓	-	-	✓	-

Robots in ALLIANCE share information about the status of a task they are per-

forming with teammates establishing mutual support and enhancing mission efficiency. Information sharing encourages robots to learn about actions and knowledge of peers. At the same time, exchanged information is also presented to the users building a basis for human-robot interaction (HRI). Due to having knowledge about teammates, robots enhance the efficiency of their coordination mechanisms as it will be presented later in this thesis. Consequently, coordination can be transferred from a central place to distributed robots.

MARTHA proposes incremental resource discovery and acquisition process as a more flexible way of adaptation to the dynamically changing environment. Human-robot interaction in MARTHA is bidirectional: on the one hand, robots receive a high-level mission from an external user and on the other hand, they continuously send their status to the Central Station (HRI). In Collaborative Tasking when a task is injected, each vehicle estimates its cost to perform the task taking into account remaining consumables, required effort, its other pending tasks, and user specified preferences. A high-level mission definition is a product of a remote operator who uses a specific device with a customized user interface (HRI).

PEIS implements dynamic join and leave of components, i.e., scalability. Each PEIS-component describes in a formal way services it offers, together with required input and output ports, dependencies, type of data, etc. By having uniform semantic-based services description, all heterogeneous components are able to communicate and cooperate. MarketE scales and the new robots only have to implement desired negotiation algorithms in order to participate in a mission. An additional feature of the policies in KAoS HART is provision of a scalability support. When a new robot joins, it automatically acquires the intelligence possessed by the others. User interface provides a user with information from robots, such as position, state, video, thereby, enabling monitoring of the system behavior (HRI).

3.2 Task Allocation in Robotic Fleets

The main challenge for a robotic fleet is to perform a mission consisting of multiple independent tasks. To have a group of robots effectively performing tasks, a designer of a robotic fleet has to address the question which robot should do which task and when. The process of assigning individual robots to tasks forming a mission is called *task allocation* [81]. Task allocation is a fundamental problem in multi-robot systems where the core requirement is to find an optimal set of heterogeneous robots that have to cooperate in order to execute a complex mission [10].

The notion of a task in multi-robot systems designates an atomic unit (task) that is necessary for achieving the overall goal of the system, i.e., a mission, and that can be achieved independently of other atomic tasks. However, robots still need to communicate which robot will execute which task. A task can be discrete, e.g., spray a certain part of a field, or continuous, e.g., monitor a mission execution. In addition, tasks can also be of different complexity, duration, and specificity [54]. On the other hand, there are robots which are either homogeneous or heterogeneous and which differ in the amount

of resources. As robots have different capabilities and tasks require different skills, the main aim of the system is to schedule tasks to robots in such a way that all tasks are successfully completed.

Moreover, the task allocation is well known to be an NP-hard problem in multi-agent systems leading to a variety of different heuristic-based approaches[40]. In [54] proposed is a well-established taxonomy of Multi-Robot Task Assignment (MRTA) problems to show how various MRTA problems can be positioned in the resulting problem space. MRTA problems are described based on the following three axes: (1) single-task robots (ST) vs. multi-task robots (MT), (2) single-robot tasks (SR) vs. multi-robot tasks (MR), and (3) instantaneous assignment (IA) vs. time-extended assignment (TA). ST denotes robots capable of executing at most one task at time, while MT marks robots which can execute multiple tasks simultaneously. SR denotes that each task requires exactly one robot to achieve it, while MR means that some tasks can require multiple robots. Finally, IA permits only an instantaneous (static) allocation of tasks to robots without planning for future allocations. On the other hand, TA supports dynamic task allocation over time. These axes serve as a guideline for classifying the task allocation problem, and, additionally, they offer a reduction from the task allocation problem to one of the well-known problems in combinatorial optimization, e.g., Optimal Assignment Problem, Set Partitioning Problem, Set Covering Problem.

Furthermore, robots cooperation complements the efficient task allocation approach and thus increases overall mission efficiency. This kind of allocation, which includes robots cooperation, finds its applicability in many domains in real world, e.g., e-commerce [48], grid computing [105], social networks [64]. Figure 3.1 illustrates 6 domain-independent task allocation approaches which are described in detail in the following sections.

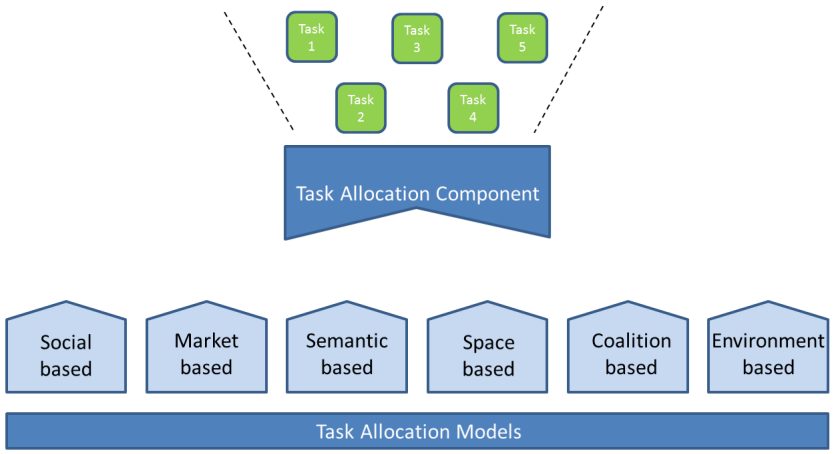

Figure 3.1: Task allocation models

3.2.1 Task Allocation in Social Networks

Common understanding of an agent social network is defined in [23] as an undirected graph where vertices are agents and each edge indicates presence of a social connection between two agents. The notion of agent can be thought off as a robot in our use case. [23] employs Greed Distributed Allocation Protocol (GDAP) to handle task allocation process in agent social networks. The task allocation process utilizing GDAP encompasses a manager agent seeking for neighbouring contractors to help him/her with the tasks. Similar to the auction-based task allocation approach, there are agents evaluating tasks offered by the manager agent and bidding for the most efficient tasks. If sufficient resources have been offered for a task, the manager contracts a bidding agent. Otherwise, if offers from all neighbours are received, but none of them offers required resources for a task execution, the task is removed from the manager's list. Due to relying only on the immediate neighbors, GDAP may cause several unallocated tasks. To overcome the main shortcoming of GDAP, i.e., unallocated tasks, [146] improves the existing GDPA, naming it Efficient Task Allocation Protocol (ETAP), by extending the manager's functionalities. The manager is able to push an unallocated task deeper in the social network, to its neighbours who are then trying to allocate the task within their neighbours. Experiments conducted in [146] showed that the ETAP can allocate more tasks than the GDAP due to the novel reallocation mechanism. On the other hand, GDAP needs less time since it relies only on immediate neighbours.

[64] explores a negotiation reputation protocol for task allocation in social networks where an agent's past behaviours in the resource allocation influence its current ability to get assigned a new task. However, not all agents in Multi-agent System in Social Networks (MAS-SN) are truthful and provide real information on available resources during a negotiation process. There exist deceptive agents that falsify their resource status information in task allocation and do not really contribute resources during task execution. To alleviate the problem of deceptive agents, the authors proposed a reputation-based allocation mechanism and a reward/punishment mechanism designed to favor the truthful agents by increasing their probabilities to receive a task. [142] also proposes a community-aware task allocation model for MAS-SN. Presented task allocation model uses heuristics to assign tasks and limits the agent's cooperation domain to the community it belongs to which means that it can only negotiate with its intracommunity agents. The disadvantage of the proposed models is the lack of shared knowledge between agents in the same community. This results in the excessive intracommunity communication during the task assignment because an allocating agent communicates also with the agents that do not have capabilities to execute a certain task.

3.2.2 Auction and Market-Based Task Allocation

An auction-based task allocation approach usually works in a way that there exists a market where tasks are sold by brokers to robots. On the one side there is a broker who generates a task for a sale and places a price on the generated task. On the other side, each robot bids on a task based on its perceived fitness to perform the task. There is an

auctioning mechanism which decides based on the price, i.e., clearing price [137], which robot gets the task, i.e., a winning robot [88]. TRACE framework introduced in [40] utilizes a price-directed approach for resource allocation and Contract Net Protocol [129] for the task allocation. During the task allocation process, an agent receives a request for a task and then determines whether or not it has the capability to carry it out. If the agent cannot carry out the task, it generates a proposal for other agents to carry out the task and sends an announcement message to all agents of its organization. The other agents calculate whether the task fits in their schedule, and if so, send a bid message to the announcement agent, i.e., the organizer [40]. The organizer takes a decision which agent will get the announced task. [83] present a market-based multi-robot task allocation algorithm that produces optimal assignment by using a pricing policy that responds to cliques of customers. The pricing policy activates when the two agents are interested in a same task, i.e., a colliding task. In that case the task price is periodically incremented until only one interested agent is left. The algorithm operates under the assumption that all tasks are known in advance and that all robots have the knowledge about all tasks. Moreover, each agent receives only one task which cannot be reassigned.

[149] proposes a distributed allocation algorithm based on the market approach utilized by robots to simultaneously and continuously allocate and decompose complex tasks. When an auction is announced, i.e., a task is offered on the market, participants compute their bids based on the profit they expect to gain by executing the offered task. After collecting bids, a robot that can perform the task with the best price is rewarded with a contract. Each robot maintains an internal schedule of tasks it has committed to and utilizes them when computing the marginal costs when bidding for a new task. Moreover, each robot can act as a trader as well and announce a task from its schedule to the others, thus facilitating peer-to-peer trade amongst the team and enabling task reallocation. Additionally, [149] treats all tasks as complex and represents them using task trees. Consequentially, the task market is extended to support bidding on any combination of nodes in the tree. [7] introduces the framework for multi-robot coordination based on a negotiation for task allocation and cooperative reaction to contingencies. For negotiation used is an adapted version of the Contract Net Protocol. Although the system proceeds with a regular mission execution after a failure occurs during task execution, the negotiation process to reallocate a task requires excessive communication among robots to proceed with task execution. The task allocation mechanism only supports allocation of simple tasks, i.e., tasks requiring only one skill for complete execution. [13] proposes a framework that uses a behaviour-based and a market-based approach to model reconfigurable robot teams. Both approaches are simultaneously used to show that they successfully avail themselves of task allocation in complex missions. It is showed that the market-based approach can be easily extended to support heterogeneous resources and different environments. Since the robots collaborate through the merge behaviours, a limiting factor could be the constraints which requires time and space coupling of robots when they want to merge, i.e., they have to be at the same time in the same place to merge.

[140] presents a distributed market-based algorithm called S+T which solves the

MRTA problem in domains where the cooperation between robots is necessary. The main idea is that a robot, if it cannot execute a task by itself, seeks for a help among it peers. The help is perceived as a service offered by other robots. The basic S+T algorithm is advanced to support resolution of deadlock situations, i.e., when the execution of tasks depends on others, and also has adaptable parameters influencing either the execution time or the energy consumption. Due to the influence on the number of exchanged message and the distance travelled, i.e., energy consumption, the robot's communication range is perceived as the most important parameter in the algorithm.

Following are the most prominent representative of the auction-based approach: MURDOCH [53], First-price auctions [150], Dynamic role assignment [15].

3.2.3 Semantic-Based Task Allocation

Another way to think of the task allocation is to investigate a matching problem that appears in e-commerce domain. With this regard, authors in [48] describe an economical approach for solving the matching problem by means of a multi-agent system representing an electronic marketplace. [114] elaborates that discovery is one of the essential activities in e-commerce enabling the matching between demands and supplies. Moreover, [114] substitutes traditional discovery solutions based on the exact matching with those building around semantics. The proposed approach utilizes OWL to describe trading objects, i.e., content, as well as requests generated by users and offers from the content providers. Outcome is an ontology which is a base for the two level filtration mechanism of advertised content. In the first level, the broker agent applies a semantic-based mechanism which compares a content requested by users to that advertised by providers. On the second level, the best content provider in terms of both, price and quality, is selected. Mapped to our use case, a user request is a task and an offer from a content provider is a robot with provided skills. On the one hand side, the proposed model rewards low cost and high quality content providers, and on the other hand side, punishes high cost and low quality content providers. According to [48], the autonomous semantic-based content discovery framework is a better solution compared to the traditional keyword-based discovery because it yields higher matching degree and increases flexibility by means of describing resources.

[10] addresses the problem of finding a set of agents that can participate in the task allocation processes. The framework utilizes semantics as a communication paradigm which assists to overcome heterogeneity issues posed by a large number of autonomous agents. The main benefit, attributed to the semantic infrastructure, is a semantic operability which enables that heterogeneous agents establish a common understanding of shared information. In that way heterogeneous agents have the same semantical interpretation of tasks exchanged among them. To ensure semantic operability, framework [10] utilizes a task ontology language (OWL-T) [138] which is a template language based on OWL for formally and semantically defining task templates to capture system demands.

Moreover, [105] addresses the resource matching issue in grid computing where a fundamental task is to decide which job will run on which resource. Traditionally, each organization uses its own language and notation for publishing its resource properties

and application requirements and thus makes the interoperation very expensive [35]. To tackle the problem, authors in [105] developed a component known as a resource broker. The resource broker architecture consists of three modules: (1) request handling module, (2) resource discovery, and (3) matchmaking module. The resource handling module is an entry point for a user who submits his/her request in RDF. Upon receiving a user request, the resource handling module adds semantic annotations to the request and forwards it to the matchmaking module. After that, the matchmaking module performs matching between the request and advertised resources based on the requested and provided capabilities. However, to get a list of resources, the matchmaking module consults the resource discovery module which returns a list of available resources in a grid. Moreover, the matchmaking algorithm classifies the matches into four different group: (1) Exact, where all requested parameters are matched by a resource, (2) Plug-In where the resource parameters are subset of requested parameters, (3) Subsume where the resource parameters are superset of requested parameters, and (4) Fail. Domain, resource description, and resource request ontologies enable the successful operation of the matchmaking algorithm. [86] addresses the problem of resource matching in grid environments. In contrast to most of the existing discovery components in grids that utilize keyword based matching, [86] proposes semantic based resource discovery mechanisms that use resource information and local resource policy information. Similar to [105], in [86] a general job execution process starts with user request processed by a request handler service which extracts application requirements. After that, the extracted requirements are semantically described using the Grid Resource Ontology template. Just after that, resource and policy information about the grid resources are fetched and stored in the knowledge base using the same ontology template as for the requirements. Finally, a semantic discovery component constructs a query based on the user requirements and the resource usage policies and executes the query on the previously populated knowledge base. The query execution relies on the inference engine to retrieve the inferred knowledge. Same as in [105], the matchmaking algorithm classifies resources in four different group: (1) Exact, (2) Plug-In, (3) Subsume, and (4) Fail. [132] proposes a flexible and extensible approach for solving resource matching issues in grids using Semantic Web Technologies. The main shortcoming of the traditional matchmaking algorithms relying on keyword based matching is a limited flexibility and inability to extend to new concepts. The main component of the framework [132] is the ontology-based matchmaker which utilizes ontologies and rules based on Horn logic [87] and F-Logic [68] for resources matching. The following three ontologies, built by using RDF-Schema, are main input parameters for the matching algorithm: (1) Resource ontology describing resources, e.g., *ComputerSystem*, (2) Resource Request ontology captures request properties, e.g., *NumberOfCPUs*, and (3) Policy ontology capturing the resource authorization and usage policies. After all input parameters are semantically described using the above ontologies, the matchmaking procedure composed of various inference rules is triggered to reason on the request properties, available resources, and usage policies, to find a resource that fulfils the request requirements. The reasoning process is enabled by the TRIPLE/XSB deductive database system [157]. The notion of

ontologies encourages the loose coupling between request and resource descriptions and thus removes inflexible coordination strings between resource providers and consumers.

The resource allocation approach, named Semantically Enhanced Resource Allocation (SERA) [34], uses Semantic Web Technologies, agents, and virtualization. The use of semantics incorporates flexibility and reasoning in the framework. Agents are suitable for monitoring and reacting to possible undesirable events, e.g., SLA violations. There are two main type of agents in the system [154]: (1) Job Agents (JA) in charge of managing users' tasks, and (2) Resource Agents (RA) responsible for managing providers' resources. Agents' behaviours are implemented complying to the Belief-Desire-Intention (BDI) [116] model where beliefs represents task requirements and resource capabilities, goals are the successful execution of users' tasks, and plans are actions to be performed in different situations. On the other hand, virtualization provides a specific execution environment to each single task without risks to the underlying system or other tasks.

The central component of the SERA framework [34] is a Semantic Scheduler that allocates resources to each task according to its requirements, its priority, and the system status. Allocation decisions are performed applying Horn rules on the ontology utilized for describing tasks and physical resources. However, prior to the execution of allocation algorithm, input parameters, i.e., task requirements and resource descriptions, have to be semantically annotated. The Description Mapper [154] is a component in charge of mapping the information provided by users and providers to an ontology, using the RDF model, which provides a common knowledge. Semantically annotated users' and providers' data are stored in a repository and together with the common ontology creates a knowledge base queried by different JAs and RAs. The purpose of queries is to select all the hosts from the repository that fulfil the resource requirements of the task [35]. The common ontology utilized in the SERA framework is based on the Grid Resource Ontology (GRO) [152]. Finally, when all input data is semantically annotated, the resource allocation for a particular task is decided between the JA and a set of RAs using the Contract Net Protocol [154]. The same architecture is utilized in the framework [36] which combines prediction techniques with semantic technologies to improve the allocation of resources to the different tasks. Although the framework supports multiple agents, it does not provide collaboration capabilities between agents.

Although the Semantic Event Notification Service (SENS) [101] does not explicitly address the task allocation problem in multi-agent system, it can be utilized to coordinate task allocation because it provides knowledge-based coordination capabilities. SENS is a publish/subscribe system which uses ontologies to capture domain knowledge. Client can subscribe for changes or extensions of domain knowledge by registering a description of this knowledge at SENS using SPARQL. When a new event arrives at SENSE, the SENS tries to infer additional knowledge by reasoning on the available data and the provided ontology. If the new knowledge is relevant for any of the subscribed clients, they receive a notification about the new knowledge. Utilization of ontologies and subscriptions enables modelling of complex coordination processes.

According to [35] and [37], the major disadvantage of utilizing semantics and inference to facilitate resource allocation is reflected in the degraded system performance. However,

both authors state that the performance degradation is reasonable because the SERA framework deals with long-running tasks which are not frequently assigned or reassigned. In a task planning domain [51], authors explore how semantic description of environments, objects, and tasks, can be used to improve task planning in complex scenarios where a robot executes tasks on objects in an unstructured environment with a great number of objects. [51] outperforms the other solutions in the task planning domain because semantics, in particular inference, account for reducing a search space and thus faster yields potential solutions.

3.2.4 Space-Based Task Allocation

A framework described in [84] is a pioneer in utilizing the space-based paradigm for task allocation. It describes tasks which build a mission stored (advertised) in a centralized shared space accessible to all agents. On the other hand, agents fetch task information from the shared space, i.e., a blackboard, and based on their capabilities calculate their ability to solve the mission. A blackboard approach is complemented with Contract Net protocol [129]. Thus, this is one of the earliest work on distributed task allocation in multi-agent system. In [10] authors introduced the task allocation framework that utilizes Linda-based tuple space as a communication infrastructure and semantic technology for overcoming heterogeneity issues. Authors decided to use space-based communication paradigm because it decouples agents and thus scales well when the number of agents increases. Moreover, the framework introduces the administrator agents responsible for managing a centralized space and for semantic matching. Managing a space includes publishing a new task using OWL-T template language, deleting a tuple, and organising the space. Semantic matching is performed by comparing a semantic template that a requesting agent sent to task repository, i.e., space. If the match is positive, the requesting agent is able to execute a task. Otherwise, the agent can try with another space or create its own place and publish its request there. Moreover, there is also a provider ontology agent which provides and maintains an ontology used to derive OWL-T task templates. This approach enables distributed agents to collaborate without needing to know each other.

[77] proposes the generic (SILBA) framework based on the SBC paradigm for accommodating different load balancing algorithms. In load balancing the notion of agent's mobility is utilized to migrate (reassign) tasks from one server to another. The arbiter pattern in SILBA is responsible for load balancing, i.e., it redirects the load between the load spaces of different local nodes. In particular, the arbiter agents query the load of a local load space and decide about re-distribution of work. The real distribution, i.e., a task transfer, is performed by IN and OUT agents that read routing information from allocation space and pull, respectively push, work from/to another node in a network.

3.2.5 Coalition-Based Task Allocation

Coalitions are dynamically formed teams, i.e., groups of robots, which stay close to each other until all tasks are executed. Initially, teams are formed based on spatial proximity

utilizing hierarchical clustering to reduce the time robots need to move to reach other teams [109]. Although the robots are required to maintain a close proximity to other teammates, they do not necessarily cooperate on the same tasks. The framework allows a robot with a low utility to migrate to another team where the similar robots have higher utility rates. To successfully realize a transfer from one to another team, shared knowledge between teams is necessary. Therefore, in [109] each team periodically shares a summary of the utilities of its robots with other teams. The resource allocation protocol of the TRACE [40] framework periodically reallocates marketable agents to different organizations in accordance with their demands. Each organization consists of a set of permanent agents which are lastingly assigned to the organization, a set of marketable agents which can be hired by other organizations, and the resource management agent responsible for renting labour. At the beginning of each reorganization cycle, buyers, i.e., organizations that wish to acquire additional labour, and sellers, i.e., organizations that wish to sell their agents, place their bids on the market. As a result, the number of agents in an organization changes, the distribution of domain knowledge, as well as the communication structure. [16] proposes two different methods for building a robot coalition. First is the sequential method which aims for producing an online solution for task assignment and the second is holistic method that does task assignment at the beginning and thus requires the information of all tasks in the system. However, the proposed solution lacks the mechanism for reassigning tasks between coalitions.

[80] describe a solution that leverages the lack of social skills among simple autonomous robots, e.g., cleaning robots. The idea is to have software that ensures cooperation among robots without changing their hardware to provide new functionalities. Thus, the authors proposed to utilize intelligent agents, as representatives of robot units, capable to perform social interaction and team formation. As a result, each robot has a corresponding agent in the system. There is as well a user acting as a central manager who decomposes tasks, assigns, and reassigns them in case of failures, and monitors the task execution. The layered control approach retains the same level of autonomy a robot had before, because it does not interfere with the robot's internal algorithms utilized during the task execution.

The main challenge in coalitions is the group size of robots that work together. [71] conducted experiments to determine the relationship between group size and efficiency. The experiments showed that the efficiency is highest when the number of robots in a group remains below 9. When there are more than 9 robots in a group, efficiency decreases due to the interferences between robots. Moreover, [123] utilizes a network flow optimization model in task allocation where a group of small air vehicles are in charge of wide area search munitions. The disadvantage of the presented model is in that each vehicle can only have one task assigned at a time resulting in frequent task assignment processes. [126] utilizes the Set Covering Problem to allocate tasks to groups of agents. In order to decrease extensive communication between agents in a group, the algorithm introduces restriction which limits the number of agents in a group. It prefers small-sized coalitions over larger coalitions. The algorithm proposed in [128] divided the task allocation problem in 4 phase: (1) task selection, (2) resource negotiation,

(3) coalition formation, and (4) task execution. After a task is assigned to an agent with insufficient capabilities to execute it, the agent negotiates resource allocation with neighbours and forms a coalition accordingly. Similar to [142], the problem is in excessive communication due to the lack of shared knowledge. This results in task negotiation between agents that do not have complementing capabilities and thus cannot form a coalition.

3.2.6 Alternative Task Allocation in Multi-Robot Systems

This section introduces frameworks for multi-robot systems which practice some other approaches for task allocation than those described above. [81] describes a dynamic task allocation mechanism where robots utilize local observations of the environment to decide their task assignments. This approach is named task allocation through utilizing emergent coordination and it is perceived at the system encompassing individual robots which coordinate their actions based on local sensing information and local interactions. On the one hand, these systems are more scalable, robust, and introduce high parallelism. On the other hand, they are difficult to design, solutions are sub-optimal, and predictive analysis of expected performance is challenging. Due to the limited sensing capabilities, robots in [81] cannot acquire global information about the system and a surrounding environment, and thus they have an internal task state where they store their observations from the environment. The robots consult these observations periodically and update their task state according to some transition function. The goal of the framework is to efficiently assign emerging tasks under the limited communication possibilities and the lack of global knowledge.

[18] deals with the task allocation in distributed data processing. Distributed entities are processors with computational power while tasks are software modules that have to be executed. Similar to our approach, the introduced concept tries to decrease communication between distributed entities by allowing complex entities, i.e., entities with more computational power, to prefer complex tasks. To achieve this, the approach uses a graph theoretic approach. [125] utilizes Genetic Algorithm to address task scheduling in grid computing. The task allocation algorithm optimizes two objectives, it minimizes the completition time and the economic cost. [33] developed energy-aware strategies for adaptive task allocation in energy harvesting wireless sensor networks. The task allocation algorithm utilizes Genetic Algorithm to address the energy prediction for each node. The main objective is to maximize the fairness in energy-driven task mapping, i.e., energy balancing. When a wireless node receives a task, it adapts the task execution time according to the available energy. Consequently, if the node is in low-energy state, it will not collaborate with the other nodes neither will reassign the task.

[144] investigates how the task allocation process can be utilized for a path planning algorithm to accommodate dynamically changing task schedules. Each task is represented as a waypoint for the robot to navigate to and the tasks arrive dynamically in the environment. A new waypoint can emerge when a robot discovers an obstacle while traveling to its current goal and thus is required to make a detour around the obstacle. The robot's path planning algorithm, D* [45], then generates the optimal plan to get to

the next waypoint. [66] develops heuristic method for the task allocation and collision-free path planning for three robots working in a shared environment. A Genetic Algorithm had been used for task allocation, and A* for path planning. The system operates under the assumption that all tasks are known in advance and only one robot is required to execute one task. Due to the centralized mission planning, distributed robots are not able to collaborate and exchange tasks. The system [98] performs task assignment and scheduling given the number of aerial robots. Both, task assignment and scheduling utilize heuristics to calculate a strategy for tasks execution. Since the calculated strategy cannot be altered during the mission, the collaboration between robots is not supported.

[119] addresses the task allocation problem in extreme teams which have the following properties: (1) the dynamic environment causes tasks to appear and disappear, (2) an agent may perform multiple tasks, (3) agents have overlapping functionalities, and (4) inter-task requirements such as simultaneous execution requirements. The proposed task allocation algorithm is built on Distributed Constraint Optimization (DCOP) [92] algorithm and is called LA-DCOP (Low-communication Approximate DCOP). The main improvement introduced by LA-DCOP over DCOP is the use of tokens to represent tasks which further enables the minimization of the communication overhead. By accepting a potential token, an agent confirms that it will perform the task once the interdependencies have been satisfied [119]. Meanwhile, the agent can perform other tasks. [63] introduces a robotic fleet responsible for delivering materials inside a hospital. The task allocation mechanism assigns a material to a corresponding robot based on the exhaustive searching for the shortest path of all robots in the fleet. The algorithm presumes that the robots only have one skill, i.e., ability to transfer a material between two points, and that they do not collaborate for task execution.

[147] addresses the group role assignment problem and proposes an efficient algorithm based on the Kuhn-Munkers (K-M) algorithm. The role assignment problem has three steps: agent evaluation, group role assignment, and role transfer. Agent evaluation validates the agent's capabilities and resources, and based on that, rates its qualification for a role. Group role assignment instantiates a group by assigning roles to its members. Role transfer reassigns roles to agents. [147] contributes by proposing a practical solution based on the K-M algorithm for the group role assignment.

3.2.7 Environment-based Task Allocation

This section addresses a problem of robots spatial interference which is perceived as a key stumbling block on the way to efficient task allocation in robotic fleets. [118] introduces a distributed algorithm for task allocation in multi agent systems where tasks and agents are dispersed in a two-dimensional space. The algorithm is based on computational geometry technique, i.e., Delaunay triangulation [24], which tries to isolate each agent and thus enables it to base its decision solely on a small set of adjacent task and agent nodes. In contrast to the traditional task allocation approaches focused mainly on negotiation or market strategies such as contracts or auctions that require substantial amount of communication, algorithm [118], due to the local decisions, reduces the communication costs significantly. Work described in [121] studies a territorial approach to the task

allocation where robots are assigned exclusive working areas that can be dynamically resized if one of the robots fails completing a task. Reflecting on the experiments conducted in [121], the authors concluded that the larger the number of robots working in the same global workspace area, the greater the interference and the uncertainty related to the time required for task execution. Moreover, the task coverage algorithms presented in [17] and [70] utilize the Boustrophedon cellular decomposition approach, Figure 3.2 a), for partitioning the robots workspace, i.e., an environment is divided in multiple cells whose union form the whole environment. A cell is a minimal partitioning unit able to host one robot at time. In [70] the robots are initially distributed in a space where each robot is allocated to a cell that it shall cover. During the operation, robots incrementally build an adjacency graph representing the operating environment and share their knowledge, i.e., the constructed graph, with others. The approach presented in [17] exploits a geometric structure which is a union of nonintersecting rectangular regions that together compose the working environment. Each region is termed a cell and in each cell a coverage path is a simple back-and-forth motion. The approach is validated on mobile robots performing various experiments described in [17].

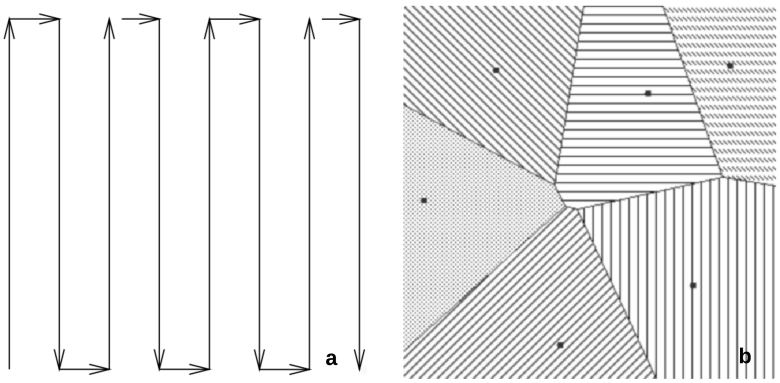

Figure 3.2: a) Boustrophedon cellular decomposition [70], b) Voronoi diagram

[62] develop a dynamic partitioning algorithm which assigns subareas to robots during the mission. A working area is divided in polygons allocated to robots that perform a cleaning task in a selected polygon. Since multiple robots may want to allocate the same polygon, robots have to cooperate to avoid collisions and double polygon allocations. Before a robot starts to allocate a polygon, it calculates an importance value for the polygon considering how far is the new polygon from the ones already allocated to the robot. After the importance value is calculated, it is communicated with other robots using Contract Net Protocol. However, a disadvantage is that the robots are not always in a communication range of other and thus cannot always perform a negotiation process with all interested parties. Thus, it could happen that some polygons are processed more than once. The authors argue that this dynamic approach is more flexible than the static one because in a case of a robot's failure, other robots dynamically take over its work. In addition to the Boustrophedon cellular decomposition, [145] and [79] utilize Voronoi

diagrams, Figure 3.2 b), as another technique for space partitioning which decouples robots sharing the working environment. Voronoi diagrams will be explained in detail in Section 6.3.

3.2.8 Comparison of Task Allocation Approaches

Table 3.4 compares 43 task allocation frameworks reviewed in the above sections with respect to the following criteria: (1) **Dynamic tasks** denotes whether a framework supports dynamically arriving tasks or only static tasks known in advance, (2) **Reassignment** describes capability to reallocate a task from one to another robot during a mission runtime, (3) **Allocate N tasks** indicates that a task allocation framework can in 1 run allocate more than 1 task to a robot, (4) **Collaboration** announces that two or more robots can work on a same task, e.g., each robot executes one part of a complex task, and (5) **Shared knowledge** means that each robot in a fleet is aware of the capabilities the other robots in the fleet have. The following task allocation frameworks ([51], [101], [71], [125], [147], [121], [17]) are not listed in Table 3.4 because they deal with specific use cases, e.g., path planning, area coverage, which neither encompass software agents nor mobile robots and thus cannot be compared against selected criteria. Task allocation frameworks are clustered based on the criteria, i.e., features, they provide.

Table 3.4 shows that majority of reviewed task allocation frameworks support dynamic arrival of tasks during a mission execution, i.e, 33 out of 43 frameworks. 8 task allocation frameworks support only dynamic arrival of tasks, while 6 more support one additional feature, i.e., collaboration. This feature is particularly important when a robotic fleet operates in unstructured environment where not all tasks are known in advance, i.e., before a mission begins. Since in this thesis we address the precision agriculture scenario which subsumes operation in unstructured environments, it would be beneficial to consider this feature in our SKIM coordination framework. 17 frameworks provide the capability to reassign already allocated tasks. Similarly, also 17 frameworks provide the collaboration capabilities between robots in a fleet. Majority of frameworks that support task reallocation or robots' collaboration, also support dynamic arrival of tasks. The system robustness, i.e., ability to proceed with the mission execution when a robot fails to execute a task, can be realized by means of dynamic task reallocation. The capability to cooperate on complex tasks, is inevitable factor to address in multi-robot systems. Thus, these two factors should be cornerstone in the SKIM. To strengthen the system robustness, a robot in a fleet should be able to proceed with a mission execution even if it looses a connection with either a central management system or with other robots in the fleet. To support this, a robot should have a plan that consists of allocated tasks, i.e., a task allocation framework should support allocation of multiple tasks to one robot. This feature is practised by 20 reviewed frameworks. On the other hand, only 3 frameworks provide shared knowledge between robots in a fleet. Shared knowledge facilitates collaboration between robots in a way that it enables a robot A to find an appropriate collaboration partner (robot B), i.e., robot B which offers particular skills that robot A needs to complete a task. Due to that, each task allocation approach that supports shared knowledge, support the collaboration as well.

Table 3.4: Comparison of task allocation frameworks

Approach	Dynamic tasks	Reassignment	Allocate N tasks	Collaboration	Shared knowledge
[23],[88],[48],[114],[86],[132],[16],[123]	✓	-	-	-	-
[150],[35],[37],[34],[154],[80],[81]	✓	✓	✓	-	-
[105],[10],[84],[126],[18],[118]	✓	-	-	✓	-
[40],[33],[144],[63]	✓	-	✓	-	-
[149],[13]	-	✓	✓	✓	-
[77],[128]	✓	✓	-	-	-
[66],[98]	-	-	✓	-	-
[146]	-	✓	✓	-	-
[64]	✓	✓	✓	✓	-
[142]	-	-	-	✓	-
[7]	-	✓	-	-	-
[140]	-	✓	-	✓	-
[53]	✓	✓	-	✓	✓
[109]	✓	-	-	✓	✓
[70]	✓	-	✓	✓	✓
[15]	✓	✓	-	✓	-
[119]	✓	-	✓	✓	-
[62]	-	-	✓	✓	-
[83]	-	-	-	-	-
Total	33	17	20	17	3

Only 15 frameworks provide three or more features, and only 3 of them provide four features. Although [83] does not provide any of the selected features, it is selected because of the interesting implicit coordination strategy applied for dealing with colliding tasks. To the best of my knowledge, there is no existing task allocation framework that supports all five features. Therefore, the SKIM coordination framework will provide a task allocation model that implements all five features.

3.3 Adaptive Autonomy in Mixed Human-Robot Teams

This section reviews the frameworks that model the human-robot interaction (HRI) by means of adaptive autonomy in missions that comprise mixed teams of humans and robots. The purpose of HRI technology is in the short-term to facilitate efficient automation of processes and in the long-term to enable operation of teams in which humans and robots jointly perform complex tasks and missions. Adaptive autonomy is the term used

to refer to autonomy of robotic units designed for flexible change, i.e., the autonomy may increase or decrease depending on the state of the system, the environment, and the requirements of the human operator [56]. In this way development of autonomous capabilities of robotic systems can be better matched to the needs of human operators and coordination among the robots in a fleet and human team members in complex unstructured environments. Humans can interact with robots in different roles, including a mission planner, a remote operator, an in-field operator for maintenance and diagnosis tasks, or a close collaborator in joint missions. The spatial and temporal dimension of human-robot interaction depends on tasks and application fields: (1) remote operation has challenges related to potentially long delays in communication, (2) interaction in the same environment is challenging because of the safety issues, and (3) multi-operator interaction is challenging because of the requirements of coordination and split control [27].

Communication between entities is fundamental for achieving a mission goal. Explicit communication is limited in terms of fault tolerance and reliability, because it is typically conducted in volatile and unstructured environments which tend to partition all members of the robot team [111]. Consequently, robots have to deal with a dynamic network topology where it is very difficult to predict future moves. In most of the existing research, the availability of a communication link between two robots is always assumed [113]. However, this assumption does not hold in unstructured and dynamic environments as the agricultural land is. To remedy communication uncertainties and constraints imposed by a volatile environment, the work in [31] proposes a data collection function within the robotic fleet which is targeted to enrich an operator in-field understanding of the mission with specific additional information beyond what he/she can directly perceive by supervising the mission. This framework combines data from three different sources: (1) data retrieved directly from robots utilizing HRI system, (2) visual perception of the in-field situation, and (3) network data obtained through the network monitoring system. Utilizing fused data, the user increases the overall mission efficiency by reducing the robots outages, i.e., a robot being disconnected from a fleet due to communication loss, and increasing their utilization based on the network information which helps the user to predict and prevent remote robots of losing a connection and thus being out of service.

The safety of human interactions with robots becomes a key issue as robots are used in interactive and uncertain environments. Interaction between humans and robotic fleets in different application fields ranges from operation to teaming in joint tasks. In the precision agriculture applications the interactions among humans and autonomous robotic fleet predominantly focus on interactions for a fleet operation, including in-field interactions [27] where the safety issue is the main parameter to address. To detect safety critical situations robots are equipped with person or animal detection system. However, to cover all possible situations of human-robot interaction, the high-level decision making system in [27] is distributed among the central platform at the base station and the user in a field. As a consequence of problem detection, the mitigating reactions include the robot autonomous behaviour and the human-assisted decision making process in the field. These redundant decision-making systems prevent the hazardous situations

and increase mission efficiency. To allow multiple humans to participate in the team at any level of control, [65] proposes a regulatory mechanism based on the regulations. A regulation is based on authorizations and obligations: authorization policies specify which actions an actor or a set of actors is allowed or not allowed to perform in a given context and obligation policies specify actions that an actor or a set of actors is required or not required to perform.

The existing work further identifies challenges in designing multi-human multi-robot interaction mechanisms in relation to scalability [133] and coordination [41]. Coordination is an issue often underestimated in the design of autonomous robotic functions [143]. Therefore, in the design of the human-robot interactions, particularly in very demanding multi-robot systems, robotic and cognitive requirements (scalability, avoiding information overload) have to be jointly accounted for [12]. To address uncertainties emerging due to the unpredictable situations, in [20] authors use multi-human/multi-robot interactions where the agents ask humans for help when they meet unpredictable situations. To model the behaviour in those unpredictable situations, [20] introduces a model called HHP-MDP (Human Help Provider - Markov Decision Processes) which describes how a controller can handle different types of requests that agents send to the set of humans. Moveover, the human can react on a request either by teleoperating a requesting agent, or by giving recommendations, i.e., giving the agent a set of subgoals to be reached before the main goal. This approach is known as the goal-biased technique. [47] model the multi-human/multi-robot interactions with the mixed-initiative planning approach which is to continuously coordinate, integrate, and monitor the operator interventions and decisions with respect to the concurrent functional activities. Due to the number of hardcoded procedures, it is not trivial to extend or modify the presented HRI model. Work presented in [139] inspects the effect of the number of controlled robots on performance of an urban search and rescue (USAR) task. The outcome of this study revealed that the human workload increased monotonically with the number of operating robots, while at the same time performance per robot decreased because a robot is neglected for a longer period of time. Although all the robots are autonomous, they perform better when teleoperated. The study suggests a limit of $8 - 12$ robots for direct human control. This implies that not all robots can be teleoperated at the same time thus some robots will be neglected. However, it is worth to notice that the observed robots did not have collaboration and cooperation activities among themselves. [141] presented a framework which increases human capacity for control by removing the independence among robots and thus allowing them to cooperate. Robots cooperation is complemented with a human control resulting in the model of adaptive autonomy. The experiments were performed in USAR environment to measure the trust a human operator has in the autonomy model. The results showed that autonomy helped the operators to explore more area and find more victims.

[124] investigates challenges in heterogeneous teams where humans operate multiple robots. [124], as well as [31], point out a human awareness of the operating fleet as a major issue that needs an attention when designing adaptive autonomy models. [124] introduces two models of adaptive autonomy: (1) System-Initiative Sliding Autonomy (SISA), and

(2) Mixed-Initiative Sliding Autonomy (MISA). SISA allows the operator to interfere with the system only when explicitly asked to do so by an autonomous robot. Whereas MISA allows a human operator to intervene with the system at any time. To compare these two representatives of adaptive autonomy, the authors conducted an experiment where they compared SISA and MISA with teleoperation (human is in a complete control over the robotic fleet) and pure autonomy (no human interaction and complete autonomy of the fleet). The results revealed that the autonomous system is consistently faster, but less reliable than the teleoperated approach. In SISA and MISA the execution time of a mission is close to that of the autonomous system. Regarding the human workload, it is the highest in the teleoperated approach, while in SISA and MISA it is task- and user-dependent, i.e., depends among the other factors on the user's experience. [124] concluded that adaptive autonomy approaches may improve the multi-agent team's reliability without compromising efficiency. [148] presented a model of human-robot cooperative control that helps to improve the resilience of the human-machine system by introducing multiple levels of autonomy. Presented model includes four levels of autonomy deployed on a robot. In a dynamic context, the human-robot system must be able to select the most appropriate autonomy level. Conducted experiments showed that the automatic selection of an autonomy level did not yield expected improvement with respect to the mission efficiency (mission execution time). The authors argue that this could be due to a lack of experience of the operators with the system.

Multiagent Adjustable Autonomy Framework (MAAF) [49] addresses the challenges posed by exploiting the unique capabilities of heterogeneous teams composed of a mixture of robots, agents and humans. The challenges pertain to improving the safety, efficiency, reliability, etc. MAAF uses a technique which introduces the notion of global goals that all team members have to be aware of, i.e., team-oriented programming. The basic idea is to provide high-level team plans and their decomposition into sub-team plans and roles. After that, those plans and roles are allocated to agents and then the framework facilitates the reasoning about whether to act autonomously or pass control to other team members. Furthermore, the notion of adjustable autonomy uses transfer-of-control strategies to allow the best teammate to have autonomy over a decision at a given time, e.g., if the agent A cannot make decision, it passes the control over to the agent B, if the agent B cannot make decision, it passes the control over to a human. Similar to MAAF, [120] introduces the notion of transfer-of-control strategy from an agent to a human or to another agent. However, [120] emphasizes that interrupting a human user has very high costs and thus such transfer of control has to be minimized. To minimize the transfer, [120] built an algorithm which uses MDP to select an optimal strategy with regard to given costs of individuals and teams.

[55] presents a model for adaptive autonomy that takes into account the cognitive task load (CTL) of a human team member and the coordination costs of switching to a new task while deciding the level of autonomy of a robot. The introduced model enables dynamic task allocation between humans and robots. A human operator can interact with the system in two ways: either by executing a task in parallel with a robot, or to react on a robot request for help. This is achieved by representing each task with all

possible levels of automation, i.e., different levels of autonomy can be modeled as several mutually exclusive subtasks. Thus, different subtasks can be dynamically allocated to different actors in the system.

Table 3.5 compares the adaptive autonomy frameworks reviewed in this section with respect to the following criteria: (1) **N operators** denotes whether the reviewed framework supports more than 1 human operator, (2) **N controlled robots** indicates whether one human operator can control more than 1 robot, (3) **N interaction levels** describes the capability of a human operator to perform supplementary tasks, e.g., a task allocation, in addition to the default interaction mechanisms, (4) **Active | Reactive** shows whether a human operator can only react on a robot's request or can also initiate one, and (5) **Flexible behaviour** denotes the capability to easily change a human's behaviour in case the human behaviour is modeled with an algorithm. When a human operator is represented by a real human, the behaviour is anyhow flexible. To be able to easily change a human's behaviour subsumes that the implemented algorithm does not have to be altered. Adaptive autonomy frameworks are clustered based on the criteria, i.e., features, they support.

Table 3.5: Comparison of adaptive autonomy frameworks

Framework	N operators	N controlled robots	N interaction levels	Active | Reactive	Flexible behaviour
[139],[141]	-	✓	-	Reactive	✓
[120],[148]	-	-	-	Reactive	✓
[20]	✓	✓	-	Reactive	-
[47]	✓	✓	-	Both	-
[124]	-	✓	-	Both	-
[49]	✓	✓	-	Reactive	✓
[55]	-	-	✓	Both	✓

Table 3.5 shows that majority of reviewed adaptive autonomy frameworks do not support multiple human operators. This is due to the coordination complexities between multiple human operators, e.g., human operators have to coordination who will process next robot's request. On the other hand, in most frameworks a human operator is responsible for controlling and interacting with multiple robots in a fleet. Consequently, the human operator can easily become overloaded which can influence a mission efficiency. Only one reviewed framework [55] supports more than one level of a human interaction with a fleet. A human operator can interact with the system either by executing a task in parallel with a robot, or to react on a robot request for help. All other reviewed frameworks support only interaction for providing help during a task execution, e.g.,

teleoperation. Majority of reviewed frameworks provision only a simple, i.e., reactive, mechanism to react on robot's requests. Only 3 frameworks enable a human operator to initiate an interaction with a robot. Finally, a flexible behaviour is present only in the systems where a human operator is represented by a real human. In other 3 framework a human behaviour is modeled and thus cannot be altered without altering the algorithm. To the best of my knowledge, there is no existing adaptive autonomy framework that supports all at least four features. Therefore, the SKIM coordination framework will provide an adaptive autonomy model that implements all features except the multiple human operators.

3.4 Summary

This section summarizes the related work analysis. Due to the three different categories of the related work: (1) coordination middlewares for robotic fleets, (2) task allocation models, and (3) adaptive autonomy in human-robot teams, this summary is structured accordingly.

Designing middleware for application in complex robotic systems requires that many challenges related to the fleet operation are addressed. This thesis classified challenges in the three domains of concern: (1) general fleet realization, (2) environmentally entailed challenges, and (3) task specific challenges. The latter two are specific for the fleet application. The benefits of some of the design options are already well understood: a solution that enables decentralized control of autonomous and heterogeneous robots has advantages over the one which lacks those features. Specific tasks and environments introduce a set of additional design constraints that are critical for operation in missions with high uncertainty. The review of existing middlewares revealed that the semantic-based task and service description is an approach that only a small number of middlewares support. This motivates the work presented in this thesis which, in particular, focuses on extension of a semantic solution that combines space-based middleware with semantic modeling, processing and reasoning.

The use of semantics for resource and task modeling is recognized as a prominent approach. The benefit of using semantics for a task allocation is twofold: (1) the developed ontology provides uniform description of heterogeneous and distributed resources, and (2) semantically annotating tasks and services yields a more accurate matching and thus results in more efficient utilization of resources. The former enables all heterogeneous robots to execute tasks produced by a third party because all entities conform to the introduced ontology. Moreover, heterogeneous robots can even generate ad-hoc tasks which are executable by other robots. The latter is expected to be a basis for the matching algorithm that satisfies exact, subsume, and plugin matching degrees which are well established guidelines in reviewed literature.

Operating a robotic fleet is a cognitively demanding task that requires an efficient user-fleet interface for decision-support in control and monitoring, diagnosis, problem detection and resolution, complementing the autonomous decision making that the robotic fleet units are capable of. Particularly for applications of robotic fleets in open

uncontrolled environments, the role of the human-operator needs to be carefully defined and supported, taking into account uncertainties in all phases of the system operation, including also human potential in resolving and equally creating critical situations. Some of the stated challenges are addressed in this thesis by utilizing an ontology to model human-robot interactions.

CHAPTER 4

Ontology-based Approach for Task Allocation

Multi-agent systems heavily rely on dynamic and intensive collaboration to provide interoperability between heterogeneous autonomous robots, i.e., agents. In this thesis, the notion of interoperability is limited to the model of a task allocation. Achieving an efficient task allocation, and thus interoperability, is a challenging task which requires to address the following issues: (1) heterogeneity of resources, i.e., robots and tasks to execute, (2) coordination which manages the possible interactions between involved parties, (3) dynamic resource discovery and utilization, (4) sharing knowledge within a robotic fleet, and (5) domain independent and flexible collaboration approach.

To address these challenges, this section utilizes ontologies to overcome the semantic discrepancies inherited by integrating heterogeneous components, i.e., to specify and conceptualize knowledge using a formal description that is machine-readable, shareable, and enables reasoning to infer new information. Due to the interoperability support between heterogeneous agents as well as reusability, ontologies enjoy significant support in multi-agent community [112]. Utilizing ontologies formalizes modeling of domain knowledge and simplifies the design of coordination patterns between robots which otherwise is a complex and cumbersome task. This thesis applies the Model-Driven Architecture (MDA) approach that separates the specification of system functionality and implementation. MDA encourages explicit modelling of heterogeneous resources and a clear separation between models and implementation [96].

This chapter begins by introducing the MDA approach and describing the relevant concepts. After that it introduces a high-level and domain independent ontology for modelling resources, i.e., tasks and robots, and operating environment and describes how this maps to the MDA approach. This is followed by the ontological model of a configuration, i.e., a robotic fleet, which executes the modeled tasks. Moreover, described is a customizable ontology defining coordination policies between heterogeneous team

members. Finally, the chapter is concluded with an overview of benefits that the integration of ontologies and MDA brings.

4.1 The Concept of Model-Driven Architecture

Model-Driven Architecture (MDA), defined by the Object Management Group (OMG), is an approach to application design and implementation which encourages efficient use of system models in the development process as well as the reuse of those models when creating families of systems. Models allow engineers to focus on important details essential to reason about the system. Models can be used to predict system qualities, to reason about specific properties when aspects of the system are changed, and as a precursor to implementing the physical system [91].

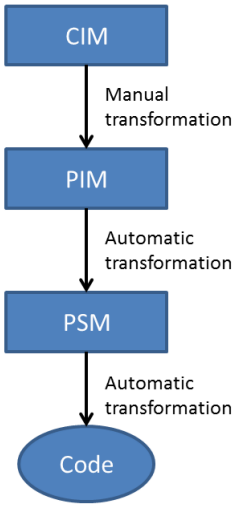

Figure 4.1: MDA approach

As already mentioned, the main goal of MDA is the separation of system functionality specification and implementation [6]. Figure 4.1 shows the structure of the MDA approach and the relation of different models: (1) Computation Independent Model (CIM), (2) Platform Independent Model (PIM), (3) Platform Specific Model (PSM), and (4) software code. CIM pertains to the construction of the models in a formal way mainly using Unified Modeling Language (UML) [90]. The system requirements are extracted and described in CIM, which is then refined into the PIM. This refinement is called transformation and it is usually done by hand. PIM separates system functionality from the implementation and thus keeps the specification independent of the platform the system may be deployed to. PSM is the result of the PIM transformation to the target platform. PSM describes the separation of system functionality on a specific technology. Later, PSM is transformed to code which will be executed on that platform.

The following sections introduce and describe multiple ontologies used as formal models to capture the domain knowledge and to specify system functionality.

4.2 Ontological Description of Heterogeneous Resources

This section addresses the challenge of heterogeneous resources which are unavoidable issue in multi-agent system. The challenge is explicitly addressed by utilizing ontologies to formally model the shared knowledge between heterogeneous components.

The proposed system ontology reuses the concepts from ontologies described in [86] and [106] which partially describe our system, but are still incomplete. Known existing ontologies, for this and related use cases, lack the support for automated mapping between tasks and robots, i.e., they do not model an explicit relation between these two. Therefore, the following concepts were required to make these ontologies suitable and intuitive for task allocation. The basic idea of the proposed model is to formally specify that *Tasks* can be performed by *Robots* providing different *Skills*, and that each *Task* can be associated with a *Robot* via *Skill*. Moreover, each *Task* requires a specific amount of resources, e.g., a spraying liquid, for execution which has to be provided by a robot which participates in the task allocation process. This approach will be later in this thesis utilized in the design and implementation of various task allocation approaches.

4.2.1 Describing Task Requirements

Common to all missions that include either single or multiple robots, is to execute a set of tasks in a given environment under some constraints [111]. Missions consist of multiple tasks with different requirements, constraints, dependencies, locations in an environment, etc. Therefore, the ontology illustrated in Figure 4.2 is modeled as a general ontology which formally describes tasks and due to its generality, it is domain independent. However, the illustrated ontology is derived from the agricultural use case. Domain independency is shown later in this thesis by modeling different scenarios using the same task model, i.e., ontology.

The ontology illustrated in Figure 4.2 models the class *Task* with three properties. First property, *hasCentralPoint*, relates a task to a $2D$ location in an environment, i.e., a place where the task resides. Second property, *needsSkill*, describes skills required by the task, i.e., which skills a robot must have to execute the task. One task has to have at least one skill, i.e., an individual of type *Skill*. Finally, the *needsAmount* property refers to the amount of resource a robot must have to execute the task. To keep the example simple, some properties are omitted from the Figure 4.2, i.e., *hasId* and *hasNumberOfSkills*.

Listing 4.1 shows an excerpt of the ontology defined in OWL (see Figure 4.2) which presents the definition of class *Task*. Class *Task* has three necessary and sufficient conditions defined as *Restrictions*. These three conditions ensure that an individual which has exactly one value for *hasCentralPoint* property and at least one value for *needsSkill* and *needsAmount* properties is classified as a *Task*.

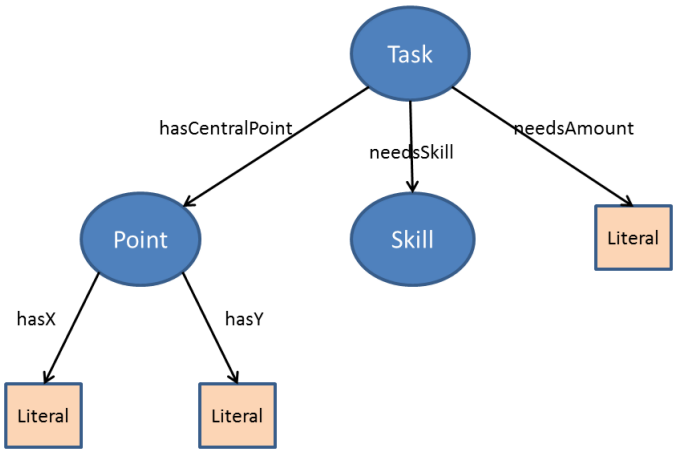

Figure 4.2: General ontology for describing tasks

Listing 4.1: Ontology excerpt - class *Task*

```
<owl:Class rdf:about="&multiagent;Task">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="&multiagent;hasCentralPoint"/>
                    <owl:onClass rdf:resource="&multiagent;Point"/>
                    <owl:qualifiedCardinality rdf:datatype="&xsd;
                        nonNegativeInteger">1</owl:qualifiedCardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="&multiagent;needsSkill"/>
                    <owl:onClass rdf:resource="&multiagent;Skill"/>
                    <owl:minQualifiedCardinality rdf:datatype="&xsd;
                        nonNegativeInteger">1</owl:minQualifiedCardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="&multiagent;needsAmount"/>
                    <owl:onClass rdf:resource="&multiagent;Literal"/>
                    <owl:minQualifiedCardinality rdf:datatype="&xsd;
                        nonNegativeInteger">1</owl:minQualifiedCardinality>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

Table 4.1 presents an example for an instance of class *Task* with an Id = 24, a central point [x=89, y=47], two needed skills [Skill_1, Skill_2] where each skill needs 3 units of resource, e.g., one unit of resource could represent 100 liters of spraying liquid. Due to the generality of the presented ontology, Skill_1 and Skill_2 denote general, domain independent skills, which can be later substituted with real, domain dependent, skills. However, this will be demonstrated later in this thesis.

To execute an instance of class *Task* presented in Table 4.1, either one robot with two skills and an appropriate amount of resources can do that, or two robots, each with one corresponding skill and an appropriate amount of resources, have to collaborate. Both

Table 4.1: A task modeled using general ontology

Id	Central Point	Needs Skill	Needs Resource	Number of Skills
24	[x=89, y=47]	[Skill_1, Skill_2]	3	2

approaches will be examined later in this thesis.

4.2.2 Describing Robots' Capabilities

Ontology for modeling robots is illustrated in Figure 4.3. Although the robot ontology is closely related to the general task ontology from Figure 4.2, the reason why these two are separated is twofold: (1) to make a clear separation between modeling tasks in a specific mission (scenario) and modeling a robotic fleet to execute those tasks, and (2) usually the focus is on modeling tasks and thus it is avoided to clutter the general task ontology with robot classes and properties. However, it is strongly encouraged, and also the best practice, to reuse classes and properties for modeling both tasks and robots, respectively.

Same as the ontology in Figure 4.2, the ontology illustrated in Figure 4.3 is modeled as a general ontology which formally describes robots and due to the generality, it is domain independent.

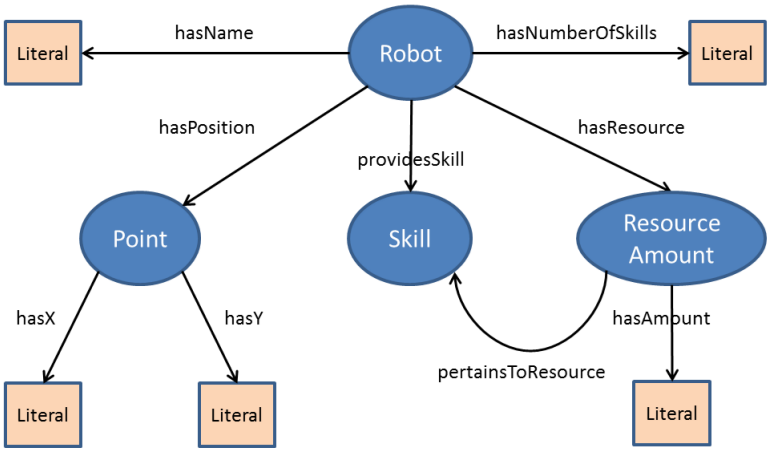

Figure 4.3: Ontological description of robots

The ontology illustrated in Figure 4.3 models the class *Robot* with three properties. First property, *hasPosition*, relates to a robot's 2D location in an environment, i.e., the place where the robot is at the current moment. This property corresponds to the property *hasCentralPoint* from the general task ontology. Second property, *providesSkill*, describes skills provided by the robot, i.e., the skills the robot can apply to execute a certain task. One robot has to have at least one skill, i.e., an individual of type *Skill*. Finally, the *hasResource* property refers to the object *ResourceAmount* which denotes

the amount of resources the robot has, i.e., property *hasAmount*, and to which skill this resource amount pertains to, i.e., property *pertainsToResource*. The property *hasAmount* corresponds to the property *needsAmount* from the general task ontology. Properties *hasName* and *hasNumberOfSkills* denote a robot's name and the number of different skills it provides, respectively. The ontology excerpt reflecting class *Robot* is similar to the one reflecting class *Task* in Section 4.2.1 and thus is omitted.

Table 4.2 presents an example instance of class *Robot* with name agent_2, location [x=6, y=6], two provided skills where the robot has 10 units of resource for the first skill, i.e., *skill_spraying*, and 20 units of resource for the second skill, i.e., *skill_flaming*. In contrast to the task presentation in Table 4.1, the robot modeled in Table 4.2 lacks generality in a sense of modeling *Skills*. To be more precise, the robot model has specific skills in contrast to the task model which has general skills that can be later specified.

Table 4.2: A robot modeled using robot ontology

Name	Position	Provides Skill	Has Resource	Number of Skills
Robot_2	[x=6, y=6]	[skill_spraying, skill_flaming]	[skill_spraying:10, skill_flaming:20]	2

The example robot modeled in Table 4.2 can execute two types of tasks under the assumption that it has a sufficient amount of resources: (1) a task which requires both skills the robot provides, and (2) a task that requires one of the skills the robot provides. Both approaches will be examined later in the thesis.

4.2.3 Mapping Ontology to Model-Driven Architecture

As previously presented in Section 4.1, the MDA presents several viewpoints, i.e., computational independent, platform independent, and platform specific. Following these considerations, presented is an MDA oriented approach for task allocation in multi-robot systems. The core component of the proposed task allocation approach focuses on modeling heterogeneous resources and thus is based on the ontologies described in Sections 4.2.1 and 4.2.2.

Similar to [96], [97], and [6], the proposed task allocation approach utilizes Semantic Web Technologies, i.e., an ontology, to describe system requirements and resource capabilities. On the other hand, UML can be used as well to construct the PIM [90]. In particular, the proposed approach uses ontologies for constructing the PIM. PIM, formally modeled with the general ontology presented in Figure 4.2, describes the tasks comprising a mission and at the same time hides the details necessary for a particular platform. The major aim at this level is to represent main properties of tasks without taking into account any specific technological details. This means that the task specification formally described with the ontology is independent from the platform which will execute those tasks.

After having a valid PIM, it can be transformed to PSM. PSM denotes the object-oriented model of tasks which are then transformed to the Java code. Validity check

can be automatically performed on the ontology-based PIM by using ontology-based reasoning. Transformation from PIM to PSM is done automatically using a software component in the following way: (1) the component extracts data from the ontology, i.e., task instances, (2) it prepares an existing object-oriented model of tasks, (3) it transform task instances from the ontology into the corresponding object-oriented mode, and finally (4) makes tasks mapped to the object-oriented model available for robots. The crucial thing in the transformation process is that the transformation component uses the same ontology, i.e., classes and properties, as the one used for constructing a PIM. Otherwise, the transformation component could misunderstand the meaning of an extracted property and consequently create an invalid task object for further execution. Both validity check and transformation are implemented in Java using Jena API [155] and Pellet reasoner [158].

A similar process is followed to create the PIM from the robot ontology presented in Figure 4.3. The same transformation procedure is utilized to generate PSM from PIM. The code generated from PSM denotes the robot configuration for executing tasks. These two are similar processes, but produce different outcomes. The first one generates tasks from the PIM, while the second one a robot configuration for executing those tasks. However, both PSMs are designated to run on the Java platform. Construction of CIM is out of the scope of this work.

Due to the platform-independent and high level approach that utilizes ontologies to model system resources, i.e., tasks and robots, an arbitrary number of tasks and robots in the system can be instantiated. Consequently, the introduced approach supports scalability in multi-robot systems.

4.3 Ontological Description of Operating Environment

This section demonstrates the generality and flexibility of the ontology-based task allocation approach complemented with MDA. Similar to the approach presented in Section 4.2, this section presents an ontology-based approach for modeling different environments where a robotic fleet strives to execute a set of tasks. Different operating environments can be presented as different scenarios where each scenario encompasses specific tasks to execute. Ontology-based scenario modeling is followed by ontology-based configuration modeling where a configuration encompasses different robots with different skills operating in a specific environment.

The framework developed in this thesis utilizes the scenario modeling approach for building relations between different scenarios and configurations of robotic fleets. In that way it is possible to select a nearly optimal configuration for every scenario and thus to increase the overall system efficiency.

4.3.1 Ontology-based Scenario Description

Having an ontology to describe a specific scenario where a robotic fleet will operate has several benefits. It allows: (1) to classify abundance of arbitrary tasks in a specific

scenario, (2) to help the system operator to better perceive and interpret a mission consisting of numerous tasks, (3) to help the system operator to select the most efficient configuration of a robotic fleet to execute those tasks, and (4) to utilize gained knowledge to tune up the parameters describing scenarios and thus increase the overall system efficiency.

The ontology illustrated in Figure 4.4 models the class *Abstract Scenario* which has three subclasses denoting specific scenarios. Each *Scenario* class has two properties: (1) *needsAmount*, and (2) *hasNumberOfSkills*. The former expects the number denoting the amount of resources a task has to require in order to belong to a specific scenario. This property is the same as the property *needsAmount* presented in the general ontology described in Figure 4.2. The latter denotes the number referring to the number of skills a task has to require in order to belong to a specific scenario. This property quantifies the number of *needsSkill* relations depicted in the general ontology described in Figure 4.2. This ontology reuses some concepts from the general ontology. For example, the task presented in Table 4.1 has two relations for the property *needsSkill* and thus has the value of property *hasNumberOfSkills* set to 2.

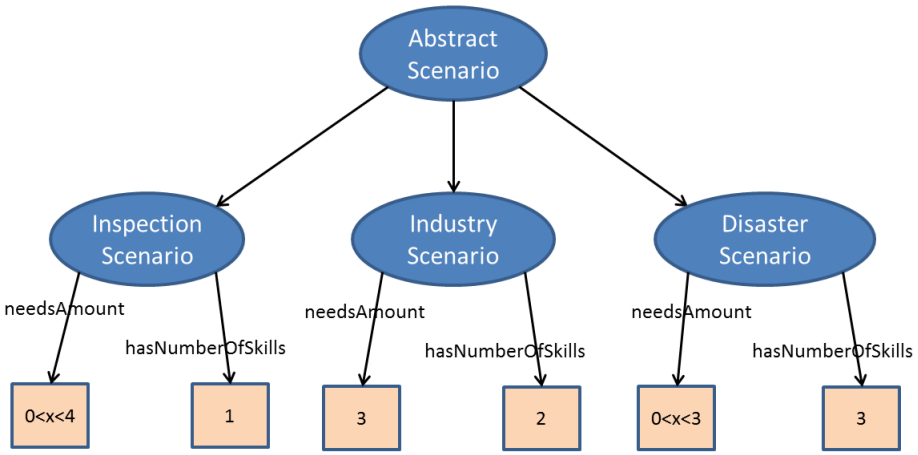

Figure 4.4: Ontological description of different scenarios

Inspection scenario is a representative of the scenario class which encompasses simple tasks, i.e., tasks requiring only one skill for complete execution. The class involves simple scenarios consisting of repetitive tasks, e.g., a surveillance mission. The amount of resources for a provided skill is an arbitrary number $0 < x < 4$. *Industry scenario* is the representative of a class including more complex tasks where multiple unique skills have to be applied and robots have to collaborate to execute tasks. *Disaster scenario* is the representative of a class composed of the most complex tasks requiring robots' collaboration and human interaction during the task's execution, e.g., Urban Search and Rescue (USAR). Since the introduced precision agriculture scenario encompasses robots which provide at most 3 different skills, the ontology in Figure 4.4 also models the scenarios which have at most 3 different skills. However, the ontological approach

is not limited only to those scenarios, i.e., to maximum 3 different skills per scenario. Rather, it can model any scenario with an arbitrary number of skills. Moreover, human interaction is not modeled in the ontology describing different scenarios, rather in the coordination ontology which will be presented in Section 4.7.

Listing 4.2 shows an excerpt of an ontology defined in OWL which presents definition of class *Industry scenario*. The excerpt reflects ontological description of a class *Industry scenario* illustrated in Figure 4.4 using OWL. In detail, it shows class *Industry scenario* which is a defined class with two necessary and sufficient conditions defined as *Restrictions*. These two conditions ensure that an individual which has exactly two skills (*hasNumberOfSkills* property) and each skill requires three units of resources (*needsAmount* property), is classified as the type of *Industry scenario* class.

Listing 4.2: Ontology excerpt - class *IndustryScenario*

```
<owl:Class rdf:about="&multiagent;IndustryScenario">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&multiagent;Task"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="&multiagent;hasNumberOfSkills"/>
                    <owl:hasValue rdf:datatype="&xsd;int">2</owl:hasValue>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="&multiagent;needsAmount"/>
                    <owl:hasValue rdf:datatype="&xsd;int">3</owl:hasValue>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&multiagent;AbstractScenario"/>
</owl:Class>
```

Due to the same properties, at first glance all three scenarios resemble each other. However, the key property which distinguishes one scenario from the others is *hasNumberOfSkills* which describes the task complexity. The notion of task's complexity, expressed as the number of required skills, i.e., *hasNumberOfSkills*, has twofold purpose: (1) to characterize the scenario class, and (2) to enable task-robot mapping during the task execution. As illustrated in Figure 4.4, the task complexity is responsible for modeling different scenarios. In addition to the task complexity, the property *needsAmount* also influences the model of a specific scenario, but not in the same measure as the task complexity does. Due to the generality of this approach, a new scenario could be easily modeled where the property *needsAmount* will be the key property to distinguish between scenarios. Later purpose of the task complexity, i.e., task-robot mapping, will be explained in Section 6.4.

4.3.2 Ontology-based Configuration

The advantages of having an ontology to describe a configuration, i.e., a robotic fleet with provided skills, are manifold: (1) bringing all heterogeneous robots in a fleet to the same operational level where they share a common understanding, (2) common understanding

enables shared knowledge and thus collaboration, and (3) ontology-based configuration clearly separates specification from an implementation.

The ontology utilized for constructing the configuration of the robotic fleet is illustrated in Figure 4.5. This ontology is an extension of the general ontology depicted in Figure 4.2 and the robot ontology shown in Figure 4.3. The class *Skill* presented in the configuration ontology in Figure 4.5 is same as the class *Skill* from the general and robot ontologies. Moreover, class *Robot* from the configuration ontology is same as the class *Robot* from the robot ontology. However, there are three subclasses defined in the configuration ontology which extend the class *Skill*. Those classes model specific skills provided by the robotic fleet described in a configuration. The configuration ontology reuses existing concepts and therefore is a specialization of the two already presented ontologies.

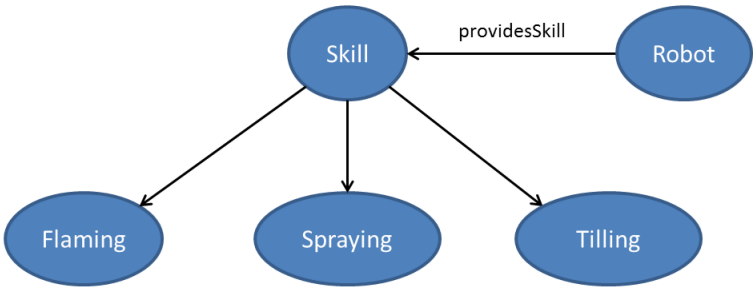

Figure 4.5: Ontological description of a configuration

Instances of the classes *Flaming*, *Spraying*, and *Tilling*, which model specific skills, are presented in the Table 4.3. Column *Name* denotes an instance name and the column *Skill* represents the class from the configuration ontology to which an individual belongs to, e.g., instance named *skill_spraying* belongs to the class *Spraying* defined in the configuration ontology. Relation belong to is defined as *type of* in semantic technology.

Table 4.3: An example configuration model using ontology - Skills

Name	Skill
skill_flaming	Flaming
skill_spraying	Spraying
skill_tilling	Tilling

Having a list of skills, as in Table 4.3, is just a prerequisite for constructing a configuration which consists of heterogeneous robots. Each robot in the configuration has to provide at least one skill, as it is modeled in robot and configuration ontologies. Thus, the instances of the type *Skill* are required.

After the instances of the type *Skill* are provided, the instances of type *Robot* can be constructed. Each robot instance is related to at least one skill instance using the

property *providesSkill* introduced in robot and configuration ontologies. Table 4.4 lists three robot instances with provided skills.

Table 4.4: A configuration model using ontology - Robots

Name	Position	Provides Skill	Has Resource	Number of Skills
Robot_1	[x=1, y=4]	[skill_spraying, skill_flaming]	[skill_spraying:10, skill_flaming:20]	2
Robot_2	[x=16, y=9]	[skill_tilling]	[skill_tilling:5]	1
Robot_3	[x=5, y=14]	[skill_flaming]	[skill_flaming:30]	1

Listing 4.3 shows an excerpt of an ontology defined in OWL which defines an individual named *Robot_1* from the configuration listed in Table 4.4. The individual is of types *NamedIndividual* and *Robot* and provides two skills, i.e., it has two values for *utilizesSkill* property. One skill is individual named *skill_flaming*, which is of type *Flaming*, and the other, named *skill_spraying*, is of type *Spraying*.

Listing 4.3: Ontology excerpt - individual *Robot_1*

```
<owl:NamedIndividual rdf:about="&multiagent;Robot_1">
    <rdf:type rdf:resource="&multiagent;Robot"/>
    <utilizesSkill rdf:resource="&multiagent;skill_flaming"/>
    <utilizesSkill rdf:resource="&multiagent;skill_spraying"/>
</owl:NamedIndividual>
```

It is worth to notice that the reasoning capabilities are used to classify the instances from Table 4.4 to be the type of *Robot* defined in the robot ontology. Since the class *Robot* defines that each instance which has at least one property *providesSkill* is of type *Robot*, the reasoner infers that each instance listed in Table 4.4 is of type *Robot*.

4.3.3 Mapping Ontology to Model-Driven Architecture

In addition to the core component of the proposed task allocation approach, described in Section 4.2.3, which focuses on modeling heterogeneous resources utilizing the general ontology described in the Section 4.2.1, this section introduces another component which facilitates the construction of a robotic fleet, i.e., a configuration. Similar to the component for modeling heterogeneous resources, the configuration is responsible for the construction of a robotic fleet and therefore utilizes the ontology described in Sections 4.3.1 and 4.3.2.

Similar to the approach described in Section 4.2.3, the component for the construction of a robotic fleet configuration utilizes an ontology to describe scenario requirements and build PIM. PIM, formally modeled with the scenario ontology presented in Figure 4.4, models different scenarios based on the tasks' complexity, i.e., the number of skills a task requires, and the amount of required sources. The task complexity is modeled as the number rather than the real skill, e.g., a spraying skill. The same pattern is applied

for modeling the amount of resources. The amount is modeled as a number without expressing a unit. The major aim at this level is to represent main properties of scenarios at the PIM level.

Constructed PIM is afterwards validated and transformed to PSM. The PSM represents a configuration of robotic fleet described in Section 4.3.2. At the moment, the transformation from PIM to PSM is done manually by observing the properties from a modeled scenario and thus constructing a robotic fleet configuration accordingly. The main parameter to take care of during the transformation is the number and the type of skills requested by tasks classified in that specific scenario. Accordingly, a robotic fleet configuration is constructed. Unfortunately, there is no single way to construct a configuration since it is a combination which includes the total number of robots and the number of provided skills per robot, e.g., a set of 3 different robots and a set of 3 different skills yields in total 21 different configuration. One robot has 7 different configuration possibilities C for skills $S = \{a, b, c\}$, i.e., $C = \{a, b, c, ab, ac, bc, abc\}$. Since robots can provide the same skills, and there are in total 3 robots, the total number of possible configurations is 21. One parameter which determines the transformation pertains to the types of skills requested by tasks and provided by robots. The robots in a configuration have to provide all skills requested by a set of tasks in order to execute them successfully. This makes sense under the assumption that the robots have sufficient amount of resources.

As already mentioned, the purpose of formally modeling different scenarios using an ontology is to investigate a relation between a specific scenario and the configuration of the robotic fleet. It strives to select the most appropriate configuration for a particular scenario.

4.4 Coordination Ontology

Ontologies have a wide use in coordinating different components in mixed human-robot teams. From facilitating the coordination issues between multiple hardware and software components residing on one robot [82], to modeling robots coordination and collaboration patterns [57], [131], [8], and also to model human-robot interactions in mixed-human robot teams [9]. However, this section is focused on utilizing an ontology to model a task mapping, i.e., a task allocation, between tasks belonging to a specific scenario and robots from a configuration described in Section 4.3. In addition, in the focus is also a simple ontology-based model of human-robot interaction.

The framework developed in this thesis utilizes the ontology-based coordination model as well as the ontology-based human-interaction model to enable robot-robot and human-robot coordination and collaboration activities. This is a final component which together with the the core component for modeling heterogeneous resources described in Section 4.2 and the component for the construction of a robotic fleet described in Section 4.3, forms a complete ontology-based framework for a coordination of robotic fleet. The framework is named SKIM and is presented Chapter 5.

4.4.1 Ontology-based Coordination

The simplest approach to coordinate the activities of interacting agents has been to hard-wire the coordination mechanism into the system structure by means of semaphores, monitors, or locks [131]. In dynamic systems, like multi-robot systems with mixed human-robot teams, such an approach is infeasible. The idea in such systems is to allow the agents to communicate their intentions with respect to the future activities and the utilization of resources. Moreover, it would be beneficial to enable the processes to reason about coordination at run time.

The first issue to address in such a system is to have all agents agree on a common vocabulary for coordination. This issue has already been addressed in Section 4.2 and thus sets a solid background for an efficient coordination mechanism based on shared knowledge.

Figure 4.6 illustrates a coordination ontology with multiple *Capability* classes related to different *Skill* classes, i.e., *Tilling*, *Flaming*, and *Spraying*, over the property *utilizesSkill*. These three classes representing different skills are the same as those presented in the configuration ontology in Figure 4.5. Moreover, the property *utilizesSkill* is the same as the following properties:

- property *needsSkill* from the general ontology in Figure 4.2, and
- property *providesSkill* from the robot ontology in Figure 4.3.

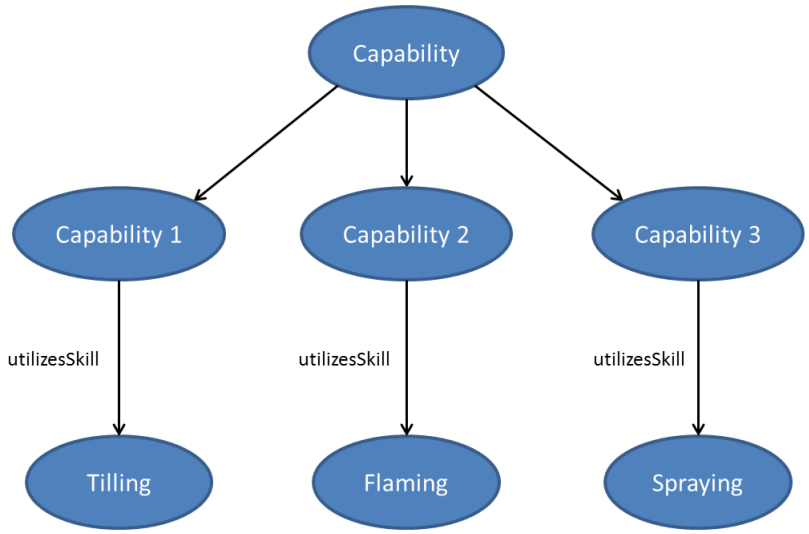

Figure 4.6: Coordination ontology

Speaking in semantic terms, those three properties are aligned which is a precondition for a successful task allocation. *Capability* classes model the task allocation process by relating different skills to different *Capability* classes. The idea is to classify all tasks and

all robots from a configuration to those *Capability* classes. This means that all tasks which have an individual of type *Flaming* as a value for the property *needsSkill* and all robots which have an individual of type *Flaming* as a value for the property *providesSkill*, e.g., Robot_1 and Robot_3 from Table 4.4, will be members of the class *Capability 2*, i.e., will be of type *Capability 2*. Classification is performed through the utilization of Pellet reasoner.

Listing 4.4 shows an excerpt of an ontology defined in OWL which presents definition of class *Capability_2* and thus complements Figure 4.6. In particular, it shows class *Capability_2* which is a defined class with one necessary and sufficient condition defined as *Restriction*. The condition ensures that an individual which has at least one value for *utilizesSkill* property of type *Flaming*, is classified as the type of *Capability_2* class. Moreover, the listing defines that *Capability_2* is a subclass of *Capability* class.

Not only does the coordination ontology, in particular various definitions of *Capability* classes, enable task allocation to the robots defined in a configuration, but also it establishes a basis for the shared knowledge in a robotic fleet. The notion of shared knowledge will be described in more detail in Section 6.4.

Listing 4.4: Ontology excerpt - class *Capability_2*

```
<owl:Class rdf:about="&multiagent;Capability_2">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&multiagent;utilizesSkill"/>
            <owl:someValuesFrom rdf:resource="&multiagent;Flaming"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&multiagent;Capabilities"/>
</owl:Class>
```

4.4.2 Ontology-based Human Interaction

Although robotic fleets are designed for limited autonomous operation in the field and interaction with humans for the operation purposes, some robotic fleets occupy working or living environments and directly serve humans, or interact with humans in joint tasks. Specific scenarios require certain capabilities of the HRI. The most challenging future scenario is the one in which humans and autonomous robots with self-optimizing and learning capabilities collaborate in joint tasks.

So far the ontology-based modeling approach is utilized to overcome the heterogeneity issues in a system, i.e., for semantically describing tasks and robots, afterwards, to model different scenarios and configurations of robotic fleets, after that, to model the coordination capabilities, and finally it is used to construct a simple model of human-robot interaction.

A simple ontology illustrated in Figure 4.7 is composed of two classes: (1) class *Skill* which is familiar from before, and (2) class *User* which is new. Class *User* models a simple human interaction mechanism with a robotic fleet. Besides that, there is one property, *utilizesSkill*, also familiar from before and introduced in the coordination ontology. However, the difference is that the property *utilizesSkill* has a parameter *min x*

where x denotes the minimum number of skills an individual has to have to be classified in *User* class, i.e., to be of type *User*. On the one hand, when a task requires x or more skills, the reasoner will classify it to the *User* class. On the other hand, when a robot from a configuration provides x or more skills, the reasoner will also classify it to the *User* class.

The value x models the human behavior, its interaction with a fleet and its involvement in decision processes. When a task is, in addition to type *Task*, also classified as *User*, a robot which gets that task will have to consult a human operator to obtain a permission to execute the task. Thus, robot's level of autonomy is directly influenced by the definition of *User* class. In particular, definition of *User* class models the adaptive autonomy. However, this model delimits the system to support only one human operator.

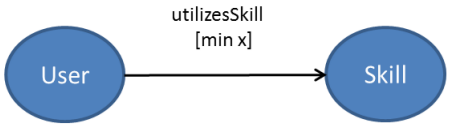

Figure 4.7: Ontology-based Human Interaction

Listing 4.5 shows an excerpt of an ontology defined in OWL which presents the definition of class *User* and thus complements Figure 4.7. It shows class *User* which is a defined class with one necessary and sufficient condition defined as *Restriction*. The condition ensures that an individual which has at least three skills, i.e., at least three relations with *utilizesSkill* property, is classified as the type of *User* class. In this excerpt, value x from the figure above is set to 3.

Listing 4.5: Ontology excerpt - class *User*

```
<owl:Class rdf:about="&multiagent;User">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&multiagent;utilizesSkill"/>
            <owl:onClass rdf:resource="&multiagent;Skill"/>
            <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">
                3</owl:minQualifiedCardinality>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

However, a question is what does it mean for either a task or a robot to be of type *User*, what are the consequences? When during a mission execution a robot of type *User* tries to execute a task of type *User*, it has to consult a human operator who will decide whether the robot has necessary skills and resources to execute the task. The robot consults a human for assistance if there are ambiguities during the execution of the task. In that case the human decides whether the robot has compliant skills and needed amount of resources to execute the task. Upon receiving confirmation from the human, the robot is allowed to proceed with regular execution of the task. This algorithm is described in detail in Section 6.4.

4.5 Flexibility supported by Ontology and Model-Driven Architecture

Heterogeneous system designers have to cope with the lack of standardisation between different devices which impedes their seamless integration. To cope with these issues, system designers are obliged to incorporate hard-wired interactions which deteriorate the overall system flexibility. The notion of flexibility pertains to the system extendability with new modules, scalability, dynamic interactions, heterogeneous resources, human interaction, adaptability to new environments, reusability, etc. Consequently, this thesis reveals two main sources which degrade the flexibility of multi-agent systems: (1) the lack of mechanisms to overcome resources heterogeneity resulting in inefficient communication, coordination, and collaboration, and (2) domain tailored and domain dependent interaction models which limit the reusability in similar application.

The thesis addressed these two problems in the above sections by harnessing the coherence of formal ontology-based modeling and Model-Driven Architecture approach. Ontologies are used to formally model various hot spots in multi-agent systems which usually obstruct the system's flexibility:

- ontology-based modeling of heterogeneous resources, i.e., tasks and agents,
- ontology-based modeling of an operating environment,
- ontology-based modeling of a configuration of robotic fleet,
- ontology-based modeling of a coordination activities, and
- ontology-based modeling of a human-robot interaction.

Up to some extent, the introduced ontologies decouple the system implementation from the system specification resulting in the increased domain independency. However, the idea was to even more decouple the implementation from specification by utilizing the Model-Driven Architecture approach for making a clear separation between a system model and its specific implementation on a target platform. Consequently, complementing the ontology-based modeling approach with the Model-Driven Architecture brings several benefits to the system design with respect to flexibility:

- multi-agent system designers ***shift focus from logical and technical details***, i.e., PSM, to CIM and PIM which helps them to devote more attention to develop conceptual models of each system component,
- increased ***portability*** due to automatic transformation of a PIM to multiple PSM, i.e, target platforms,
- ***integration and interoperability*** enabled by overcoming the resource heterogeneity and introducing shared knowledge,

- ***reusability*** of ontology-based models,
- when deployed on another platform, ***adaptability*** ensures that the PIM can be deployed and utilized without any changes, and
- ***evolution*** support ensures that new requirements are addressed in CIM and PIM which are then automatically transformed to PSM, and
- ability to perform ***validity check*** before translating PIM to PSM increases the system ***robustness***.

To conclude, the chapter reflects how the synergy between Semantic Web Technologies and Model-Driven Architecture at the same time provides a formal model of a system and facilitates the design of multi-agent systems while increasing the interoperability, scalability, adaptability, reusability, and robustness.

Next chapter proposes the design of main building blocks of the Shared Knowledge Interaction Modelling (SKIM) framework. SKIM framework is designed to investigate model of shared knowledge as a basis for adaptive autonomy in mixed teams and to evaluate its impact on task allocation.

CHAPTER 5

SKIM - Shared Knowledge Interaction Modeling Framework

This chapter introduces the design of the main building block of the Shared Knowledge Interaction Modelling (SKIM) framework [29]. The SKIM coordination framework is designed to investigate the following challenges: (1) model shared knowledge as a basis for adaptive autonomy in mixed teams and to investigate its impact on task allocation, and (2) implement different coordination approaches to evaluate the performance of finding a set of robots to execute a certain task. To address these challenges, the SKIM framework utilizes the semantic approach to describe the collaboration and coordination activities by means of ontologies: SKIM Resource Ontology (SKIM-RO) and SKIM Coordination Ontology (SKIM-CO) which are described in more detail in the upcoming sections.

The chapter starts by introducing and describing the task allocation scenario in detail that provides motivation and requirements for the presented work. After that the core characteristics of knowledge-based task allocation with Semantic XVSM are explained. This section is then complemented with the two models describing task transfer from the central part to the distributed robots. After that, the focus is switched to robots which support different knowledge-based resource matching mechanisms as well as different decision-making mechanisms. Then, a design which supports system robustness during a task execution is introduced. Finally, the chapter is concluded with the design of three different coordination approaches for task allocation.

5.1 Application Scenario: Task Allocation

This section describes the application scenario exploited for extracting requirements and providing a motivation for the design of the SKIM framework in detail. The application scenario is based on a specific precision agriculture use case based on the robotic fleet

for weed control elaborated within the European Project RHEA-Robot Fleets for Highly Effective Agriculture and Forestry Management [153].

5.1.1 Initial Precision Farming Scenario

Sustainable precision crop management can be based on the use of a fleet of heterogeneous robots equipped with advanced sensors and actuators as developed within the project RHEA. Figure 5.1 illustrates a precision farming concept of the RHEA project where the core of the RHEA concept is a centralized fleet management system that assists the system operator in choosing a suitable strategy for field treatment, taking into account weed infestation map and available robots, their implements, and sensors [46]. The selection of the treatment strategy, i.e., building a mission, takes into account many parameters, e.g., the type of tasks to be performed, the number and features of available robots, and field information. After the mission is defined, it is decomposed in the number of tasks and mapped to corresponding robots. A centralized control system is responsible for both task-robot mapping and robots coordination during the mission execution. During the mission, heterogeneous and distributed robots report their status to the base station where a human operator supervises the mission and acts as a central point in the system. The base station is a place that manages, coordinates, makes decision, collects data, instructs, and monitors all robots in the network.

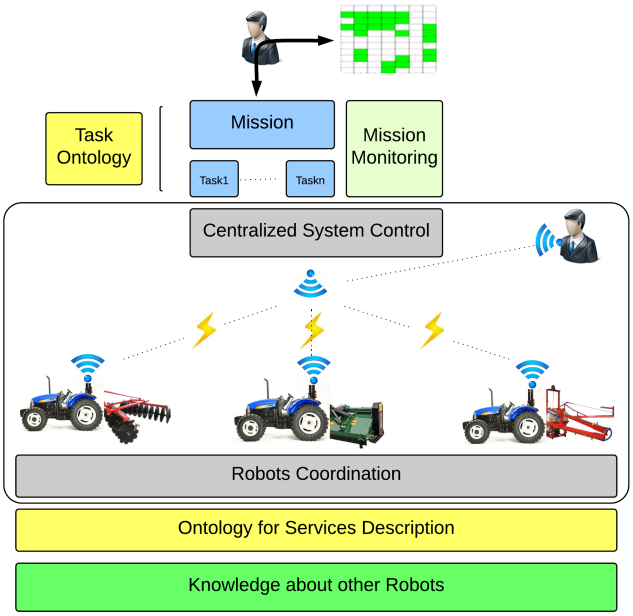

Figure 5.1: Extended RHEA Scenario

A human operator present in the field can establish remote control over the fleet complementing the central user when necessary. In this basic RHEA scenario the robots autonomy is limited to a narrow set of basic functions like small adjustments

related to, e.g., path correction. Consequently, the SKIM interest is to understand how autonomy, i.e., the robot self-awareness, can be extended and the implications of this extension on the robots coordination. In this respect, SKIM is particularly focused on the aspects pertaining to the modeling of tasks and resources and services that the robots embody. Figure 5.1 extends the basic RHEA concept with the following modules: (1) a module which represents the tasks model (task ontology illustrated with a yellow square), (2) a module for robotic coordination (illustrated with a grey rectangle), (3) a module supporting self-awareness and shared knowledge models (illustrated with a green rectangle), and (4) a module providing models of robotic resources and services (ontology for service description illustrated with a yellow rectangle). This thesis refers to the concept illustrated in Figure 5.1 as the extended RHEA scenario.

5.1.2 Extended Precision Farming Scenario

Due to the centralized control in RHEA scenario, robots are not involved in any collaboration and act as isolated nodes. This solution has benefits and drawbacks. On the one hand, centralized control is far less complex than distributed control. On the other hand, lack of a collaboration and coordination capabilities among robots has a negative impact on the mission; although robots are coupled both in a time and in space, as they operate in the same field, they are not directly aware of each other. Therefore, the thesis extends the initial assumptions of the RHEA scenario with the cooperation and collaboration capabilities at robots. In this new scenario, robots are aware of their local context and capabilities of other robots in the fleet. It means that they can autonomously select to perform a task within a complex mission, which requires e.g., a combination of different implements (skills), such as spraying and flaming implements.

In both scenarios, basic and extended RHEA, a human operator at the base station designs a mission. The mission consists of a number of tasks that require some amount of the resources, e.g., spraying, flaming or tilling. Each robot has a specific implement and is able to execute one or more different tasks, depending on the implement type. While the task allocation in the basic RHEA scenario is static and performed before the mission starts, the distributed task allocation in the new, extended, scenario is based on dynamic mapping between tasks and available robots.

In the extended scenario, when the human produces a task, the task is written into a space and then distributed to robots' local triple stores. Thus, the space is distributed over robots in a fleet. Each robot queries the local triple store for a task that matches its capabilities. When such a task is found, it is executed. However, the case that two robots try to execute the same task has to be prevented in order to avoid deadlock which could lead to collisions. Section 5.7.3 describes how the SKIM framework addresses this issue. Moreover, the SKIM framework combines space-based middleware with semantic modeling, processing and reasoning to enable the robots to have direct awareness of the fleet and to make autonomous decisions about the tasks they can jointly perform. Semantics enable a robot to automatically infer the task-robot mapping based on the data in the dynamically updated triple store, and to dynamically select a task for immediate execution.

5.2 Knowledge-based Task Allocation with Semantic XVSM

The main requirement for a successful task allocation in a robotic fleet is an effective coordination mechanism which prevents robots from collisions, dead-locks, etc., and enables them to cooperate and collaborate during a mission execution. The semantics of the language employed in SKIM to define coordination rules is based on formal logics. Since the advantages of using logic-based languages are not always obvious to the developers of coordination laws, this sections strives to explain them.

The section describes the core characteristics of knowledge-based task allocation by investigating the three main elements of coordination and thus task allocation: (1) how the space-based middleware, as a coordination medium, supports the concepts from the Semantic Web, (2) how the coordinated entities interact by utilizing the semantic space, and (3) how the ontology can enable the modeling of coordination.

5.2.1 Adding Semantics to the Coordination Medium

The SKIM framework, as a simulation environment for performance evaluation of task allocation algorithms, is built on Semantic XVSM already described in Section 2.2.3. In order to utilize information stored in the knowledge base, Semantic XVSM provides the following features:

- ***Bulk data as single data unit***: opposed to space-based systems which access a single tuple, Semantic XVSM provides an interface for reading and writing an entire sets of triples describing a particular object in the knowledge base.
- ***Reasoning support***: Semantic XVSM implements a reasoning mechanism for inferring new knowledge from the stored triples and also makes this new knowledge accessible in the semantic space.
- ***Knowledge extraction***: for extracting particular fragments of the knowledge stored in a semantic space, Semantic XVSM offers a powerful capability to retrieve data by introducing minor restrictions on SPARQL query.

Hence, Semantic XVSM can be used to store, reason about, and extract knowledge. However, the knowledge is not created at once, rather, it is successively published in the space. Even before the knowledge publishing, it is required to consider which types of knowledge will be described and handled in the space. Knowledge is typically structured into the: (1) Domain Ontology, (2) Coordination Ontology, and (3) Instance Data. SKIM framework also addresses these 3 knowledge structures illustrated in Figure 5.2.

- ***SKIM-RO as Domain Ontology***: SKIM Resource Ontology (SKIM-RO) describes the domain model, i.e., generally employed concepts and relationships in the application domain.

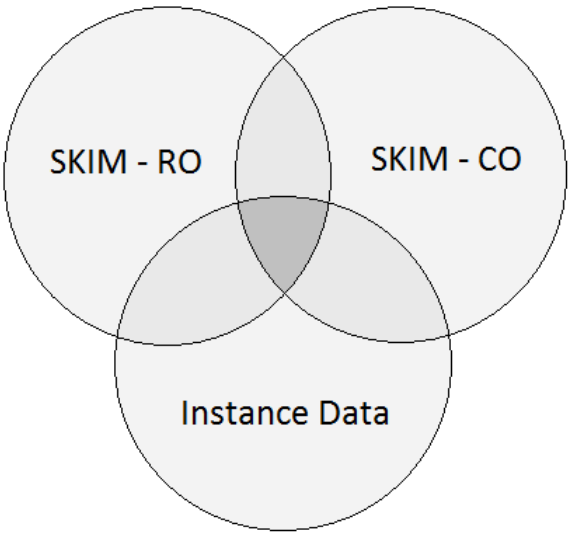

Figure 5.2: Knowledge stored in Semantic XVSM

- ***SKIM-CO as Coordination Ontology***: SKIM Coordination Ontology (SKIM-CO) describes the concepts and dependencies that drive the coordination processes.
- ***Instance Data***: describes the current state of the knowledge base and it is usually the biggest part of that base.

SKIM - Resource Ontology (SKIM-RO)

SKIM-RO describes resources, including robot capabilities and task requirements and it consists of the following ontologies presented in Chapter 4:

- general ontology for describing tasks described in Section 4.2.1,
- the ontology for describing robots presented in Section 4.2.2, and
- the ontology for modeling different scenarios introduced in Section 4.3.1.

These three ontologies combined into the SKIM-RO support tasks modeling by utilizing the class *Task*, robots modeling by using the class *Robot*, as well as modeling of working environment, i.e., a scenario, facilitated with the extensions of class *Abstract Scenario*.

SKIM - Coordination Ontology (SKIM-CO)

SKIM-CO describes coordination constraints for robot-robot and robot-human interactions and it consist of the following ontologies presented in Chapter 4:

- the ontological description of a configuration described in Section 4.3.2,
- the core coordination ontology presented in Section 4.4.1, and
- the ontology for modeling a human interaction introduced in Section 4.7.

These three ontologies combined into the SKIM-CO support the creation of configurations by defining specific skills provided by robots which extend the class *Skill*, the modeling of coordination processes by defining different classes which extend the class *Capability*, and the modeling of human-robot interaction by modifying the class *User*.

SKIM framework, based on the Semantic XVSM, utilizes the semantic approach to describe and facilitate the collaboration activities by means of ontologies: SKIM Resource Ontology (SKIM-RO) and SKIM Coordination Ontology (SKIM-CO). SKIM-RO describes resources, including robot capabilities and task requirements, and SKIM-CO describes coordination constraints for robot-robot and robot-human interactions. Hence, these ontologies are used as the model of shared knowledge and the decisions are results of automated reasoning on them.

5.2.2 Connecting the Coordination Entities for Task Allocation

This section introduces coordinated entities and their interaction with respect to the application scenario described in Section 5.1.1. It is considered in more detail how the coordinated entities are connected to the Semantic XVSM and how interaction between coordinated entities and the Semantic XVSM is implemented.

In addition to the API for adding, removing, and querying the knowledge base, Semantic XVSM provide reactive operations like subscription/notification mechanisms, introduced in Section 2.2.3, utilized in this thesis for synchronizing and coordinating multiple connected entities. A single atomic coordination unit in SKIM framework, which provides reasoning capabilities for coordination purposes, consists of four different phases that typically occur in Semantic XVSM when coordinating multiple distributed entities. Figure 5.3 illustrates these four phases:

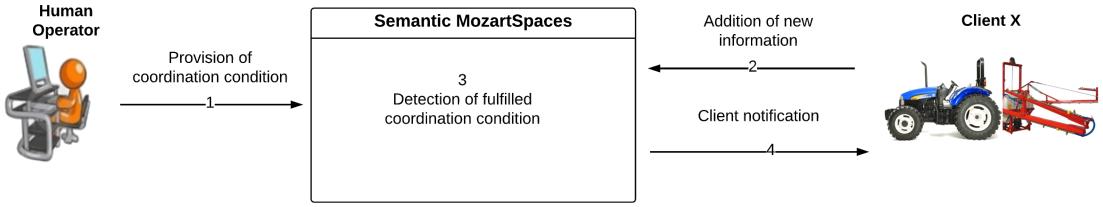

Figure 5.3: Four phases of a coordination step

- ***Provision of coordination information***: a human operator captures the coordination requirements in SKIM-CO and writes the ontology to the Semantic XVSM. Captured coordination conditions describe the mapping of tasks to available robots.

- ***Addition of new information***: a client X, i.e., a robot or a human operator, publishes a task to the semantic space. A robot could publish a sub-task to the semantic space, e.g., when it executed a task only partially where the other part of the task, i.e., a sub-task, is again published, while a human operator can publish a complete task to the semantic space. The publishing mechanism is introduced in Section 5.3.3.

- ***Detection of fulfilled coordination condition***: the semantic space evaluates the new state of the knowledge base and detects the outcome of the coordination condition. If the knowledge base is consistent, i.e., validity checks returned no exception which means that the added information conforms to the underlying ontologies, the outcome of the coordination condition is a task mapped to either one or multiple robots. The resulting mapping conforms to the one of resource matching approaches described in Section 5.4.

- ***Client notification***: the client X, i.e., a robot or another human operator, is notified about the outcome of the coordination condition utilizing mechanisms described in Section 5.6, i.e., an allocated task. The received notification further triggers one of the decision-making processes described in Section 5.5.

Figure 5.4 shows the above four phases as they occur in the application scenario for task allocation in robotic fleet. In contrast to Chapter 4 which introduces components illustrated in Figure 5.4, this section presents the interplay of these components. It is assumed that the semantic space contains the SKIM-RO, i.e., domain knowledge, SKIM-CO, i.e., coordination knowledge, and the instance data described in the previous section. The instance data consists of a task *t_1* which has the property *needsSkill* pointing to the skill *s_1* which is of type *Flaming*. To keep the figure simple, it is omitted that the task *t_1* is of type *Task* and that the class *Flaming* is subclass of the class *Skill*. As a reminder, these classes are defined in the ontologies introduced in Chapter 4.

First phase is depicted in Figure 5.4 (a) where a human operator provides a coordination condition. The coordination condition is a triple with the class *Capability* as a subject, property *utilizesSkill* as a predicate, and the class *Flaming* as an object. The triple defines that each individual, which has a skill of type *Flaming* as an object, is a member of class *Capability*. Second phase, Figure 5.4 (b), denotes the robot, i.e., a tractor, which publishes new information to the semantic space. The information is again a triple with an individual *r_1* as a subject, property *providesSkill* as a predicate, and the individual *s_1* as an object. It is important to notice that the individual *s_1* is same as the skill *s_1* which already resides in the semantic space. Moreover, due to the limited space, it is omitted that the individual *r_1* is of type *Robot*. Furthermore, third phase illustrated in Figure 5.4 (c) describes a reasoning process which does a classification. It is worth to notice here that the property *utilizesSkill* is same as the property *needsSkill* and the property *providesSkill*. In semantic terms, those three properties are aligned which is a precondition for a task allocation. Due to the definition of the class *Capability* where each individual which has a skill of type *Flaming* as an object is the class member, individuals

t_1 and *r_1* are classified as the members of class *Capability*. The classification occurred because both individuals have an individual *s_1* as an object which is further of type *Flaming*. Consequently, fourth phase described in 5.4 (d) notifies the robot that, due to the classification of both, *t_1* and *r_1* as members of the class *Capability*, the individual *r_1* is able to execute the task *t_1*.

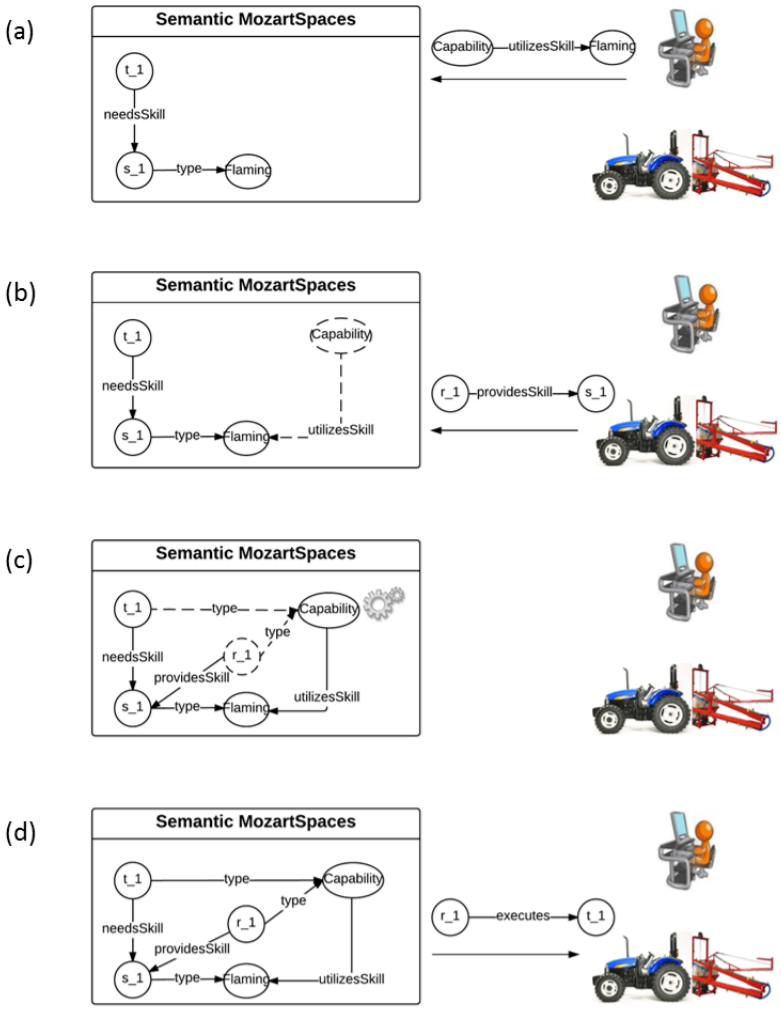

Figure 5.4: Four phases of coordination step in the application scenario

5.2.3 Modeling Coordination Patterns with OWL

In classic data-driven coordination models, the robots define the rules for determining coordination activities and the spaces are only used for synchronizing the coordinated entities as well as data exchange. On the other hand, with a semantic space it is

rather possible to describe a coordination law with the appropriate coordination ontology which shifts the coordination rules from distributed robots to the semantic space. The coordination rules are managed centrally at the semantic space. The benefit is perceived as an increased flexibility and maintainability of the developed applications.

For modeling the coordination law, all the coordination concepts, as well as relevant dependencies, have to be translated to the OWL concepts which build up the coordination ontology. In the task allocation example illustrated in Figure 5.4 (b), the coordination ontology is used to define the necessary and sufficient conditions of the class *Capability*. The existential restriction, as a construct defined in OWL, is utilized for defining the necessary and sufficient conditions of the class *Capability*. Existential restrictions describe classes of individuals that participate in *at least one* relationship along a specified property to individuals that are members of a specified class [60]. For example, the class *Capability* is defined as the class of individuals that have *at least one (some)* ***utilizesSkill*** relationship to members of ***Flaming***. Existential restrictions are by far the most common type of restrictions in OWL ontologies. By employing the reasoner, the semantic space finds an individual that fulfils the described conditions.

Modeling the coordination law in the semantic space by utilizing OWL constructs provides for a clearer separation of computation and coordination logic. If the business requirements change and thus the coordination law, the modification can be easily addressed by adapting the coordination ontology in the space. It is not necessary to modify robot's code.

5.3 Task Transfer Models

This section introduces different task transfer models which are the constituent part of a coordination step as illustrated in Figure 5.3; in particular, its fourth step when a robot receives a task to execute. The SKIM framework provides two different models for robots to query tasks from the Semantic XVSM and one model for a robot to generate and send a task to the Semantic XVSM. A green circle in Figure 5.5 illustrates those models. First two steps in Figure 5.5, i.e., steps 1 and 2 denote a human producing tasks and the semantic space persisting those tasks. Steps 3, 5, and 6 denote the task transfer models, while step 4 is responsible for a task allocation utilizing the coordination ontology SKIM-CO. Following are the task transfer models:

- Query-based task allocation denoted in step 3,
- Inference-based task allocation denoted in step 5, and
- the task generation denoted in step 6.

The task transfer models are described in more detail in the following sections.

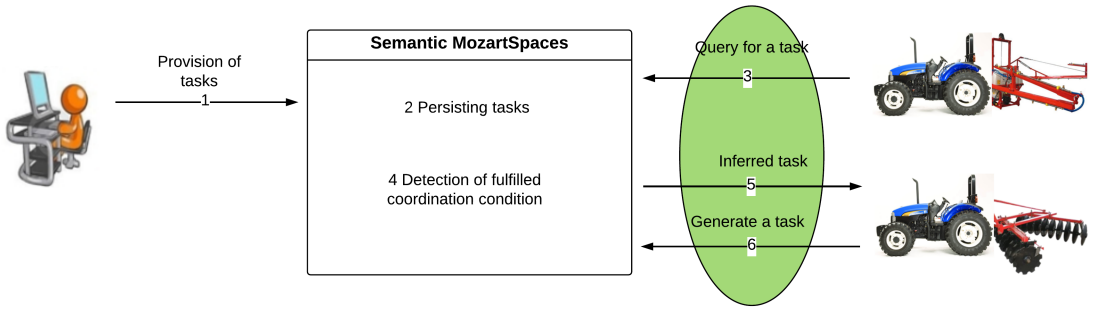

Figure 5.5: Task transfer models

5.3.1 Query-based Task Allocation

The advantage of Semantic XVSM is that SPARQL queries can be used for task selections. For this purpose, a new semantic selector is created that can be combined in a chain with other MozartSpaces selectors on the XVSM Containers. A selector chain is a sequence of selectors, where the result of one selector is piped to the next selector as an input.

Listing 5.1 describes a general, high-level, SPARQL query for a task allocation. The query is created on a robot and submitted to the semantic space where it is processed and the result is returned back to the robot. Listing 5.1 shows a selection query with a graph pattern consisting of four triples: (1) triple which denotes that it wants to select something which is of the type *Task*, (2) triple which selects skills belonging to a task, (3) triple which selects an amount of resource required by a task, and (4) triple which returns the type of skill.

Listing 5.1: SPARQL-based task allocation

```
PREFIX : <http://mozartspaces.org/semantic/multiagent#>
SELECT ?task ?skill ?amount
WHERE {
    ?task rdf:type :Task .
    ?task :needsSkill ?s .
    ?task :needsAmount ?amount .
    ?s rdf:type ?skill .
}
```

As mentioned above, this SPARQL query is very general and high-level since it does not include any robot related parameters, like skills a robot provides or amount of resource a robot has. The SPARQL query can be extended to enable the use of external context entries in the query. The context can be added at the robot side. Context entries can be used as a parameter for SPARQL queries, so that more general and flexible queries are supported, e.g. a context entry can describe the state of a robot. A complete SPARQL

query for task allocation which utilizes context entry to provide the state of robot is presented in the implementation chapter (Section 6.6.1).

Upon executing the SPARQL query from listing 5.1 on the set of data instances presented in Figure 5.7 (a), the result set could for example contain variable bindings as shown in Table 5.1.

Table 5.1: SPARQL-based task allocation results

?task	?skill	?amount
: *t_1*	"Flaming"	7
: *t_1*	"Spraying"	2

Table 5.1 manifests the task *t_1* which requires two skills, flaming and spraying, with an appropriate resource amount, 7 units of flaming liquid and 2 units of spraying liquid, for a complete execution.

5.3.2 Inference-based Task Allocation

In the inference-based task allocation approach, the task allocation is performed in Semantic XVSM by reasoning on SKIM-CO. This task transfer model is already introduced in Figure 5.4 (d). In contrast to the query-based task allocation approach where a robot initiates the task allocation, in the inference-based model the task allocation process is triggered automatically in the semantic space. The reasoning engine facilitates task allocation resulting in the set of tasks mapped to a set of matching robots. Finally, as illustrated in Figure 5.6, notification mechanisms delivers the allocated tasks to the assigned robots.

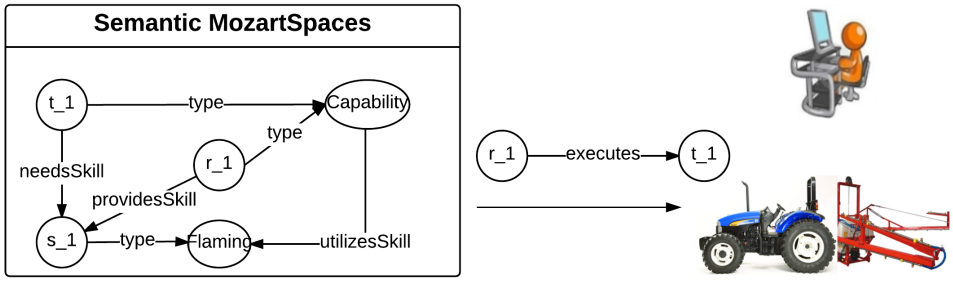

Figure 5.6: Inference-based Task Allocation

The task allocation model is described in more detail in the implementation chapter (Section 6.4.1) with an accompanying task allocation algorithm.

5.3.3 Task Generation

Task generation is a different task transfer model compared to the query- and inference-based transfer models. The former models task generation on a robot side, while the

latter two relate to the task generation in the semantic space, i.e., the sources of task generation are different. Accordingly, task destinations are different as well. Task generation is categorized as the transfer model because task are transferred between different components in the Semantic XVSM.

Figure 5.7 illustrates a process where a robot with only one skill, i.e., one *providesSkill* relation, tries to execute a task with two *needsSkill* relations. Due to the overlapping skills the robot and the task have, i.e., the skill *s_2*, the inference-based task allocation mechanism assigned task *t_1* to the robot, Figure 5.7 (a). On the other hand, it is obvious that the robot will not be able to completely execute task *t_1* because task *t_1* requires two skills, while the robot only has one. However, it will neither drop task *t_1* nor fail to execute it. Rather, due to the support of resource matching mechanisms described in Section 5.4, it will partially execute task *t_1*, Figure 5.7 (b). The partial execution of task *t_1* results in the generation of a new task which has one *needsSkill* relation less than the original task. Figure 5.7 (c) shows how the robot created a new, sub-task, which is basically the same instance as task *t_1*, but with one missing *needsSkill* relation. The missing relation is the part of original task executed by robot.

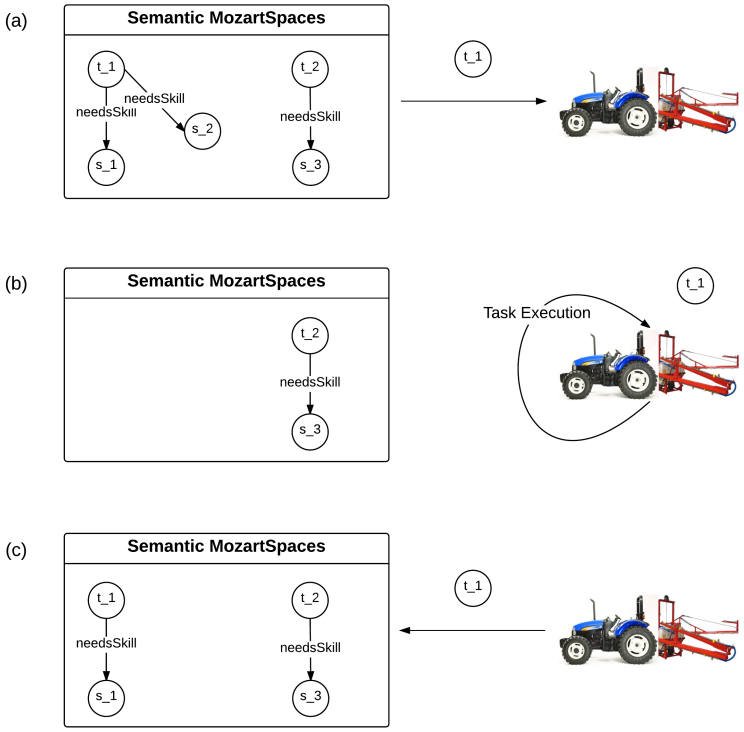

Figure 5.7: Generating a new task

The introduced task transfer model enables partial execution of tasks where one robot can execute one part of a task and the other robot the other part. There could also be more than two robots executing one task. This approach is a cornerstone for

collaboration and cooperation of robots within a fleet.

5.4 Knowledge-based Resource Matching

A task allocation mechanism is in charge of mapping task requirements to the services provided by robots to infer which task can be executed on which robot. Regardless which task allocation mechanism is utilized, query- or inference-based, they both take the advantage of resource matchmaking approaches to decide which task goes to which robot. Matchmaking approaches enable a successful task allocation even when not all task requirements are fulfilled by one robot.

SKIM reuses semantic relationship notion between the task requirements and advertised robot services introduced in [86], [8], and [48], to describe the matching degree between requirements and provided services. Figure 5.8 describes the three most common matching degrees: (1) ***exact***, (2) ***plug-in***, and (3) ***subsume***. In addition, there is also forth matching degree named ***fail*** which denotes that none of the previous three degrees is achieved. An icon illustrating a power plug-in in Figure 5.8 presents a robot in the task allocation scenario and an icon illustrating a power outlet presents a task in the task allocation scenario.

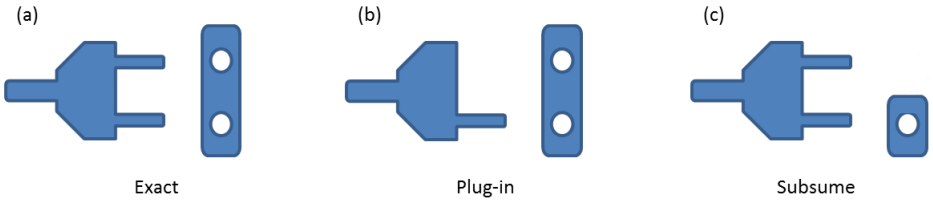

Figure 5.8: Matchmaking models

Due to the ontology support in Semantic XVSM, the semantical description of resources, i.e., tasks and robots, utilizing SKIM-RO enables these 3 matching models. Each matchmaking model is described in more detail in the following sections.

5.4.1 Exact

Exact matchmaking model is illustrated in Figure 5.8 (a) where both, a robot represented as a power plug-in, as well as a task represented as a power outlet, have the same number of provided and requested skills. Although both, the robot and task, provide and require two skills, the concept is also applicable when the number of requested and provided skills is x.

Translated to the task allocation scenario referenced throughout the chapter, it is assumed that in the semantic space exists an instance, e.g., *t_1*, of type *Taks*, defined in SKIM-RO, which has two *needsSkill* relations, i.e., the task *t_1* requires two skills to be completely executed. These two skills are instances *s_1* and *s_2* of type *Flaming* and *Spraying*, respectively. On the other hand, there exists an instance, e.g., *r_1*, of type

Robot which has two *providesSkill* relations, i.e., the robot *r_1* provides two skills for a task execution. These two skills are the same instances *s_1* and *s_2* of type *Flaming* and *Spraying*, respectively. It is important to point out that both instance, the task instance *t_1* as well as the robot instance *r_1*, have the same skills *s_1* and *s_2*.

When either of two task allocation mechanisms is triggered on the above populated knowledge base, i.e., instance data, the result is perfect, i.e., exact, match and the task *t_1* is allocated to the robot *r_1* for an execution. Consequently, the robot *r_1* is able to completely execute the allocated task (under the assumption that it has a sufficient amount of resources).

5.4.2 Plug-in

Plug-in matchmaking model is illustrated in Figure 5.8 (b) where a robot represented as a power plug-in provides less skills than a task represented as a power outlet requests, i.e., there is a mismatch between requested and provided skills.

Translated to the task allocation scenario referenced throughout the chapter, it is assumed that in the semantic space exists an instance, e.g., *t_1*, of type *Taks* which has two *needsSkill* relations, i.e., the task *t_1* requires two skills to be completely executed. These two skills are instances *s_1* and *s_2* of type *Flaming* and *Spraying*, respectively. On the other hand, there exists an instance, e.g., *r_1*, of type *Robot* with only one *providesSkill* relation, i.e., the robot *r_1* provides one skill for a task execution. The provided skill is the instances *s_1* of type *Flaming*. It is important to point out two facts: (1) the task instance *t_1* and the robot instance *r_1* have different number of skills, and (2) the skill *s_1* requested by the task *t_1* is same as the skill *s_1* provided by the robot *r_1*.

Although there is a mismatch in the number of requested and provided skills, the task allocation mechanism does not fail and still produces a task-robot mapping. When either of two task allocation mechanisms is triggered on the above populated knowledge base, i.e., instance data, the result is plug-in match and the task *t_1* is allocated to the robot *r_1* for an execution. In contrast to the exact match, the robot *r_1* partially executes the allocated task which triggers the generation of a new sub-task described in Section 5.3.3. It is assumed that the robot *r_1* has a sufficient amount of resources to partially execute the task *t_1*.

5.4.3 Subsume

Subsume matchmaking model is illustrated in Figure 5.8 (c) where a robot represented as a power plug-in provides more skills than a task represented as a power outlet requires, i.e., there is a mismatch between requested and provided skills. This model is opposite to the plug-in model.

Translated to the task allocation scenario referenced throughout the chapter, it is assumed that in the semantic space exists an instance, e.g., *t_1*, of type *Taks* which has one *needsSkill* relation, i.e., the task *t_1* requires one skill to be completely executed. The provided skill is the instances *s_1* of type *Flaming*. On the other hand, there exists

an instance, e.g., *r_1*, of type *Robot* which has two *providesSkill* relations, i.e., the robot *r_1* provides two skills for a task execution. These two skills are the instances *s_1* and *s_2* of type *Flaming* and *Spraying*, respectively. It is important to point out two facts: (1) the task instance *t_1* and the robot instance *r_1* have different number of skills, and (2) the skill *s_1* requested by the task *t_1* is same as the skill *s_1* provided by the robot *r_1*.

Although there is a mismatch in the number of requested and provided skills, the task allocation mechanism does not fail and still produces a task-robot mapping. When either of two task allocation mechanisms is triggered on the above populated knowledge base, i.e., instance data, the result is subsume match and the task *t_1* is allocated to the robot *r_1* for an execution. Similar to the exact match, the robot *r_1* completely executes the allocated task. It is assumed that the robot *r_1* has a sufficient amount of resources to partially execute the task *t_1*. Opposite to the plug-in model, subsume model is utilized when a robot provides more skills than requested by a task.

5.5 Decision-making Mechanisms

The task allocation performed centrally in the Semantic XVSM and the task transfer from the Semantic XVSM to a robot entails the autonomous triggering of either one or the combination of multiple decision-making mechanisms deployed on each robot. Decision-making mechanisms are algorithms where an input is an allocated task and an output is a notification denoting whether the allocated task is completely or partially executed. It could happen that the task execution fails as well. Following are three decision-making mechanisms supported in SKIM framework:

- On-the-robot decision-making mechanism illustrated in Figure 5.9 (a),
- In-the-fleet decision-making mechanism illustrated in Figure 5.9 (b), and
- Human-enabled decision-making mechanism illustrated in Figure 5.9 (c).

The following sections describe each of these decision-making mechanisms in more detail.

5.5.1 On-the-robot Decision Making

The term on-the-robot decision-making reflects an autonomous local-decision making procedure used when a robot can reach a decision utilizing only local knowledge. In that case the robot does neither need to interact with other robots in a fleet nor with a human operator.

Prerequisites for triggering on-the-robot decision-making algorithm are successful task allocation and task transfer to a target robot. Figure 5.9 (a) depicts a robot, let us name it *r_1*, which has three allocated tasks transferred to its local queue where they wait for execution, i.e., *t_1*, *t_2*, and *t_3*. After robot *r_1* finishes execution of its current task,

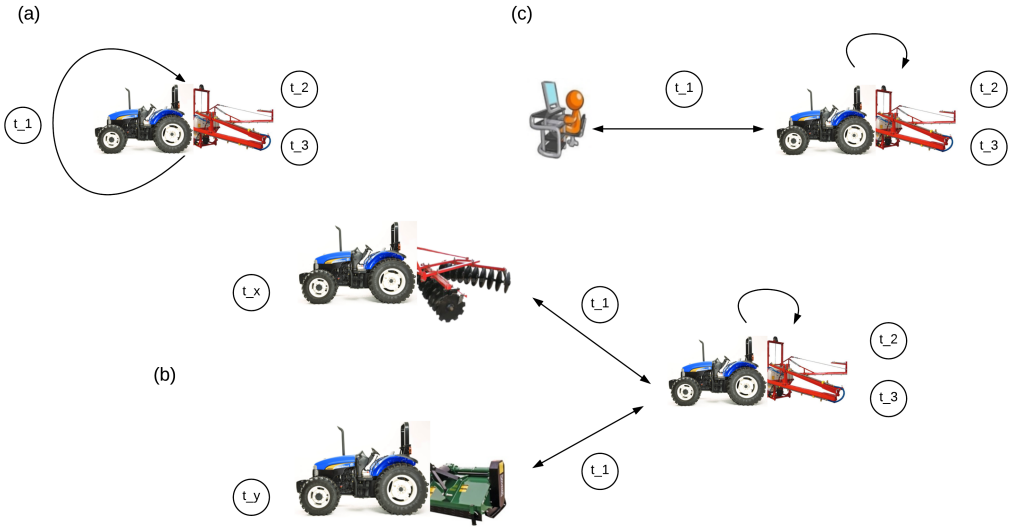

Figure 5.9: Decision-making mechanisms

it is ready to take another task from the local queue and proceed with the execution of the next task, i.e., *t_1*. To be able to execute *t_1* locally, without interaction with a human or a fleet, the following conditions have to be satisfied:

- The skills provided by robot *r_1* have to either exactly match or subsume the skills requested by task *t_1*.
- If robot *r_1* has all necessary skills to execute task *t_1*, it has to ensure that it has enough resources to execute task *t_1*.
- Due to the fleet heterogeneity, it could happen that the inference-based task allocation mechanism allocates one task to multiple robots. This type of allocation results in colliding tasks requiring the additional coordination and synchronization between the robots that have been assigned a same task. Therefore, task *t_1* has to be exclusively allocated to robot *r_1* to execute it locally.

The above conditions are validated in the same order as they are introduced. This means that in order to proceed to the next condition, the previous one has to be satisfied. After the all conditions are positively evaluated, task *t_1* is executed. If one of the conditions fails, task execution fails as well and the task will be executed by another robot. Handling an execution failure is described in the next chapter.

5.5.2 In-the-fleet Decision Making

In contrast to on-the-robot decision-making mechanism where a robot makes autonomous decision locally without interacting with a fleet or a human, in in-the-fleet decision-making

mechanism the robots in a fleet communicate and exchange data that can be used to perceive a global state and trigger a global decision.

Prerequisites for triggering in-the-fleet decision-making algorithm are successful task allocation and task transfer to a target robot. Figure 5.9 (b) depicts a robotic fleet consisting of three heterogeneous robots providing different skills, let us name them *r_1*, *r_2*, and *r_3*. Robot *r_1* is the same as robot *r_1* on Figure 5.9 (a). Each robot in the fleet has some allocated tasks transferred to its local queue where they wait for execution. However, the focus is on robot *r_1* which has three allocated tasks, i.e., *t_1*, *t_2*, and *t_3*.

After robot *r_1* finishes execution of a previous task, it is ready to take another task from the local queue and proceed with the execution of the next task, i.e., *t_1*. During task *t_1* execution, there are two different situations that can emerge and trigger in-the-fleet decision making mechanism:

- If robot *r_1* detects that task *t_1* is a colliding task it utilizes shared knowledge to coordinate the execution with other fleet members (described in Section 5.7.3).

- When robot *r_1* detects that it does not have sufficient skills, it exhibits plug-in matching degree for task *t_1*, it generates a new task which requires other fleet members to collaborate on the task.

Colliding Task

After successfully passing the resource amount check test for task *t_1*, robot *r_1* inspects whether task *t_1* is a colliding task, i.e., task *t_1* is allocated not only to robot *r_1* but also to some other robots (colliding robots), e.g., *r_2* and *r_3* as depicted in Figure 5.9 (b).

In that case robot *r_1* notifies colliding robots that it is going to execute task *t_1* and that they should remove it from their local queues. What is important here is desynchronization of colliding robots to avoid that more than one robot tries to execute the same colliding task at the same time.

Task Generation

A second use case example which triggers in-the-fleet decision making mechanism is when robot *r_1* detects that it does not have sufficient skills for completely executing task *t_1*, it exhibits plug-in matching degree for task *t_1*. However, robot *r_1* can partially execute task *t_1* and, consequently, generate a new sub-task which requires some other robots to collaborate on it.

This procedure is already described in Section 5.3.3 and it is a cornerstone for collaboration and cooperation of robots within a fleet.

5.5.3 Human-enabled Decision Making

Effective human-enabled decision making systems rely on attaining and utilizing an appropriate knowledge of involved entities, their activities, environment, and a mission. From the human perspective, this is referred to as human-robot awareness [115]. Conversely, from the robots perspective it is typically defined as situation awareness. Both refer to the state or ability to perceive, or to be conscious of events and objects residing in a surrounding environment. A lack of awareness, i.e., missing to acquire information, decreases the overall task performance. Therefore, awareness directly influences the efficiency of decision making system.

The SKIM framework addresses the notion of human-robot awareness by designing different components in charge of both, acquiring additional knowledge about a robotic fleet, and at the same time utilizing the harvested knowledge in various decision-making processes. The following components are the building blocks for a human-enabled decision making system.

- Human-robot interaction module ensures that a human has sufficient knowledge of the locations, identities, activities, status and surroundings of the robots, and that it can either partially or completely utilize this knowledge during the interaction with a robot.
- Robot-human interaction module ensures that both, a robot and a human have specific knowledge to react on an occurred event. The robot has to be aware of the human knowledge and ability to solve a particular event.
- Human's ability to gain and utilize the overall mission awareness and understanding of the overall goals of the joint human-robot activities.

To efficiently combine autonomous control embodied within the robots in their decision making systems and the human operator intelligence, SKIM exposes relevant data and control interfaces for human intervention. While human-robot interactions in the field are primarily considered in the context of resolving critical situations, they may also be used for optimization of the robotic fleet.

Robot-human interaction enables the human to interact with a fleet in two different ways. First, observing the task allocation process, the human can apply shared knowledge to allocate a task which was not allocated in the inference-based task allocation process. Robot-human interaction refers to when a robot asks a human for assistance while executing a complex task.

The human operator with a direct insight into the operational processes and status of all robots can positively influence the effectiveness of the mission and facilitate decision-making processes in critical situations. In order to support the operator in his/her role in the overall process, the fleet has to act as an information system for the operator and provide him/her with the appropriate and timely delivered information. Beneficial information for the human is one which is not directly perceived by observing the mission, but rather, it is inferred from the existing information.

The following chapter describes the implementation of building blocks and accompanying algorithms which enable the human-based decision making system.

5.6 Robustness during Task Execution

Robustness in the task execution process in multi-robot systems raises the following two requirements: (1) system's ability to proceed with a mission execution when one or more robots failed to execute a task, and (2) a robot's ability to resume with a mission execution after it failed to execute a task. Former requires system robustness and the later robot's robustness. Since these two requirements are intertwined, this section describes the general approach which addresses both, in-the-system and on-the-robots robustness.

Due to the various control architectures supported in the SKIM coordination framework, i.e., centralized and distributed, the robustness concept differs for those two as well. Figure 5.10 illustrates those two robustness concepts. The first refers to the centralized and the second one to the distributed architecture styles. Independently of the selected architecture style, each concept has the following four steps (already introduced in previous section):

- task allocation,
- task selection,
- task execution (it is assumed that a robot fails here) and
- task generation.

The robustness concept combines existing mechanisms, like task allocation/selection and task generation, to model a robot's and system's behaviour in a case of failure.

Figure 5.10 (a) illustrates the robustness concept utilized by the centralized architecture style in the SKIM coordination framework. In this model, each robot selects a task from a central repository, i.e., the Semantic XVSM. Selected tasks are numerated and painted in blue in Figure 5.10 (a). The robot on the left side in Figure 5.10 (a) fails to execute the allocated task *t_1*. Successfully executed tasks are painted in green, i.e., tasks *t_2* and *t_3*, while the failed task, i.e., *t_1* is painted in red. To proceed with the regular mission execution without a loss of information, the robot which failed to execute task *t_1*, rewrites the same task *t_1* to the central repository where it will again be available to other robots. If the robot fails, a human operator will eventually notice that task *t_1* is not executed and will write it in the space. Although it failed to execute the task, the robot does not remain in failed state, rather it tries to fetch another task for execution. A reason for the failed execution could be an insufficient resource amount for a specific skill. Consequently, if the robot provides more skills it can still try to execute another task which requires another skill for which it has a sufficient resource amount.

On the other hand, Figure 5.10 (b) illustrates the robustness concept implemented in the distributed architecture style in the SKIM coordination framework. In this approach,

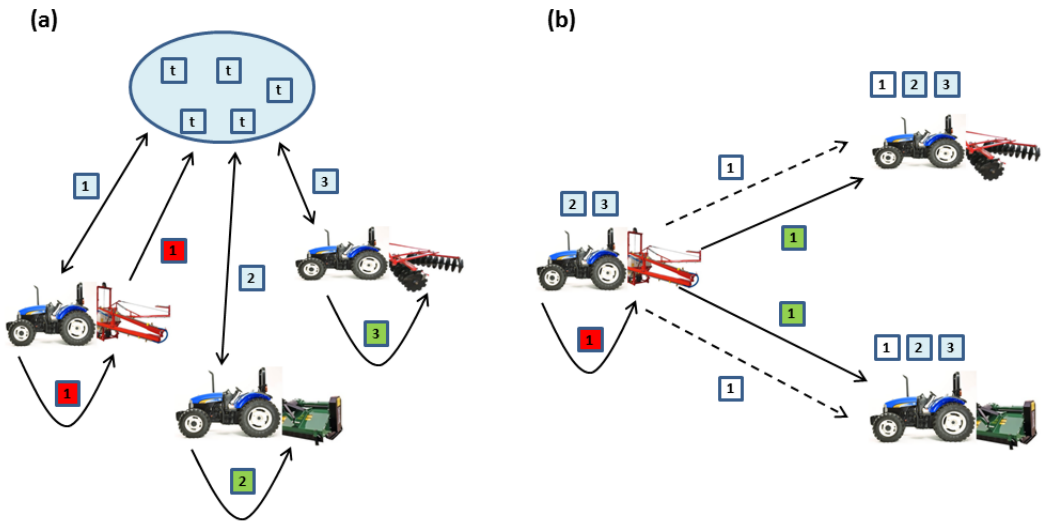

Figure 5.10: Robustness: (a) centralized systems, (b) distributed systems

all tasks are allocated to all robots no matter whether a robot can execute a task or not. The robot on the left side in Figure 5.10 (b) selects the task *t_1* for execution. At the same time, it notifies the other two robots that it is going to execute task *t_1*, i.e., dotted arrows with task *t_1* painted in white. This simple desynchronization component prevents two robots from executing a same task at the same time and thus ending up in a deadlock. Upon receiving notification, the other two robots delete task *t_1* from their local repositories. Meanwhile, the robot failed to execute task *t_1* which is now painted in red. Consequently, the robot sends task *t_1* to the other two robots in order to preserve the mission state and avoid information loss. This is depicted in Figure 5.10 (b) with two black arrows and task *t_1* painted in green. If the robot fails, a human operator will eventually notice that task *t_1* is not executed and will write it in the space where it will be automatically allocated to the other two robots, as it initially was.

Moreover, in case when a robot fails to execute a complex task, i.e., a task that requires more than one skill, it generates a new subtask with remaining, i.e., unexecuted, atomic skills. If the robot receives a complex task which requires two skills, i.e., it consists of two atomic tasks, and the robot provides both skills, and then it fails already on the first atomic task. In that case the robot will create a copy of the received task. Otherwise, if it fails while executing the second atomic task, assuming that the first is already successfully executed, it will create a new subtask. In that way the existence of duplicate tasks is avoided.

The coordination framework operates under the assumption that in both approaches a robot cannot fail during a task execution. This implies that the robot should calculate in advance whether it can handle a task or not and then proceed accordingly. This, however, limits the framework support for runtime failures. Nevertheless, both approaches ensure that the failed task reappears in the system and thus the others get an opportunity to

execute it. Moreover, after a robot fails to execute a task, it can still receive another task for execution. In that way, both requirements introduced at the beginning of section hold.

5.7 SKIM-based Implementations for Task Allocation

This section describes the design decisions of three different implementations of the SKIM coordination framework. Each implementation is based on Semantic XVSM and each implementation utilizes semantic capabilities to a different extent. Those implementations are named: (1) centralized SKIM (cSKIM), (2) distributed SKIM (dSKIM), and (3) hybrid SKIM (hSKIM). Although each implementation solves the same problem, i.e., task allocation and heterogeneous robots coordination in distributed environment, they use different approaches. cSKIM utilizes a central repository as a main coordination and task allocation component, while dSKIM is a completely distributed approach where robots collaborate to solve allocated tasks. Moreover, hSKIM is a hybrid approach which inherits some mechanisms from both, cSKIM and dSKIM. Additionally, hSKIM also considers a human and enables human-robot interaction.

Figure 5.11 illustrates all relevant components described in the previous sections. Illustrated components are not only limited to the design of SKIM framework, but can also be addressed when designing a distributed system for solving task allocation and coordination problems using semantic technologies in general. During a design phase of a coordination framework, two types of components have to be addressed: (1) main and (2) auxiliary components. Former ensure that each framework design begins with addressing coordinated entities and their interactions since this facilitates recognition of central components and, consequently, semantically models them. After that it is worth addressing a suitable transfer model as an underlying component which supports task allocation. Finally, modeling a corresponding decision-making mechanism leads to task execution. On the other hand, auxiliary components described in previous sections provide optional functionalities such as different models of resource matching and task generation. These could increase the efficiency of a coordination framework, but are not necessary for basic operations.

The design of cSKIM, dSKIM, and hSKIM framework implementations conforms to the modeling approach described in Figure 5.11. Design of each implementation addresses and models main and auxiliary components, respectively.

The following sections describe the design of each implementation, while Chapter 6 describes their implementation. Finally, Chapter 7 evaluates these three implementations.

5.7.1 centralized SKIM (cSKIM)

Centralized SKIM (cSKIM) uses a central task repository residing in the semantic space as a main notion that drives coordination. The central task repository provides different tasks which distributed robots fetch by issuing SPARQL queries.

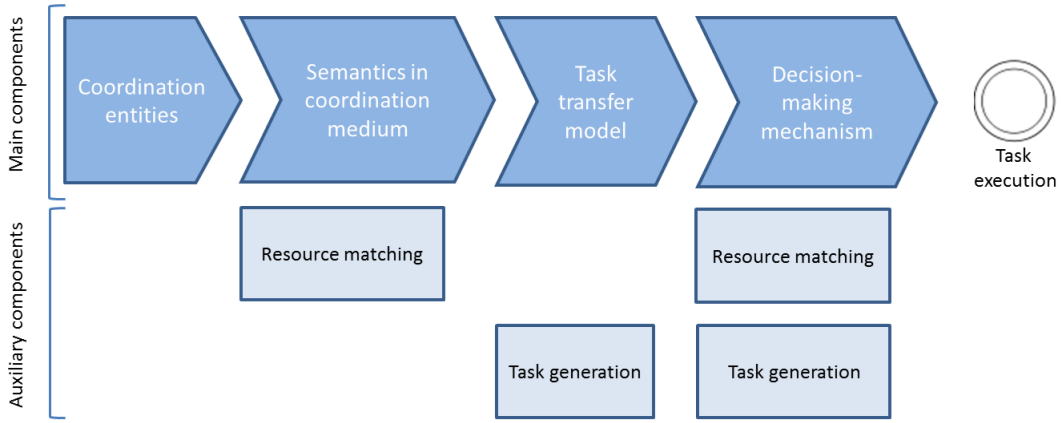

Figure 5.11: Components for modeling different versions of the SKIM framework

Figure 5.12 depicts the main entities (already addressed in Section 5.2.2 in Figure 5.3) which participate in cSKIM implementation and describes their interaction activities. First, a human operator generates a mission consisting of diverse tasks requiring different skills and resources. After that, the generated tasks are transferred to the semantic space where they are persisted in an underlying triplestore. To make the persisted tasks accessible, the semantic space supports SPARQL queries for selecting (allocating) tasks. At this point, a robot builds a SPARQL query which describes the skills the robot provides as well as the amount of resources it has. Upon receiving the SPARQL query from the robot, the semantic space executes the received query on the underlying triplestore to find tasks which match the skills and resources stated in the query. If there exists at least one result for the executed query, the coordination condition is fulfilled. The notion of coordination is perceived as an exclusive task allocation where only one requesting robot receives the resulting task.

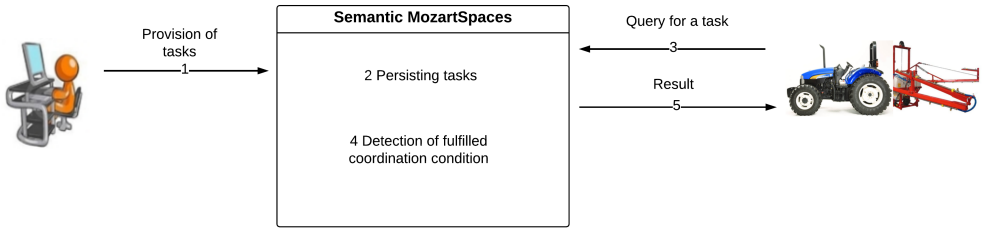

Figure 5.12: cSKIM - entity coordination

Architecture

Regarding the design of coordination, computation, and interaction activities, the cSKIM entities illustrated in Figure 5.12 conform to the modeling approach described in Figure

5.11. cSKIM addresses each of the main and auxiliary components in the following way:

- ***Coordination entities***: main entities participating in cSKIM implementation and their coordination and interaction activities are illustrated in Figure 5.12.
- ***Semantics coordination medium***: cSKIM utilizes the benefits of applying SKIM-RO for modeling diverse tasks and heterogeneous robots and thus enabling their interaction. Interaction is realized through SPARQL queries which perform matchmaking between tasks and robots. The matchmaking process supports all three resource matching models, i.e., exact, plug-in, and subsume.
- ***Task transfer model***: cSKIM is built on top of the query-based task transfer mode where a robot issues a SPARQL query to the semantic space and, consequently, receives a result. However, cSKIM supports task generation when an allocated task is only partially executed.
- ***Decision-making mechanism***: cSKIM supports only the on-the-robot decision-making approach because it neither interacts directly with other robots in a fleet nor does it interact with a human. Interaction with other robots is only implicit through the central task repository hosted in the semantic space.

There exist two coordination components: (1) the coordination model developed around the central task repository hosted in the semantic space and (2) the on-the-robot coordination model responsible for robot's internal activities and the interaction with the central task repository as well. The section introduced and described main building blocks utilized in constructing the cSKIM framework implementation. The following section introduces the design of the dSKIM framework.

5.7.2 distributed SKIM (dSKIM)

In contrast to cSKIM that uses a central task repository, dSKIM is based on a naive algorithm that distributes all tasks to all robots, which then notify each other about the tasks they are going to perform. Due to the lack of semantic-based task allocation, each robot receives a complete set of tasks, even the tasks it cannot perform.

Figure 5.13 introduces main entities participating in dSKIM implementation and describes their interaction activities. First, a human operator generates a mission consisting of diverse tasks requiring different skills and resources. After that, the generated tasks are transferred to the semantic space where they are mapped to robots in a fleet. In contrast to cSKIM with an exclusive task allocation to a robot, in dSKIM all tasks are mapped to all robots in the fleet resulting in non-feasible mappings. The non-feasible mappings subsume a task allocation where the matching level between provided and requested skills is characterized as fail. This means that a robot and a task do not have any overlapping skills.

Due to the mapping of all tasks to all robots, many colliding tasks emerge which require an extensive collaboration and cooperation between robots to execute those tasks.

Colliding tasks are those tasks which can be executed by more than one robot. Handling those tasks ensures that two or more robots do not execute a same task at the same time which could result in a collision between robots.

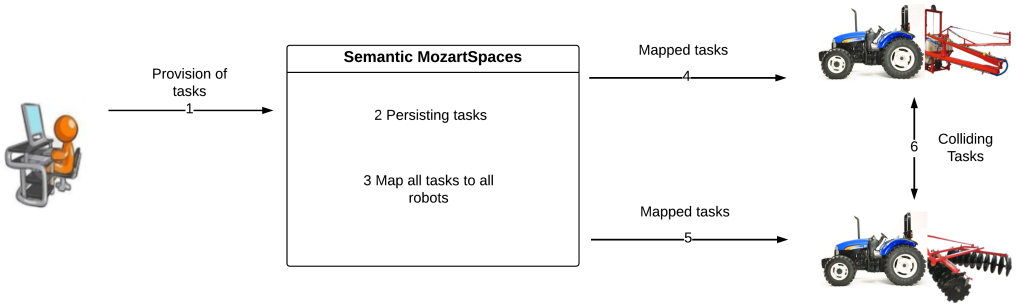

Figure 5.13: dSKIM - entity coordination

Architecture

Regarding the design of coordination, computation, and interaction activities, the dSKIM entities illustrated in Figure 5.13 conform to the modeling approach described in Figure 5.11. dSKIM addresses each of the main and auxiliary components in the following way:

- ***Coordination entities***: main entities participating in dSKIM implementation and their coordination and interaction activities are illustrated in Figure 5.13. Coordination activities between distributed robots are described in more detail in next sections.
- ***Semantics coordination medium***: dSKIM utilizes the benefits of applying SKIM-RO for modeling diverse tasks and heterogeneous robots and thus enabling their interaction. Robots issue SPARQL queries on local repositories where they only select tasks they are able to execute, while ignoring all other tasks. Matchmaking process supports all three resource matching models, i.e., exact, plug-in, and subsume.
- ***Task transfer model***: dSKIM utilizes a modified version of the inference-based task allocation transfer mode which does not use inference capabilities to allocate tasks, rather it just allocates all tasks to all robots. Moreover, dSKIM supports task generation when an allocated task is only partially executed. However, a generated task is not written to the central task repository as in cSKIM, rather, it is transferred to all robots in a fleet.
- ***Decision-making mechanism***: dSKIM supports both, on-the-robot and in-the-fleet, decision-making approaches because it also interacts directly with other robots in a fleet. Due to the handling of colliding tasks, the interaction with other robots is explicit.

Due to the distributed design of dSKIM, there exist two coordination components: (1) coordination model developed around the central task repository hosted in the semantic space and (2) on-the-robot coordination model responsible for robot's internal activities and the interaction with other robots as well. The section introduced and described main building blocks utilized in constructing the dSKIM framework implementation. The following section introduces the design of the hSKIM framework implementation.

5.7.3 hybrid SKIM (hSKIM)

In contrast to cSKIM and dSKIM which do not utilize coordination ontology and reasoning capabilities for task allocation, they utilize only resource ontology, in hSKIM, the task allocation is performed by reasoning on SKIM-CO. The result of reasoning is a set of tasks mapped to a set of matching robots. Robots receive tasks from the central task repository as in cSKIM, but locally run their own decision-making system and reason on SKIM-CO ontology to determine their own level of autonomy, e.g., whether to involve the centralized task allocation component to find collaborative robots and whether to involve the human in task assignment.

Figure 5.14 introduces the main entities participating in hSKIM and describes their interaction activities. First, a human operator generates a mission consisting of diverse tasks requiring different skills and resources. After that, the generated tasks are transferred to the semantic space where they are persisted in an underlying triplestore. Subsequently, SKIM-CO and a reasoner are used to allocate generated tasks to the robots and thus model the coordination activities between robots. It could happen that a human has to be consulted to provide additional information on task allocation. Upon allocating tasks to robots, these are transferred to robots, utilizing notifications in Semantic XVSM, which proceed further with the execution procedures.

During the task execution, it could emerge a need for the robot to consult a human on how to proceed with the execution of allocated task. In that case, the human receives a notification which requires him/her to help with the task execution. Required instructions are then propagated from the human back to the requesting robot. Moreover, the robot can also be allocated a colliding task which requires collaboration and a cooperation with other robots to execute the task. Colliding tasks are those tasks which can be executed by more than one robot. Handling those tasks ensures that two or more robots do not execute a same task at the same time which could result in a collision between robots. Moreover, similar to cSKIM and dSKIM, hSKIM also enables robots to generate new sub-tasks when an allocated task cannot be completely executed by one robot.

Architecture

Regarding the design of coordination, computation, and interaction activities, the hSKIM entities illustrated in Figure 5.14 conform to the modeling approach described in Figure 5.11. hSKIM addresses each of the main and auxiliary components in the following way:

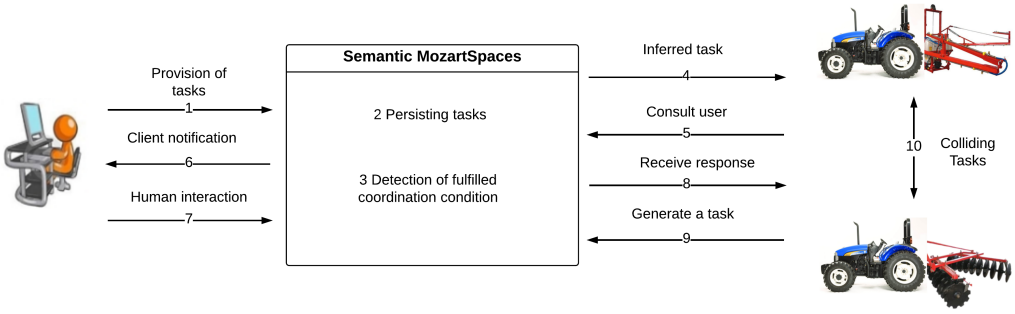

Figure 5.14: hSKIM - entity coordination

- ***Coordination entities***: main entities participating in hSKIM implementation and their coordination and interactions activities are illustrated in Figure 5.14. Coordination activities between a human, central task repository, and distributed robots, are described in more detail in next sections.
- ***Semantics coordination medium***: hSKIM utilizes the benefits of applying SKIM-RO for modeling diverse tasks, heterogeneous robots, as well as a human operator, and thus enabling their interaction. Additionally, hSKIM uses SKIM-CO for modeling coordination activities between robots as well as between a human operator and robots. Robots issue SPARQL queries on local repositories. Matchmaking process supports all three resource matching models, i.e., exact, plug-in, and subsume.
- ***Task transfer model***: hSKIM utilizes the inference-based task allocation transfer mode. Moreover, hSKIM supports task generation when an allocated task is only partially executed.
- ***Decision-making mechanism***: hSKIM supports both, on-the-robot and in-the-fleet, decision-making approaches. Additionally, a robot also interacts with a human operator. Due to the handling of colliding tasks, the interaction with other robots is explicit.

Due to the distributed design of hSKIM, there exist two coordination components: (1) coordination model developed around the central task repository hosted in the semantic space and (2) on-the-robot coordination model responsible for robot's internal activities and the interaction with other robots as well as with a human operator. Thus, the following two sections describe the centralized, on-the-robot, and human coordination patterns in more detail.

Hybrid coordination model

The hSKIM coordination model developed around the central task repository hosted in the semantic space. The central coordination model is responsible for fetching tasks,

persisting them, allocating them to robots, and transferring them to remote robots. Moreover, it could also consult a human operator to provide additional information during the task allocation process.

The system starts with an initialization procedure which creates an instance of Semantic XVSM where the fetched tasks are stored. After that, the configuration, described in Section 4.3.2, containing robots and provided skills, is loaded and used for building robot instances. To enable inference-based task allocation, the SKIM-CO is written in a container residing in the semantic space. The loaded configuration is then used to construct robots which are, together with received tasks, persisted in the same container where SKIM-CO resides. Having SKIM-CO, tasks, and robots, in a same container in the semantic space, enables inference-based task allocation. Finally, the initial procedure finishes by allocating tasks to robots and transferring them. However, during the task allocation process, a request for a human help can emerge. Due to the broader fleet knowledge, the human can provide additional information regarding the task allocation.

Due to the hSKIM support for executing tasks partially which, consequently, generates new tasks, an important part of coordination model is the component which receives data from running robots and processes them. Upon receiving data, the component checks whether the received data is instance of type *Task*. If the condition is true, it writes a received task in the local container where SKIM-CO resides. The received task will then be automatically allocated to a corresponding robot. Otherwise, it ignores received data. The received task denotes a sub-task created by a robot which was unable to completely execute an allocated task.

Robot coordination

The on-the-robot coordination model is responsible for robot's internal activities and the interaction with other robots in a fleet, as well as with a human operator. Thus, it complements the above described hybrid coordination model.

Same as in the hybrid coordination model, a robot starts with an initialization procedure which looks up the instance of Semantic XVSM created in the hybrid coordination model described in previous section. After the initialization, a loop begins in which the robot processes the allocated tasks received from the hybrid coordination component. Upon receiving tasks, the robot constructs a SPARQL query and then executes it on the local container. If the query does not return any task, the loop execution is paused until the next execution interval. The execution interval is manually configured as the number of milliseconds.

On the other hand, if the query returns a task to execute, it is first checked whether a received task is too complex for the robot to execute it. A task is too complex if it requires a human interaction and the robot is unable to support it. Otherwise, if a task is not too complex, the robot checks whether it has received a collision notification for that particular task from some other robot. The collision notification denotes that some other robot is about to process that task. In that case, the robot drops the tasks and waits until the next period to get another task from the local container. If there is no

relevant collision notification, the robot, able to support a human interaction, inspects whether a task requires a human interaction. If so, the human operator is consulted to provide a permission to the robot to execute the task. On the other hand, if there is no need for a human interaction, the robot inspects whether the task is colliding task and if so, it notifies other robots that it is going to execute that particular task.

A simple collision avoidance mechanism prevents that two colliding robots try to execute a same task at the same time and thus ending up in a deadlock. More precisely, the deadlock could result in a collision on a field. Current approach for collision avoidance in SKIM framework is based on following concepts: (1) utilizing random numbers to desynchronize distributed robots in time, (2) applying an area decomposition algorithm, described in next chapter, which prevents spatial interference between robots, and (3) each robot prioritizes local tasks based on a distance between a task and itself. However, this approach requires additional improvements which are addressed in future work.

After notifying colliding robots and interacting with a human (if necessary), the robot validates permission to execute the task and proceeds accordingly. The robot validates that it has enough resources to execute the assigned task. If it has adequate amount of resources, it proceeds with checking the overlapping skills, i.e., skills requested by the received task and the skills it provides. If the task requires a skill which the robot does not provide, then the skill is marked as a missing skill. In the case of missing skills, the robot creates a new task, i.e., a sub-task, containing the missing skills and writes the new task to the container residing in the instance of Semantic XVSM which then dynamically performs inference-based task allocation. Finally, the robot executes the allocated task. It executes the task either completely or partially. The task is completely executed if there are no missing skills, and partially otherwise.

Due to the extension of coordination paradigm from the central part of the system to the robots in the fleet, each robot has a component which facilitates processing of data received from other robots. It first checks whether the received data is of type *Mapped Task* which denotes tasks allocated from the central part of the system. If this is the case, the mapped tasks are written to the local container and colliding tasks are detected. Otherwise, the component checks whether the received data is type of *Task* denoting a sub-task received from another robot. In that case the component resolves a colliding task by removing it from the local repository. The last possibility is that a received data could be of type *ConsultUser* containing a permission to execute a task. If not, then the received data denotes a task allocated directly by a human. To avoid adding more complexity to the diagram, a case which ignores received data is omitted from the diagram. It is important to notice that the component for data receiving and processing runs in parallel with the component for querying and executing tasks.

Human interaction

The human operator in hSKIM has twofold purpose:

- to decide whether a robot can execute a specific task. The human receives data denoting that the robot has requested a permission to execute certain task, and

- to manually assign a task to a robot. The human receives data from the hybrid coordination component to provide additional information on allocating a specific task.

In first case, the human operator utilizes a simple algorithm to decide on a permission to execute a certain task. The algorithm is based on a cost function including following parameters: (1) a skill matching degree between a robot and a task, and (2) resource availability, i.e., resources for completing a task. In later case, the human operator utilizes knowledge on the fleet, thus complementing the inference-based task distribution. Regardless of the occurring case, a response is always propagated directly to the component it emerged on, i.e., either a robot or the hybrid coordination component.

5.8 Comparison of SKIM Coordination Approaches

The design chapter is concluded with a tabular comparison of the three SKIM-based coordination approaches. Table 5.2 compares them against the most prominent components represented in each of them.

Table 5.2: Comparison of SKIM coordination approaches

Approach	Semantic support	Task transfer	Decision-making	Human operator
cSKIM	SKIM-RO	query-based	on-the-robot	-
dSKIM	SKIM-RO	custom	on-the-robot in-the-fleet	-
hSKIM	SKIM-RO SKIM-CO	inference-based	on-the-robot in-the-fleet	✓

Evaluating the cSKIM and dSKIM architectures, it could be noticed that both, cSKIM and dSKIM, only provide a limited ontology support through the utilization of SKIM-RO and a SPARQL support. On the other hand, hSKIM utilizes also SKIM-CO for coordination purposes and a human interaction as well. Moreover, hSKIM supports inference-based task transfer as a more sophisticated method than the query-based and the custom task transfer methods. It is more sophisticated because it relies on the underlying SKIM-CO.

cSKIM is also limited from the perspective of interaction with other entities. It does neither support interaction with other robots in a fleet nor with a human. Is solely relies on on-the-robot decision-making mechanisms without any support to contact other robots. In contrast to cSKIM, dSKIM supports both on-the-robot and in-the-fleet decision-making mechanisms. Therefore, it supports an extensive coordination and collaboration with other robots. Finally, hSKIM also supports both on-the-robot and in-the-fleet decision-making mechanisms. However, in addition to an an extensive coordination and collaboration with other peers, it supports human interaction as well.

hSKIM implementation uses SKIM-CO ontology-based model of shared knowledge to enable collaboration between a human operator and robots. On contrary, neither does the cSKIM nor dSKIM support human interaction.

Due to the emerging support for human operators in mixed human-robot teams where humans and robots can operate as peers, hSKIM implementation represents a framework which meets the posed requirements for a flexible human interaction. Moreover, due to the semantic support, hSKIM is flexible and thus can easily be adapted to changing requirements.

Next chapter introduces implementation details of the components addressed in the previous section. Described components are enablers to build a generic architecture that the developed SKIM framework complies with. Moreover, implemented are three different coordination approaches based on the semantic extension of the Space-Based Computing architectural style.

CHAPTER 6

Implementation Details of SKIM Framework

This chapter complements the previous one while it demonstrates implementation details of the components addressed there. The chapter starts with the description of components that build the generic architecture that the developed framework complies with (Section 6.1). To have a better understanding of task allocation and robots coordination models which utilize shared knowledge, a formal task allocation model is given in Section 6.2. Notations presented in the formal task allocation model are later used in algorithms for robot-robot and robot-human interactions.

Due to the spatial interference emerging in unstructured environments where multiple robots operate, the developed framework utilizes an area decomposition algorithm to divide a working area into cells which are dynamically assigned to robots and thus decreases spatial interference between robots. The developed area decomposition algorithm is introduced in Section 6.3.

As the shared knowledge model developed in this thesis is based on ontologies, it provides uniform input for algorithms devised to map tasks to robots and facilitates robot-robot and robot-human interactions. Devised algorithms are described in Section 6.4. A human role and its behaviour is described in Section 6.5. Finally, implementation details of the different coordination frameworks are introduced in Section 6.6

6.1 Proposed System Architecture

The proposed framework architecture is based on the Model-Driven Architecture approach built on ontology-enabled components illustrated in Figure 6.1. Following are the ontology-based components which constitute the developed framework: (1) automatic ontology-based *Task* generation, (2) ontology-based *Scenario* classification, (3) manual ontology-based *Robot* and *User* generation, i.e., configuration, (4) Semantic XVSM

coordination framework [26], and (5) ontology-based model of shared knowledge. Most of these components have already been introduced in Section 4.

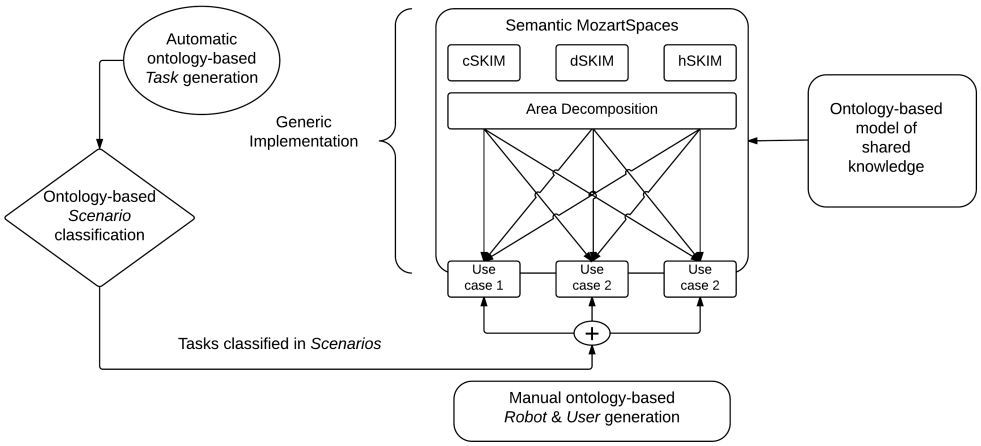

Figure 6.1: SKIM architecture

The framework is based on the ontology described in Section 4 which structures the model into two specific ontologies: SKIM Resource Ontology (SKIM-RO) and SKIM Coordination Ontology (SKIM-CO). SKIM-RO describes resources, including robot capabilities and task requirements, and SKIM-CO describes coordination constraints for robot-robot and robot-human interactions. Accordingly, SKIM-RO defines classes *Task and Scenario*, and SKIM-CO classes *Robot, User, and Capability.* **Automatic ontology-based task generation** is a software component which produces instances of class *Task* defined in SKIM-RO. The *Task* description includes three properties: (1) number and kind of required skills, (2) amount of resource per skill, and (3) spatial position. Number and kind of required skills is the main parameter utilized for task allocation. The **ontology-based scenario classification** uses class *Scenario* from SKIM-RO and automatic reasoning to classify generated tasks into three different scenarios with respect to different task complexities. A semantic description of scenario is a defined subclass of class *Scenario* with restrictions on properties from SKIM-RO. Classified tasks and **manually generated instances of classes *Robot* and *User*** define use cases which are input for different implementations of **coordination framework** referred to as centralized SKIM (cSKIM) [26], distributed SKIM (dSKIM), hybrid SKIM (hSKIM). Manually generated instances of classes *Robot* and *User* defined in SKIM-CO are referred to as a **configuration** (see Section 4.3.2).

In contrast to cSKIM that uses a central task repository, dSKIM is based on a naive algorithm that distributes all tasks to all robots, which notify each other about the tasks they are going to perform. Due to the lack of semantic-based task allocation, each robot receives a complete set of tasks, even the tasks it cannot perform. Neither does the cSKIM nor dSKIM support human interaction. On the other hand, the hSKIM implementation uses **SKIM-CO ontology-based model of shared knowledge** among humans and

robots. In hSKIM the task allocation is performed by reasoning on SKIM-CO. The result of reasoning is a set of tasks mapped to a set of matching robots. Robots receive tasks from the central task repository as in cSKIM, but locally run their own decision-making system and reason on SKIM-CO ontology to determine their own level of autonomy, e.g., whether to involve the centralized task allocation component to find collaborative robots and whether to involve the human in task assignment.

6.2 Task Allocation Model

Let $R = \{r_1, ..., r_\mathrm{m}\}$ be a set of robots, $T = \{t_1, ..., t_\mathrm{n}\}$ set of tasks, and $S = \{s_1, ..., s_\mathrm{o}\}$ set of skills. The set of skills of a robot $r_\mathrm{i} \in R$ is $S_i^\mathrm{r} \subseteq S$ and the set of skills required by a task $t_\mathrm{j} \in T$ is $S_j^\mathrm{t} \subseteq S$. Then the task allocation is defined as the mapping $\forall t_\mathrm{j} \in T$, $1 \leq j \leq n$, to a set of robots R_j^t which satisfies following:

$$S_j^\mathrm{t} \subseteq \cup_{\forall r_\mathrm{x} \in R_j^t} S_\mathrm{x}^\mathrm{r} \tag{6.1}$$

The skills required by a task t_j will be matched if there exists set of robots R_j^t such that the union S_x^r of skills provided by each robot $r_\mathrm{x} \in R_j^t$ contains the set S_j^t. If $|R_j^t| > 1$, then collaboration and coordination between robots in R_j^t is required. Our assumption in modeling human knowledge about task allocation is that a human is to be involved in solving task t_j if $|S_j^t| \geq k$. Parameter k denotes a threshold for a human interaction with a fleet and defines class *User* with a restriction on property k in SKIM-CO.

6.3 Dynamic Area Decomposition for Task Allocation

To prevent colliding robots, i.e., robots that are assigned a same task, from interfering with each other in a field, it is beneficial to introduce spatial and time dependencies between tasks. On the one hand, as mentioned in Chapter 1, SKIM coordination framework does not address the latter. On the other hand, to reduce spatial interference between robots, the coordination framework implements area decomposition algorithm based on a computational geometry technique of Voronoi diagram [5]. This approach is applicable to domains where geographical positions of robots and tasks are known, which to a great extent corresponds to the use case addressed throughout this thesis and described in Chapter 1.

Construction of Voronoi diagram is based on the Euclidian distance between two points p and q by $dist(p, q)$. Let $P = \{p_1, ..., p_\mathrm{n}\}$ be a set of n distinct points in the plane, where points are the sites. Voronoi diagram of P is defined as the subdivision of the plane into n cells, one for each site in P, with a property that a point q lies in the cell corresponding to a site p_i if and only if $dist(q, p_\mathrm{i}) < dist(q, p_\mathrm{j})$ for each $p_\mathrm{j} \in P$ with $j \neq p$. More details on the construction of Voronoi diagram can be found in [5].

The area decomposition algorithm implemented in the coordination framework considers the set of n tasks $T = \{t_1, ..., t_\mathrm{n}\}$ as points in P. Each task is composed of the random number of equal square building blocks. Since tasks can have different shapes

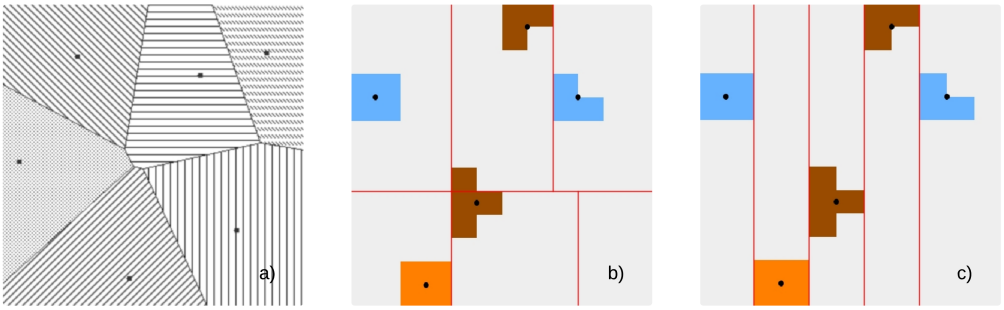

Figure 6.2: Area decomposition

and sizes, for each task $t_i \in T$, $1 \leq i \leq n$ calculated is a point $rt_i(x, y)$ denoting the center of mass. Those centers of mass are used for calculating the Euclidian distance between each task and for constructing the Voronoi cell for a task. Figure 6.2 a) illustrates a Voronoi cell which is a polygon with an arbitrary shape.

Due to the requirements posed by the addressed precision agriculture scenario, i.e., an agricultural field as an operating environment where robots are limited to performing parallel trajectories and turns on the field edges, the following constraints are imposed on a cell construction:

- the shape of a cell is either a rectangle or a square, and
- width and height of a cell are equal to the length of the building block of a task multiplied by a integer value.

Described constraints forced the adaptation of existing Voronoi diagram. Thus, developed is the algorithm 6.1 which addressed the above constraints. Moreover, rectangle or squared cells enable robots to inspect whole cells and not just a task in it.

A cell construction in lines $11 - 16$ creates lines, i.e., cell borders, parallel to either x or y axis thus conforming to the constraint that each cell is either a square or a rectangle. As a result of introduced constraints, it could happen that some tasks partially lay out of their cells and thus span to other cell. A brown object in the middle of Figure 6.2 b) denotes a spanning task which requires robots collaboration for the task execution.

Moreover, a parameter k instructs the decomposition process to either prefer vertical or horizontal decomposition. Figures 6.2 b) and c) illustrate how the shape of cells depends on the parameter k. When the horizontal decomposition is preferred, i.e., $k = 5$, cells are more squared (Figure 6.2 b)). On the other hand, cells are long and narrow when vertical decomposition, i.e., $k = 9$, is active (Figure 6.2 c)). As it could be seen comparing Figures 6.2 b) and c), choosing the right value for the parameter k could result with no spanning tasks. In contrast to a spanning task, one robot can execute a task which does not span multiple cells.

The outcome of algorithm is a list of lines denoting cells' borders. Because the algorithm never drops all lines, it always yields a list of lines denoting cells' borders.

Algorithm 6.1: Area decomposition algorithm

Data: set of tasks T
Result: a list of lines denoting cells

order all tasks t_{i} and $t_{\mathrm{i+1}} \in T$ such that $rt_{\mathrm{i}}x < rt_{\mathrm{i+1}}x$;
for *each task* $t_i \in T$ **do**
    create a line between points rt_{i} and $rt_{\mathrm{i\text{-}1}}$;
    if *line* $rt_i rt_{i\text{-}1}$ *intersects with existing lines* **then**
        choose a task t_{j}, $j < i$, such that a line $rt_{\mathrm{j}}rt_{\mathrm{i}}$ does not intersects with existing lines;
        set $rt_{\mathrm{i\text{-}1}} = rt_{\mathrm{j}}$;
    else
        retain line;
    end
    calculate the point h such that $dist(h, rt_{\mathrm{i}}) = dist(h, rt_{\mathrm{j}})$;
    create a line denoting a border between cells containing t_{i} and $t_{\mathrm{i\text{-}1}}$;
    if $|dist_y(rt_i, rt_{i\text{-}1})| < k$ *or* $|dist_x(rt_i, rt_{i\text{-}1})| > k$ **then**
        create a new vertical line parallel to $y - axis$ passing through the point h;
    else
        create a new horizontal line parallel to $x - axis$ passing through the point h;
    end
    if *the new line does not intersect with existing lines* **then**
        add the line to the list containing cell borders;
    else
        drop the new line;
    end
end
return *the list of lines denoting cells*;

However, the number of lines denoting borders depends on the setup, i.e., the number of tasks, their size, and position as well. In particular, it influences the number and shape of cells.

6.4 Shared Knowledge in Robotic Fleet

The SKIM framework is designed with the objective to model shared knowledge as a basis for adaptive autonomy in mixed teams. The semantic approach drives the modeling of shared knowledge which enables the collaboration activities between involved entities by means of ontologies: SKIM-RO and SKIM-CO. Hence, these ontologies are used as the model of shared knowledge and the decisions are results of automated reasoning on them.

Previous section describes how the SKIM-RO is utilized for semantically annotating heterogeneous resources and thus providing uniformly described tasks and robots. Uni-

formly described resources represent inputs for the algorithms described in this section which utilize SKIM-CO for task allocation and coordination activities between robots as well as between a human and robots.

This section starts with the implementation of inference-based based task allocation process which practices reasoning capabilities on SKIM-CO to allocate tasks to robots. After that, described is the implementation of an algorithm which enables robot-robot interactions. The section is concluded by describing the implementation of human interactions with the fleet.

6.4.1 Shared Knowledge Model for Task Allocation

Algorithm 6.2 is the inference-based task allocation process which receives instances of classes *Task* and *Robot* as well as the SKIM-CO, i.e., the configuration, as an input data. Received task are previously processed in the area decomposition algorithm where they were utilized to calculate operating cells for robots. The Algorithm 6.2 is performed in the Semantic XVSM.

Algorithm 6.2: Shared knowledge for task allocation

```
   Data: map a set of tasks T to a set of robots R using SKIM − CO
   Result: a map with tasks allocated to robots
 1 write SKIM − CO to container C;
 2 for each task t in T do
 3 |  write t to C;
 4 end
 5 for each robot r in R do
 6 |  write r to C;
 7 end
 8 doReasoning;
 9 for each Capability class c_h in SKIM − CO do
10 |  get classified tasks cT_h;
11 |  get classified robots cR_h;
12 |  if cT_h is empty then
13 |  |  consultUser;
14 |  else
15 |  |  for each robot r in cR_h do
16 |  |  |  add tasks in cT_h to r;
17 |  |  end
18 |  end
19 end
20 notify robots in R on mapped tasks cT ⊆ T;
```

In this algorithm the SKIM-CO ontology is written in container C to enable reasoning capabilities and thus inference-based task allocation. Container is a basic concept of

Semantic XVSM. SKIM-CO can have multiple *Capability* classes $C = \{c_1, ..., c_p\}$ where each class $c_h \in C$ is defined with the specific type of skills required by received tasks, e.g., one *Capability* class can be defined to encompass only tasks requiring skill $s_x \in S$ and another class requiring skill $s_y \in S$. After that, both tasks from T and robots from R are written to container C. Upon writing all received data in C, reasoning is performed on SKIM-CO, tasks T, and robots R. The result of reasoning is a set of tasks cT mapped to a set of matching robots cR.

Additionally, task $t_j \in T$ and robot $r_i \in R$ are classified in the class *User* if $|S_j^t| \geq k$ and $|S_i^r| \geq k$.

When tasks and robots are classified, each *Capability* class $c_h \in C$ from SKIM-CO is examined to fetch classified tasks cT_h and robots cR_h. If there exists a task which is not classified into any *Capability* class c_h, e.g., partially executed task, the human is consulted to assign the tasks to a robot. Otherwise, each robot $r \in cR_h$ is assigned a set of tasks cT_h. In the end, Algorithm 6.2 utilizes notification mechanisms from Semantic XVSM to notify all robots in R on mapped tasks $cT \subseteq T$ and thus provides an input for Algorithm 6.3 described in next section.

6.4.2 Shared Knowledge Model for Robot-Robot Interactions

Algorithm 6.2 transfers inferred robot-task mappings to distributed robots. Robot-task mapping is defined as a data structure $Map < Robot, Task[\,] > ts$ where a robot is a key and a value is an array of tasks allocated in Algorithm 6.2. The map is an input to Algorithm 6.3.

Robot $r \in R$ running Algorithm 6.3 filters its own tasks from the received map ts by checking which key k in map ts is equal to r. After fetching own tasks $ts.tasks$, robot r writes them in a local container Cl. Additionally, there are tasks which could be executed by multiple robots, i.e., collision tasks. Task $t_j \in T$ with a set of required skills S_j^t, is a collision task if there exists set of robots $R_j^t = \{r_1, ..., r_q\}$ with the union of provided skills S_i^r, where $|R_j^t| > 1$, such that Eq. (6.1) holds. Since each robot r knows about other robots and their tasks, it detects collision tasks $Tcoll$. When robot r takes task t from container Cl, it checks whether it received a collision notification on the same task t from a colliding robot. At this moment it is important that colliding robots are desynchronized to avoid that two or more robots take the same colliding task at the same time. This challenge is addressed in Section 5.7.3. If there is no collision notification nor the task is too complex for robot r, the robot can proceed. Task t is too complex for robot r if t is classified in class *User* and if $|S^r| < k$. Latter condition means that a robot r has to few skills to autonomously execute task t.

A robot r consults a human for assistance also if there are ambiguities during task t execution. In that case the human runs an algorithm which calculates whether robot r has compliant skills and needed amount of resources to execute task t. Upon receiving confirmation from human, robot r is allowed to proceed with regular execution where it checks whether task $t \in Tcoll$. If this holds, robot r notifies only colliding robots $Rcoll \subseteq R$ that it is going to execute task t. This is different compared to dSKIM where

Algorithm 6.3: Shared knowledge for robot-robot interaction

```
   Data: a map with robots as keys and assigned tasks as values
 1 get assigned tasks ts.tasks;
 2 write ts.tasks to local container Cl;
 3 detect colliding tasks Tcoll and robots Rcoll in Cl;
 4 while Cl is not empty do
 5 |   take task t from Cl;
 6 |   if (|S_j^t| ≥ k and |S_i^r| < k) or collNotification then
 7 |   |   dropTask;
 8 |   else
 9 |   |   if |S_j^t| ≥ k and |S_i^r| < k then
10 |   |   |   consultUser;
11 |   |   else
12 |   |   end
13 |   |   if t ∈ Tcoll then
14 |   |   |   send collNotification;
15 |   |   else
16 |   |   end
17 |   |   execute task t;
18 |   end
19 end
```

all robots are notified upon task execution. After necessary interaction with the human and other robots, r executes task t.

A simple collision avoidance mechanism prevents that two colliding robots try to execute a same task at the same time and thus ending up in a deadlock. The deadlock could result with two robots running into each other on a field. Thus, the current approach for collision avoidance in SKIM framework is based on following concepts: (1) utilizing random numbers to desynchronize distributed robots in time ensuring that only one robot is always the first to take a colliding task and notify the others, (2) applying an area decomposition algorithm, described in Section 6.3, which prevents spatial interference between robots, and (3) each robot prioritizes local tasks based on a distance between a task and its current location. However, this approach requires additional improvements which are addressed in future work.

6.5 Human Role in Adaptive Autonomy

Operating a robotic fleet is a cognitively demanding task that requires efficient user-fleet interface for decision-support in control and monitoring, diagnosis, problem detection and resolution, complementing the autonomous decision making that the robotic fleet units are capable of. Particularly for applications of robotic fleets in open uncontrolled

environments, the role of the human-operator needs to be carefully defined and supported, taking into account uncertainties in all phases of the system operation, including also human potential in resolving and equally creating critical situations. The most challenging future scenario is the one in which humans and autonomous robots with self-optimizing and learning capabilities collaborate in joint tasks. Currently however the main focus is on control with different levels of autonomy.

As described in the precision agriculture use case in Chapter 1, there are two main roles assigned to human operators, i.e., a fleet owner who leases out his heterogeneous robotic fleet, i.e., a configuration, consisting of multiple tractors with different implements able to perform different tasks, and a farmer who owns a field which has to be cultivated. The fleet owner can also act as fleet operator responsible for selecting a suitable strategy for a field treatment. When during a mission execution a robot tries to execute a task that requires a human intervention, it has to consult the operator who decides whether the robot has necessary skills and resources to execute the task. The robot consults the fleet operator for assistance if there are ambiguities during the execution of the task. In that case the operator decides whether the robot has compliant skills and needed amount of resources to execute the task. Upon receiving confirmation from the human, the robot is allowed to proceed with regular execution of the task.

The section first describes a system developed within the project RHEA which facilitates a human operator in gaining situational awareness by providing it with relevant fleet information. Afterwards, described are two implementations of human-involved decision algorithms which utilize data fused from the shared fleet knowledge with the knowledge acquired in awareness processes.

6.5.1 Gaining Situational Awareness

Human mobility is a great advantage in the context of acquiring situational awareness in robotic fleet, as the human has direct access to the environment and, in some situations, is co-located with the robots too. The human mobility within the operating environment allows for fast knowledge acquisition by a direct interaction with the elements in the environment.

In [31] a system is developed which enriches a human in-field experience and understanding of mission by adding information beyond what can directly be perceived by supervising the mission. Designed are two complementary information components that use different data sources: (1) the human-robot interaction (HRI) system enables the human operator in a field to obtain the current robot status, e.g., position, speed, heading, the status of the implements, and (2) the communication network monitoring system which offers information regarding the quality of communication network collected from the wireless network routers. In particular the network monitoring interface helps the human to understand and mitigate communication uncertainties which may arise during the mission due to the problems in transmission link quality, which are characteristic for the volatile and unstructured environments and are hard to detect and can jeopardize a whole mission.

Figure 6.3 illustrates the system which enriches the human perception of operating environment. It is deployed on distributed robots which utilize wireless routers to communicate with the central repository and the human operator as well. The system consists of distributed client applications running on wireless routers which collect both, data coming from an underlying decision-making system as well as network data related to e.g., the network topology, signal strength, bitrate and traffic load, directly from the routers. Collected data is transferred to the central repository that processes and stores received data and also make it accessible for the human operator.

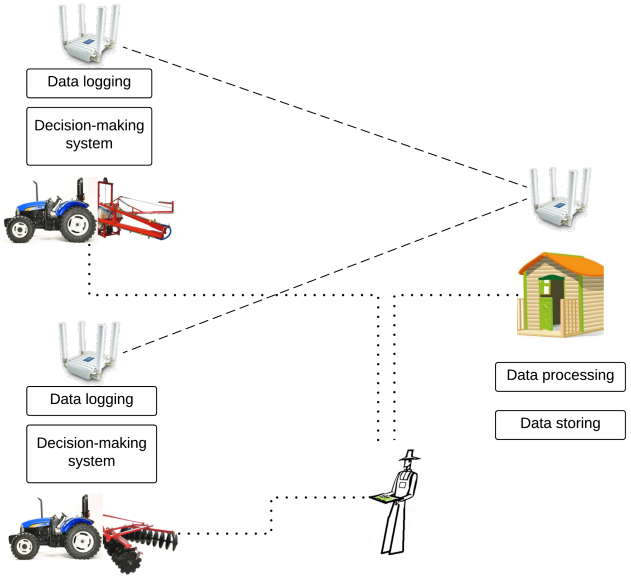

Figure 6.3: Gaining situational awareness

As a result, the human operator can continuously monitor network topology and transmission link quality. Based on the network information the human can predict and prevent remote robots of losing a connection with a central station and thus being out of service. Combining information from three different sources: (1) data retrieved directly from robots utilizing HRI system, (2) visual perception of the in-field situation, and (3) network data obtained through the network monitoring system, the human can make decisions about whether to increase or decrease robot's autonomy. By doing so, the human can increase the overall mission efficiency by reducing the robots outages and increasing their utilization based on the network information which helps to predict and prevent remote robots of losing a connection and thus being out of service.

6.5.2 Shared Knowledge Model for Robot-Human Interactions

Algorithms 6.2 and 6.3 introduced two different models of human interaction for facilitating decisions emerged due to the ambiguities occurred during the task allocation and robot-robot interaction.

On-the-human extended task allocation

Algorithm 6.2 triggers reasoning on SKIM-CO which classifies two types of tasks: (1) input tasks at the beginning of a mission, and (2) ad-hoc tasks which emerge during the mission due to the skills mismatch between a robot and a selected task, i.e., partially executed tasks. The classification has two outcomes independently of the task type: (1) a task is mapped to a robot, or (2) there is no suitable robot for the given task. The outcome of mapping depends on SKIM-CO. However, an unmapped task does not imply that the task could not be executed. In that case, the task is assigned to the human who is assumed to have knowledge to solve the task allocation problem. In the simulation the human actor performs the task allocation algorithm which utilizes knowledge on the fleet, thus complementing the inference-based task distribution.

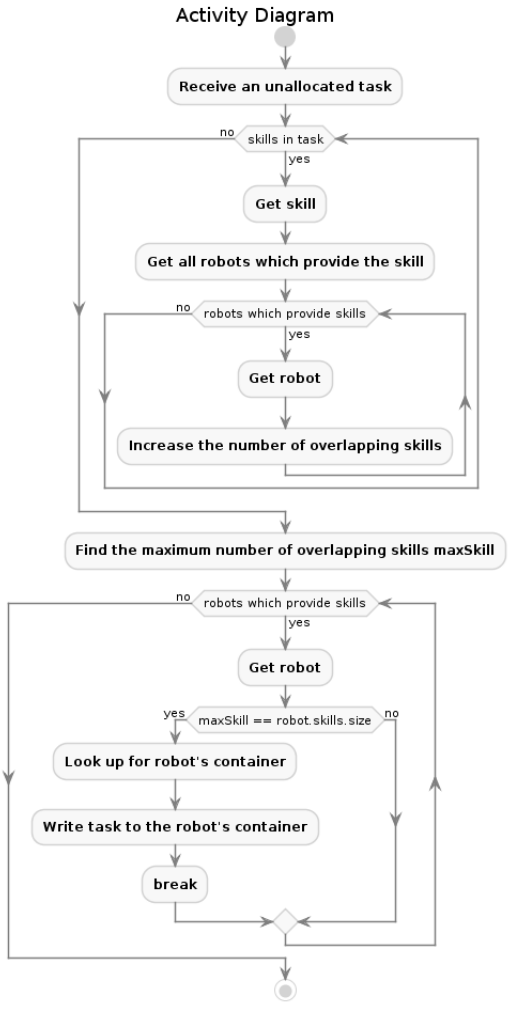

Figure 6.4: On-the-human extended task allocation

Figure 6.4 illustrates on-the-human extended task allocation algorithm which is triggered after a robot consults a human for help. Although the human decision process is a cognitive task, in the scope of this thesis it is implemented as the algorithm. This is due to the fact that the SKIM framework is only tested in simulations without an opportunity to test it on a real robotic fleet operating in a field. In such a case human's cognitive capabilities would be beneficial for task allocation.

First, for each requested skill in an unallocated task, the algorithm retrieves all robots that provide that skill. Retrieved robots are stored in a data structure map where for each robot, stored as a key, there is a counter, stored as a value, denoting the number of overlapping skills. Overlapping skills are those skills, required from the unallocated task, which are same as the skills provided by a robot. After that, the algorithm finds the maximum number of overlapping skills denotes as *maxSkill*. Finally, from the set of robots that provide the overlapping skills, selected is one which has the highest number of overlapping skills, i.e., who has the number of overlapping skills equal to *maxSkill*. To allocate the task to the robot, the human has to use look up mechanism provided in Semantic XVSM to get a reference to the robot's container and to write the task there. The execution of algorithm finishes after allocating the task to a first robot which has the number of overlapping skills equal to *maxSkill*. Due to the ambition to decrease the number of colliding tasks and thus the exchange of coordination messages between robots, an unallocated task is allocated to only one robot. On-the-human extended task allocation process utilizes knowledge on the fleet to facilitate task allocation thus complementing the inference-based task distribution.

On-the-human extended robot coordination

In algorithm 6.3, if a robot r takes a task t from container Cl which requires $|S^t| \geq k$ skills, robot r will have to consult the human regarding the allocation of task t. Robot r is capable to execute task t if $|S^r| \geq k$. Otherwise, if a robot with a higher matching degree exists, it will be chosen. The decision-making algorithm performed by the human actor is based on a cost function including following parameters: (1) skill matching degree between a robot and a task, and (2) resource availability, i.e., resources for completing a task.

Figure 6.5 illustrates on-the-human extended robot coordination algorithm which has first part similar to the algorithm described in Figure 6.4. For each skill required in a task received in a consult request sent by a robot, retrieved are all robots that provide that skill. Retrieved robots are stored in a data structure map where for each robot, stored as a key, there is a counter, stored as a value, denoting the number of overlapping skills. Overlapping skills are those skills, required from the task received in the consult request, which are same as the skills provided by a robot. After that, the algorithm finds the maximum number of overlapping skills denotes as *maxSkill*.

The part which differs from the algorithm described in Figure 6.4 is the condition which is fulfilled when a robot with the highest number of overlapping skills is found and when the found robot is the same as the robot who sent the consult request. If a robot that sent a consult request has the highest number of overlapping skills, the procedure is

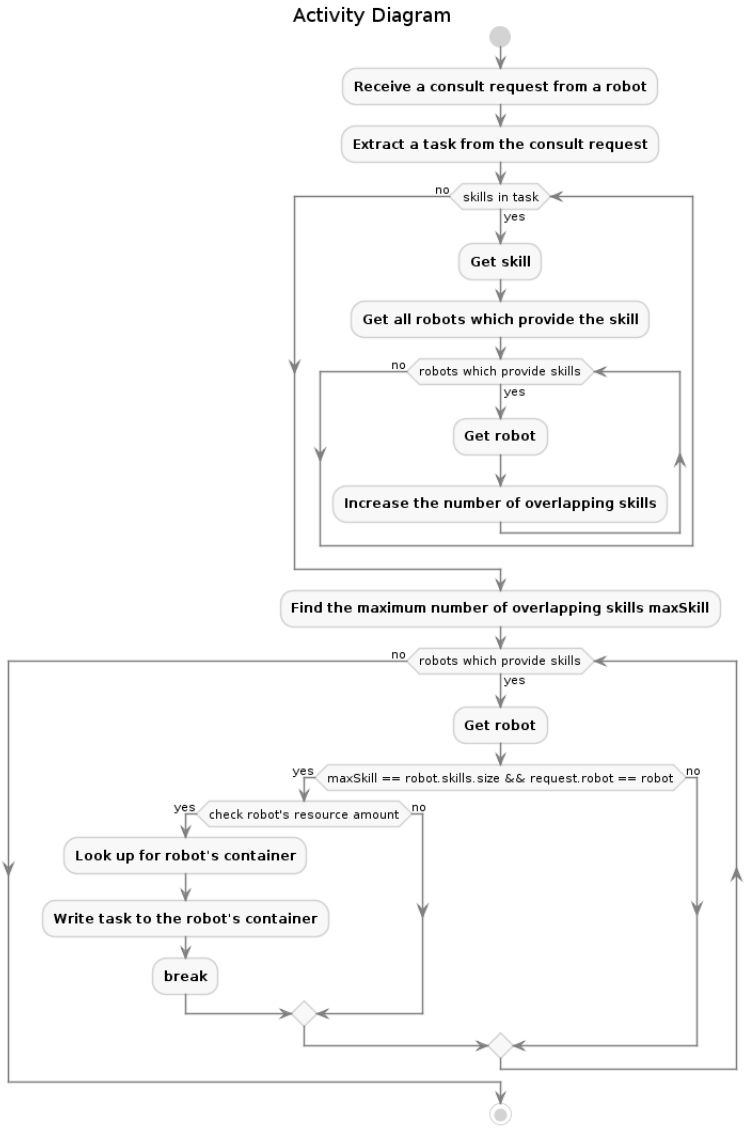

Figure 6.5: On-the-human extended robot coordination

the same as in the algorithm in Figure 6.4. The human has to use look up mechanism provided in Semantic XVSM to get a reference to the robot's container and to write the task there. Otherwise, if another robot with a higher matching degree exists, requesting robot will not get a permission to execute that task and eventually a robot with highest matching degree will consult a human for a help and it will be allocated a task.

6.6 Implementation of Coordination Approaches for Task Allocation

The section presents implementation details of each SKIM-based implementation of coordination framework. Each implementation builds on the semantically annotated entities reflecting tasks in an operating environment and robots executing them. Semantically described resources enable a robot to execute the context enriched SPARQL query when selecting tasks to execute. Depending on the implementation, the SPARQL query is either executed on a local or on a remote container which hosts the set of tasks. Selected tasks are further used as an input for different coordination mechanisms which depend on the implementation.

The section starts with the description of implementation details which pertain to cSKIM. Described are the SPARQL query for selecting tasks and a sequence diagram which illustrates involved entities and their interactions. Then, the implementation details of dSKIM are presented, as well as an associated sequence diagram. The section is concluded with the implementation of hSKIM which builds on the task allocation and coordination algorithms described in Section 6.4 and human-interaction mechanisms introduced in Section 6.5, respectively.

6.6.1 centralized SKIM (cSKIM)

As already mentioned in the design chapter, cSKIM uses a central task repository as a coordination paradigm between distributed robots. The implementation of this central task repository is based on Semantic XVSM with SPARQL interface and an underlying triple store (Jena). The main part of Semantic XVSM is the data model that exposes the mapping process between MozartSpaces and semantic entries. The basic concept utilized to realize the central task repository is a container hosted in a single runtime instance of the space where the container is addressable by URL and therefore can be accessed as any other resource on the Internet. The container, representing the central task repository, hosts semantically annotated tasks that build a mission and which can be accessed, i.e., selected, using a SPARQL query. As inherited from the core implementation, Semantic XVSM offer a multitude of coordination patterns for retrieving stored entries, e.g., First-In First-Out (FIFO), Last-In First-Out (LIFO), Random, Key coordinators, as well as a proprietary SemanticCoordinator.

The robot entity with accompanying resources is described using the SKIM-RO ontology complemented with the semantic annotation concept. Each robot offers a set of services, i.e., based on an implement (physical device) for executing a special type of a task, and amount of available resources, e.g., amount of a liquid for spraying tasks. To utilize provided services, a robot selects a task from the remote central task repository by issuing the SPARQL query described in Listing 6.1. Issued query wraps up a task selection algorithm that implements an ontology based matchmaking mechanism that determines semantic relationship between the advertised task descriptions and services offered by a robot. The algorithm uses two parameters for selecting a matching task: (1)

type of skills requested by advertised task, (2) a robot's distance from a task. As a result of the query, the Semantic XVSM returns an entry with task description which satisfies requirements stated in the query.

SPARQL for Task Selection

The SPARQL query for selecting tasks residing in the remote central task repository, described in Listing 6.1, uses the same SKIM-RO ontology as advertised tasks and robots do. The query defines a variable *?entry* which is of type *Task* and at the same time retrieves the context entry describing an executor robot. Context entries can be used as a parameter for SPARQL queries, so that more general and flexible queries are supported. After fetching context information and defining the variable for storing task data, the query searches for tasks which require skills that overlap with those offered by the robot. Finally, the query calculates a distance between a task and the robot, and sorts the list of tasks in a way that a closest task is on the first place, i.e., the robot will first execute a task closest to his current position.

Listing 6.1: SPARQL for selecting tasks from a container in Semantic XVSM

```
?entry sxvsm:hasValue ?entryValue .
?entryValue a ma:Task .

OPTIONAL {
            select ?entryValue (count(?ns) as ?neededSkills) {
                   ?entryValue ma:needsSkill ?ns .
                   } group by ?entryValue
            }
            {
            select ?entryValue (count(?ts) as ?overlappingSkills) {
                   ?entryValue ma:needsSkill ?ts .
            GRAPH ?context {
                  ?contextEntry ma:hasSkill ?ts .
                  }
            }group by ?entryValue
}
FILTER (!bound(?neededSkills) || ?overlappingSkills > 0)

?entryValue  ma:centralPoint [ ma:isPoint ?p ; ma:posX ?x ; ma:posY ?y ] .
    GRAPH ?context {
        ?contextEntry ma:hasPosition [ ma:isPoint ?a ; ma:posX ?b ; ma:posY ?c ] .
    }

BIND ((abs(?x - ?b) + abs(?y - ?c)) as ?distance)

FILTER (?distance <= 20)
```

Listing 6.2 shows the complete code implemented on a robot which is used for building the SPARQL query. Class *SemanticAPI* provides the static method *createQuery* for creating a query represented as an object of type *SemanticSelector*. The *createQuery* takes two parameters: (1) a string representation of a query, i.e., a query from Listing 6.1, and (2) the number of returned semantic entries. Upon creating the query object of type *SemanticSelector*, additional parameters are added. First, a variable *distance* is added to enable ordering of resulting semantic entries. After that, two prefixes describing ontology concepts are attached. Finally, a context entry representing the robot which constructs the query is appended.

Listing 6.2: SPARQL query

```
SemanticSelector query = SemanticAPI.createQuery(Dictionary.wherePart, 1);
query.getQuery().addGroupByVariables("distance");
query.addOrderByVariables("ASC(?distance)");
query.addPrefix("ma", URI.create(Dictionary.NS));
query.addPrefix("sxvsm", URI.create(Ontology.ONTOLOGY_URI));
query.addContextEntry("context", SemanticAPI.serializeEntryValue(robot));
```

The advantage of Semantic XVSM is that SPARQL queries can be used for entry selections. For this purpose, a new *SemanticSelector*, complementing *SemanticCoordinator*, is created and it can be combined in a chain with other MozartSpaces selectors. A selector chain is a sequence of selectors where the result of one selector is piped to the next selector as an input.

Containers and Notifications

Due to the centralized architecture, the coordination between entities in cSKIM is simple and it is realized with the containers listed in Table 6.1. Listed containers belong to the instances of two coordination entities, the centralized coordination model, described in Section 5.7.3, named as cSKIM and multiple instances of type *Robot*. cSKIM hosts only one container where all instances of type *Task* are stored and prepared for querying. The container utilizes *SemanticCoordinator* which is a custom MozartSpaces coordinator for selecting entries by using SPARQL queries. It operates only in a combination with the sematic back-end. Furthermore, the notification mechanism registered on the container enables cSKIM to be aware when a new task is written in the container.

On the other hand, instances of type *Robot* retrieve a reference to the container residing in cSKIM utilizing lookup mechanism provided in Semantic XVSM (Listing 6.3). The obtained reference has twofold purpose:

- to execute the SPARQL query on the remote container to select a task (Listing 6.3), and
- to create a new task and store it on the remote container (Listing 6.4).

The result of SPARQL query executed on the remote container is a list which contains objects of type *SerializedSubResourceTree* as denoted in Listing 6.3. Returned objects are deserialized using the static method *createQuery* in class *SemanticAPI* and casted to instances of type *TaskReasoning*. Similar, Listing 6.4 denotes a procedure when a robot creates a new task by serializing an instance of type *Task* using the static method *serializeEntryValue*. Result is an object of type *SerializedSubResourceTree* which is then written to the remote container.

Listing 6.3: Execute a SPARQL query

```
ContainerReference centralTaskContainer =
    capi.lookupContainer(CentralizedCoordinationPattern.semanticTaskContainer);
List<SerializedSubResourceTree> result = capi.take(centralTaskContainer, query, 5000, tx);
```

Table 6.1: cSKIM - containers

Entity	Container	Coordination	Notification
cSKIM	Container	SemanticCoordinator	✓
Robot	ref → Container	-	-

Listing 6.4: Create a new task

```
SerializedSubResourceTree entryValue = SemanticAPI.serializeEntryValue(tNew);
capi.write(centralTaskContainer, new Entry(entryValue));
```

Figure 6.6 illustrates a sequence diagram which denotes all components required for establishing interaction activities between cSKIM, as a central coordination model, and robot instances. ORM entity in Figure 6.6 denotes the module in Semantic XVSM responsible for mapping a Java object, which represents a task, to a semantic entry. And it supports the mapping in the opposite direction as well. Capi entity in Figure 6.6 is a core component of MozartSpaces which supports elementary operations such as reading, writing, container lookup, etc. However, due to the space limitations, the diagram denotes only one robot instance.

At the beginning, cSKIM retrieves tasks generated by an external component. Retrieved tasks are not semantically annotated, i.e., they are just regular Java objects. After retrieving tasks, cSKIM loads a configuration which contains the instances of type *Robot*. During the configuration load, cSKIM creates a container, listed in Table 6.1, which will store retrieved tasks. Subsequently, cSKIM annotates received tasks using SKIM-RO, serializes them, and writes them to the container created in previous step. By writing the semantically annotated tasks to the container, cSKIM finishes its internal processing and makes the stored tasks accessible over SPARQL interface. At the same time, cSKIM starts listening to new tasks by registering a notification to the container and starts remote robots.

When started, a remote robot utilizes the lookup mechanism to fetch a container reference. Then the robot creates a query using the procedure described in Listing 6.2. As described in Listing 6.3, once having the container and query instances, the robot uses it to execute a query on the remote container hosting tasks. Since the second parameter in the method *createQuery* is set to 1, the result will contain only one instance of type *SerializedSubResourceTree* which is then deserialized to an instance of type *Task*. After the robot deserializes received values and has the *Task* instance, it starts the execution phase where it checks whether it has the available amount of resources to execute the task and also whether it provides all skills requested by task. The case illustrated in Figure 6.6 denotes that the robot does not provide all skills requested by task and thus it creates a new task, serializes it, and writes it to the remote container (Listing 6.4). Just before writing the new task to the remote container, the execution phase ends.

The sequence diagrams described in next two sections reuse concepts illustrated in this sequence diagram.

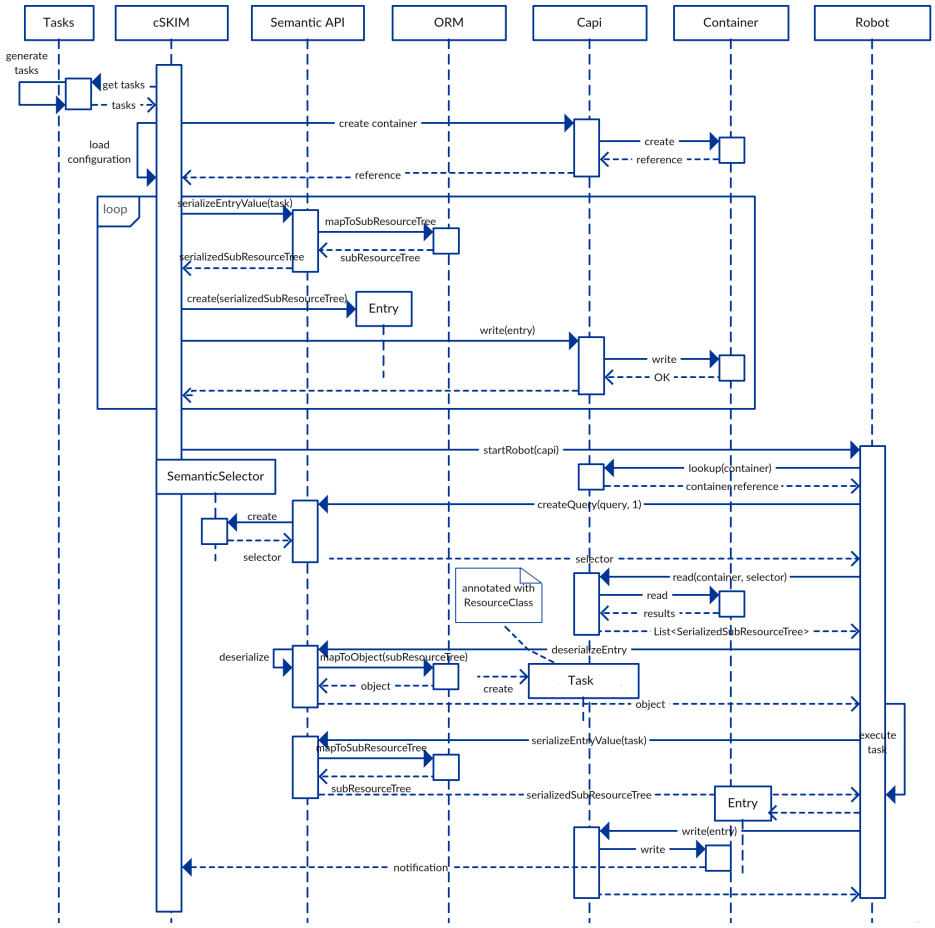

Figure 6.6: cSKIM - sequence diagram

6.6.2 distributed SKIM (dSKIM)

In contrast to cSKIM that uses a central task repository, dSKIM is based on a naive algorithm that distributes all tasks to all robots, which notify each other about the tasks they are going to perform. Due to the lack of semantic-based task allocation, each robot receives a complete set of tasks, even the tasks it cannot perform.

Containers and Notifications

Although dSKIM does not have a central task repository, it still has a container, i.e., a container named *Container* in Table 6.2, hosted by Semantic XVSM where the tasks fetched from the external component are stored. In contrast to cSKIM, the container does not act as a central task repository, rather, dSKIM distributes tasks to robots which then store them in their local repositories implemented by means of Semantic XVSM containers. Local task repositories, hosted on each robot, are similar to the central task

Table 6.2: dSKIM - containers

Entity	Container	Coordination	Notification
dSKIM	Container	FifoCoordinator	-
	local TC	SemanticCoordinator KeyCoordinator	-
	local PE	AnyCoordinator	-
Robot	local NT	AnyCoordinator	-
	ref → Container	FifoCoordinator	✓
	ref → PE_i	AnyCoordinator	✓
	ref → NT_i	AnyCoordinator	✓

repository in cSKIM. Therefore, the local containers, i.e., a container named *Local TC* in Table 6.2, represent the local task repository, hosts semantically annotated tasks, i.e., instances of type *Task*, and robots, i.e., instances of type *Robot*, that build a mission and which can be accessed, i.e., selected, using a SPARQL query. The container utilizes *SemanticCoordinator* which is a custom MozartSpaces coordinator for selecting entries by using SPARQL queries. In addition to *SemanticCoordinator*, the container also supports *KeyCoordinator* for writing and deleting tasks using their names as keys. Having a local task repository eliminates the need for an interaction with the central component.

In addition to a *Local TC*, each robot has two extra containers, containers named *Local PE and Local NT* in Table 6.2, for storing the current task and a new task, respectively. Since there is no central coordination component as in cSKIM, robots in dSKIM have to coordinate and collaborate with each other. This is realized by means of notifications exchanged between robots. Notifications are used to carry information about a colliding task that the robot is going to execute, i.e., when a robots selects a colliding task to execute, a notification is automatically generated and all other colliding robots are notified. Upon receiving a collision notification, a robot either deletes a colliding task from its local repository or, if it is just about to execute the colliding task, it stops its execution and drops the colliding task.

Moreover, Table 6.2 lists few more containers, i.e., container references, belonging to the coordination instances of type *Robot*: (1) a reference to the container hosted in dSKIM, (2) multiple references where each reference PE_i, $i \in R$, adheres to one robot, and (3) multiple references where each reference NT_i, $i \in R$, adheres to one robot. The reference to dSKIM container is necessary for registering a notification on that container. The registered notification is triggered when dSKIM writes tasks and robots to that container. In that way, by receiving a notification, a robot receives mapped tasks and robots. Moreover, PE_i container references register notifications to other robots to listen to the colliding tasks the other robots are going to execute. Upon receiving a notification, a robot updates its *Local TC* by deleting a colliding task from it. Finally, NT_i container references register notifications to other robots to listen when a new task emerges. Upon receiving a notification, a robot updates its *Local TC* by writing a

new task to it. Therefore, both PE_i and NT_i enable coordination and collaboration between distributed robots. Due to the semantically annotated tasks and robots using SKIM-RO, seamless collaboration between them is possible.

Task Selection

A task selection process in dSKIM resembles the one in cSKIM with a difference that it is not executed on a remote container hosting tasks, rather on a *Local TC*. Each robot offers a set of services, i.e., based on an implement (physical device) for executing a special type of a task, and amount of available resources, e.g., amount of a liquid for spraying tasks. To utilize provided services, a robot selects a task from the *Local TC* by issuing the SPARQL query described in Listing 6.1. Issued query wraps up a task selection algorithm that implements an ontology based matchmaking mechanism that determines semantic relationship between the advertised task descriptions and services offered by a robot. Same as in cSKIM, the task selection algorithm uses two parameters for selecting a matching task: (1) types of services requested by advertised task, (2) a robot's distance from a task. As a result of the query, the Semantic XVSM returns an entry with task description which satisfies requirements stated in the query.

Entities Coordination

Figure 6.7 illustrates a sequence diagram which denotes all components required for establishing interaction activities in dSKIM, i.e., between distributed heterogeneous robots. However, due to the space limitations, the diagram denotes only one robot instance.

First step, i.e., a retrieval of generated tasks, is same as in cSKIM described in Figure 6.6 and thus is omitted from Figure 6.7. However, after retrieving tasks, dSKIM loads a configuration which contains the instances of type *Robot*. During the configuration load, dSKIM creates a container, listed in Table 6.2 as *Container*, which will serve for transferring the received tasks to distributed robots.

After extracting the robots from configuration, dSKIM starts the robots. Once when started, a remote robot utilizes lookup mechanism to fetch a reference to the container created in dSKIM. The robot registers a notification to the received container reference to be able to receive the map with all robots and associated tasks. Received tasks will be later written in the created *Local TC*. As already mentioned, the robot creates two additional containers, i.e., *Local PE* and *Local NT*, where the other robots from a fleet will register their notifications and utilize them to coordinate on colliding tasks. The former container is used to store the current task a robot executes, while the latter is used for storing a new task created during the execution phase. A new task is created due to the mismatch in the number of provided and requested skills between a robot and a task.

When the robots from a fleet are up and running, dSKIM annotates received tasks using SKIM-RO, serializes them, creates a map with robot instances as keys and all tasks as values, and writes the map to the container created in the previous step. By

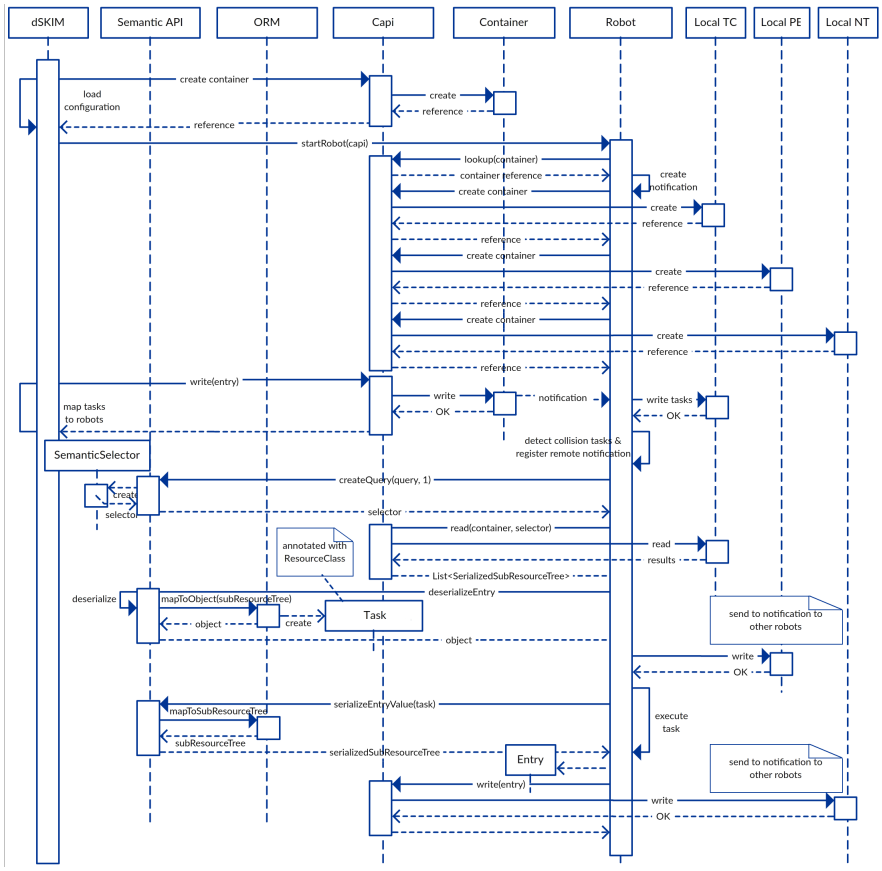

Figure 6.7: dSKIM - sequence diagram

writing the map with semantically annotated tasks and robots to the container, dSKIM finishes its internal processing. In contrast to cSKIM, dSKIM does not listen to new tasks since the coordination paradigm is shifted from the central task repository to distributed robots. Simultaneously, due to the registered notifications on the container hosted in dSKIM, each robot receives a notification with the map where robots are keys and tasks are values. Upon receiving the map, the robot extracts tasks that belong to it, it compares itself to all keys in the map, and writes them in the *Local TC*. Due to the lack of inference-based task allocation, all tasks are allocated to all robots and thus all tasks are colliding tasks. Thus, a robot does not perform any special procedure to detect colliding tasks. Consequently, it registers remote notifications to all robots.

After that, the robot creates a query using the procedure described in Listing 6.2. As described in Listing 6.3, once having the container and query instances, the robot uses it to execute a query on the *Local TC* hosting tasks. Since the second parameter in the method *createQuery* is set to 1, the result will contain only one instance of type *SerializedSubResourceTree* which is then deserialized to an instance of type *Task*.

Before entering in the execution phase, the robot writes a selected task to the *Local*

PE which triggers a notification distributed to all other robots in the fleet. It would be more precise to say that the notification is distributed to all colliding robots, but since in dSKIM all tasks are allocated to all robots, it subsumes that all robots are colliding robots as well. Due to the space limitation and to avoid repetition of same steps, Figure 6.7 omits a step when the task is first written to the *Capi* which then writes it in *Local PE*. Rather, *Robot* writes it directly in *Local PE*.

After the robot deserializes received values and has the *Task* instance which is also written in *Local PE*, it starts the execution phase where it checks whether it has the available amount of resources to execute the task and also whether it provides all skills requested by task. The case illustrated in Figure 6.7 denotes that the robot does not provide all skills requested by task and thus it creates a new task, serializes it, and writes it to the *Local NT*. Writing the new task to the *Local NT* triggers a notification distributed to all robots in the fleet. Upon receiving a notification, each robot adds the new task to its *Local TC*. Just before writing the new task to the *Local NT*, the execution phase ends.

The dSKIM coordination approach introduces a shift in paradigm from the centralized coordination approach described in cSKIM with one central task repository to the completely distributed approach where each robot in a fleet has a local task repository synchronized with the global fleet state. However, neither does the cSKIM nor dSKIM support human interaction or utilizes the coordination capabilities of SKIM-CO.

6.6.3 hybrid SKIM (hSKIM)

In contrast to cSKIM and dSKIM, the hSKIM implementation uses the SKIM-CO ontology-based model of shared knowledge among humans and robots. In the hSKIM the task allocation is performed by reasoning on SKIM-CO. The result of reasoning is a set of tasks mapped to a set of matching robots. As in cSKIM, robots receive tasks from the central task repository, but locally run own decision-making system and reason on SKIM-CO ontology to determine the level of autonomy, e.g., whether to involve the centralized task allocation component to find collaborative robots and whether to involve the human in task assignment.

Containers and Notifications

The hSKIM implementation inherits some concepts from both, cSKIM as well as dSKIM. It combines the concept of a centralized task repository introduced in cSKIM with a distributed coordination mechanisms described in dSKIM. Moreover, on top of that is a human operator able to interact with a central task allocation system and distributed robots as well.

From cSKIM, hSKIM inherits the notion of a centralized task repository which consists of two containers hosted by Semantic XVSM and listed in Table 6.3: (1) one container named *Container*, and (2) the other named *ProCont*. The former container, similar to cSKIM, is used for storing semantically annotated task and robots, as well as SKIM-CO. On the other hand, similar to dSKIM, the latter is used to store a map

Table 6.3: hSKIM - containers

Entity	Container	Coordination	Notification
hSKIM	Container	SemanticCoordinator	✓
	ProCont	FifoCoordinator	-
	ref → DTC	AnyCoordinator	-
User	DTC	AnyCoordinator	✓
	ref → TC_i	KeyCoordinator	-
	local TC	SemanticCoordinator KeyCoordinator	✓
Robot	local PE	AnyCoordinator	-
	ref → ProCont	FifoCoordinator	✓
	ref → Container	SemanticCoordinator	-
	ref → DTC	AnyCoordinator	-
	ref → PE_i	AnyCoordinator	✓

denoting mappings between robots and tasks. In contrast to dSKIM where the map contains all tasks mapped to all robots, hSKIM utilizes the inference-based task allocation approach to map only those tasks to a robot which it is able to execute, either partially or completely.

Moreover, *Container* deploys a *SemanticCoordinator* because the inference-based task allocation approach uses a simple SPARQL query to retrieve tasks and robots mapped to a *Capability* class as described in Algorithm 6.2. *Container* also registers a notification mechanism in order to be notified when a new task is dynamically created by a robot during a mission execution. On the other hand, *ProCont* neither implements *SemanticCoordinator* nor a notification mechanisms. Its sole purpose is to distribute mapped tasks to robots which then store them in their local repositories implemented by means of Semantic XVSM containers.

To enable a human interaction during a task allocation process, hSKIM obtains a reference on *DTC* container. During the task allocation, an unallocated task could exist. In that case, the task is written to the *DTC* container and a human operator is notified. The human operator applies its knowledge on a fleet and allocates the task directly to a corresponding robot by writing it in its *Local TC*.

Furthermore, Table 6.3 lists two containers that an instance of type *Robot* implements together with four references to remote containers. Same as in dSKIM, robots in hSKIM have local task repositories, hosted on each robot. Those local task repositories are similar to the central task repository in cSKIM. Therefore, the local containers, i.e., a container named *Local TC* in Table 6.3, represent the local task repository, host semantically annotated tasks, i.e., instances of type *Task*, and robots, i.e., instances of type *Robot*, that build a mission and which can be accessed, i.e., selected, using a SPARQL query. The container utilizes *SemanticCoordinator* which is a custom MozartSpaces coordinator for selecting entries by using SPARQL queries. In addition to *SemanticCoordinator*,

container also supports *KeyCoordinator* for writing and deleting tasks using their names as keys.

It is important to point out that *Local TC* implements a notification mechanism which is utilized in the human-robot interaction. When a robot requests a help from a human operator, the operator writes a resulting entry in robot's *Local TC*. This triggers a notification which, when processed by the robot, makes it aware of the received help.

In addition to a *Local TC*, each robot has an extra container named *Local PE* in Table 6.3, for storing the current executing task. Similar to dSKIM, robots coordinate each other by means of notifications exchanged between robots. Notifications are used to carry information about a colliding task that the robots is going to execute, i.e., when a robot selects a colliding task to execute, a notification is automatically generated and other colliding robots are notified. Upon receiving a collision notification, a robot either deletes a colliding task from its local repository or, if it is just about to execute the colliding task, it stops its execution and drops the colliding task.

Apart from the created containers, each robot obtains four references on remote containers: (1) a reference on *ProCont* container hosted in hSKIM, (2) a reference on *Container* container hosted in hSKIM, (3) a reference on *DTC* container hosted in a human operator space, and (4) a reference on *PE_i* hosted on each robot i, $i \in R$. The reference to *ProCont* container is necessary for registering a notification on that container. The registered notification is triggered when hSKIM writes the map with tasks and robots to that container. In that way, by receiving a notification, a robot receives mapped tasks and robots. The reference on *Container* container hosted in hSKIM is necessary to handle a dynamic task created during a mission execution. A new, dynamically created, task is then written to the *Container* where the inference-based task allocation algorithm allocates it to a robot. Moreover, the reference on *DTC* container facilitates robot-human interaction, i.e., a robot writes a request for help in that container when it consults a human during a task execution. Finally, the reference on PE_i container registers notifications to other robots to listen to the colliding tasks the other robots are going to execute. Upon receiving a notification, a robot updates its *Local TC* by deleting a colliding task from it.

There is also a *User* entity in Table 6.3 which hosts *DTC* container and has a reference to *TC_i*. The former is used when either hSKIM task allocation process needs a human help in allocating a specific task or when a robot needs a human help during the task execution. In any case, both hSKIM and a robot, writes a equest in a remote *DTC* container using an obtained reference. Moreover, *DTC* registers a notification mechanisms which notifies a human operator upon receiving a request. The reference to *TC_i*, is used when a human operator responds to a robot's request. The operator writes a response directly in robot's *Local TC*.

Task Selection

A task selection process in hSKIM is same as the one in cSKIM and thus is not described here.

Entities Coordination

Figure 6.8 illustrates a sequence diagram which denotes all components required for establishing interaction activities in hSKIM, i.e., between distributed heterogeneous robots, between a centralised component and robots, as well as between a human operator and all other entities. However, due to the space limitations, the diagram denotes only one robot instance.

First step, i.e., a retrieval of generated tasks, is same as in cSKIM described in Figure 6.6 and thus is omitted from Figure 6.8. After retrieving tasks, hSKIM loads a configuration which contains the instances of type *Robot*, an instance of type *User*, and SKIM-CO. During the configuration load, hSKIM creates containers *Container* and *ProCont*, which will serve for inference-based task allocation and the transfer of mapped tasks to distributed robots. When the containers are created, hSKIM writes the loaded SKIM-CO in an instance of *OntologyEntry* and writes it to *Container*. Due to the complexity of the sequence diagram and repeating multiple writing steps, Figure 6.8 omits a step where hSKIM writes semantically annotated tasks, i.e., instances of *Task*, robots, i.e., instances of *Robot*, and a *User* to *Container*.

After extracting relevant entities and SKIM-CO from configuration, hSKIM starts both, a human operator denoted as a *User* and the robots. Once when started, a human operator creates *DTC* container described in section above. On the other hand, a remote robot utilizes lookup mechanism to fetch a reference to the *ProCont* container created in hSKIM. The robot registers a notification to the received container reference to be able to receive the map with robots and associated tasks. After that the robot creates two containers: *Local TC* and *Local PE*. The former is used to store allocated tasks, i.e., tasks received through the notification mechanisms registered on *ProCont*, while the latter is used to store the current task a robot executes. Moreover, the robot fetches a reference to *DTC* container as well.

When the robots and a human operator are up and running, hSKIM performs inference-based task allocation algorithm and, consequently, writes a map with robot instances as keys and tasks as values to the *ProCont* which triggers a notification sent to all robots. Although the tasks are allocated to robots, hSKIM still listens for new tasks dynamically created during a mission execution which are then again allocated to corresponding robots.

Simultaneously, due to the registered notifications on the *ProCont*, each robot receives a notification with the map where robots are keys and tasks are values. Upon receiving the map, the robot extracts tasks that belong to it, it compares itself to all keys in the map, and writes them in the *Local TC*. Due to the inference-based task allocation, not all tasks are allocated to all robots as in dSKIM and thus not all tasks are colliding tasks. Thus, a robot has to detect colliding tasks by examining the received map with all robots and tasks, and consequently, has to register remote notifications to colliding robots.

Then the robot creates a query using the procedure described in Listing 6.2. As described in Listing 6.3, once having the container and query instances, the robot uses it to execute a query on the *Local TC* hosting tasks. Since the second parameter in the method *createQuery* is set to 1, the result will contain only one instance of type

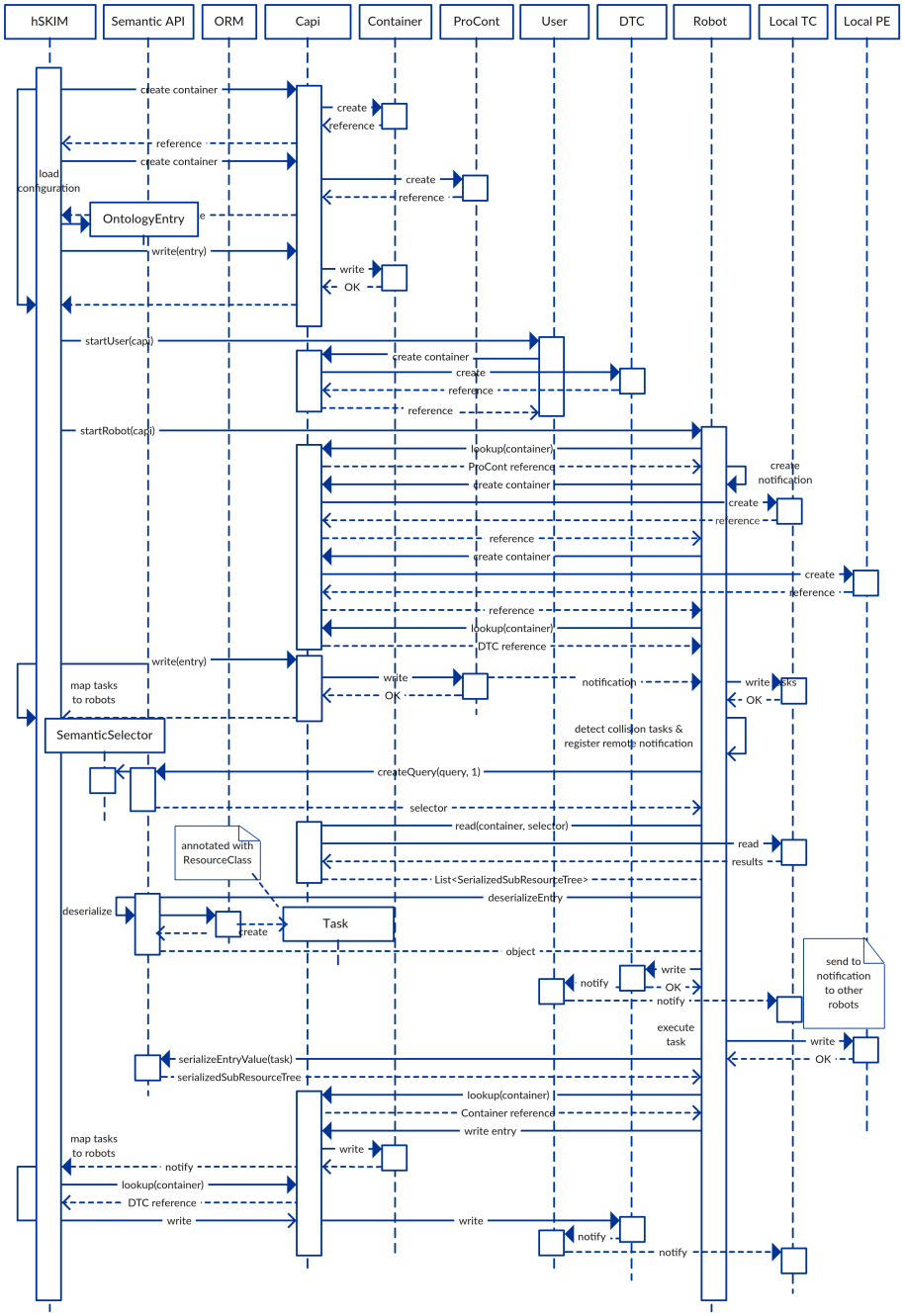

Figure 6.8: hSKIM - sequence diagram

SerializedSubResourceTree which is then deserialized to an instance of type *TaskReasoning*. As illustrated in Figure 6.8, the received task requires a human help and the robot writes it to *DTC* which notifies the user, i.e., the human operator, that provides the help.

Provided help is communicated to the robot by means of writing a response in robot's *Local TC*.

Before entering in the execution phase, the robot writes a selected task to the *Local PE* which triggers a notification distributed to all colliding robots in the fleet. Due to the space limitation and to avoid repetition of same steps, Figure 6.7 omits a step when the task is first written to the *Capi* which then writes it in *Local PE*. Rather, *Robot* writes it directly in *Local PE*. After the robot deserializes received values and has the *Task* instance which is also written in *Local PE*, it starts the execution phase where it checks whether it has the available amount of resources to execute the task and also whether it provides all skills requested by task.

The case illustrated in Figure 6.8 denotes that the robot does not provide all skills requested by task and thus it creates a new task, serializes it, and writes it to *Container* hosted in hSKIM. Writing the new task to *Container* triggers a notification in hSKIM which further triggers the execution of inference-based task allocation algorithm which maps the new task to applicable robots. However, the case is that the algorithm cannot allocate the task and hSKIM consults the human operator, i.e., the user, for a help. The user is consulted by writing the task to *DTC* container. The human, once when notified about new task in its container, allocates the task directly to the robot by writing it into its *Local TC* which triggers a notification on a robot side.

Although it is quite comprehensive and complex, the sequence diagram illustrated in Figure 6.8 addresses four basic mechanisms developed in hSKIM:

- it illustrates the activities and entities involved in inference-based task allocation process,
- it denotes how distributed robots utilize a notification mechanism to coordinate colliding tasks,
- it shows the procedure when a robot consults a human operator for a help during a task execution, and
- it demonstrates creation of a new task and involvement of a human operator in its allocation to a robot.

Finally, it is worth to point out that all containers hosts on robots conform to the naming convention in order to be remotely addressable. The naming convention is coded in the implementation of each coordination approach.

Next chapter presents the results of the evaluation of the general SKIM coordination framework which is based on the semantic extension of the Space-Based Computing architectural style, i.e., Semantic XVSM, and discusses these results with regard to the specified research issues.

CHAPTER 7

Evaluation and Discussion

This chapter presents the results of the evaluation of the general SKIM coordination framework which is based on the semantic extension of the Space-Based Computing architectural style, i.e., Semantic XVSM. The SKIM coordination framework, in particular cSKIM, dSKIM, and hSKIM, is evaluated in different operating environments utilizing different configurations of robotic fleet regarding the criteria defined in Section 7.1.

The chapter is structured as follows: Section 7.1 describes the strategy followed in the conducted experiments. The limitations of Semantic XVSM, with respect to a number of tasks and an optimal fleet size, are perceived through the experiments in Section 7.2. Section 7.3 describes three scenario classes and different configurations for the evaluation of different SKIM-based coordination approaches. These coordination approaches are evaluated in Section 7.4 concerning the criteria defined in Section 7.1. Section 7.5 focuses on the evaluation of hSKIM approach with respect to the different levels of adaptive autonomy and Section 7.6 describes how the ontology-based Model-Driven Architecture approach facilitates the addressing of changing requirements in a robotic fleet, i.e., in a configuration, as well as in an operating environment, i.e., in a scenario. Finally, Section 7.7 gives an overview, complemented with recommendations, on selecting an appropriate coordination approach based on an available scenario and a configuration.

7.1 Testing Strategy

The SKIM coordination framework operates with heterogeneous robotic fleets that execute various tasks in unstructured environments. Two main parameters to configure in the SKIM are: (1) a fleet size, and (2) a number of tasks. Since the SKIM coordination framework is built on Semantic XVSM, it is necessary to find out the limitations with respect to these two parameters. Although [69] did initial tests to acquire the reading and writing performances of Semantic XVSM, what is still missing are performance tests for determining an optimal fleet size and a number of tasks that the system can handle.

Experiments conducted in Section 7.2 derive an optimal fleet size as well as a maximal number of tasks the system can handle in limited time.

Derived fleet size served as a basis for constructing five different configurations for a robotic fleet listed in Section 7.3. Defined configurations consist of heterogeneous robots, i.e., robots provide different skills. Since the SKIM is envisaged to operate in the environments of different complexity, Section 7.3 describes three scenario classes where the defined configurations of a robotic fleet can be employed to perform a mission.

For the comparison of SKIM coordination approaches in different scenarios using different configurations, a precise definition of criteria is indispensable. The SKIM framework is used to evaluate and compare cSKIM, dSKIM, and hSKIM coordination approaches with respect to the following identified criteria: (1) the success rate of task allocation, (2) communication overhead, (3) a robot utilization rate, (4) load balancing, and (5) mission execution time. Task allocation success rate is defined as follows: if a task is completely executed after the *ith* iteration, the success rate of that task is *1/i*. Iteration denotes a robot's attempt to execute a task. Communication overhead represents the number of exchanged messages in the system during a mission execution. Utilization rate denotes the percentage of skill a robot utilized in a mission (see Section 7.4.3). Load balancing is measured as a number of tasks a robot executed and mission execution time is the total mission duration in [s]. Identified criteria pertain to the evaluation and comparison of task allocation approaches and thus can serve as a benchmark in multi-robot system. The experiments in Section 7.4 evaluate the listed criteria and show their interdependencies to identify which fleet configuration is the most appropriate in which scenario considering the selected criteria.

Due to the hSKIM support for adaptive autonomy, Section 7.5 focuses on the evaluation of hSKIM coordination approach which models human interaction. Human interaction subsumes a human's involvement in the task allocation process as well as participation in the decision-making process which decides whether a robot is suitable to execute a specific task. Since the level of robot's autonomy directly influences the inference-based task allocation algorithm which does task classification, i.e., classifies tasks which require human interaction, it is fundamental to detect which autonomy level yields best performances with respect to the above listed criteria.

7.2 Performance of Semantic XVSM

This section presents results of multiple experiments utilized to acquire initial efficiency measurements of Semantic XVSM to determine the expected performances and scalability limitations of the SKIM framework. The framework was tested in various experimental setups running on a laptop equipped with $i5-3230M$ CPU at 2.60 GHz x 4, 8 GB of RAM, Ubuntu 12.04 LTS 64-bit OS, Java 1.7, and Eclipse 4.3 with -Xmx 512m.

An experimental setup is derived from the precision agriculture scenario addressed in the project RHEA and described in Section 1.1. Although there are three different types of tasks in the reference agriculture scenario, i.e., spraying, flaming, and tilling tasks, the experiments in this section are limited to the one type of tasks and robots, i.e., spraying.

By imposing this limitation, we strive to setup the concurrent environment where all the robots compete for the same tasks. In that way scalability and load balancing of Semantic XVSM can be measured more precise than in scenarios with heterogeneous tasks and robots. The latter scenarios are addressed in Section 7.4. The general experiment in our experimental setup consists of an agent that produces tasks and stores them in a central repository (in-memory), i.e., a container in Semantic XVSM, and multiple robots that query the central repository to fetch and execute tasks. A task and a robot are described in Section 4.2.1 and Section 4.2.2, respectively. The experimental setup is divided in two groups of experiments: first addresses experiments where first all tasks are produced and then executed (sequential execution), and second where tasks are simultaneously produced and executed. Division in these groups is necessary because in the most multi-robot systems, which address the task allocation problem, a complete set of tasks is either known in advance, before a mission starts, or tasks arrive dynamically (see Section 3.2.8 which compares various task allocation models). A mixture of two is also possible where the majority of tasks are known in advance, while some of them can emerge during the mission execution. The following parameters are measured for each group: (1) production time for tasks, (2) execution time for a set of task with respect to the number of robots, (3) the relation between task production, task execution, and mission duration. Finally, compared are the measured parameters from sequential and simultaneous executions.

To design the experiments with respect to the number of tasks and robots executing these tasks, it is necessary to find limitations of Semantic XVSM and use these as a basis in the experiments. To discover a limitation concerning the number of tasks in a system, constructed is a baseline scenario where one robot, with the execution time set to 100 ms, executes 10000 pre-produced (produced before the execution starts) tasks. After running for one hour, the robot was unable to execute all tasks due to the long query times influenced by the size of task pool. It is important to notice that the system did not fail after one hour. Consequently, we limited the execution time to 10 min. In 10 min time slot the robot managed to execute 750 out of 10000 tasks. After a couple simulations where the size of initial task pool was calibrated, the limitation for an upper bound for the pool size emerged. The limitation is set to 3000 tasks because the robot managed to execute all of them in 10 min and 38 s. The other limitation that concerns the number of robots, i.e., a fleet size, is investigated in the upcoming experiments where the fleet size varies from 1 to 20 robots. It is worth to notice that 20 is not the limit for the fleet size. Rather, it is just used in experiments to perceive what would be the optimal fleet size. The optimal fleet size is given in the results analysis.

To conform to the discovered limitations, the experiments were designed to measure production and execution times for the following 7 groups of tasks: 10, 20, 50, 100, 500, 1000, 3000. The size of a robotic fleet was scaled accordingly: 1, 3, 5, 10, and 20 robots in a fleet. There are 42 experiments in the sequence execution mode, 7 experiments measuring duration of a task production with respect to the different groups of tasks, and 35 measuring mission execution time. There are 5 different robotic fleets (different sizes) and for each fleet measured is the mission execution time regarding the number of

provided tasks. There are 38 experiments in the simultaneous execution mode. Similar to the sequence execution mode, there are 35 experiments measuring mission execution time. Additionally, there are 3 experiments where the fleet with 3 robots executed 50, 100, and 500 tasks with a variable task production frequency. These experiments are conducted to understand how different production frequencies influence the mission duration. In general, each experiment was repeated 5 times resulting in total with 400 simulations. The listed values are averages from these simulations. There was no need to do more repetitions because the standard deviation is small, i.e., less than 0.1 for simulations that last up to 1 min, regardless whether is it task production or task execution, and less than 1 for all other simulations.

It is important to point out that in all experiments the robots in a fleet were started one after the other with 250 ms delay between two starts and that each robot simulates 100 ms execution time for one task. Since in the simultaneous execution mode both, the task production agent and robots, simultaneously write and read/take data from the container, Semantic XVSM often triggers rescheduling. Due to the rescheduling, there is an arbitrary delay (usually around 20 ms) between writing two tasks to the container. Consequently, we did not manually set the production frequency, rather used the one implied by rescheduling mechanisms. Although the occurrence of rescheduling can be mitigated by increasing robot's execution time, it did not influence our results and thus we decided to keep 100 ms execution time.

7.2.1 Execution Modes

As already mentioned, the experiments are divided in two groups where each group conducts experiments for one execution modes, i.e., either sequential or simultaneous.

There is a production agent in the sequential execution mode which, before a mission starts, writes all tasks that build the mission in a container in Semantic XVSM. After the agent finishes writing the tasks in the container, a robotic fleet starts with the execution of tasks written in the container. Each robot in the fleet concurrently accesses the container and utilizes SPARQL query to take a task.

In the simultaneous execution mode the tasks production and execution runs in parallel. The challenge in this mode is to balance between the distribution of task production and robots' load. On the one hand, the task distribution has to ensure that there is enough tasks in a task pool (container) so that there are no idle robots. On the other hand, having too many tasks degrades system performance and extends mission duration due to the long query times. This relation is addressed in results presented in Section 7.2.5.

Sometimes it can occur that in the simultaneous execution mode a robot gets a timeout (after 200 ms) when trying to get a task from the task container. In particular, this could happen when a large robotic fleet, with 10 or more robots, execute relatively small set of tasks, 50 or less. In that case, the robot will try 5 times to query the task container to check for a new task. If it does not get a new task, it terminates. This behaviour implies that at the end of a mission robots will still try to get new tasks. The

duration of additional retries, when there are no more tasks in the repository, is not calculated in the total mission duration.

7.2.2 Task Production

This section investigates how much time is needed to produce a certain number of tasks in sequential and simultaneous execution modes. Table 7.1 lists the production times in [s] for these two modes regarding the number of tasks and a fleet size (number of robots). The fleet size does not influence the production time in the sequential mode because all tasks are pre-produced. Listed production times are illustrated in Figure 7.1 and Figure 7.2, respectively.

Table 7.1: Task production time in [s]

		Tasks						
Mode	**Robots**	**10**	**20**	**50**	**100**	**500**	**1000**	**3000**
Sequential		0.37	0.59	1.20	1.98	5.82	9.55	22.14
Simultaneous	1	0.91	1.34	2.29	3.66	9.16	14.72	42.62
	3	1.69	2.44	3.25	4.44	13.26	23.77	76.75
	5	1.64	2.48	3.83	5.51	14.85	27.54	94.40
	10	2.12	2.95	4.71	6.13	15.24	28.51	90.70
	20	2.27	3.65	5.23	6.84	16.14	31.56	99.21

Figure 7.1 illustrates the relationship between the number of produced tasks and the time in the sequential execution mode. The graph shows the strong positive relationship between these two variables (the coefficient of determination $R^2 = 0.9912$) that can be modeled with the exponential function. When there are no other entities, e.g. robots accessing the container with tasks, except the production agent, Semantic XVSM supports writing 500 tasks in around 5 s.

Figure 7.2 illustrates how does the task production in the simultaneous execution mode depends on the number of tasks as well as on the number of robots that access the container with tasks. Similar to the graph in Figure 7.1, this graph shows the strong positive relationship between the task production time and the number of tasks and robots. Although the growth of production time can be approximated with the exponential function, it does not ideally follow it when the number of tasks is around 3000. The $R^2 = 0.9439$ is a bit less compared to the one in the sequential mode ($R^2 = 0.9912$). The fleet size influences the production time as well. It takes longer to produce the same amount of tasks when a fleet has more robots. The production time is approximately doubled comparing a fleet with 1 and with 20 robot, especially when the number of tasks is larger than 500. When there are more than 50 tasks in the pool, fleets with 5 and more robots introduce similar production times. This implies that the more robots operate on a same container, the rescheduling is triggered more frequently leading to the increased production time.

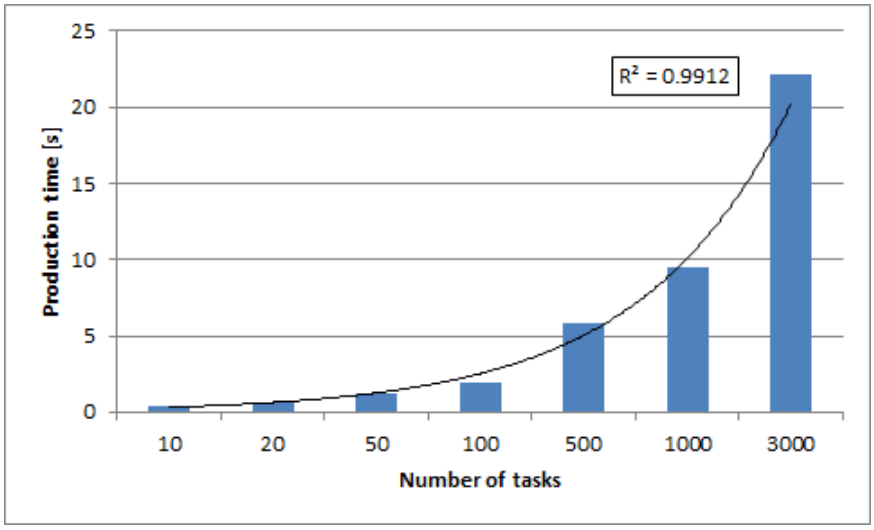

Figure 7.1: Producing tasks - sequential mode

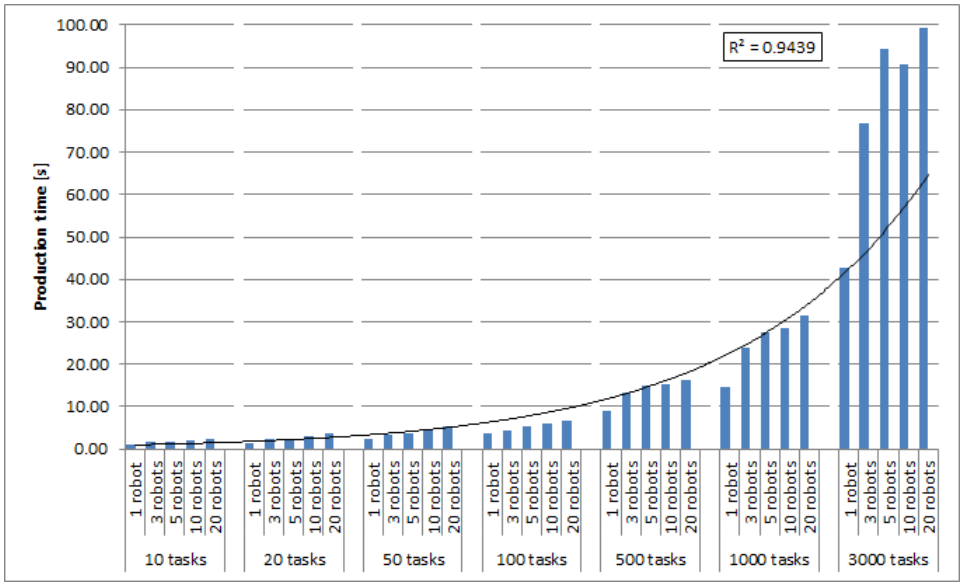

Figure 7.2: Producing tasks - simultaneous mode

Comparing values listed in Table 7.1 and illustrated in Figure 7.1 and Figure 7.2, it can be perceived that the production times in the simultaneous mode with a fleet of size 1 are approximately doubled compared to the sequential modes, e.g., 50 task in sequential mode are produces in 1.20 s and in simultaneous in 2.29 s and 3000 tasks are produced in 22.14 s and 42.62 s, respectively. This is due to the multiple robots that take tasks from the task container while the producer is still writing them and thus

trigger rescheduling in Semantic XVSM. Both execution modes exhibit the exponential production time with respect to the number of tasks. This may be due to storing and maintaining tasks in-memory rather than persisting them.

7.2.3 Task Execution

This section investigates a time required to execute all generated tasks in sequential and simultaneous execution modes utilizing 5 robotic fleets which differ in size. Table 7.2 lists task execution times in the sequential mode with respect to the number of executing tasks and a fleet size. The values are then illustrated in Figure 7.3.

Table 7.2: Task execution time in [s] - sequential mode

	Tasks						
Robots	**10**	**20**	**50**	**100**	**500**	**1000**	**3000**
1	2.69	4.33	8.50	15.9	71.62	152.75	637.71
3	1.78	2.43	3.84	6.20	30.29	75.01	413.76
5	1.82	2.37	3.57	5.11	21.12	54.37	359.95
10	1.84	2.57	3.38	4.83	19.94	51.20	348.50
20	1.83	2.48	3.42	5.07	19.82	51.16	353.13

Observing task execution times in Table 7.2 and Figure 7.3, it can be perceived that there exists the strong positive relationship between the number of tasks to execute and the required time when a robotic fleet has fixed size. X-axis in Figure 7.3 denotes a fleet size. The coefficient of determination averaged for 5 robotic fleets is 0.98 and it means that the execution time increases when the number of tasks increases assuming that a fleet size is fixed. More interesting is the relation between a fleet size and the execution time. The coefficient of determination calculated for fleets with sizes 1, 3, and 5 and averaged for all groups of tasks is -0.91. This infers the strong relationship between the execution time which decreases when the fleet size increases up to including 5 robots independently on the number of executing tasks. The increase in a fleet size, above 5 robots, is not followed by decrease in the execution time.

Comparing the execution times for fleets with 1, 3, and 5 robots, it can perceived that the execution times decrease more when switching from 1 robot to a fleet with 3 robots than switching from 3 to 5 robots. When a mission has 10 tasks, a fleet with 3 robots is more efficient (execution time is 1.78 s) than a fleet with 5 robots (execution time is 1.82). On the other hand, when executing 50 tasks, change from 1 to 3 robots reduce the execution time by approximately 55% while increase from 3 to 5 robots reduce the execution time by approximately 7%. The similar pattern applies to the all other task groups except when there are 3000 tasks.

What cannot be seen neither in Table 7.2 nor Figure 7.3, is the load balancing. In all experiments, except those where fleets of 10 and 20 robots were performing 10, 20, and 50 tasks, all robots executed an equal number of tasks when the total number of tasks

was even, and the difference was at most 2 tasks when the total number of tasks was odd. In those exceptional experiments, unequal task distribution occurred due to the 250 ms delay when starting robots. Robots that started first executed more tasks than those started later. When delay was decreased to 50 ms, tasks distribution was almost equal.

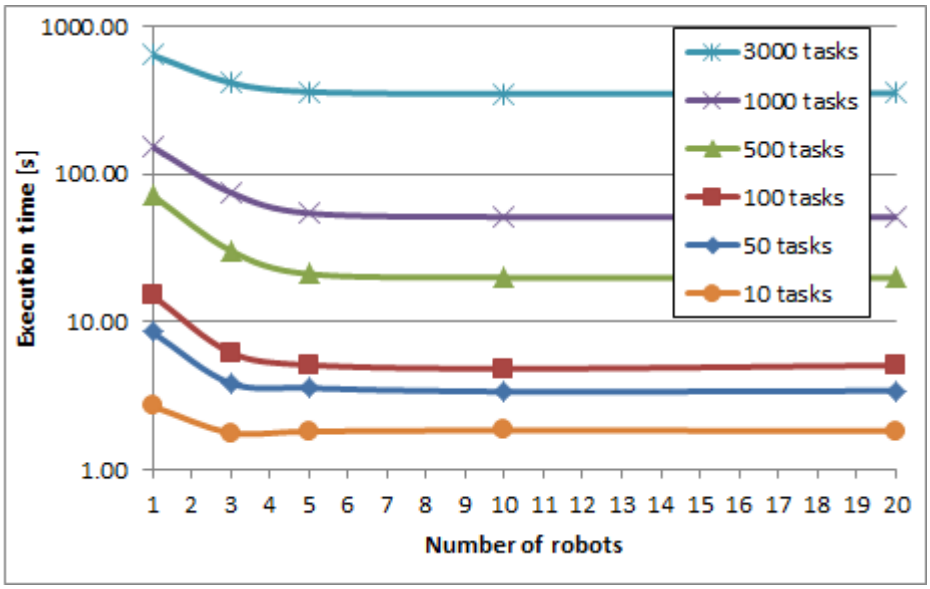

Figure 7.3: Task execution - sequential mode

Table 7.3 lists task execution times in the simultaneous mode with respect to the number of executing tasks and a fleet size. The values are then illustrated in Figure 7.4.

Table 7.3: Task execution time in [s] - simultaneous mode

	Tasks						
Robots	**10**	**20**	**50**	**100**	**500**	**1000**	**3000**
1	2.93	4.61	9.00	15.98	73.04	156.46	639.38
3	2.21	3.10	5.37	8.88	26.70	62.72	320.01
5	1.77	2.60	4.03	5.70	17.32	33.03	99.45
10	2.25	3.00	4.81	6.32	18.08	32.80	94.86
20	2.50	3.71	5.36	7.08	17.86	37.67	105.06

Similar to the sequential execution mode, observing task execution times in Table 7.3 and Figure 7.4, it can be perceived that there exists the strong positive relationship between the number of tasks to execute and the required time when a robotic fleet has fixed size. More interesting is the relation between a fleet size and the execution time. The coefficient of determination calculated for fleets with sizes 1, 3, and 5 and averaged for all groups of tasks is -0.97. Similar to the sequential execution mode, the strong

relationship between the execution time which decreases when the fleet size increases up to including 5 robots, implies that increase in a fleet size, above 5 robots, is not followed by decrease in the execution time. For example, when the number of tasks is less than 500, a fleet with 5 robots has lower execution times compared to fleets with 10 and 20 robots, respectively. This could be due to the frequent rescheduling since there are a lot of robots interacting with the task container.

Comparing the execution times for fleets with 1, 3, and 5 robots, it can perceived that the execution times decrease more when switching from 1 robot to a fleet with 3 robots than switching from 3 to 5 robots. For example, when executing 20 tasks, change from 1 to 3 robots reduce the execution time by approximately 30% while increase from 3 to 5 robots reduce the execution time by approximately 15%. The same pattern applies to the all other task groups except when there are 3000 tasks.

In the experiments where a fleet of 10 robots executes more than 500 tasks, load balancing is unequal and only 5 robots execute around 95% of tasks. The same occurs when a fleet has 20 robots. Only around 40% of them participate in the execution. Due to that, the fleet with 5 robots has similar execution times as fleets with 10 and 20 robots, respectively. For the same fleet sizes the load balancing remains unequal even when the number of tasks is decreased. This could be due to rescheduling and also due to the 250 ms delay when starting robots. Robots that started first, executed more tasks than those started later.

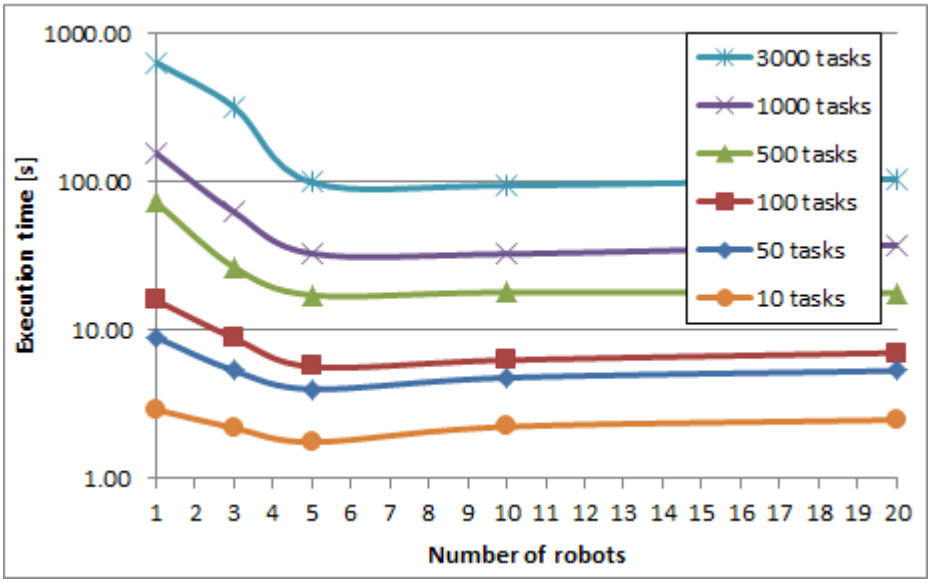

Figure 7.4: Task execution - simultaneous mode

Comparing task execution in sequential and simultaneous modes, it can be perceived that the simultaneous mode operates faster when a robotic fleet has more than 1 robot and when there are more than 100 tasks in the task container. This is due to the approximately constant size of task container in the simultaneous mode compared to the

sequential mode where at the beginning of execution the size of task container equals the total number of tasks. Less tasks in the task container imply faster retrieval of tasks using SPARQL queries. This is described in Section 7.2.5. The query time does not influence the execution time when there are less than 100 tasks and thus the sequential mode operates faster because there is no simultaneous writing and reading from the task container.

7.2.4 Mission Duration

In the sequential mode the mission duration is calculated by summing up production and execution time, while in the simultaneous mode the mission duration equals the execution time. Figure 7.5 illustrates mission duration times for five groups of tasks with respect to the different sizes of robotic fleet. Values for 1000 and 3000 tasks follow the same pattern as the values for 500 tasks and thus they are omitted. Same applies to 20 tasks which have the same pattern as 10 tasks.

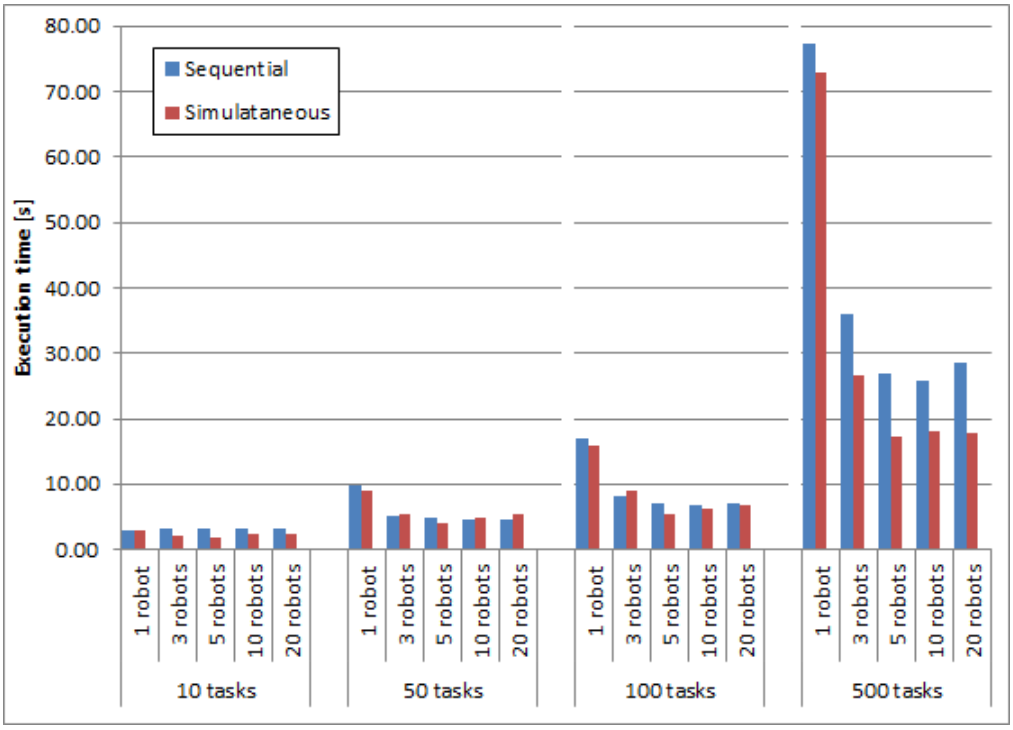

Figure 7.5: Comparison of mission duration in sequential and simultaneous modes

In general, the simultaneous mode performs faster, i.e., has shorter mission duration time. There are few exceptions as well. For example, when there are 50 task in the task container, fleets with 3, 10, and 20 perform slower than the same in the sequential mode. The same occurs when a fleet with 3 robots perform 100 tasks. The largest difference occurs when fleets with more than 1 robot execute more than 100 tasks. In these cases the

simultaneous mode is approximately 25% faster when there are 500 tasks, approximately 50% faster for 1000 tasks, and approximately 70% for 3000 tasks. Regardless of the execution mode, fleets with 5, 10, and 20 robots have similar mission duration times. Especially when there are more than 100 tasks.

Figure 7.5 also depicts that, in both modes, the duration time decreases more when switching from 1 to 3 robots than from 3 to 5 robots. It may be concluded that with respect to the performances it is optimal to utilize a fleet with 3 robots.

7.2.5 Production vs. Execution vs. Mission Duration

This section investigates how does the distribution of tasks production influences task execution and thus mission duration. In our experiments, the distribution of tasks production can be one of the following: (1) all tasks are produced in advance, before a mission starts, (2) all tasks are produced at the beginning of mission although the task production and execution start at the same time, and (3) tasks are produced in parallel with their execution till the end of mission. Former relates to the sequential execution mode, while latter two to the simultaneous execution mode.

Figure 7.6 illustrates how the three different distributions of task production, characterized with a production frequency, influence task execution and thus mission duration. *Pre-prod* denotes that all tasks are produced before a mission starts, $f = \infty$ *Hz* denotes that the number of tasks produced in one second depends on the underlying capabilities of Semantic XVSM, and $f = 33.3$ *Hz* strives to distribute task production during the whole mission. The same pattern applies to 10 and 20 tasks as well as to 1000 and 3000. Presented results pertain to a fleet with 3 robots because the previous sections illustrated that this is the optimal fleet size.

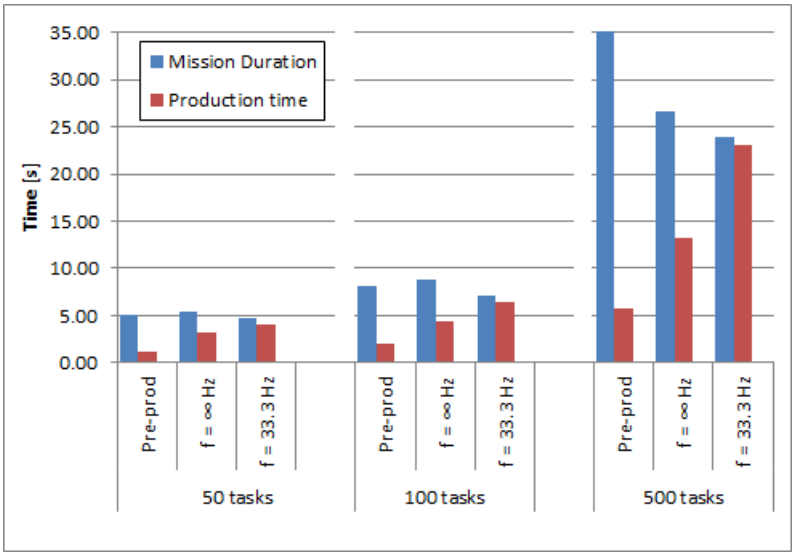

Figure 7.6: Influence of the production duration on the total mission duration

When all tasks are pre-produced, the production phase takes approximately 25% of the total mission duration when the number of tasks is less than 500. For 500 and more tasks, the production share in the total mission duration decreases, e.g., it is around 17% for 500 tasks. When the production frequency was $f = \infty$ Hz, shares of the production phases are approximately 50%. For the production frequency $f = 33.3$ Hz, the production phases finished just before the mission ends. Although the production phase finished just before the mission, all robots in the fleet executed a same number of tasks in each experiment. This implies that there were no idle robots.

To conclude, mission duration decreases when the duration of production phase increases. The optimum is reached when the production phase ends just before a mission.

Figure 7.7 shows how the size of task pool (task container) and the query time change during a mission. The experiments was design in a way that 500 tasks were pre-produced and 1 robot was executing them. The mission duration is similar to the execution time in the sequential execution mode when 1 robot executed 500 tasks, i.e., 72 s.

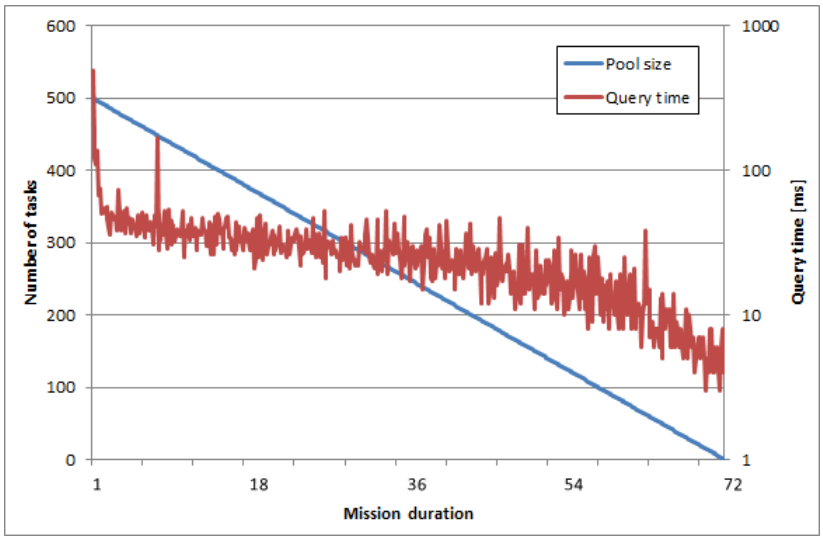

Figure 7.7: SPARQL query time

Both, the size of task pool as well as the query time, decrease towards the end of mission. The size of task pool decreases linearly, while the query time is approximately constant in the first 45 s and then decreases towards the end of mission. After the pool size drops below 150 tasks, the query time starts to decrease. This explains why mission duration decreases when the duration of production phase increases. Longer production phase subsumes fewer tasks in a pool and thus lower query times.

The conducted experiments revealed that the optimal setup, with respect to the mission duration and gained benefit regarding a fleet size, is when 3 robots operate in a mission where tasks are dynamically produced.

7.3 Scenarios and Configurations

Although the SKIM framework is developed within the project RHEA which focuses on a specific precision agriculture use case where a robotic fleet is utilized for weed control, this section aims for emphasizing SKIM generality by introducing different scenarios where the various configurations of a robotic fleet can be employed to perform a mission. The size of robotic fleet is set to 3 because the experiments in Section 7.2 revealed that this is the optimal fleet size with respect to the mission duration and gained benefit.

Table 7.4 describes three scenario classes and for each class is listed the number of initial tasks, i.e., pre-produced tasks, the total number of tasks, i.e., initial tasks and dynamic tasks that are created during a mission execution, as well as a number of skills per task. For the demonstration purposes, the initial number of tasks in each scenario class is 10. This is due to the fact that a number of tasks neither influence task allocation rate nor the robot's utilization rate. This holds under the assumption that initial tasks have a same number of skills per task. Although a number of tasks influence a number of exchanged messages as well as a mission duration, such experiments are out of scope of this thesis.

These scenario classes have been selected because they cover both, missions with simple tasks as well as missions with complex tasks where the robots in a fleet have to collaborate and interact with a human. The scenarios are comprised of independent tasks, e.g., the number of surveillance actions that have to be performed within a specified time interval in several geographic locations. Although the scenario classes have the same number of initial tasks, the number of total tasks is different for each class. It varies and depends on: (1) a scenario class, (2) a configuration, and (3) a level of adaptive autonomy. Dynamic tasks are created during a mission execution due to the robots' collaboration on complex tasks. Scenario classes also have different number of required skills per task, i.e., task's complexity. The notion of task's complexity, expressed as the number of required skills, has a twofold purpose: (1) to characterize the scenario class and (2) to enable task-robot mapping during the task execution.

Table 7.4: Description of scenarios

Scenarios	Initial tasks	Total tasks	Skills per task
Inspection	10	10	1
Industry	10	[10,20]	2
Disaster	10	[10,30]	3

Inspection scenario is a representative of the scenario class which encompasses simple tasks, i.e., tasks requiring only one skill for the complete execution. Collaboration between robots is unnecessary and thus the number of initial and total tasks is the same. The class involves simple scenarios consisting of repetitive tasks similar to those found in surveillance mission. *Industry* scenario is a representative of the scenario class including complex tasks where two unique skills are required. To apply multiple unique

skills on a complex task, robots in a fleet have to collaborate to completely execute the complex task. Consequently, the number of total tasks varies. *Disaster* scenario is a representative of the scenario class composed of the most complex tasks requiring robots' collaboration and human interaction during the task's execution, e.g., Urban Search and Rescue (USAR), resulting in a variable number of total tasks.

Due to the varying number of skills required per scenarios in Table 7.4, different configurations of a robotic fleet are manually generated as shown in Table 7.5. In contrast to the homogeneous robotic fleets (robots only had a spraying skill) utilized in the experiments in Section 7.2, configurations listed in Table 7.5 imply heterogeneous robots with different skills. Each configuration consists of 3 robots and each robot has the number of skills $s \in [1, 3]$. The number of skills is limited to 3 because there are in total 3 different skills in the precision farming scenario described in Section 1.1, i.e., a spraying, a flaming, and a tilling skill. Although the SKIM framework supports tasks and robots which require, respectively provide, more than 3 skills, such a setup is out of scope of this thesis. Simple robots have only one skill (white), more complex have two skills (blue), and the most sophisticated have three skills (dark gray).

Table 7.5: Different configurations of a robotic fleet

Configuration	**Robot 1**	**Robot 2**	**Robot 3**
1	1	1	1
2	3	3	3
3	1	1	3
4	2	2	3
5	1	2	3

Configurations in Table 7.5 can be grouped in two classes: (1) a peer class, and (2) a team leader class. The peer class encompasses configurations where all robots have the same number of skills independently of the type of skills, e.g., configurations 1 and 2. The team leader class encompasses configurations where exists a dominant robot. The dominant robot is a robot which has more skills than the other robots in the same configuration. However, a robot in a configuration of type team leader doesn't lead the other robots in the configuration.

7.4 Performance Evaluation of Coordination Approaches for Task Allocation

In this section the SKIM framework is used to evaluate and compare cSKIM, dSKIM, and hSKIM coordination approaches with respect to the following criteria: (1) the success rate of task allocation, (2) communication overhead, (3) a robot utilization rate, (4) load balancing, and (5) mission execution time. The results are acquired in three scenarios, presented in Table 7.4, utilizing five configurations, listed in Table 7.5, of a robotic fleet and three mentioned coordination approaches.

A simple experimental setup, somewhat resembles the extended RHEA scenario where a user generates tasks for a mission. Generated tasks require some skills for execution, e.g., a spraying, a flaming or a cutting skill or their combination. Since there are in total 3 different skills in the RHEA scenario, simulations performed in the SKIM framework are also limited to 3 different skills. Although the SKIM framework supports tasks and robots which require, respectively provide, more than 3 skills, such a setup is out of scope of this thesis. Moreover, the amount of resources each task requires is suppressed in the following experiments and it is assumed that each robot has a sufficient amount of resources to execute an assigned task. In experiments which consist of multiple heterogeneous tasks, neither is the priority nor order of task execution relevant. This subsumes that atomic tasks which build up a complex task can be executed in an arbitrary order, i.e., there is no hierarchy between tasks. However, in some use cases the order of task execution could matter, e.g., a flaming must be done before a spraying. In all experiments a single task execution time is simulated 2500 ms (which in real scenario corresponds to approximately 4 meters trajectory of a robot moving at 6 km/h). In all experiments, the level of robot's autonomy, adjusted through the configurable parameter k introduced, is set to 3. This subsumes that a human operator will only interact with a system during the execution of the most complex tasks, i.e., tasks that require 3 skills for complete execution. It is worth to notice that the SKIM coordination framework does not introduce any limitation on the mission duration. A mission is considered to be finished only after all tasks have been successfully executed.

The experiments were designed to measure the behaviour of 3 coordination approaches in 3 different scenarios utilizing 5 configurations with respect to the 5 criteria introduced in Section 7.1. There are in total 225 experiments. Test runs were designed in a way that it is possible to extract the values for the 5 criteria in one simulation provided that a coordination approach, a scenario, and a configuration are fixed, i.e., one simulation provides results for 5 experiments. This yields in total 45 different simulations. Each simulation was repeated 5 times and presented results show average values. There was no need to do more repetitions because the developed coordination approaches, together with the underlying Semantic XVSM, build up a very stable system where the all measurements were closely aggregated, i.e., a standard deviation was below 5%.

7.4.1 Task Allocation Success Rate

Task allocation success rate is defined as follows: if a task is completely executed after the *ith* iteration, the success rate of that task is *1/i*. An iteration denotes a robot's attempt to execute a task. If a robot attempts to execute an allocated task that requires only one skill, the task will be executed in $1st$ iteration yielding the 100% success rate. Provided that the robot has sufficient resources. On the other hand, if a task requires multiple skills and there is no robot which provides all required skills, multiple robots have to collaborate to execute that task. If two robots have to collaborate to execute the task, both will attempt to execute the task, but each will only partially succeed in the execution. It means that the task will be executed in $2nd$ iteration. Furthermore,

defined is the total success rate as the average of the success rates of all tasks in a specific scenario.

Table 7.6 lists the success rates for cSKIM, dSKIM, and hSKIM utilizing 5 configuration evaluated in 3 scenarios. Figure 7.8 illustrates the values from Table 7.6

Table 7.6: Task allocation success rate in [%]

	Inspection			**Industry**			**Catastrophe**		
	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM
1	100	100	100	50	50	50	34	34	34
2	100	100	100	100	100	100	100	100	100
3	100	100	100	55	65	50	47	34	100
4	100	100	100	85	80	85	70	70	100
5	100	100	100	80	70	80	65	65	100

When a robotic fleet operates in *Inspection* scenario, task allocation rate is 100% and it neither depends on the coordination approach nor the selected configuration. This is due to the simple tasks which only require one skill and thus each allocated task is executed in 1*st* iteration. When configuration 2 is utilized, task allocation rates are 100% in all 3 coordination approaches operating in *Industry* and *Catastrophe* scenarios. This is because each robot in configuration 2 has either subsume or exact matching degree, i.e., it provides either equal or more skills than required by a task, and thus each allocated task is executed in 1*st* iteration. In *Industry* and *Catastrophe* scenarios, configuration 1 yields 50% and 34% allocation rates. *Industry* scenario encompasses tasks that require 2 skills, while the robots in configuration 1 provide only one skill each. This implies that 2 robots attempt to execute a task yielding 50% allocation rate. Similar, tasks in *Catastrophe* scenario require 3 skills implying that 3 robots have to collaborate on task execution and thus is the allocation rate is approximately 34%, e.g., each robot executes 1/3 of a task.

In *Industry* scenario, configuration 3 yields similar success rates for all coordination approaches. Since there are 2 robots with only 1 skill and all tasks require 2 skills, a team leader robot participates in each task execution. Consequently, for a task execution required are up to 2 iterations. Due to the higher share of robots with 2 or more skills, configurations 4 and 5 yield higher allocation rates compared to configuration 3. In configuration 4, all robots provide at least 2 skills and thus it yields higher allocation rates than configuration 5. The same applies in *Catastrophe* scenario where configuration 3 has lower allocation rates compared to configurations 4 and 5.

hSKIM yields 100% allocation rate in all configurations operating in *Catastrophe* scenario, except configuration 1. The allocation rate in configuration 1 diverges because a human operator was excluded from the control loop (k is set to 3). Maximum allocation rate in other 4 configurations is due to the interoperability of the reasoning-based task

allocation algorithm and human interaction triggered during the execution of the most complex tasks (Algorithm 6.3). Only the robots which provide 3 skills were able to execute tasks.

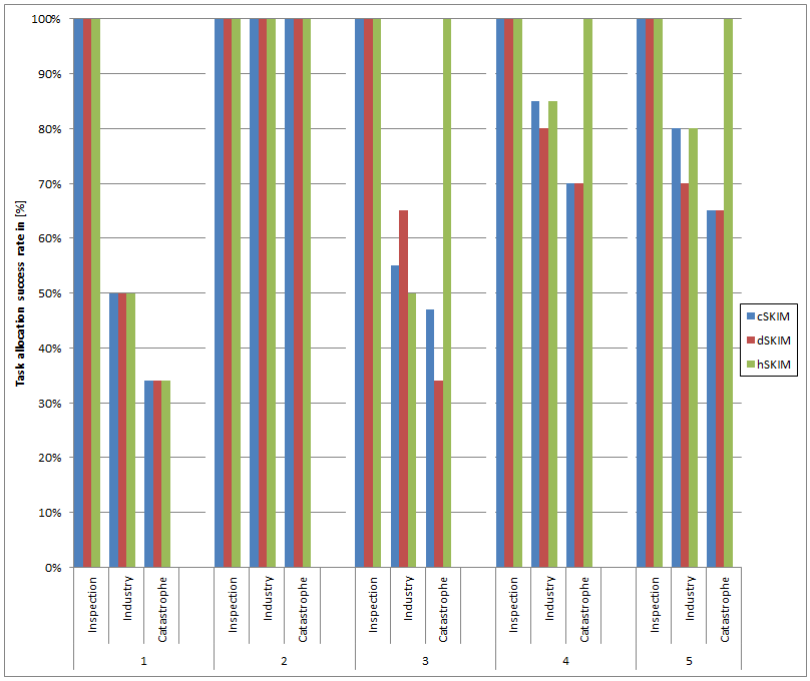

Figure 7.8: Task allocation success rate

7.4.2 Communication Overhead

Table 7.7 lists the number of messages exchanged in each coordination approach for the 5 configurations evaluated in 3 scenarios as specified in Section 7.3. A message contains information about a task and it is exchanged between two distributed robots, or between the space and one distributed robot. Figure 7.9 illustrates the values from Table 7.7.

Figure 7.9 shows that the fleets that operate in *Catastrophe* scenario require the highest number of messages to finish a mission. In particular, configuration 1 requires the highest number of messages (113). This is due to the simple robots which extensively collaborate to finish tasks, and thus have to coordinate on colliding tasks, as well the frequent usage of inference-based task allocation algorithm. The algorithm is triggered on each partial task and the robots generate two partial tasks for one complex task. For example, robot 1 with 1 skill tries to execute task 1 which requires 3 skills. First, it will notify the other two robots that it is going to execute the colliding task 1. Since it only provides 1 skill, it will only partially execute the task (1/3). A new (partial) task will be generated and the inference-based task allocation algorithm will allocate it to corresponding robots. After that, the other two robots will try to execute it.

Table 7.7: Number of messages

	Inspection			**Industry**			**Catastrophe**		
	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM
1	20	23	3	50	53	53	80	73	113
2	20	23	23	20	23	23	20	23	43
3	20	23	13	47	44	54	68	66	23
4	20	23	16	29	35	38	38	41	23
5	20	23	13	28	41	39	34	44	23

Finally, last robot will complete the task. In contrast to configuration 1, configuration 2 yields the lowest number of messages due to the complex robots which do not need to collaborate on a mission. Since configurations 4 and 5 are similar with respect to the robots with provided skills, they need a similar number of messages in every scenario for each coordination approach.

In *Inspection* scenario, dSKIM requires 15% more messages, in all five configurations, compared to cSKIM. Contrary, hSKIM on average requires fewer messages than the cSKIM in the same scenario. Due to the inference-based task allocation in hSKIM, the number of exchanged messages, using configuration 1, in *Inspection* scenario is decreased by 85% compared to cSKIM. Except when using configuration 3 in *Industry* scenario, cSKIM outperforms the other two coordination approaches. This occurs due to the more complex coordination mechanisms between distributed robots present in dSKIM and hSKIM coordination approaches.

In *Catastrophe* scenario, the number of exchanged messages in dSKIM is within approximately 30% compared to cSKIM. On the other hand, in the same scenario the number of messages in hSKIM depends on the configuration class. The number of messages is increased when a configuration from a peer class is utilized, i.e., configurations 1 and 2, and it is decreased when using a configuration from the team leader class, i.e., configurations 3, 4, and 5. In configuration 2 each robot is capable to execute any task in *Catastrophe* scenario. This results with numerous colliding tasks that require more coordination, and thus more exchanged messages, between robots compared to the configuration from the team leader class. On the other hand, team leader configurations have only one robot which executes tasks and thus no coordination with other robots is necessary.

Using configuration 3 and hSKIM in the *Catastrophe* scenario, decreases the number of messages by 66% compared to cSKIM. Configurations 4 and 5, in the same setup, decrease the number of messages by 39% and 32%, respectively. Due to the adaptive autonomy and the human interaction model, which do not allow robots with less than k skills to execute tasks with k and more skills, hSKIM prevent excessive collaboration of simple robots and thus decreases the number of messages. Decreased number of

exchanged messages is related to the increased task allocation rate.

In general cSKIM requires fewer messages to complete the same mission compared to dSKIM and hSKIM. However, in cSKIM the central component is bottleneck because the whole mission stops if this component is unavailable to robots and thus they cannot fetch tasks for execution. Therefore, hSKIM could be an alternative while it distributes allocated tasks to robots and even outperforms cSKIM in some cases.

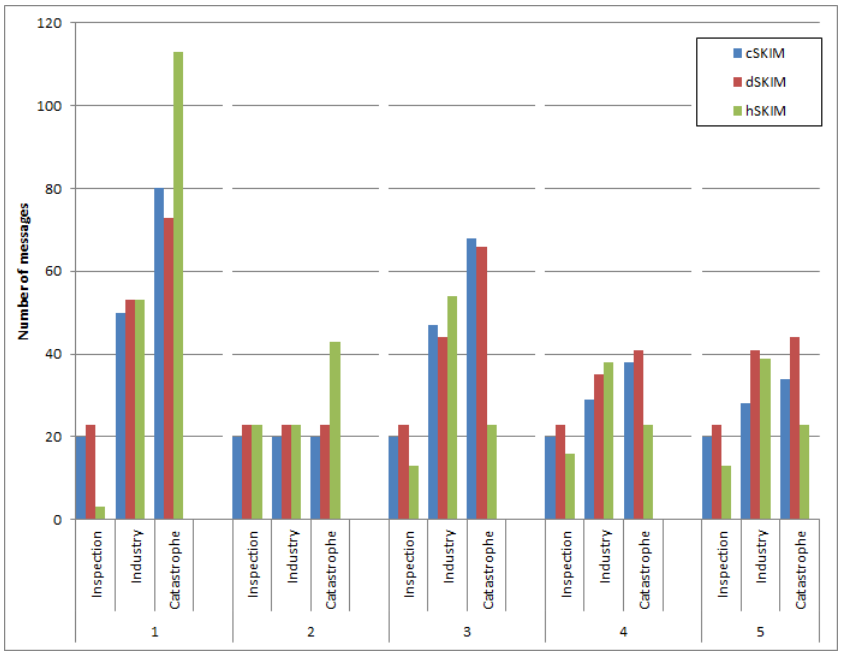

Figure 7.9: Number of messages

7.4.3 Utilization Rate

Utilization rate U_i of robot i is calculated in (Eq. 7.1) where $r_{i,j}$ is ratio between the number of skills robot i utilizes while performing a task j and the total number of provided skills. l_i denotes the number of tasks executed by robot i. Since a robot can also partially execute a specific task, the ratio $r_{i,j}$ could be < 1. The sum is divided by l_i to get an average utilization rate over all executed tasks.

$$U_i = \frac{\sum_{j=1}^{l} r_{i,j}}{l_i} \tag{7.1}$$

Table 7.8 lists the values of fleet utilization rates for 5 configurations evaluated in 3 scenarios using 3 coordination approaches. Figure 7.10 illustrates the values from Table 7.8.

When a configuration is a member of the peer class, i.e., configurations 1 and 2, utilization rate does not depend on the coordination approach. This is because each

Table 7.8: Utilization rate in [%]

	Inspection			**Industry**			**Catastrophe**		
	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM
1	100	100	100	100	100	100	100	100	100
2	34	34	34	67	67	67	100	100	100
3	67	80	87	77	75	80	80	83	100
4	45	44	45	68	63	69	79	80	100
5	60	58	62	77	67	74	90	94	100

robot in configuration 1 provides only one skill and thus applies plug-in or exact matching degree during the execution of an allocated task. Therefore, the provided skills are always fully utilized which yields 100% utilization rate. Utilization rate in configuration 2 is 100% only in *Catastrophe* scenario where robots execute tasks which require 3 skills, i.e., same as the robots in configuration 2 provide. In other two scenarios, robots provide more skills than requested by tasks thus yielding lower utilization rate. In other two scenarios, *Inspection* and *Industry*, configuration 2 yields 33% and 66% utilization rate, respectively. This is because robots have subsume matching degree and thus not utilize the all provided skills.

During a fleet operation in *Inspection* scenario utilizing a configuration from the team leader class, e.g., configuration 3, 4, or 5, hSKIM slightly outperforms dSKIM, while dSKIM slightly outperforms cSKIM. Robotic fleets that have a configuration with a dominant robot and which operate in *Industry* scenario yield similar utilization rates. Due to the higher share of robots with 1 skill in configuration 3 than in configurations 4 and 5, configuration 3 yields a bit higher utilization rates in *Industry* scenario than configurations 4 and 5. On the other hand, robots in configuration 3 have to collaborate more during a mission execution and thus yield more messages compared to configurations 4 and 5.

Similar to the task allocation success rate, when configurations with a dominant robot are deployed in *Catastrophe* scenario, hSKIM outperforms the other two by 5% using configuration 5, by approximately 20% with configuration 3, up to 25% with configuration 4. This is due to the inference-based task allocation algorithm which utilizes parameter k to classify the most complex tasks and robots as those that require a human interaction. This implies that a human operator grants a permission to execute complex tasks, i.e., tasks that requires 3 skills, only to the complex robots, i.e., those that provide 3 skills.

Increased utilization rate is related to the decreased number of exchanged messages. This implies that a robot which is capable to execute a task autonomously, i.e., has either exact or subsume matching degree, generates fewer coordination messages.

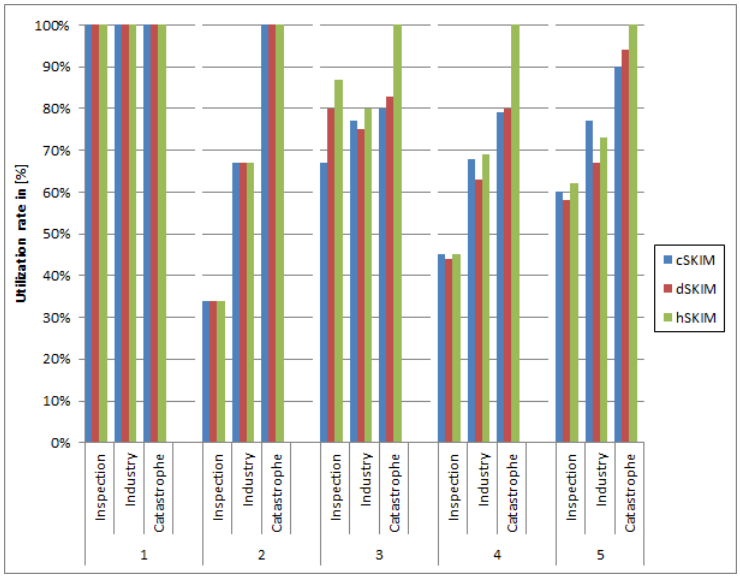

Figure 7.10: Utilization rate

7.4.4 Load Balancing

Load balancing denotes the number of tasks each robot executed in a specific scenario. Due to the variety of skills provided by robots, i.e., different configurations, and due to the different task complexities, i.e., skills requested by tasks, not every robot will execute the same amount of tasks. Table 7.9 shows the number of task each robot executed utilizing 5 configurations evaluated in 3 scenarios specified in Section 7.3. There were in total 30 tasks, 10 tasks per scenario. Figure 7.11, Figure 7.12, and Figure 7.13 illustrate the number of tasks each robot executed in *Inspection*, *Industry*, and *Catastrophe* scenarios, respectively.

Independent from a selected scenario, when a deployed configuration is a member of the peer class, i.e., configurations 1 and 2, robots in all 3 coordination approaches execute almost a same number of tasks. Almost the same means that a robot could execute at most two tasks more or two tasks less compared to the other robot. Similar occurs with configurations that have a dominant robot and operate in *Inspection* scenario. If a robot executes more tasks than the other two, it could be that it provides a skill which more tasks request in that particular scenario. It is worth to notice that the number of tasks in *Inspection* scenario is always 30, i.e., each coordination approach executes 10 tasks. This is because all the robots completely execute allocated tasks.

Neither does the configuration nor the selected coordination approach influence the number of tasks a robot executes in *Industry* scenario. Executed tasks can have different distribution in each simulation. Eventually some simulations will yield the same task distribution. The values listed in Table 7.9 are thus averaged values. Although there is indeterminism in the number of executed tasks, it could be perceived that robot 3, on

Table 7.9: Number of executed tasks

		Inspection			Industry			Catastrophe		
		cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM
1	**Robot 1**	4	4	4	6	6	6	10	10	10
	Robot 2	3	3	3	8	8	7	10	10	10
	Robot 3	3	3	3	6	6	7	10	10	10
2	**Robot 1**	3	3	4	4	3	3	3	3	3
	Robot 2	3	3	3	3	4	3	3	3	4
	Robot 3	4	4	3	3	3	4	4	4	3
3	**Robot 1**	2	3	3	6	4	6	8	10	0
	Robot 2	3	4	5	6	5	8	8	7	0
	Robot 3	5	3	2	7	8	6	10	10	10
4	**Robot 1**	4	2	3	5	5	4	5	5	0
	Robot 2	3	4	4	4	4	5	5	6	0
	Robot 3	3	4	3	4	5	4	6	5	10
5	**Robot 1**	3	3	3	4	4	4	5	6	0
	Robot 2	4	3	5	5	5	4	6	6	0
	Robot 3	3	4	2	5	7	6	6	5	10

average, executes more tasks in the team leader configurations that the other 2 robots.

The number of total tasks in *Industry* scenario varies from 30 when applying configuration 2 up to 60 with configuration 1. This is because the robots in configuration 1 can only collaborate to execute tasks which require 2 skills. This implies that the higher the share of complex robots in a fleet is, the less the number of new tasks is. For example, configuration 3 has 2 robots with only 1 skill and 1 robot with 3 skills and it yields in total 56 tasks for all 3 coordination approaches in *Industry* scenario. On the other hand, configuration 4 has 2 robots with 2 skills and 1 with 3 skills and it yields in total 40 tasks.

In *Catastrophe* scenario robots in dSKIM execute almost the same number of tasks as those in cSKIM. On the other hand, major discrepancy occurs when hSKIM utilizes a configuration from the team leader class in *Catastrophe* scenario. Due to inference-based task allocation as well as the human involvement, only robot 3, the one with the highest number of skills, is allowed to execute tasks. The other two robots tried to partially execute tasks where their skills fit, i.e., plug-in matching degree, but they were unable to do so because the inference-based task allocation classified all tasks as complex ones thus enforcing a human operator to grant execution permission to only those robots with exact matching degree. As in *Industry* scenario, configuration 1 yields in total 90 tasks,i.e., 30 for each coordination approach, while configuration 2 yields in total 30 tasks, i.e., 10

tasks for each coordination approach.

Table 7.9 shows that there are no idle robots no matter which coordination approach, a configuration, or a scenario, is selected. The only exception is when a human interacts in hSKIM. Although the implemented SKIM framework exhibits fair task distribution and efficient resource utilization, i.e., robots, independently of the selected coordination approach, it is recommended to avoid using a configuration with a dominant robot in hSKIM while operating in *Catastrophe* scenario because all the other robots, except the dominant robot, are idle and thus underutilized.

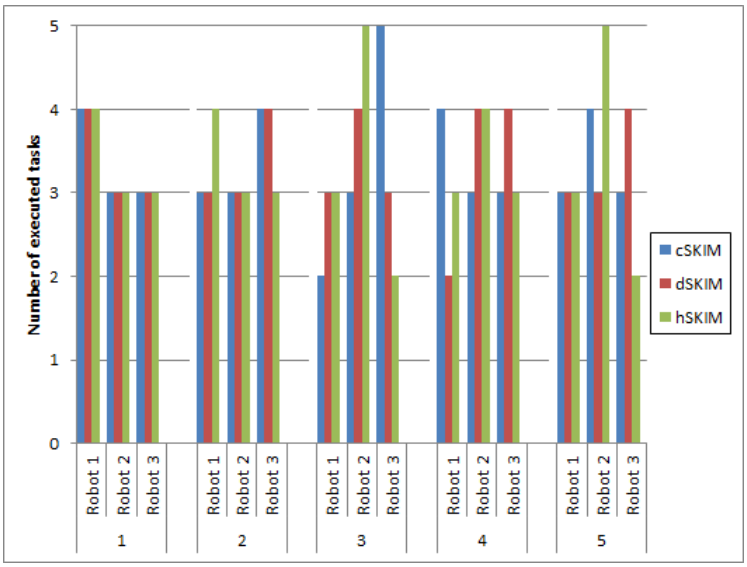

Figure 7.11: Number of executed tasks in *Inspection* scenario

7.4.5 Mission Execution Time

Table 7.10 shows an execution time of a mission that consists of 10 tasks. Mission execution time is measured using 5 configurations deployed in 3 scenarios specified in Section 7.3. Listed values denote mission execution times in [s] for cSKIM, dSKIM, and hSKIM. Execution time depends on hardware as well as the time a robot needs to execute a single task. Values for these parameters are introduced in Section 7.2.

cSKIM outperforms, executes all missions faster, dSKIM and hSKIM independently of selected configuration and scenario. Due to the complex coordination mechanisms in dSKIM, it has longer execution times compared to cSKIM. Employing configurations 4 and 5 in *Industry* and *Catastrophe* scenarios, dSKIM execution time remains within approximately 15% compared to cSKIM. On the other hand, dSKIM exhibits the longest mission execution time, compared to cSKIM, when it operates in *Industry* scenario utilizing configuration 3. It took dSKIM 100% longer than cSKIM. This occurs because of intensive collaboration between robots.

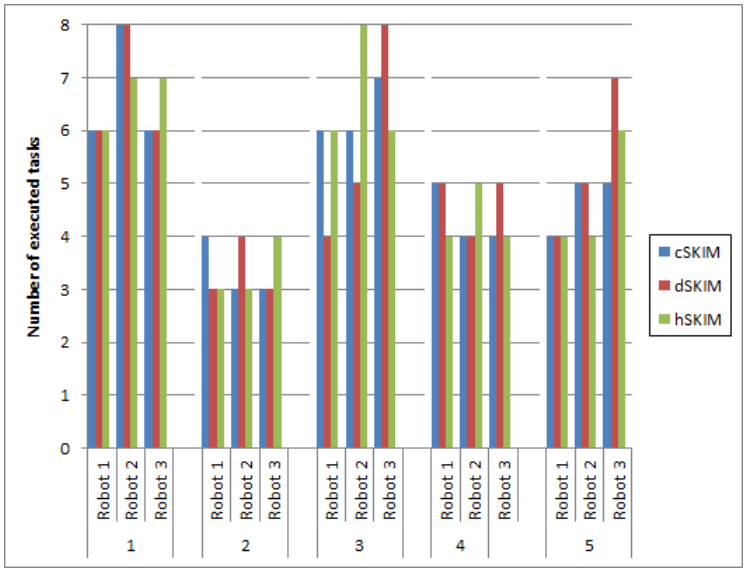

Figure 7.12: Number of executed tasks in *Industry* scenario

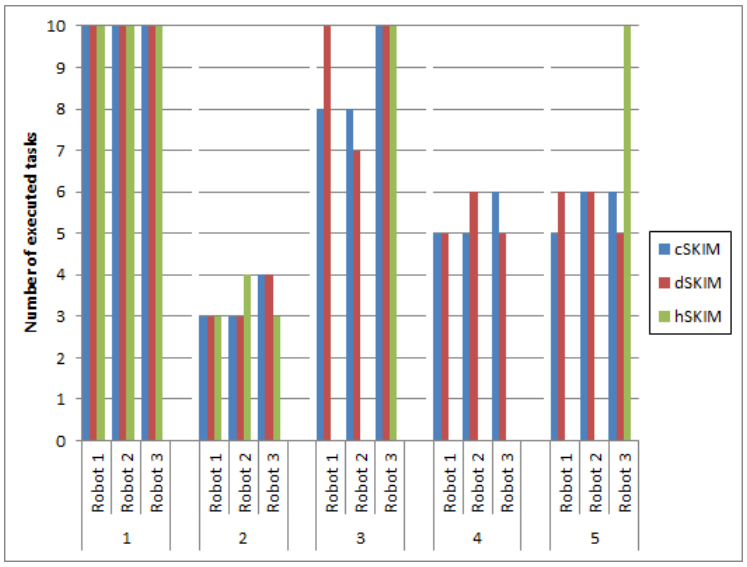

Figure 7.13: Number of executed tasks in *Catastrophe* scenario

Moreover, inference-based task allocation algorithm in hSKIM implies longer execution times compared to cSKIM and dSKIM. In most cases these are approximately doubled. Due to the ability of only one robot in configuration 3 to execute all tasks as well as an extensive collaboration between simple robots and a human operator, hSKIM performs slowest under the configuration 3 in *Industry* scenario. hSKIM needs approximately 230% longer compared to cSKIM. This is due to the utilization of inference-based and

Table 7.10: Mission execution time in [s]

	Inspection			**Industry**			**Catastrophe**		
	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM	cSKIM	dSKIM	hSKIM
1	10	13	14	20	23	41	26	30	49
2	10	12	26	10	13	21	10	12	20
3	13	21	21	18	36	60	26	30	40
4	10	12	23	13	14	25	15	17	29
5	10	13	21	13	14	30	15	17	32

human-based task allocation algorithms that were triggered to allocate partial tasks. On the other hand, hSKIM has the shortest execution time in *Inspection* scenario utilizing configuration 1, i.e., it requires 40% longer than cSKIM. In this case both, tasks and the robots, are simple (1 requested/provided skill) and thus a human intervention is unnecessary. In general, hSKIM has the shortest execution times when operating with fleets with complex robots, i.e., configurations 2, 4, and 5. This is due to the inference-based task allocation which reduces a number of colliding tasks and thus decreases complex coordination between distributed robots.

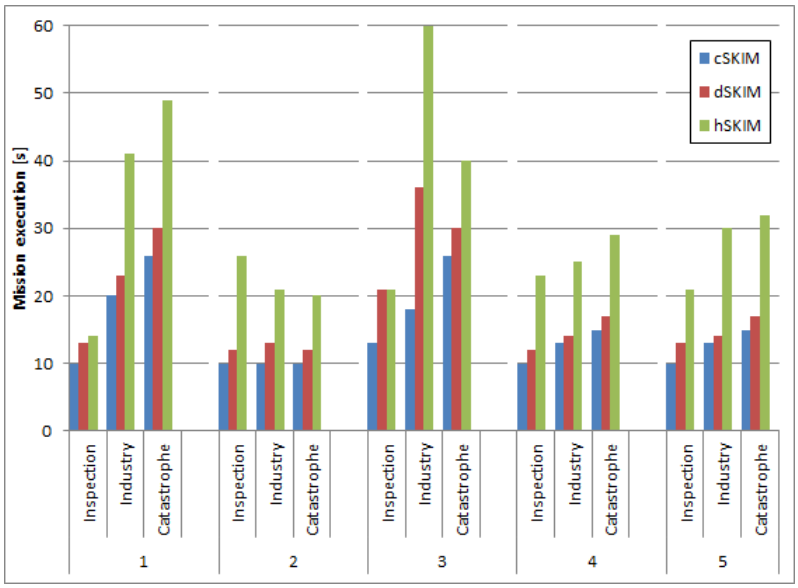

Figure 7.14: Mission execution time

Table 7.11: Configuration 3

	Tilling	Spraying	Flaming
Robot 1	✓		
Robot 2		✓	
Robot 3	✓	✓	✓

Table 7.12: Configuration 4

	Tilling	Spraying	Flaming
Robot 1	✓	✓	
Robot 2		✓	✓
Robot 3	✓	✓	✓

7.5 Performance Evaluation of Adaptive Autonomy

This section focuses on the evaluation of hSKIM coordination approach which models human interaction. The level of robot's autonomy, i.e., adaptive autonomy, is adjusted through the configurable parameter k introduced in Section 6.4. The purpose of experiments in this section is to analyze the influence of adaptive autonomy on the selected criteria introduced in the previous section. Consequently, the value for k will be changed from 1 to 3 inclusive.

Conducted experiments assume that a human operator is perfect, i.e., he/she always needs the same amount of time for making a decision and he/she always makes a right decision. In contrast to the underlying coordination framework which has knowledge about robots in a fleet, it is assumed that a human operator, in addition to the fleet knowledge, also has the knowledge about the operating environment. In addition to human's involvement in the task allocation process, it also performs the decision-making algorithm which decides whether a robot is suitable to execute a specific task. Since both human functionalities, task allocation and decision-making, are implemented as algorithms, the experiments omit a real human operator whose behaviour is simulated with those algorithms.

During the literature review in Chapter 3, it is perceived that almost all robot teams are heterogeneous and there is at least one robot acting as a team leader. Usually, the team leader provides more skills than the other team members do. Therefore, to retain the heterogeneity of robotic fleet, with respect to the provided skills, configurations 3 and 4 are utilized in the following experiments. These two configurations are listed in Table 7.11 and Table 7.12, respectively. Both configurations have a team leader robot and two other robots with fewer skills compared to the team leader robot. Those two robots provide 1 skill in configuration 3 and 2 skills in configuration 4. Changing the value of k, directly influences the inference-based task allocation algorithm which does task classification, i.e., classifies tasks which require human interaction. Therefore, it is expected that changing the value of k influences not only the frequency of human interactions in the system, but also has the impact on other observed parameters, e.g., communication overhead and utilization rate.

The experiments were designed to measure the behaviour of hSKIM coordination approaches in 3 different scenarios utilizing 2 configurations with respect to the 5 criteria listed in Section 7 and 1 new criteria, i.e., in total 6 criteria. Three different values were utilized for parameter k. There are in total 108 experiments. Test runs were designed in

a way that it is possible to extract the values for the 6 criteria in one simulation provided that a coordination approach, a scenario, and a configuration are fixed. This yields in total 18 different simulations. Each simulation was repeated 5 times and presented results show average values. There was no need to do more repetitions because the developed coordination approaches, together with the underlying Semantic XVSM, build up a very stable system where the all measurements were closely aggregated, i.e., a standard deviation was below 5%.

7.5.1 Task Allocation Success Rate

To recall, task allocation success rate is defined as follows: if a task is completely executed after the *ith* iteration, the success rate of that task is *1/i*. Furthermore, defined is the total success rate as the average of the success rates of all tasks in a specific scenario.

Table 7.13 denotes values in [%] of success rates for configurations 3 and 4 evaluated in 3 scenarios, i.e., *Inspection*, *Industry*, and *Catastrophe*, in hSKIM. The value for parameter k is varied from 1 to 3.

Table 7.13: Task allocation success rate in adaptive autonomy in [%]

	Inspection			**Industry**			**Catastrophe**		
Conf \ k	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	100	100	100	100	100	55	100	100	100
4	100	100	100	100	100	85	100	100	100

Interpreting results listed in Table 7.13, it can be seen that the task allocation rate is 100% when the human interaction parameter, i.e., k, is set either to 1 or 2. This is due to the complete fleet knowledge the human has and which enables him/her to allow a task execution only for those robots which have either exact or subsume matching degree. This is independent of the selected configuration and scenario. In *Inspection* scenario all tasks require only one skill to be completely executed and, in both configurations, all robots provide at least one skill. Consequently, all tasks are executed in the $1st$ iteration and thus is the task allocation rate 100%.

When the human interaction parameter is set to 1 and hSKIM operates in *Industry* scenario, all tasks require a human interaction during the execution, i.e., all tasks are classified as *User* tasks in SKIM-RO. Classification takes place because all tasks in *Industry* scenario have 2 required skills and the parameter k is set to 1 which means that all tasks that require at least 1 skill will be classified as *User* tasks. On the one hand, when configuration 3 is deployed in this setup, robots 1 and 2 will not be able to execute any task. They will try to execute a task, but when they consult a human to get a permission to execute the task, the human will refuse to grant a permission because there exists a robot with a highest matching degree, i.e., robot 3. The decision algorithm ensures that a robot which asked for permission has either exact or subsume matching

degree. Otherwise, permission is denied. On the other hand, when configuration 4 is deployed in this setup, all three robots participate in task execution because each robot provides at least 2 skills. It could still happen that a robot which does not have a highest matching degree selects a task and asks a human for permission. However, it will not get a permission to execute the task. Thus, in both configurations, only robots which have either exact or subsume matching degree are allowed to execute tasks. Therefore, robots collaboration during a task execution is omitted and the task allocation rate is 100%.

Similar to the above, when the human interaction parameter is set to 2 and hSKIM operates in *Industry* scenario, all tasks require a human interaction during the execution, i.e., all tasks are classified as *User* tasks in SKIM-RO. On the one hand, when configuration 3 is deployed in this setup, robots 1 and 2 will not be able to execute any task because they only provide 1 skill and thus are not allowed to execute tasks classified as *User* tasks, i.e., those tasks are marked as too complex for them. Compared to the previous case, they do not even try to execute them, i.e., they do not consult a human on them because they are marked as too complex for those robots. On the other hand, when configuration 4 is deployed in this setup, all three robots participate in task execution because each robot provides at least 2 skills. Moreover, it could still happen that a robot which does not have a highest matching degree selects a task and asks a human for permission. However, it will not get a permission to execute the task. Thus, same as in the previous case, only robots which have either exact or subsume matching degree are allowed to execute tasks. Therefore, robots collaboration during a task execution is omitted and the task allocation rate is 100%.

In contrast to the previous cases where the task allocation rate is always 100% independently of the selected configuration and scenario, this is not the case when parameter k is set to 3. Due to the parameter k set to 3, none of the tasks in *Industry* scenario, which need 2 skills, will require a human interaction. Therefore, even robots which exhibit the subsume matching degree will be able to execute some tasks. Consequently, multiple robots collaborate on a task execution. As a product of the collaboration on the same task, atomic tasks (sub-tasks) are generated which then have to be allocated to available robots. Since in configuration 3 robots provide less skills than those in configuration 4, it means that more robots have to collaborate to execute a task. Consequently, majority of tasks will not be executed in $1st$ iteration and thus is the task allocation rate lower utilizing configuration 3. Finally, in *Disaster* scenario the task allocation rate is 100% because all tasks require a human interaction and only robots with the highest matching degree, i.e., robot 3, are allowed to execute those tasks.

7.5.2 Communication Overhead

Table 7.14 shows the number of messages exchanged during the tasks execution in configurations 3 and 4 evaluated in 3 scenarios utilizing hSKIM. Same as in the previous section, the value for parameter k is varied from 1 to 3. Figure 7.15 illustrates the values listed in Table 7.14.

Figure 7.15 shows that the number of exchanged messages increases with the scenario complexity, i.e., it is lowest in the *Inspection* scenario and highest in *Catastrophe* scenario.

Table 7.14: Number of messages in adaptive autonomy

	Inspection			**Industry**			**Catastrophe**		
Conf \ k	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	27	13	13	61	37	52	81	42	43
4	36	16	16	45	59	38	81	79	43

This is because the more complex the tasks are, more robots have to participate in their execution and thus more messages have to be exchanged to coordinate collaborating robots.

When parameter k is 1 and a fleet operates in *Inspection* scenario, configuration 3 requires less messages to execute a mission than configuration 4. This is due to the increased number of colliding tasks which robots in configuration 4 have to coordinate on. Increased number of colliding tasks is a result of more provided skills in configuration 4 compared to configuration 3. Moreover, robots are consulting a human in both configurations which additionally increases the number of exchanged messages. 2 messages are required each time when a robot consults a human, 1 for a request and the other for a response. Using the same value for k, but operating in *Industry* scenario, configuration 3 yields more messages than configuration 4. This is related with the same scenario in the task allocation rate in Section 7.5.1 where robots 1 and 2 try to execute a task, but when they ask a human to get a permission to execute the task, the human will deny to give a permission because there exists a robot with a higher matching degree, i.e., robot 3. Both configurations yield the same number of messages in *Catastrophe* scenario where only robots which have either exact or subsume matching degree are allowed to execute tasks.

When parameter k is 2 and a fleet operates in *Inspection* scenario, configuration 3 requires less messages to execute a mission than configuration 4. This is due to the increased number of colliding tasks which robots in configuration 4 have to coordinate on. Compared to the same scenario when parameter k is 1, the number of exchanged messages is decreased because there is no human interaction during the mission execution. Using the same value for k, but operating in *Industry* scenario, configuration 3 yields less messages than configuration 4. This is due to the fact that robots 1 and 2 will not be able to execute any task because they only provide 1 skill and thus are not allowed to execute tasks classified as *User* tasks, i.e., those tasks are marked as too complex for them. On the other hand, same robots in configuration 4 are allowed to execute those same tasks but have to consult a human to get a permission. The same rationale applies to the *Catastrophe* scenario.

Finally, when k is 3 and a fleet operates in *Inspection* scenario, both configurations yield the same number of messages as in the case when k is 2. Using the same value for k, but operating in *Industry* scenario, configuration 3 yields more messages than configuration 4. This is related with the same scenario in the task allocation rate in

Section 7.5.1 where none of the tasks in *Industry* scenario, which need 2 skills, will require a human interaction. Even robots which exhibit the subsume matching degree will be able to execute some tasks. Consequently, multiple robots collaborate on a task execution and more messages are required for coordination. Both configurations yield the same number of messages in *Catastrophe* scenario where only robots which have either exact or subsume matching degree are allowed to execute tasks. However, the number of messages is decreased compared to the same scenario when k is 1. This is due to the fact that robots 1 and 2 will not be able to execute any task because they only provide 1 skill and thus are not allowed to execute tasks classified as *User* tasks, i.e., those tasks are marked as too complex for them.

It is recommended to set the autonomy level to 2 when having the configuration 3 on disposal because it outperforms the other two autonomy levels, i.e., the system requires less messages for a mission execution. Switching to configuration 4, the autonomy level set to 3 outperforms the other two in *Industry* and *Catastrophe* scenarios. In this use case, the human operator utilizes its complete fleet knowledge to permit a task execution only for those robots which have either exact or subsume matching degree.

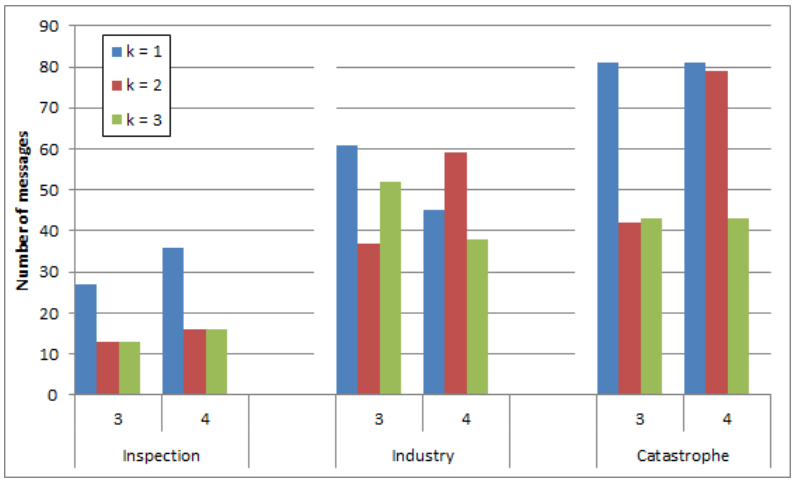

Figure 7.15: Number of messages in adaptive autonomy

7.5.3 Utilization Rate

Table 7.15 shows the values in [%] of fleet utilization rates during the tasks execution in configurations 3 and 4 evaluated in 3 scenarios utilizing hSKIM. Same as in the previous sections, the value for parameter k is varied from 1 to 3. Figure 7.16 illustrates the values listed in Table 7.15.

As shown in Figure 7.16, neither does the selected configuration nor the parameter k influence the utilization rate in *Catastrophe* scenario. It is always 100% because only robot 3 is allowed to execute tasks due to the highest matching degree. Utilization rates in *Inspection* scenario have similar values for all k values. However, configuration 3 yields

Table 7.15: Utilization rate in adaptive autonomy in [%]

	Inspection			Industry			Catastrophe		
Conf \ k	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	78	78	87	67	67	80	100	100	100
4	45	45	45	96	89	69	100	100	100

higher utilization rates than configuration 4. This is due to the lower number of skills that robots 1 and 2 provide in configuration 3 compared to configuration 4.

In *Industry* scenario, when k is either 1 or 2, configuration 3 yields lower utilization rate compared to configuration 4. This happens because in configuration 3, due to the highest matching degree, only robot 3 is allowed to execute tasks. On the other hand, in configuration 4 each robot can have a highest matching degree and thus execute a task. On contrary, due to the allowed collaboration when k is 3 in *Industry* scenario, configuration 3 yields higher utilization rate. Although robots in both configurations collaborate, configuration 3 yields higher rate because robots provide less skills than those in configuration 4. This results with the increased number of exchanged messages (see Table 7.14) and lower task allocation rate (see Table 7.13).

In general, observing the above results it can be concluded that the closer the matching degree between robots and tasks is, i.e., the number and types of skills provided by robots and requested by tasks, the higher utilization rate is. This can be seen in *Inspection* scenario where configuration 3 yields higher utilization rate compared to configuration 4 and also in *Industry* where the situation is opposite, i.e., configuration 4 yields higher utilization rate, except when k is 3. In this use case the human operator utilizes fleet knowledge during task execution by means of preferring those robots during task execution which yield higher utilization rates.

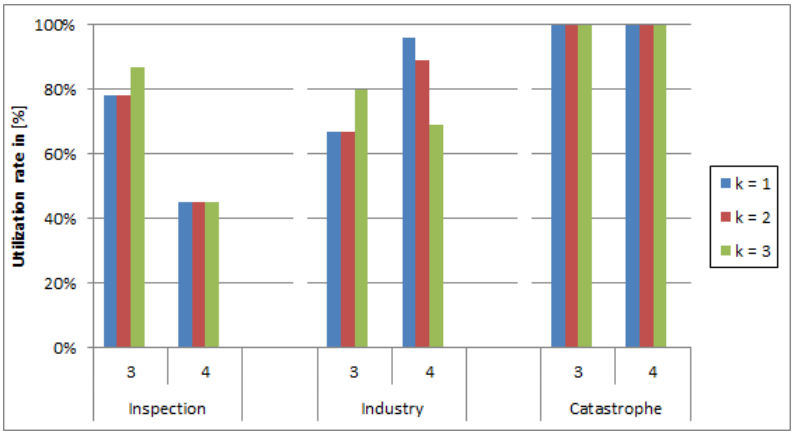

Figure 7.16: Utilization rate in adaptive autonomy

7.5.4 Load Balancing

Table 7.16 shows the number of tasks each robot in a fleet executes during a mission utilizing hSKIM. Listed values are for configurations 3 and 4 and are related to 3 scenarios. Same as in the previous sections, the value for parameter k is varied from 1 to 3. Figure 7.17 (left) illustrates the values for configuration 3 and Figure 7.17 (right) illustrates the values for configuration 4.

Table 7.16: Load balancing in adaptive autonomy

		Inspection			**Industry**			**Catastrophe**		
		1	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	**Robot 1**	4	3	3	0	0	6	0	0	0
	Robot 2	3	3	5	0	0	8	0	0	0
	Robot 3	3	4	2	10	10	6	10	10	10
4	**Robot 1**	2	3	3	3	3	4	0	0	0
	Robot 2	5	4	4	3	1	5	0	0	0
	Robot 3	3	3	3	4	6	4	10	10	10

In *Inspection* scenario the number of executed tasks is almost equally balanced across all robots. There are few outliers when a robot executes either 2 or 5 tasks. The results become more interesting in *Industry* scenario, especially when parameter k is either 1 or 2. For both k values, in configuration 3, robot 3 executes all tasks. As explained in previous sections, this is because it has the highest matching degree compared to the other two robots. This comes at price that configuration 3 yields lower utilization rate and requires more messages than the configuration 4 in the same setup. When k is 3 in *Industry* scenario, the total number of tasks, including sub-tasks as well, that appear in the system is much higher when configuration 3 is utilized compared to configuration 4, i.e., 20 in configuration 3 and 13 in configuration 4. This is because the robots which provide 1 skill are still able to, at least partially, execute tasks in *Industry* scenario when k is 3. Due to the partial execution, there are many new tasks in the system. Consequently, the same setup yields higher utilization rate, but it requires more messages and has lower task allocation rate.

Due to the highest matching degree, robot 3 executes all tasks in *Catastrophe* scenario independently of the selected configuration and the value for parameter k.

7.5.5 Human Activity

Table 7.17 shows how many times a human operator is consulted during a mission execution utilizing hSKIM. Results are listed with respect to configurations 3 and 4 and 3 scenarios. Same as in previous sections, the value for parameter k is varied from 1 to 3. Figure 7.18 illustrates the values listed in Table 7.17.

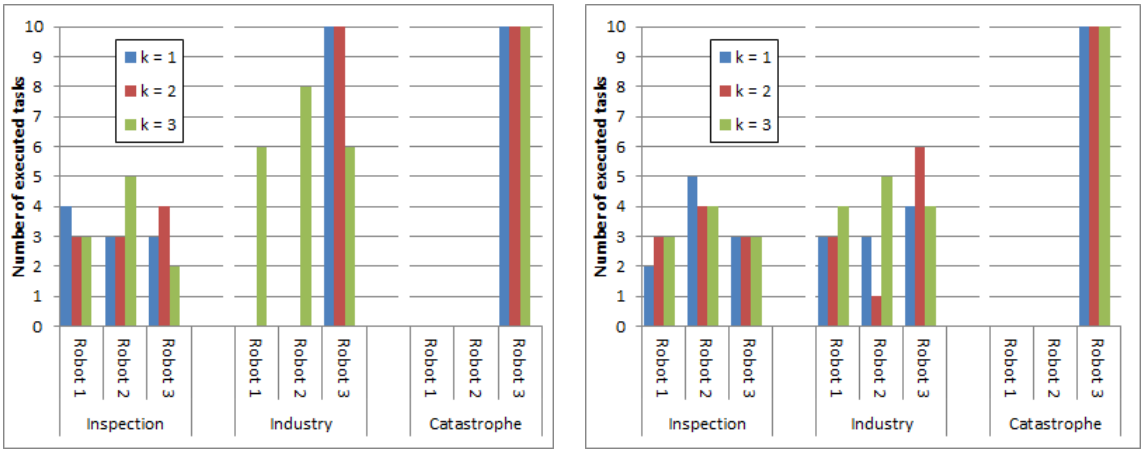

Figure 7.17: Number of executed tasks utilizing configuration 3 (left) and configuration 4 (right)

Table 7.17: A number of times a human has been consulted

	Inspection			**Industry**			**Catastrophe**		
Conf \ k	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	10	0	0	26	10	0	29	10	10
4	10	0	0	12	18	0	29	28	10

As expected, human activity decreases with the increase of parameter k. The higher parameter k is, the lower is human activity. When parameter k is 1 in *Industry* scenario, robots in configuration 3 consult the human more frequent than those in configuration 4. This is related with the same scenario in the task allocation rate in Section 7.5.1 where robots 1 and 2 try to execute a task, but when they consult a human to get a permission to execute the task, the human will refuse to grant a permission because there exists a robot with a higher matching degree, i.e., robot 3. This phenomenon reflects also in the increased number of exchanged messages in the same setup.

When parameter k is 2 in *Industry* and *Catastrophe* scenarios, robots in configuration 3 consult the human less frequent than those in configuration 4. This is due to the fact that only one robot in configuration 3, i.e., robot 3, is allowed to execute tasks in *Industry* and *Catastrophe* scenarios. When k is 3, in both configurations only robot 3 is allowed to execute tasks. Thus, both configurations yield the same number of human interactions in *Catastrophe* scenario.

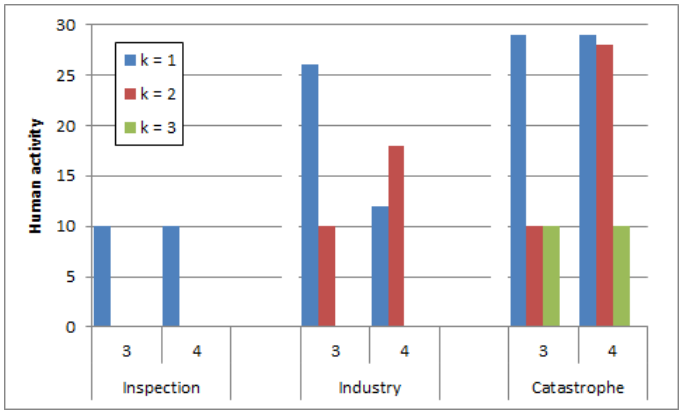

Figure 7.18: A number of times a human has been consulted

7.5.6 Mission Execution Time

Table 7.18 shows the mission duration in [s] for configurations 3 and 4 in 3 scenarios utilizing hSKIM. Same as previous sections, the value for parameter k is varied from 1 to 3. Figure 7.19 illustrates the values listed in Table 7.18.

Table 7.18: Mission execution time in [s]

	Inspection			**Industry**			**Catastrophe**		
Conf \ k	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
3	29	30	28	46	49	50	55	55	55
4	29	30	30	35	40	38	58	56	56

In general, the mission duration increases with the scenario complexity, i.e., it is lowest in the *Inspection* scenario and highest in *Catastrophe* scenario. This is because the more complex the tasks are, more robots participate in their execution and thus the coordination requires more resources, e.g., more coordination messages have to be exchanged to coordinate collaborating robots. Additional factor responsible for this phenomenon is the more frequent execution of inference-based task allocation algorithm, as well as human interaction, during the complex missions.

Missions durations do not vary much with respect to the value of parameter k. Mission duration varies up to approximately 15% in *Industry* scenario when a robotic fleet utilizes configuration 4 and k changes from 1 to 2. On the one hand, in *Inspection* and *Catastrophe* scenarios, both configurations have similar execution times, i.e., difference is not more than 3 s. On the other hand, in *Industry* scenario, configuration 3 performs up to 12 s slower than configuration 4. This is due to the fact that only robot 3 in configuration 3 is allowed to execute tasks. The other robots also try to execute tasks, they ask a human

operator for permission, but does not get it. Consequently, the number of messages in this setup is increased as well, especially when k is 1 and 3.

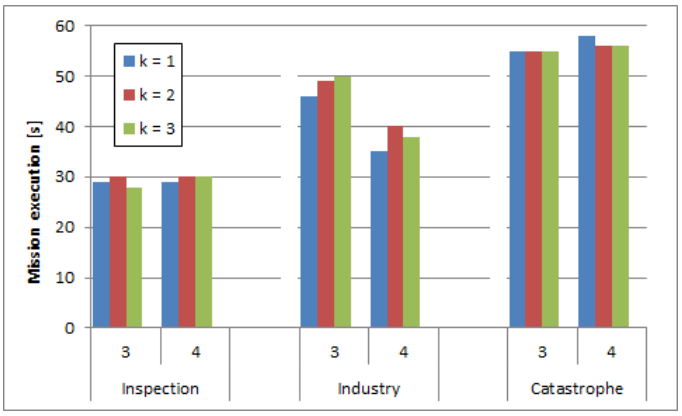

Figure 7.19: Mission execution time in adaptive autonomy

7.6 Flexibility and Robustness with Model-Driven Architecture

This section qualitatively evaluates the influence of Model-Driven Architecture approach introduced in Section 4. In particular, the evaluation refers to the use cases and simulations addressed in Section 7.4 and Section 7.5. Examined use cases serve as a basis for evaluating following criteria addressed in MDA in Chapter 4: (1) how SKIM abstracts heterogeneous resources in a system, (2) cross-domain operation, and (3) how SKIM abstracts coordination and adaptive autonomy in a fleet.

7.6.1 Abstraction of Heterogeneous Resources

Heterogeneous resources in an operating system can belong to either tasks or robots. Due to the ontologically modeled resources, it is more precise to say that heterogeneous resources belong either to class *Task* defined in SKIM-RO or class *Robot* defined in SKIM-CO. Nevertheless, the aim of this section is to show that both, SKIM-RO and SKIM-CO, ontologies for describing resources support changing requirements. In particular, that they support modeling of different tasks and different (heterogeneous) robots. According to MDA, it is important that the underlying coordination framework, i.e., SKIM, remains unchanged.

Section 7.3 introduces various input instances which are, in Section 7.4 and Section 7.5, used as input parameters for numerous simulations performed on various implementations of SKIM coordination framework. Defined input instances include various types of tasks and robots as well. Former are differentiated based on a number of skills required to execute a task, and latter are differentiated based on a number of skills a robot provides

for a task execution, i.e., a configuration. General model of a task is captured in SKIM-RO where class *Task* defines properties that an instances has to have in order to be classified as a *Task*. The general task description, i.e., class *Task*, does not limit any instances with respect to the number of required skills. Consequently, this infers that any instance which has at least one required skill is type of *Task*. The same notion applies for robots as well.

Since the underlying SKIM implementation operates only on class definitions, e.g., classes *Task* and *Robot*, it does not depend neither on the number of skills a task requests nor on the number of skills a robot provides. Thus, as it is shown in experiments conducted in Section 7.4 and Section 7.5, that each SKIM implementation, i.e., cSKIM, dSKIM, and hSKIM, successfully and completely executes any mission independently of operating scenario, i.e., different tasks, and utilized configuration, i.e., heterogeneous robots.

Although the current ontologies, i.e., SKIM-RO and SKIM-CO, are limited to model only skills and the amount of resources, they can easily be extended with additional parameters. New parameters have to be handled in task allocation algorithms as well. Current ontologies can be utilized to model any mission which encompasses tasks that require some skills for execution, e.g., demining a field. Due to the resources definition abstracted in ontologies, i.e., SKIM-RO and SKIM-CO, SKIM coordination framework successfully handles heterogeneous resources in a system.

7.6.2 Domain-independent Fleet Operation

Operating domain of a robotic fleet consists of various tasks. Usually, one type of an operating domain encompasses same or similar tasks. In the scope of this thesis those are tasks with the same or similar requested skills. The MDA approach introduced in Chapter 4 utilizes an ontology, i.e., SKIM-RO, to facilitate modeling of different types of operating domains named scenarios in the scope of this thesis. Similar to the Section 7.6.1, the aim of this section is to show that ontologically modeled scenarios do not influence a mission execution performed in the SKIM coordination framework.

Section 7.3 introduces three different representatives of class *Scenario* defined in SKIM-RO used to classify generated tasks into three different scenarios with respect to the different task complexities. The scenario classes have the same number of tasks, but different number of required skills per task, i.e., task's complexity. To emphasize SKIM generality in the sense of application domains, the framework is evaluated in Section 7.4 and Section 7.5 using all three representatives of class *Scenario*. Although the introduced scenarios differ in terms of a number of skills and the amount of resources, the evaluation was only focused on a number of skills.

Since the underlying SKIM implementation operates only on class definitions, e.g., representatives of class *Scenario*, it does not depend on a scenario to which a task belongs to and thus on a number of skills the task requires. As it is presented in experiments conducted in Section 7.4 and Section 7.5, each SKIM implementation, i.e., cSKIM, dSKIM, and hSKIM, successfully and completely executes any mission independently of operating scenario, i.e., tasks' complexity.

Although the existing SKIM-RO utilizes only the number of skills and the amount of resources to model an operating domain, it can be easily extended with additional parameters, e.g., the size of a task. Due to the domain description abstracted in SKIM-RO ontology, SKIM coordination framework retains generality with respect to the application domains.

7.6.3 Abstraction of Coordination and Adaptive Autonomy in a Fleet

Coordination and adaptive autonomy are abstracted in an ontology-based model of shared knowledge captured in SKIM-CO. In contrast to cSKIM and dSKIM which lack a support for ontology-based coordination and human interaction, hSKIM utilizes coordination model defined in SKIM-CO in terms of *Capability* and *User* classes that enable task allocation and human interaction, e.g., whether to involve the centralized task allocation component to find collaborative robots and whether to involve a human in task assignment. Similar to the previous sections, the aim of this section is to show that the customization of ontologically defined coordination capacities is loose-coupled with the underlying coordination framework, i.e., it does not require hSKIM to change any code to address changes in SKIM-CO.

Coordination capacities defined in SKIM-CO encompass classes *Capability* and *User*. Representatives of the former act as input parameters for the inference-based task allocation algorithm and thus are responsible for mapping tasks to robots in a fleet. Latter models the level of human interaction with the system. Human interaction is bounded to: (1) provide a permission for a robot to execute a specific task, (2) to allocate tasks which the inference-based task allocation algorithm cannot allocate, and (3) to limit simple robots to execute complex tasks, i.e., those robots which provide less skills than a task requires. Latter is important in order to prefer robots with higher matching degree, i.e., exact or subsume, during a task execution.

Tests conducted in Section 7.5 demonstrate the influence of the level of adaptive autonomy in hSKIM coordination framework. Due to the operation on class definitions, e.g., representatives of classes *Capability* and *User*, instead of on real instances, hSKIM implementation successfully and completely executes any mission independently on the level of autonomy. Moreover, hSKIM can also execute a mission if representatives of class *Capability* are missing. In that case, the inference-based task allocation algorithm will be disabled and a human will have to allocate all tasks instead.

Semantics enable abstraction of coordination rules and adaptive autonomy in SKIM-CO and thus prevents hSKIM from changing code to host different coordination patterns and different levels of autonomy. The benefit of using semantics for modeling coordination in a robotic fleet can be perceived by means of an automatized tasks classification which further drives the task allocation in the robotic fleet.

7.7 Recommendation for Coordination Approach and Human Interaction

Finally, the last section gives an overview, together with recommendations, on selecting an appropriate coordination approach based on an available scenario and a configuration considering evaluated parameters. First, the section compares three coordination approaches and visualizes results in Figure 7.20. After that, the focus is shifted on a comparison of three different levels of adaptive autonomy implemented in hSKIM. Latter comparison is visualized in Figure 7.21. Due to the numerous evaluated parameters and different input values, this section cannot visualize and compare all possible outcomes. For complete set of results, Sections 7.4 and 7.5 have to be consulted.

Figure 7.20 visualizes results from the simulations performed in Section 7.4. X-axis on the graph illustrated in Figure 7.20 has four values denoting parameters evaluated in Section 7.4, i.e., TA - task allocation rate, UT-utilization rate, COMM - communication overhead, and DUR - mission duration. Y-axis denotes three scenarios, while Z-axis present two configuration classes, a peer class and a team leader class. Markers on the graph represent different decisions which recommend the most appropriate coordination approach for given (x,y,z) values. In particular, a marker denotes which coordination approach to use under a given configuration and a scenario with respect to the specific parameter.

Being interested in maximizing task allocation and utilization rates while having a configuration with a dominant robot on disposal, it is recommended to use hSKIM coordination approach independently of an operating scenario (blue triangles associated to TA and UT values). In particular, hSKIM outperforms the other two coordination approaches when it operates in complex *Catastrophe* scenarios. On the other hand, it does not matter which coordination approach is utilized when a configuration is a member of peer class (black circles associated to TA and UT values). In that case, all three coordination approaches yield same performances.

Observing the communication overhead, i.e., the number of exchanged messages, while operating in *Inspection* scenarios, hSKIM decreases the number of exchanged messages compared to the other two coordination approaches independently of provided configuration (blue triangles associated to COMM). In contrast, when operating in *Industry* scenarios, cSKIM needs less messages to complete a mission compared to dSKIM and hSKIM. cSKIM outperforms the other two independently of selected configuration (green squares). In *Catastrophe* scenarios, decision on a recommended coordination approach depends on provided configuration. On the one hand, when operating on a configuration that is a member of peer class, cSKIM outperforms the other two. On the other hand, when a configuration is a member of team leader class, hSKIM performs best, i.e., it needs less messages than the other two approaches. Finally, when the mission execution time is a parameter to minimize, cSKIM is the approach to use. Due to the inference-based task allocation algorithm and human interaction, hSKIM performs slowest, i.e., it has longest mission execution times.

In general, hSKIM coordination approach should be preferred whenever a config-

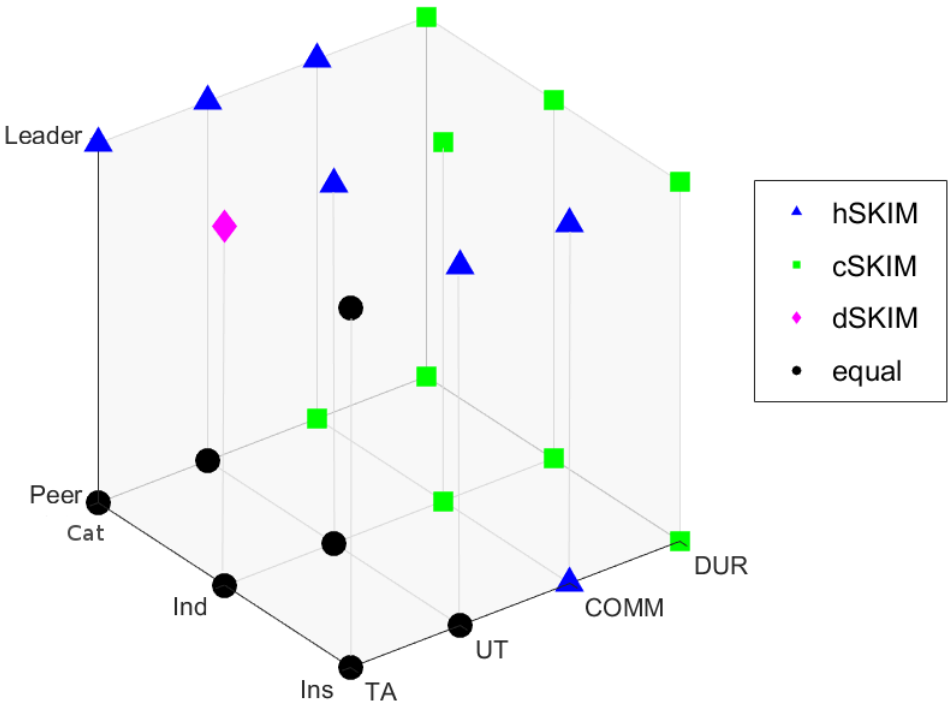

Figure 7.20: Recommendation for a coordination approach

uration belongs to team leader class with an exception when the mission execution time matters. Blue triangles, associated with *Leader* value on z-axis in Figure 7.20, illustrate hSKIM prevalence. Furthermore, having a configuration from peer class while focusing on communication overhead and execution time, cSKIM should be preferred coordination approach (green squares). For other two parameters, all approaches have same performances (black circles).

When hSKIM coordination framework is selected as an output observing Figure 7.20, there is still a decision to be made with respect to the level of autonomy. Figure 7.21 illustrates which autonomy level performs best in a specific scenario utilizing a configuration from team leader class. Therefore, hSKIM as an output from Figure 7.20 serves as an input for Figure 7.21. X-axis on the graph illustrated in Figure 7.21 has four values denoting parameters evaluated in Section 7.5, i.e., TA - task allocation rate, UT-utilization rate, COMM - communication overhead, and HA - human activity. Y-axis denotes three scenarios, while Z-axis present two configurations, i.e., configurations 3 and 4. Moreover, markers on the graph represent different decisions which recommend the autonomy level for given (x,y,z) values. In particular, a marker denotes which autonomy level in hSKIM is preferred under a given configuration and a scenario with respect to the specific parameter. As discusses in Section 7.5, parameter which denotes the autonomy

level has a value which is a non-negative integer between 1 and 3 inclusive.

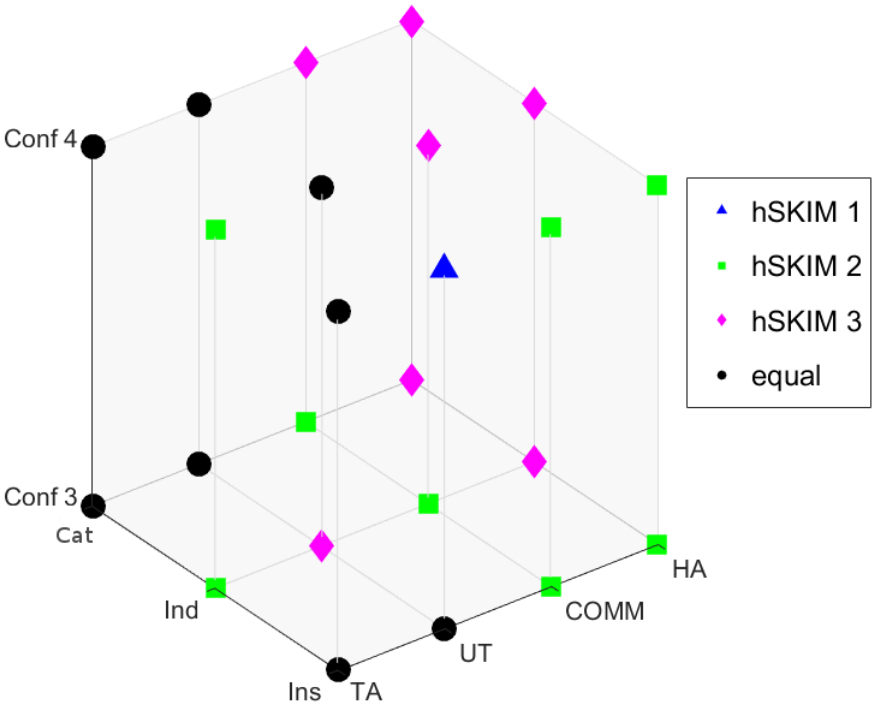

Figure 7.21: Recommendation for human interaction

Having the maximization of task allocation and utilization rates in focus, in most cases all autonomy levels perform equally (black circles in Figure 7.20). Exception occurs when a robotic fleet operates in *Industry* scenarios and when a focus is on task allocation parameter. In that case, independently of selected configuration, the autonomy level set to 2 outperforms the other two autonomy levels. When a focus shifts to the communication overhead, the autonomy level set to 2 outperforms the other two autonomy levels when configuration 3 is utilized (green squares). Switching to configuration 4, the autonomy level set to 2 outperforms the other two in *Industry* and *Catastrophe* scenarios (pink diamonds). Observing the human activity, it is the lowest for the autonomy level set to 2 in *Inspection* (green squares). In other two scenarios, the lowest human activity is a consequence of setting the autonomy level to 3 (pink diamonds). In neither case, the activity does not change with the selected configuration.

In general, the autonomy level set to 2 should be preferred approach when a focus is on minimizing the communication overhead. Moreover, the autonomy level set to 3 should be preferred approach when a focus is on minimizing a human activity. On the other hand, the autonomy level does not influence to a great extent task allocation

and utilization rates. Detailed test results, as well as the relations between observed parameters, are discussed in Section 7.5.

CHAPTER 8

Conclusion

Today's multi-robot systems can be seen as complex systems in the sense that they have to deal with heterogeneous resources, have to communicate, coordinate and collaborate, as well as to interact with human operators. All these processes are performed in distributed, heterogeneous, unstructured, dynamic, and decentralized environments. Therefore, developers of robotic middlewares have to deal with issues like heterogeneity and varying size of resources, variety of communication paradigms, dynamically emerging resources, adaptability to a human presence. To address the listed issues, it was shown in this thesis how the extension of existing space-based paradigm, Semantic XVSM, by introducing the Semantic Web Technologies facilitated modeling of heterogeneous resources. In particular, a concept of an ontology was utilized to abstract heterogeneous resources, implement coordination rules, and model adaptive autonomy. The following sections summarize the obtained results and describe future directions.

8.1 Results and Contributions

The following subsections wrap up the results and contributions with respect to the research questions which were formulated in Chapter 1.

8.1.1 Heterogeneity and Complexity in Unstructured Environments

The first research question asked for the abstraction of heterogeneous resources in Semantic XVSM and whether the abstraction methodology can increase system flexibility.

Contribution 1 **Semantic XVSM abstracts heterogeneity issues**

Developers that are challenged to integrate heterogeneous components in one system are often faced with the lack of standardisation that exists between different devices and thus impedes seamless integration. To cope with these issues, system designers

are obliged to incorporate hard-wired interactions which deteriorate the overall system flexibility. The notion of flexibility pertains to the system's extendibility with new features, scalability, dynamic interactions, heterogeneous resources, human interaction, adaptability to changing requirements, reusability, etc.

With respect to the above listed challenges, this thesis detected two main sources which degrade the flexibility of multi-robot systems: (1) the lack of mechanisms to overcome resources heterogeneity resulting in inefficient communication, coordination, and collaboration, and (2) domain tailored and domain dependent interaction models which limit the reusability in similar application domains. These two issues are addressed in Chapter 4 where the presented approach harnesses the coherence of formal ontology-based modeling and Model-Driven Architecture approach. This resulted in two ontologies used to formally model various hot spots in multi-robot systems which usually obstruct the system flexibility and extendibility.

Introduced Model-Driven Architecture approach, implemented by means of ontologies, decoupled the system implementation from the specification resulting in increased domain independency and improved resources selection. It made the implementation general and independent of the selected configuration and operating environment. Furthermore, it is reflected how the synergy between Semantic Web Technologies and Model-Driven Architecture, provides a formal model of a system and facilitates the design of multi-robot systems. At the same time it increases the interoperability, scalability, adaptability, reusability, and robustness.

8.1.2 Distributed Task Allocation

The second research question dealt with the transition from the centralized to the distributed coordination approach together with the characteristics of a setup which benefits most from the distributed approach complemented with human interactions.

Contribution 2 Semantic XVSM facilitates distributed task allocation

This thesis utilized Semantic XVSM to show the complete transition from the centralized architecture to the distributed, human-aware, architecture style. The transition is reflected with the implementation of three different coordination approaches introduced in Chapter 5, i.e., centralized SKIM (cSKIM), distributed SKIM (dSKIM), and hybrid SKIM (hSKIM). First, the simplest one, i.e., cSKIM, was implemented where the coordination between distributed robots was realized by means of selecting tasks from a central repository and thus limiting robots to be aware of each other. After that, dSKIM was implemented where the coordination mechanisms were completely transferred to distributed robots. However, neither of these two supported human interaction. Consequently, hSKIM was implemented and it used coordination ontology (SKIM-CO) for modeling shared knowledge among humans and robots.

Due to the shared knowledge utilized for task allocation and for modeling human interactions, hSKIM is much more mature than the other two. The central feature, i.e., inference-based task allocation algorithm, is hereby implemented by the system's

reasoning engine, which employs logic-based inference mechanisms to infer new knowledge from explicitly available knowledge. In particular, this pertains to inferring task-robot mappings and modeling human interactions. Furthermore, logic-based inference also provides a very expressive mechanism for defining coordination rules. In many cases, ontologically supported coordination modeling allows for simpler and more intuitive definitions of coordination rules than the development of procedural algorithms.

For the evaluation of three coordination approaches, a benchmark framework that measured different metrics such as task allocation rate, communication overhead, load balance, was established in Chapter 7. The benchmark results showed that in terms of task allocation performance, the hSKIM coordination approach with adjustable autonomy outperforms traditional coordination approaches, i.e., cSKIM and dSKIM, in complex scenarios where tasks require multiple skills and a fleet is composed of the robots with different number of skills. Therefore, it can be concluded that, although the centralized task allocation approach performs well in simple scenarios, distributed inference-based task allocation approach is more suitable for complex scenarios where a collaboration and coordination between heterogeneous fleet members plays an important role.

8.1.3 Shared Knowledge for Adaptive Autonomy

The third research question addressed the notion of human interactions in a system and evaluated the impact of shared knowledge on adaptive autonomy.

Contribution 3 Shared knowledge as an effective means to evaluate the benefits of adaptive autonomy

The SKIM framework is designed with the objective to model shared knowledge as a basis for adaptive autonomy in mixed teams. The semantic approach drives the modeling of shared knowledge which enables the collaboration activities between involved entities by means of ontologies: SKIM-RO and SKIM-CO. Hence, the decisions are results of automated reasoning on them.

hSKIM implements two algorithms, described in Chapter 6, that introduced two different models of human interaction for facilitating decisions emerged due to the ambiguities occurred during the task allocation and robot-robot interactions. The first algorithm triggers reasoning on SKIM-CO which classifies two types of tasks: (1) input tasks at the beginning of a mission, and (2) ad-hoc tasks which emerge during the mission due to the skills mismatch between a robot and a selected task. The classification has two outcomes independently of the task type: (1) a task is mapped to a robot, or (2) there is no suitable robot for the given task. The outcome of mapping depends on SKIM-CO. However, an unmapped task does not imply that the task could not be executed. In that case, the task is assigned to a human who is assumed to have knowledge to solve task allocation problem. In the simulation the human actor performs the task allocation algorithm which utilizes knowledge on the fleet, thus complementing the inference-based task distribution. The second algorithm pertains to a robot which consults the human operator during a task execution. The human operator is consulted to take a decision

in a case when conflicting robots try to execute the same task. The decision-making algorithm performed by the human actor is based on a cost function including following parameters: (1) skill matching degree between a robot and a task, and (2) resource availability, i.e., resources for completing a task.

For the evaluation of different autonomy levels, a benchmark framework that measured similar metrics as the framework for evaluating three coordination approaches was established in Chapter 7. The benchmark results showed that the medium autonomy level should be a preferred approach when a focus is on minimizing the communication overhead. On the other hand, as expected, the highest autonomy level requires less human activities. Moreover, the task allocation and utilization rates are not influenced to a great extent when the autonomy level changes. However, detailed results are discussed in the benchmark in Chapter 7.

8.2 Future Work

Future work refers to research topics which strive for the improvement of hSKIM coordination approach:

1. Enhance the semantic modelling of the human decision-making mechanism with the focus on developing a support for different human decision-making models. The focus should be on introducing additional parameters in the existing model of adaptive autonomy, e.g., a parameter which describes the number of requests a human can handle per unit of time, as well as a support for having multiple humans providing different models in the same mission. This aspect is critical in missions where multiple humans, with different capabilities and skills, collaborate with a robotic fleet.

2. Extend the existing task model by introducing dependencies between tasks that build up a complex task. Introduce a hierarchy between tasks with respect to the priority and the order of task execution. This implies a new property in the ontology which will be used to build dependencies between task instances and which will have to be checked before a robot starts with a task execution.

3. Improve the existing desynchronization approach which currently utilizes random waiting times before a robot tries to execute a task and thus increases overall task execution time. The existing approach could be enhanced by introducing a distributed shared token which guaranties that a robot which owns a token has an exclusive right to select a task at time t and after that has to pass the token to another robot. Although the waiting time in this approach depends on the number of robots in a fleet, it could be potentially lower than in the current approach. However, it is necessary to address following criteria in this approach: (1) to decide which robot will initially get a token, (2) in which order robots pass token, and (3) what do the robots do while waiting on a token.

4. Upgrade the existing fail-over concepts to be able to handle situations when a robot fails, e.g., it becomes isolated from a robotic fleet due to the hardware issues or power loss. In that case the robot will be unable to write an allocated task back to the space. Thus, there should be a component which monitors executed tasks and generates a notification if some tasks are not executed due to the robots' failure.

However, the priority should be on the extension of human interaction model which should support more complex interactions and should scale with respect to the number of human operators.

Bibliography

Bibliography

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the martha project. *Robotics Automation Magazine, IEEE*, 5(1):36–47, Mar 1998.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[3] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation,*, 14(6):926–939, Dec 1998.

[4] M. S. Barisits. Design and Implementation of the next Generation xvsm Framework. Master's thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, June 2010.

[5] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[6] S. Biffl, R. Mordinyi, and A. Schatten. A model-driven architecture approach using explicit stakeholder quality requirement models for building dependable information systems. In *Fifth International Workshop on Software Quality, 2007. WoSQ'07: ICSE Workshops 2007.*, pages 6–12, May 2007.

[7] S. C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *Proceedings 1999 IEEE International Conference on Robotics and Automation, 1999.*, volume 2, pages 1234–1239 vol.2, 1999.

[8] N. Bouten, A. Hristoskova, F. Ongenae, J. Nelis, and F. De Turck. Ontology-driven dynamic discovery and distributed coordination of a robot swarm. In *Proceedings*

of the 6th IFIP WG 6.6 International Autonomous Infrastructure, Management, and Security Conference on Dependable Networks and Services, AIMS'12, pages 2–13, Berlin, Heidelberg, 2012. Springer-Verlag.

[9] J. M. Bradshaw, P. J. Feltovich, M. J. Johnson, L. Bunch, M. R. Breedy, T. Eskridge, H. Jung, J. Lott, and A. Uszok. Coordination in human-agent-robot teamwork. In *International Symposium on Collaborative Technologies and Systems, 2008. CTS 2008.*, pages 467–476, May 2008.

[10] Z. Brahmi and M. Gammoudi. Semantic shared space-based complex tasks allocation method for massive mas. In *2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009.*, pages 428–434, Aug. 2009.

[11] M. Broxvall. The peis kernel: A middleware for ubiquitous robotics. In *Proc. of the IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications*, 2007.

[12] J. Burke, R. Murphy, E. Rogers, V. J. Lumelsky, and J. Scholtz. Final report for the darpa/nsf interdisciplinary study on human-robot interaction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,*, 34(2):103–112, May 2004.

[13] Z. Butler and J. Hays. Task allocation for reconfigurable teams. *Robotics and Autonomous Systems*, 68:59 – 71, 2015.

[14] A. Campbell and A. S. Wu. Task and role allocation within multi-agent and robotics research, 2007.

[15] L. Chaimowicz, M. F. M. Campos, and V. Kumar. Dynamic role assignment for cooperative robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 293–298, 2002.

[16] J. Chen and D. Sun. Coalition-based approach to task allocation of multiple robots with resource constraints. *IEEE Transactions on Automation Science and Engineering*, 9(3):516–528, July 2012.

[17] H. Choset. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, 9(3):247–253, 2000.

[18] W. W. Chu, L. J. Holloway, M.-T. Lan, and K. Efe. Task allocation in distributed data processing. *Computer*, 13(11):57–69, Nov 1980.

[19] J. Conesa-Munoz, M. Gonzalez-de Soto, P. Gonzalez-de Santos, and A. Ribeiro. Distributed multi-level supervision to effectively monitor the operations of a fleet of autonomous vehicles in agricultural tasks. *Sensors*, 15(3):5402–5428, March 2015.

[20] N. Cote, A. Canu, M. Bouzid, and A.-I. Mouaddib. Humans-robots sliding collaboration control in complex environments with adjustable autonomy. In *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012*, volume 2, pages 146–153, Dec. 2012.

[21] S. Crass. A Formal Model of the Extensible Virtual Shared Memory xvsm and its Implementation in Haskell. Master's thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, Feb. 2010.

[22] S. Craß, E. Kühn, and G. Salzer. Algebraic foundation of a data model for an extensible space-based collaboration protocol. In *Proceedings of the 2009 International Database Engineering*, IDEAS '09, pages 301–306, New York, NY, USA, 2009. ACM.

[23] M. de Weerdt, Y. Zhang, and T. Klos. Distributed task allocation in social networks. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 76:1–76:8, New York, NY, USA, 2007. ACM.

[24] B. Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[25] T. Doenz. Design and Implementation of the next Generation xvsm Framework Operations, Coordination and Transactions. Master's thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, June 2011.

[26] D. Drenjanac, L. Klausner, E. Kühn, and S. D. K. Tomic. Semantic shared spaces for task allocation in a robotic fleet for precision agriculture. In E. Garoufallou and J. Greenberg, editors, *Metadata and Semantics Research*, volume 390 of *Communications in Computer and Information Science*, pages 440–446. Springer International Publishing, 2013.

[27] D. Drenjanac and S. Tomic. User interactions with robotic fleets for intelligent agriculture. In *Proc. Robotics and Associated High-Technologies and Equipment for Agriculture Applications of automated systems and robotics for crop protection in sustainable precision agriculture, Pisa, Italy*, Sep 2012.

[28] D. Drenjanac, S. Tomic, J. Aguera, and M. Perez-Ruiz. Wi-fi and satellite-based location techniques for intelligent agricultural machinery controlled by a human operator. *Sensors*, 14(10):19767–19784, October 2014.

[29] D. Drenjanac, S. Tomic, and E. Kühn. A semantic framework for modeling adaptive autonomy in task allocation in robotic fleets. In *IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2015*, pages 15–20, June 2015.

[30] D. Drenjanac and S. D. K. Tomic. Middleware Challenges in Robotic Fleets for Precision Agriculture. In *Proc. Recent Advances in Robotics and Mechatronics, Istanbul, Turkey*, Oct. 2013.

[31] D. Drenjanac and S. D. K. Tomic. User Interactions And Network Monitoring Ease Decision-Making In a Robotic Fleet For Precision Agriculture. In *Proceedings of the 2nd International Conference on Robotics and associated High-technologies and Equipment for Agriculture and forestry, Madrid, Spain*, jun 2014.

[32] D. Drenjanac, S. D. K. Tomic, L. Klausner, and E. Kühn. Harnessing Coherence of Area Decomposition and Semantic Shared Spaces for Task Allocation in a Robotic Fleet. *Information Processing in Agriculture*, June 2014.

[33] N. Edalat and M. Motani. Energy-aware task allocation for energy harvesting sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):1–14, 2016.

[34] J. Ejarque, M. de Palol, I. Goiri, F. Julia, J. Guitart, R. Badia, and J. Torres. Sla-driven semantically-enhanced dynamic resource allocator for virtualized service providers. In *IEEE Fourth International Conference on eScience, 2008. eScience '08.*, pages 8–15, Dec 2008.

[35] J. Ejarque, M. de Palol, I. Goiri, F. Julia, J. Guitart, J. Torres, and R. Badia. Using semantics for resource allocation in computing service providers. In *IEEE International Conference on Services Computing, 2008. SCC '08.*, volume 2, pages 583–587, July 2008.

[36] J. Ejarque, A. Micsik, R. Sirvent, P. Pallinger, L. Kovacs, and R. M. Badia. Semantic resource allocation with historical data based predictions. 2010-11-23 2010.

[37] J. Ejarque, R. Sirvent, and R. Badia. A multi-agent approach for semantic resource allocation. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010*, pages 335–342, Nov 2010.

[38] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming: Introduction. *Commun. ACM*, 44(10):29–32, Oct. 2001.

[39] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[40] S. S. Fatima and M. Wooldridge. Adaptive task resources allocation in multi-agent systems. In *The 5th International Conference on Autonomous Agents*, AGENTS '01, pages 537–544, New York, NY, USA, 2001. ACM.

[41] P. Fazli and A. Mackworth. Human-robot teams in entertainment and other everyday scenarios. In *the International Conference on ACE*, ACE 2009, 2009.

[42] D. Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In F. Aagesen, C. Anutariya, and V. Wuwongse, editors, *Intelligence in Communication Systems*, volume 3283 of *Lecture Notes in Computer Science*, pages 43–53. Springer Berlin Heidelberg, 2004.

[43] D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 2002.

[44] D. Fensel, R. Krummenacher, O. Shafiq, E. Kühn, J. Riemer, Y. Ding, and B. Draxler. Tsc triple space computing. *ei Elektrotechnik und Informationstechnik*, 124(1-2):31–38, 2007.

[45] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field d* algorithm. *Journal of Field Robotics*, 23:79–101, 2006.

[46] C. Fernandez-Quintanilla, D. Jose, J. Conesa-Munoz, and A. Ribeiro. A five-step approach for planning a robotic site-specific weed management program for winter wheat. In *1st International Workshop on Robotics and Associated High Technologies and Equipment for Agriculture, Montpellier, France*, Sep 2011.

[47] A. Finzi and A. Orlandini. Human-robot interaction through mixed-initiative planning for rescue and search rovers. In *Proceedings of the 9th Conference on Advances in Artificial Intelligence*, AI*IA'05, pages 483–494, Berlin, Heidelberg, 2005. Springer-Verlag.

[48] N. Fornara, G. Jezic, M. Kusek, I. Lovrek, V. Podobnik, and K. Trzec. Semantics in multi-agent systems. In S. Ossowski, editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 115–136. Springer Netherlands, 2013.

[49] E. Freedy, O. Sert, J. McDonough, G. Weltman, M. Tambe, T. Gupta, W. Grayson, and P. Cabrera. Multiagent adjustable autonomy framework (maaf) for multi-robot, multi-human teams. In *International Symposium on Collaborative Technologies and Systems, 2008. CTS 2008.*, pages 498–505, May 2008.

[50] E. Freeman, K. Arnold, and S. Hupfer. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1st edition, 1999.

[51] C. Galindo, J. Fernandez-Madrigal, J. Gonzalez, and A. Saffiotti. Using semantic information for improving efficiency of robot task planning. In *Proc. of the ICRA-07 Workshop on Semantic Information in Robotics*, Rome, Italy, 2007.

[52] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, Jan. 1985.

[53] B. Gerkey and M. Mataric. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768, Oct 2002.

[54] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 2004.

[55] T. R. A. Giele, T. Mioch, M. A. Neerincx, and J. C. Meyer. Dynamic task allocation for human-robot teams. In *ICAART 2015 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 1, Lisbon, Portugal, 10-12 January, 2015.*, pages 117–124, 2015.

[56] M. Goodrich, T. McLain, J. Anderson, J. Sun, and J. Crandall. Managing autonomy in robot teams: Observations from four experiments. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pages 25–32, March 2007.

[57] C. Goumopoulos, A. D. Kameas, and A. Cassells. An ontology driven system architecture for precision agriculture applications. *Int. J. Metadata Semant. Ontologies*, 4(1/2):72–84, May 2009.

[58] M. Gritti, M. Broxvall, and A. Saffiotti. Reactive self-configuration of an ecology of robots. In *Proc. of the ICRA-07 Workshop on Network Robot Systems*, pages 49–56, Rome, Italy, 2007.

[59] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *Proceedings of the Third Latin American Web Congress*, LA-WEB '05, pages 71–, Washington, DC, USA, 2005. IEEE Computer Society.

[60] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. *A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools.* The University of Manchester, 2011.

[61] P. Iñigo Blasco, F. Diaz-del Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Diaz. Robotics software frameworks for multi-agent robotic systems development. *Robot. Auton. Syst.*, 60(6):803–821, June 2012.

[62] M. Jager and B. Nebel. Dynamic decentralized area partitioning for cooperating cleaning robots. In *Robotics and Automation, 2002. ICRA '02. IEEE International Conference*, volume 4, pages 3577–3582 vol.4, 2002.

[63] S. Jeon and J. Lee. Multi-robot multi-task allocation for hospital logistics. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*, pages 339–341, Jan 2016.

[64] Y. Jiang, Y. Zhou, and W. Wang. Task allocation for undependable multiagent systems in social networks. *Parallel and Distributed Systems, IEEE Transactions on*, 24(8):1671–1681, Aug 2013.

[65] M. Johnson, P. Feltovich, J. Bradshaw, and L. Bunch. Human-robot coordination through dynamic regulation. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008.*, pages 2159–2164, May 2008.

[66] K. Jose and D. K. Pratihar. Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. *Robotics and Autonomous Systems*, 80:34 – 42, 2016.

[67] D. Khushraj, O. Lassila, and T. Finin. stuples: semantic tuple spaces. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.*, pages 268–277, Aug 2004.

[68] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, July 1995.

[69] L. Klausner. Semantic XVSM Desing and Implementation. Master's thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, Apr. 2013.

[70] C. S. Kong, N. A. Peng, and I. Rekleitis. Distributed coverage with multi-robot system. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2423–2429, May 2006.

[71] M. J. Krieger, J. B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992–5, 2000.

[72] E. Kühn. *Virtual Shared Memory for Distributed Architectures*. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[73] E. Kühn, S. Craß, G. Joskowicz, A. Marek, and T. Scheller. *Coordination Models and Languages: 15th International Conference, COORDINATION 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, chapter Peer-Based Programming Model for Coordination Patterns, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[74] E. Kühn, R. Mordinyi, L. Keszthelyi, and C. Schreiber. Introducing the concept of customizable structured spaces for agent coordination in the production automation domain. In *the 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 625–632, 2009.

[75] E. Kühn, R. Mordinyi, L. Keszthelyi, C. Schreiber, S. Bessler, and S. Tomic. Introducing aspect-oriented space containers for efficient publish/subscribe scenarios in intelligent transportation systems. In *Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 313–316, Sept 2009.

[76] E. Kühn and G. Nozicka. Post-Client/Server Coordination Tools. In *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents [ASIAN 1996 Workshop]*, Lecture Notes In Computer Science, pages 231–254, London, UK, 1998. Springer.

[77] E. Kühn and V. Sesum-Cavic. *A Space-Based Generic Pattern for Self-Initiative Load Balancing Agents*, pages 17–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[78] V. Kumar, D. Rus, and G. S. Sukhatme. Networked robots. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 943–958. Springer Berlin Heidelberg, 2008.

[79] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. An algorithm of dividing a work area to multiple mobile robots. In *Proceedings. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots'*, volume 2, pages 286–291 vol.2, Aug 1995.

[80] E. Lavendelis, A. Liekna, A. Nikitenko, A. Grabovskis, and J. Grundspenkis. Multi-agent robotic system architecture for effective task allocation and management. In *Proceedings of the 11th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications, and Proceedings of the 11th WSEAS International Conference on Signal Processing, Robotics and Automation, and Proceedings of the 4th WSEAS International Conference on Nanotechnology*, EHAC'12/ISPRA/NANOTECHNOLOGY'12, pages 167–174, Stevens Point, Wisconsin, USA, 2012. World Scientific and Engineering Academy and Society (WSEAS).

[81] K. Lerman, C. Jones, A. Galstyan, and M. J. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, pages 225–241, 2006.

[82] G. H. Lim, I. H. Suh, and H. Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,*, 41(3):492–509, May 2011.

[83] L. Liu and D. Shell. Optimal market-based multi-robot task allocation via strategic pricing. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[84] T. C. Lueth and T. Laengle. Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system. In *In IEEE/RSJ IROS*, pages 1516–1523, 1994.

[85] D. C. Mackenzie. Collaborative tasking of tightly constrained multi-robot missions. In *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings of the 2003 International Workshop on Multi-Robot Systems*. Kluwer Academic Publishers, 2003.

[86] K. Mandeep. Semantic resource discovery with resource usage policies in grid environments. *IJCSI International Journal of Computer Science*, 9(3):301–307, 2012.

[87] K. Marriott and P. J. Stuckey. *Introduction to Constraint Logic Programming*. MIT Press, Cambridge, MA, USA, 1998.

[88] M. J. Mataric, G. Sukhatme, and E. Ostergaard. Multi-robot task allocation in uncertain environments. *Autonomous Robots*, 14(2-3):255–263, 2003.

[89] M. J. Matarić and G. S. Sukhatme. Task-allocation and coordination of multiple robots for planetary exploration. In *Proceedings of the International Conference on Advanced Robotics*, pages 61–70, Buda, Hungary, Aug 2001.

[90] J.-N. Mazon, J. Trujillo, M. Serrano, and M. Piattini. Applying mda to the development of data warehouses. In *Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '05, pages 57–66, New York, NY, USA, 2005. ACM.

[91] S. J. Mellor, S. Kendall, A. Uhl, and D. Weise. *MDA Distilled*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[92] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 161–168, New York, NY, USA, 2003. ACM.

[93] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. A Review of Middleware for Networked Robots. *Intl. Journal of Computer Science and Network Security*, 9(5):139–148, May 2009.

[94] R. Mordinyi. *Managing Complex and Dynamic Software Systems with Space-Based Computing*. PhD thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, June 2010.

[95] R. Mordinyi, E. Kühn, and A. Schatten. Space-based architectures as abstraction layer for distributed business applications. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS),*, pages 47–53, Feb 2010.

[96] R. Mordinyi, T. Moser, E. Kühn, S. Biffl, and A. Mikula. Foundations for a model-driven integration of business services in a safety-critical application domain. In *35th Euromicro Conference on Software Engineering and Advanced Applications, 2009. SEAA '09.*, pages 267–274, Aug 2009.

[97] T. Moser, R. Mordinyi, S. Biffl, and A. Mikula. Efficient system integration using semantic requirements and capability models - an approach for integrating heterogeneous business services. In *11th International Conference on Enterprise Information Systems, ICEIS 2009, Milan, Italy*, pages 56 – 63, May 2009.

[98] J. Muñoz-Morera, I. Maza, C. J. Fernandez-Agüera, and A. Ollero. *Task Allocation for Teams of Aerial Robots Equipped with Manipulators in Assembly Operations*, pages 585–596. Springer International Publishing, Cham, 2016.

[99] A. Murphy, G. Picco, and G. Roman. Lime: a middleware for physical and logical mobility. In *21st International Conference on Distributed Computing Systems, 2001.*, pages 524–533, Apr 2001.

[100] M. Murth. *Efficient Coordination with Semantic Shared Data Spaces.* PhD thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, Apr. 2010.

[101] M. Murth and E. Kühn. A semantic event notification service for knowledge-driven coordination. In *1st International Workshop on Emergent Semantics and cooperaTion in opEn systEMs*, 2008.

[102] E. Nardini. *Semantic coordination through programmable Tuple spaces.* PhD thesis, University of Bologna, Apr. 2011.

[103] E. Nardini, A. Omicini, and M. Viroli. Semantic tuple centres. *Sci. Comput. Program.*, 78(5):569–582, May 2013.

[104] E. Nardini, A. Omicini, M. Viroli, and M. I. Schumacher. Coordinating e-health systems with tucson semantic tuple centres. *SIGAPP Appl. Comput. Rev.*, 11(2):43–53, Mar. 2011.

[105] N. Neela and S. Kailash. Resource matchmaking in grid - semantically. In *The 9th International Conference on Advanced Communication Technology,*, volume 3, pages 2051–2055, Feb 2007.

[106] A. Nilsson, R. Muradore, K. Nilsson, and P. Fiorini. Ontology for robotics: A roadmap. In *International Conference on Advanced Robotics, 2009. ICAR 2009.*, pages 1–6, June 2009.

[107] L. j. b. Nixon, E. Simperl, R. Krummenacher, and F. Martin-Recuerda. Tuplespace-based computing for the semantic web: A survey of the state-of-the-art. *Knowl. Eng. Rev.*, 23(2):181–212, June 2008.

[108] A. Omicini and F. Zambonelli. Coordination of mobile agents for information systems: the tucson model, 1998.

[109] J. E. Parker. Task allocation for multi-agent systems in dynamic environments. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1445–1446, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[110] L. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation,*, 14(2):220–240, Apr 1998.

[111] L. Parker. Multiple mobile robot systems. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 921–941. Springer Berlin Heidelberg, 2008.

[112] L. Paull, G. Severac, G. Raffo, J. Angel, H. Boley, P. Durst, W. Gray, M. Habib, B. Nguyen, S. Ragavan, G. Sajad Saeedi, R. Sanz, M. Seto, A. Stefanovski, M. Trentini, and H. Li. Towards an ontology for autonomous robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),*, pages 1359–1364, Oct 2012.

[113] Y. Pei, M. W. Mutka, and N. Xi. Coordinated multi-robot real-time exploration with connectivity and bandwidth awareness. In *2010 IEEE International Conference on Robotics and Automation (ICRA),*, pages 5460–5465, May 2010.

[114] V. Podobnik, K. Trzec, and G. Jezic. A multi-agent system for auction-based resource discovery in semantic-aware b2c mobile commerce. In *International Transactions on Systems Science and Applications*, volume 2, pages 169–182, 2007.

[115] G. Randelli. *Improving Human-Robot Awareness through Semantic-driven Tangible Interaction*. PhD thesis, "Sapienza" University of Rome, Department of Computer and System Sciences, Rome, Italy (forthcoming), 2011.

[116] A. S. Rao and M. P. Georgeff. Bdi agents: From theory to practice. In *Procedings of the first International Conference on Multi-agent Systems (ICMAS-95*, pages 312–319, 1995.

[117] A. Saffiotti and M. Broxvall. Peis ecologies: Ambient intelligence meets autonomous robotics. In *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI*, pages 275–280, 2005.

[118] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3*, AAMAS '02, pages 1191–1198, New York, NY, USA, 2002. ACM.

[119] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 727–734, New York, NY, USA, 2005. ACM.

[120] P. Scerri, D. Pynadath, and M. Tambe. Adjustable autonomy for the real world. In H. Hexmoor, C. Castelfranchi, and R. Falcone, editors, *Agent Autonomy*, volume 7 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 211–241. Springer US, 2003.

[121] M. Schneider-Fontan and M. Mataric. Territorial multi-robot task division. *IEEE Transactions on Robotics and Automation,*, 14(5):815–822, Oct 1998.

[122] C. Schreiber. Design and Implementation of MozartSpaces, the Java Reference Implementation of xvsm. Master's thesis, Vienna University of Technology/Space-based Computing Group, Argentinierstrasse 8 , 1040 Vienna, Sept. 2008.

[123] C. Schumacher, P. R. Chandler, and S. R. Rasmussen. Task allocation for wide area search munitions. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 3, pages 1917–1922 vol.3, May 2002.

[124] B. P. Sellner, F. Heger, L. Hiatt, R. Simmons , and S. Singh. Coordinated multi-agent teams and sliding autonomy for large-scale assembly. *Proceedings of the IEEE - Special Issue on Multi-Robot Systems*, 94(7):1425 – 1444, July 2006.

[125] S. Shakya and U. Prajapati. Task scheduling in grid computing using genetic algorithm. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT),*, pages 1245–1248, Oct 2015.

[126] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 655–661, 1995.

[127] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In D. Rus and S. Singh, editors, *Experimental Robotics VII*, volume 271 of *Lecture Notes in Control and Information Sciences*, pages 323–332. Springer Berlin Heidelberg, 2001.

[128] V. Singhal and D. Dahiya. Distributed task allocation in dynamic multi-agent system. In *2015 International Conference on Computing, Communication Automation (ICCCA),*, pages 643–648, May 2015.

[129] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29(12):1104–1113, Dec. 1980.

[130] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[131] V. Tamma, C. van Aart, T. Moyaux, S. Paurobally, B. Lithgow-Smith, and M. Wooldridge. An ontological framework for dynamic coordination. In Y. Gil, E. Motta, V. Benjamins, and M. Musen, editors, *The Semantic Web ISWC 2005*, Lecture Notes in Computer Science, pages 638–652. Springer Berlin Heidelberg, 2005.

[132] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based resource matching in the grid the grid meets the semantic web. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 706–721. Springer Berlin Heidelberg, 2003.

[133] A. Tews, M. J. Mataric, and G. S. Sukhatme. A scalable approach to human-robot interaction. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation, ICRA 2003, September 14-19, 2003, Taipei, Taiwan*, pages 1665–1670. IEEE, 2003.

[134] R. Tolksdorf, E. P. Bontas, and L. J. B. Nixon. Towards a tuplespace-based middleware for the semantic web. In *Proc. of 2005 Int'l Conf. on Web Intelligence, Compiege*, 2005.

[135] R. Tolksdorf and D. Glaubitz. Coordinating web-based systems with documents in xmlspaces. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 356–370. Springer Berlin Heidelberg, 2001.

[136] R. Tolksdorf, L. Nixon, E. Bontas, D. Nguyen, and F. Liebsch. Enabling real world semantic web applications through a coordination middleware. In A. Gomez-Perez and J. Euzenat, editors, *The Semantic Web: Research and Applications*, volume 3532 of *Lecture Notes in Computer Science*, pages 679–693. Springer Berlin Heidelberg, 2005.

[137] S. Tomic and D. Drenjanac. Agent-based simulation of new energy markets. In *8th International Conference on the European Energy Market (EEM), 2011*, pages 93–98, May 2011.

[138] V. X. Tran and H. Tsuji. Owl-t: An ontology-based task template language for modeling business processes. In *Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, SERA '07, pages 101–108, Washington, DC, USA, 2007. IEEE Computer Society.

[139] P. Velagapudi, P. Scerri, K. Sycara, H. Wang, M. Lewis, and J. Wang. Scaling effects in multi-robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008.*, pages 2121–2126, Sept 2008.

[140] A. Viguria, I. Maza, and A. Ollero. S+t: An algorithm for distributed multirobot task allocation based on services for improving robot cooperation. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008.*, pages 3163–3168, May 2008.

[141] J. Wang, M. Lewis, and P. Scerri. Cooperating robots for search and rescue. In *the 5th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, 2004.

[142] W. Wang and Y. Jiang. Community-aware task allocation for social networked multiagent systems. *IEEE Trans. Cybernetics*, 44(9):1529–1543, 2014.

[143] D. Woods, J. Tittle, M. Feil, and A. Roesler. Envisioning human-robot coordination in future operations. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,*, 34(2):210–218, May 2004.

[144] B. Woosley and P. Dasgupta. Multirobot task allocation with real-time path planning. pages 574–579, 2013.

[145] L. Wu, M. A. Garcia, D. Puig, and A. Sole. Voronoi-based space partitioning for coordinated multi-robot exploration. *Journal of Physical Agents*, 1(1), 2007.

[146] D. Ye, Q. Bai, M. Zhang, K. T. Win, and Z. Shen. An efficient task allocation protocol for p2p multi-agent systems. In *ISPA*, pages 11–18. IEEE, 2009.

[147] H. Zhu, M. Zhou, and R. Alkins. Group role assignment via a kuhn munkres algorithm-based solution. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,*, 42(3):739–750, May 2012.

[148] S. Zieba, P. Polet, and F. Vanderhaegen. Using adjustable autonomy and human-machine cooperation to make a human-machine system resilient - application to a ground robotic system. *Inf. Sci.*, 181(3):379–397, Feb. 2011.

[149] R. Zlot and A. Stentz. Complex task allocation for multiple robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005.*, pages 1515–1522, April 2005.

[150] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 3, pages 3016–3023, 2002.

Web References

[151] N3QL - RDF Data Query Language. `http://www.w3.org/DesignIssues/N3QL.html`, 2004. [Online: accessed 30-May-2014].

[152] Grid Resource Ontology. `http://www.unigrids.org/ontology.html`, 2005. [Online: accessed 1-October-2014].

[153] Rhea project. `http://www.rhea-project.eu/`, 2010. [Online: accessed 3-November-2011].

[154] Semantically Enhanced Resource Allocation. `http://www.bsc.es/sites/default/files/public/computer_science/grid_computing/sera_architecture.pdf`, 2010. [Online: accessed 1-December-2014].

[155] Apache Jena. https://jena.apache.org/, 2011. [Online: accessed 15-December-2014].

[156] Sesame. http://rdf4j.org/, 2011. [Online: accessed 18-January-2015].

[157] The XSB research group. http://xsb.sourceforge.net, 2012. [Online: accessed 15-December-2012].

[158] Pellet: An Open Source OWL DL reasoner for Java. https://github.com/complexible/pellet, 2013. [Online: accessed 15-December-2014].

[159] D. Beckett. RDF 1.1 XML Syntax). http://www.w3.org/TR/rdf-syntax-grammar/, 2014. [Online: accessed 15-July-2015].

[160] D. Beckett and T. Berners-Lee. Turtle - Terse RDF Triple Language. http://www.w3.org/TeamSubmission/turtle/, 2011. [Online: accessed 15-July-2015].

[161] T. Berners-Lee. The Semantic Web. http://www.w3.org/standards/semanticweb/, 2000. [Online: accessed 30-June-2015].

[162] T. Berners-Lee. Notation 3 Logic. http://www.w3.org/DesignIssues/Notation3, 2005. [Online: accessed 15-July-2015].

[163] T. Bray, D. Hollander, A. Layman, and R. Tobin. Namespaces in XML 1.0. (Third Edition). http://www.w3.org/TR/REC-xml-names/, 2009. [Online: accessed 15-July-2015].

[164] M. Dean, B. Grosof, H. Boley, P. Patel-Schneider, and I. Horrocks. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/, 2004. [Online: accessed 12-July-2015].

[165] D. Grant, Jan Beckett. RDF Test Cases. http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/, 2004. [Online: accessed 15-July-2015].

[166] F. Harmelen and D. McGuinness. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/, 2004. [Online: accessed 2-July-2015].

[167] M. Kifer and H. Boley. RIF Core Design. http://www.w3.org/TR/2007/WD-rif-core-20070330/, 2007. [Online: accessed 12-July-2015].

[168] B. McBride. RDF Schema 1.1. http://www.w3.org/TR/rdf-schema/, 2004. [Online: accessed 30-June-2015].

[169] B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004. [Online: accessed 30-June-2015].

[170] E. Prud'hommeaux, S. Harris, and A. Seaborne. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query/, 2013. [Online: accessed 2-July-2015].