2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Software Development Framework for Cooperating Robots with High-level Mission Specification

Hyesun Hong, Woosuk Kang, and Soonhoi Ha[1]

Abstract— **In recent years, there has been a growing interest in multiple robots performing a single task through different types of collaboration. There are two software challenges when deploying collaborative robots: how to specify a cooperative mission and how to program each robot to accomplish its mission. In this paper, we propose a novel software development framework to support distributed robot systems, swarm robots, and their hybrid. We extend the service-oriented and model-based (SeMo) framework [1] to improve the robustness, scalability, and flexibility of robot collaboration. To enable a casual user to specify various types of cooperative missions easily, the high-level mission scripting language is extended with new features such as team hierarchy, group service, one-to-many communication. The script program is refined to the robot codes through two intermediate steps, strategy description and task graph generation, in the proposed framework. The viability of the proposed framework is evidenced by two preliminary experiments using real robots and a robot simulator.**

I. Introduction

In recent years, there has been a growing interest in multiple robots performing a single mission collaboratively in various types. On the one hand, in a distributed robot system, robots are usually assigned different tasks to accomplish a common goal collaboratively. On the other hand, in swarm robotics, robot behavior is defined collectively without specification of the role of individual robots. A mixture of those two different collaboration styles is also possible. In the deployment of cooperating robots, there are two software challenges: how to specify the cooperative mission and how to program each robot to accomplish the mission. For general robot programming, we need to take into account a wide range of robot platforms, from insect-sized small mobile robots [2]–[4] to humanoid robots.

A traditional method to program a robot is to use the robot-specific programming environment provided by the robot manufacturer [5], [6]. This method is not adequate for the behavior specification of cooperating robots that may be heterogeneous. To increase the reusability of software, several robotic software platforms have recently been developed. The most widely used is robot operating system (ROS) [7], which provides a set of APIs (application programming interfaces) abstracting the hardware platform, and libraries and tools to enable robot programming agnostic of the hardware platform. While ROS is useful for programming of an individual robot, there is no API to specify a cooperative mission of multiple robots [8], [9]. Also, ROS programming is not easy for a casual user that has little knowledge of computer programming [10].

Two approaches have been proposed for software development of multiple robots: 1) bottom-up approach and 2) top-down approach. The bottom-up approach is to program the behavior of individual robots and their interaction with a predefined set of APIs for communication and synchronization between robots. This approach is widely used by extending the existent robot programming environment for distributed robot systems. Since it gives the developer complete control over the design, the developer is exposed to the overwhelming burden of design details such as synchronization and robust programming [11]–[13]. The top-down approach, on the other hand, is to abstract multiple robots into groups and specify their behaviors as a single robotic motion. Since it lacks expressive power to fine-tune specific robot behaviors, it is difficult to specify a cooperative mission in which heterogeneous robots behave differently [13]. Thus, it is applicable for swarm robotics.

To support both distributed robot systems and swarm robotics, we propose a novel software development framework that integrates the top-down and the bottom-up approaches synergistically. The proposed framework is based on the service-oriented and model-based (SeMo) framework [1] that supports distributed robot systems only. The unique characteristic of the SeMo framework is to separate high-level mission specification and low-level robot programming. In SeMo, a cooperative mission is specified as a sequence of services that the robots perform at a high level by a scripting language: grouping robots into teams and specifying the behavior of each team, assuming that all robots in a team perform the same specified service. The robots that perform different tasks should be assigned to different teams. While it supports communication between teams, it does not allow robots in a team to communicate with each other. Moreover, it has no consideration of robot failure, which is a common assumption in swarm robotics. Since the behavior of the robots is statically determined, if one robot fails to perform its role, the entire mission is affected.

In this paper, we extend the SeMo framework significantly to improve the robustness, scalability, and flexibility of robot collaboration, by adding some key features of swarm robotics in the scripting language. Unlike the fixed team formation in the SeMo framework, we add a team hierarchy, which allows the developer to form a group of robots dynamically in a team. A team of robots may have several groups that

*This research was supported by a grant to Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration, and by Agency for Defense Development (UD190018ID).

[1]Hyesun Hong, Woosuk Kang, and Soonhoi Ha are with the Department of Computer Engineering, Seoul National University, Seoul, South Korea {hshong, wecracy, sha}@snu.ac.kr

perform different services at the same time. Also, a new notion of a service, called *group service*, is introduced, which corresponds to the cooperative mission specification in the top-down approach. Moreover, intra-team communication via broadcasting and local information sharing, which are essential for swarm robotics, are supported in the extended framework. Thus, the proposed framework enables a casual user to specify various types of cooperative missions for distributed robot systems, swarm robots, and their hybrid.

The high-level mission specification is translated into extended dataflow graphs with the *strategy* description file in the SeMo framework. The file describes how to translate a high-level service into a set of tasks that each robot can perform. For each robot, it is assumed that all tasks are programmed and prepared in the task library, which corresponds to the bottom-up approach. The individual robot program is automatically generated from the extended dataflow graph. It relieves the programmer of the aforementioned burden of the bottom-up approach and reduces the risk of manual programming error drastically.

The viability of the proposed methodology is validated with two experiments; one with a distributed robot system and the other with a robot simulator. The former demonstrates the added features for distributed robot programming and the other for swarm robotics. We compare the number of lines of generated code to verify the productivity of the software.

We summarize the key contributions of this paper as follows.

- We propose a novel software development methodology for cooperating robots combining the top-down and bottom-up approaches synergistically. High-level mission specification enables a novice programmer to perform a mission scenario with multiple robots. And the extended dataflow model is used for robot behavior programming.
- A novel scripting language is extended to support swarm robotics as well as distributed robot systems with added features for dynamic group formulation, the definition of group services, local information sharing, and so on. It improves the scalability, flexibility, and robustness of multiple robots.
- From the extended syntax of the scripting language, we generate the extended dataflow graph to realize the specified mission and individual robot programs are automatically synthesized from the task graph specification.
- To support sharing information among multiple robots, we extend the dataflow model by adding another type of port for multicasting. We use multicasting to share information in the distributed fashion.

II. Background

In this section, we review the SeMo framework briefly with a distributed robotics example.

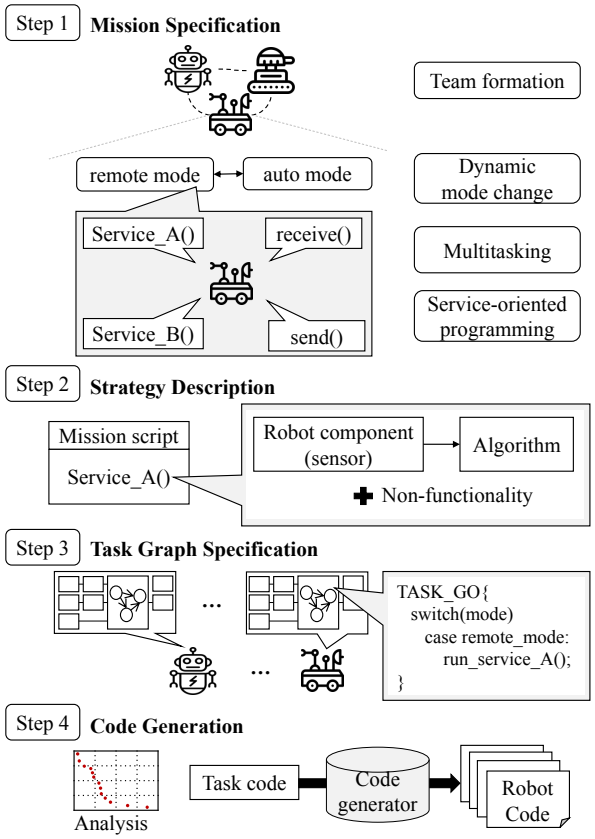


Fig. 1. Overview of the SeMo framework

A. An Example of Distributed Robotics System

A cooperative mission to scout a specific area is given to a set of heterogeneous robots. After they all move to a specific destination, a group of robots works together to find colored papers within a particular area and the other robots watch around the area to detect any danger. When any danger is detected, they signal to the searching robots to hide. When they are considered safe, they continue searching for papers again. The robots share the information of the colored papers they have found with each other, and that information is periodically reported to the user.

B. Service-oriented and Model-based Framework

SeMo [1] is a software development framework that separates high-level mission specification and robot behavior programming. The overall flow of the software development methodology is shown in Fig.1, which can be understood as the refinement process among four levels of abstraction in software development.

The first step is *mission specification* at the top level of abstraction with a scripting language. It involves team composition and service-oriented behavior specification of each team, allowing dynamic mode change of operation and multi-tasking. Fig.2 shows a snippet of mission specification associated with the example mentioned above. Robots are grouped into teams (line 1 and 2) and the behavior of each team is defined with a *composite service*; for instance, *ApproachDestination* (lines 3-5) and *Search* (lines 6-16) are

two composite services for SlaveTeam. The internal behavior of a *composite service* is defined by a sequence of services that the robots will perform. To express multi-tasking, the notion of *plan* [14] is introduced. There are two plans, *Listen* and *Action*, defined for SlaveTeam, as can be found on lines 17-19 in the example of Fig.2. Besides, the robot may have various operating modes, and mode change is triggered by events generated in a composite service. In Fig.2, how to express mode transition can be found on lines 20-25. Refer to [1] for more detailed information on the mission specification.

The second step is *strategy description*, which provides information on how to transform service into a set of tasks that each robot can perform. In case there are several methods to perform a service, we need to select one based on the provided information. Non-functional requirements such as execution time and power budget can be described at this step as well.

The next step is *task graph specification* that describes the internal behavior of each robot to accomplish its mission, based on the extended synchronous dataflow (SDF) model [15]. The task graph is composed of the tasks and channels between tasks. Unlike the mission specification, it is assumed that the task code is written by experts. Recently compute-intensive services such as vision and machine learning have become popular in robots. A compute-intensive service can be specified by a task subgraph that can be mapped to multiple processors for parallel processing. The extended SDF model, which defines formal semantics for communication between tasks and task execution conditions, allows us to make the task scheduling decision at compile-time and estimate the performance and resource requirements. The extended model uses a finite state machine to represent dynamic behavior and a special type of task, called library task, to manage shared resources among multiple robots.

The final step is automatic code generation, from the extended dataflow model to the target code that runs on each processor. From the same task graph, it generates different codes for different robot platforms automatically. It could improve the software design productivity significantly.

III. Proposed Methodology

In this section, we explain how we extend the SeMo methodology to increase the robustness, scalability, and flexibility of robot collaboration, by adding some key features of swarm robotics in the scripting language.

A. Service-oriented Mission Specification

The syntax of the scripting language is formally defined by the Backus-Nauer form (BNF). The suffixes "*", "+", and "?" mean "repeated zero or more times," "repeated one or more times" and "zero or one time," respectively. Due to page limitations, we focus on the extended syntax, omitting the unchanged syntax as the SeMo framework. In SeMo, the cooperative mission is described by a set of team services that may be changed dynamically depending on the mode of operation. The first extension is made to the composite

```
1  MasterTeam: iRobotCreate irobot                    # team formation
2  SlaveTeam: TurtlebotBurger burger[2], Ev3Robot ev3[2]
3  SlaveTeam.Action.ApproachDestination {
4    move (Paris palace of congress )  # composite service definition
5    throw find_color_paper } repeat (SlaveTeam.not_arrived)
6  SlaveTeam.Action.Search {
7    [[
8    group (instance of TurtlebotBurger) {
9      loop(2 SEC)
10       if  (SlaveTeam.lightness < 200 )
11         publish(SlaveTeam, SlaveTeam.Message = suggestHiding)}
12   others {
13     searchPaper();
14     publish(SlaveTeam, SlaveTeam.Message = suggestReturning) }
15   ]]
16 } ….
17 SlaveTeam.AUTODRIVE_MODE{          # multitasking per mode
18   set( Listen, CommOperator )
19   set( Action, ApproachDestination) } ….
20 SlaveTeam.main {                   # dynamic mode change definition
21   case (AUTODRIVE_MODE):
22     catch(find_color_paper): mode = SEARCH_MODE
23     catch(remote_control): mode = RC_MODE
24   case (SEARCH_MODE): ….
25   default: mode = AUTODRIVE_MODE }
```

Fig. 2. Mission scripting language example

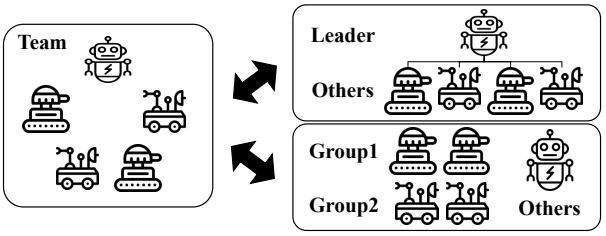


Fig. 3. Dynamic group allocation

service definition of a team as displayed below from lines 3-10 in the BNF form.

```
1 <Service> ::= <TeamName>.<PlanName>.<CompServiceName>
2              { <Stmt>+ } <RepeatStmt>?
3 <Stmt> ::= <GeneralStmt> | <GroupStmt>
4 <GroupStmt> ::= [[ <LeaderDef> <OtherStmt>? ]]
5             | [[ <GroupDef>+ <OtherStmt>? ]]
6 <LeaderDef> :: = leader(<GroupCondition>) {<GeneralStmt>+}
7 <GroupDef> ::= group(<GroupCondition>) {<GeneralStmt>+}
8 <OtherStmt> ::= others {<GeneralStmt>+}
9 <GroupCondition> ::= instance of <RobotType>+
10                  | capable of <RobotCapability>+
```

We introduce the team hierarchy in which the robots in a team can be grouped dynamically. While team formation is statically defined at the beginning of the mission script, groups are defined inside the composite service to support dynamic grouping. We support two types of group structure, leader-follower structure [12] and peer-peer structure [16], [17] as defined in line 4 and 5 in the above BNF description. As illustrated in Fig.3, the leader-follower structure is a vertical structure in which one robot is designated as the leader to control the other robots. The peer-peer structure, on the other hand, has a horizontal relationship between robots.

We can group robots based on the type or the capability of the robot (line 9 and 10 in the BNF description). For

example, robots with a camera sensor can perform a video shooting service. In the composite service of a team, we may divide the robots into multiple groups that are assigned different services. A pair of symbols, [[and]], are used to specify concurrent execution of services by multiple groups of the robots. Note that a robot can belong to one group at most; if it satisfies more than one grouping condition in a composite service, it has to choose one group arbitrarily. In the example of Fig.2, *SlaveTeam* is divided into two groups in the *Search* service, as shown in lines 7-15. One group that is of type *TurtlebotBurger* detects a dangerous condition, while the other group that consists of the remaining robots search for the color papers.

The second extension is the introduction of group service that can be performed by two or more robots together without the specification of the role of individual robots. It is an example of a top-down specification for swarm robotics. How to perform a group service is elaborated in the strategy description in the proposed methodology. The service *SearchPaper* described in Fig.2 can be defined as a group service, for instance. Since we assume that a robot in a group may fail during operation, the group service is performed collaboratively by available robots.

As the third extension, one-to-many communication is added. The SeMo framework supports one-to-one communication only with two APIs, *send* and *receive*. Since multicasting can be realized by multiple one-to-one communications, it can specify the cooperative mission of a distributed robotics system as long as all robots are live during operation. Unfortunately, any robot may fail in swarm robotics, so it is recommended to use broadcasting. To this end, a pair of new APIs, *publish* and *subscribe*, is introduced. In line 11 and 14 of Fig.2, broadcasting communication is used for each group to send a message.

Last but not least, a new semantics for robot synchronization is defined. Since the performance of robots has a large variation, the robots in a team may sit in different execution states. For group services, however, we need to synchronize the robots. Thus, in the proposed semantics, all robots involved in a group service are synchronized at the start of the group service. In the case of the leader-follower structure of grouping, we may need to select a new leader at the group formation step if the previous leader fails during operation. The leader selection scheme is not described in the mission specification phase but the strategy description phase.

In summary, the proposed extension offers greater flexibility and robustness than the previous work, assuming that any robot may fail during the execution of group services, and grouping of robots can be made at run-time dynamically. Fig.2 shows how a group service is used in the service-oriented mission specification as a hybrid system example of distributed robotics and swarm robotics.

B. Model-based Task Graph Specification

In the proposed framework, the internal behavior of each robot is represented by task graphs following an extended

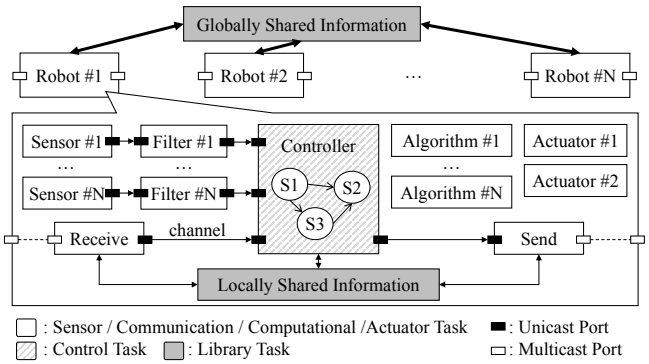


Fig. 4. Task graph specification for the internal behavior of a robot, supporting two types of information sharing

dataflow model of computation. The task graph consists of tasks and channels, as shown in Fig.4. A task is a software component that performs the service specified in the mission script. Following the dataflow semantics, a task communicates with other tasks through connected channels via ports. The task graph can naturally specify data dependencies between tasks. The task graphs are generated from the mission specification automatically, with the help of the strategy description that will be explained in the next subsection.

A major difference between a single robot and cooperating robots lies in information sharing. We distinguish two types of shared information: global information and local knowledge. Global information is maintained by a special type of the task, called *library* task [18] that defines a set of function interfaces inside. By mapping the library task to a designated robot, all robots can access the same global information by sending a query to the robot. Since the robot is a single point of failure, it is essential to keep the robot live at all times.

On the other hand, robots may want to share information locally with near robots, without guaranteeing the global consistency of the knowledge. This local knowledge sharing is useful for performing a group service. Adopting the knowledge sharing technique proposed in [19], we disseminate the local knowledge through broadcasting. Since the broadcasting message includes the creation time of knowledge, the robots can maintain up-to-date knowledge. For instance, if one robot discovers a yellow paper and receives a broadcasting message from a neighbor robot that a red paper is found, the robot broadcasts an up-to-date knowledge that both red and yellow papers are found. By delivering the received up-to-date knowledge to neighbors, all robots will be able to have globally shared knowledge eventually. This knowledge sharing method is an efficient way of sharing information between agents without paying a high cost of time synchronization. When time synchronization is critical for correct operation, the library task should be used for shared information management. While the task graph model in the SeMo framework only supports one-to-one communication using channel through ports, we add another type of port for multicasting, as shown in Fig.4.

Multicast is a group communication in which data transmission is addressed to a group of destinations. For instance,

in Fig.4, task *Robot*#1 communicates with task *Robot*#2 and *Robot#N* using a multicast port that is associated with an identifier(*id*). Since no explicit communication link is needed for multicast communication, we may vary the size of the communication group dynamically, allowing us to add or remove robots. Note that a multicast port is implemented with a buffer, and multicast communication does not guarantee the transfer of information between tasks. Therefore, multicast is not suitable when data must be delivered for sure.

C. Strategy Description

Since there is a large gap between the high-level mission script and the task graph specification of each robot behavior, an intermediate level of abstraction, called strategy description, is added in the SeMo framework. In this step, how to refine high-level services and values written in the mission script to the tasks in the task graph model is presented in the XML markup language. Since such refinement is dependent on the robot platform, the strategy description should be written by an expert that is familiar with the robot platform. For example, when detecting a color value, some robots can use the color sensor directly, while others need to use the color filter after capturing an image from the camera.

Since several extensions are made in the mission specification, the corresponding extension should be made in the strategy description. In particular, we need to clarify how to synchronize robots for the leader selection and execution of group service. It is also necessary to specify how to share the information and who has the information. For example, the service *SearchPaper* can be refined into a set of fine-grained services to accomplish the following action: move on wheels while detecting a new color paper, share the color information when it detects the new color, and check whether the goal is satisfied or not. A group service includes broadcasting information for knowledge sharing. Moreover, it is necessary to make available robots continue to carry out the group service even if some robots fail during operation. Besides, which algorithm to use for selecting leaders of groups should be expressed in this layer. Due to space limitations, we omit a detailed explanation and an example strategy description.

D. Automatic Task Graph Generation

Based on the strategy description file, the task graph specification can be automatically generated from a given mission specification. Since we assume that tasks associated with services and values are defined by robot developers and registered in the database a priori, we can instantiate all the required tasks from the mission specification in a straightforward fashion without any interconnection among tasks. With the instantiated tasks, we have to add dependency arcs between tasks by analyzing the dependency between services in the mission specification. In case the mission requires knowledge sharing, we identify the requested types of knowledge sharing and insert the additional tasks for knowledge sharing and associated communication ports and channels.

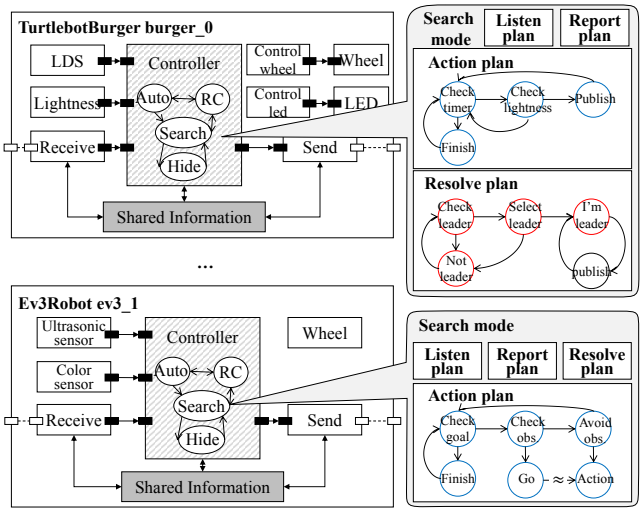


Fig. 5. Automatic generation of task graph from mission specification and strategy description

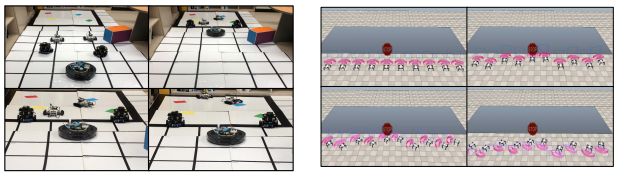
The most challenging is to synthesize the control task that realizes the dynamic behavior of each robot. In the mission specification, a robot may change the operation mode, and grouping is changed dynamically. Such dynamic behavior is expressed by a hierarchical finite state machine (FSM) inside a control task, *Controller* in Fig.4. The mode and plan information from the mission script is translated into the state transition diagram in the top-level FSM. In each mode, the sequence of services is translated into a bottom-level FSM.

The inside of the control task is shown in Fig.5 for the mission specification of Fig.2. Note that we generate different FSMs for two robots that belong to the same team since they belong to different groups in the *Search* composite service. Since the group conditions can be inferred from the candidate robots of the group in advance, the robots only contain a set of services that need to be performed. For example, a sensing task that checks brightness should be included only inside the TurtlebotBurger robot task, and only the parts that are necessary to search for color papers are created for Ev3Robot. If a leader exists in the team, the leader candidate robotic task should include a leader selection algorithm and periodically check the leader. Fig.5 shows that there are additional states that check the leader and separate what the leader or the rest need to do in the resolve plan.

From the generated task code, we synthesize the target code for each robot automatically, which corresponds to individual robot programming. Since it is done automatically, robot programming effort is minimized, while the detailed modeling of robot behavior is still possible in the proposed methodology.

IV. Experiments

To examine the viability of the proposed methodology, preliminary experiments are conducted with two cooperative mission scenarios; one is the example explained in section II-A with real robots, and the other is a simple swarm robotics



(a) (b)

Fig. 6. (a) Heterogeneous robots work together to find colored paper, (b) Quadruped robots share information to avoid falling

TABLE I
LINES OF THE GENERATED CODE

Robot	iRobotCreate	TurtleBot3 Burgers		Ev3Robot
		Raspberry Pi	OpenCR	
Mission	211			
Strategy	203	654		
Task code	278	213	128	240
Control task	207	475	-	455
Comm. task	155	270	-	196
Data Structure	1,168	2,141	375	1,163
Scheduler, etc	19,198	20,235	5,848	19,795
Robot code	20,916	23,334	6,351	21,849

example demonstrated with the V-Rep robot simulator [20]. Our experiments are demonstrated in the video[1] from mission specification to task code generation and actual robot execution. Some snapshots are shown in Fig. 6.

A. Scouting Mission with Heterogeneous Robots

For the scouting mission of section II-A, three different types of robots are used: one iRobotCreate [21], two TurtleBot3 Burgers [22], and two Ev3Robots [23]. The iRobotCreate is controlled by a Raspberry pi board with Ubuntu. TurtleBot3 Burger is equipped with a laser distance sensor, a light sensor, a LED, two wheels, and a Raspberry Pi 3 Model B+ (ARM Cortex-A53, quad-core, 1.4GHz) as a single-board computer and OpenCR 1.0 board [24](ARM Cortex-m7, 216MHz) as a micro-controller. Ev3Robot has a color sensor, a distance sensor, two motors, and a brick(ARM926EJ-S, 300MHz). All robots communicate with each other using Wi-Fi.

To verify the robustness of the group service, we test a scenario where the leader robot in the SlaveTeam fails during the operation. Since the leader robot is in charge of communication with the MasterTeam, its failure will fail the mission. However, the proposed framework selects a new leader after detecting the failure of the leader by periodic monitoring and completes the mission correctly. This experiment also confirms that we generate the target codes of different robots automatically from the translated task graph. Table I shows the number of lines in the generated code. The scenario represented by only 211 lines in the mission script language is converted into control task code, communication task code, and task code associated with the hardware component of the robot. For TurtleBot3 Burger, two wheels, LED, and brightness sensor are connected to the OpenCR board, so there are no additional control task and communication tasks. Only serial communication code is added to transfer the value of each task to the control task in the Raspberry Pi board. The generated actual robot code contains data structures for tasks, channels, and libraries, and target specific code for initialization and wrapup actions. In addition, target-independent APIs, used in the task graph, are redefined as target-specific APIs.

[1]We uploaded the video in YouTube: https://youtu.be/yugHlo_wr-0

B. Swarm Robotics Example

In this example, a number of 4-leg biomimetic robots that are equipped with a proximity sensor and a camera are marching in line, which is defined as a group service in the mission specification. When a robot detects an obstacle, it shares this knowledge with neighboring robots to stop marching and turn back. Without knowledge sharing, the other robots would crash the obstacle. The proposed framework, however, generates the code with knowledge sharing correctly to avoid crashing.

Furthermore, we change the number of robots from three to ten to test the scalability in swarm robotics. Since the detailed robot movement takes long in the V-Rep robot simulator, we used a small number of robots in this experiment. In the proposed framework, what a user has to do is to modify the number of robots in the mission scripting language. Then the codes for all robots are automatically generated regardless of the number of robots involved.

V. RELATED WORK

The robot software development frameworks of multiple robots have recently been researched.

While ROS is focusing on individual robots, several studies use two or more robots for a cooperative mission [25]. Even though ROS supports various languages such as Python, Java, and C++, there are no specific APIs defined for a cooperative mission. Hence it is difficult to specify a collaborative mission for robots [8]. Since ROS is based on a central node called *rosmaster* that provides naming and registration services for the rest of the nodes to discover one another, it is known that a robot may get disconnected in multi-robot systems due to unreliable network [9]. Thus, using one master inside each reliable network, which is typically one master per robot, is taken as a solution [26].

Karma [27] is a framework for programming and managing swarms of micro-aerial vehicles (MAVs) based on a centralized hive-drone model. A drone is an individual MAV that performs the specified commands without in-field communication. And the hive, as the central coordinator, orchestrates the drone and executes a given mission. The user specifies the cooperative mission by considering many drones as a swarm rather than individuals. The centralized hive maps the drones by location, and each drone works in that area and returns to the hive to share information, which leads to long information latency. Although this method has the advantage of easy decision making thanks to the centralized hive and

simplified programming of the MAV, it is not applicable for general distributed robot systems. Dolphin [28] is a programming language for autonomous vehicle networks. It assumes a centralized program like Karma, allowing a human to orchestrate an entire network of vehicles based on a global specification. Since the high-level abstraction helps the user to write a program without in-depth technical knowledge, Dolphin is included in Groovy as a domain-specific language. This work is similar to the proposed methodology in that it is easy for users to use with scripting languages, but it does not support swarm robotics.

Proto [29] is a functional programming language for homogeneous robots. It suggests a primitives library for group behaviors such as flock, scatter, disperse, and cluster-by [30]. It also provides five constructs, including behavior assignment to groups and activation of the group action [30]. The program specifies a swarm-like behavior and neighborhood-based computation. Several studies extend Proto. While Proto is cumbersome because it is a functional language and has LISP-like syntax, Protelis [31] is a more recent Proto-based language built into Java. And Protoswarm [32] extends to program the swarm of robots. They focus on the swarm robotics mission that is performed by homogeneous robots, which is different from the proposed methodology.

The language called Buzz [13] supports heterogeneous robots. Similar to our research, Buzz has taken both a top-down approach and a bottom-up approach. Buzz includes several constructs designed explicitly for top-down swarm-level development, such as primitives for group formation and management, local communication, and global consensus. Buzz is also designed to work with small systems. Since Buzz allows seamless mixing of bottom-up and top-down constructs in one language, it is difficult for novice programmers to use.

We summarize the comparison between the proposed methodology with some selected related work frameworks in Table II in terms of the following characteristics: 1) development approach, 2) the orchestration type, 3) the supporting type of cooperation, and 4) support of dynamic task allocation. The development approach can be a top-down approach, a bottom-up approach, or a mixture of these two. While most studies are using either the top-down approach or the bottom-up approach, Buzz and our proposed work, SeMo, take the mixture approach. The orchestration indicates whether multiple robots are centrally managed or not. Many studies, even the latest research [28], manage robots centrally, assuming that human intervention and control are necessary. ROS is classified as centralized in terms of multiple robots because communication between nodes (robots) is made through the name server called *rosmaster*. Since swarm robots usually involve a large number of nameless robots, decentralized orchestration is usually assumed. In contrast, SeMo considers both orchestration types. In the case of cooperation type, most of the researches consider only swarm robots or distributed robots, but not both. SeMo, however, supports both types of cooperation, which is a unique characteristic to the best of our knowledge. Lastly, the dynamic task allocation represents whether robots can assign groups dynamically or not, depending on their environments or situations. Dynamic task allocation is necessary for robust operation of a cooperating mission.

TABLE II
FRAMEWORK FOR PROGRAMMING MULTIPLE ROBOTS

Framework	Approach	Orchestration	Cooperation type	Dynamic allocation
ROS	Bottom-up	Centralized	Individual/ Distributed	No
Karma	Top-down	Centralized	Swarm	Yes
Dolphin	Top-down	Centralized	Distributed	No
Proto	Top-down	Decentralized	Swarm	Yes
Buzz	Mixture	Decentralized	Swarm	Yes
Ours	Mixture	Centralized & Decentralized	Both	Yes

TABLE III
COMPARISON WITH OTHER LANGUAGES

-	Python	Protoswarm	Dolphin	Buzz	Ours
Number of Keywords	31	28	61	64	38
Example 1	-	11	22	43	16
Example 2	-	19	24	43	11
Example 3	-	20	22	47	21
Example 4 (Service)	-	-	-	30	3

In Table III, we compare our mission script with other languages to compare the coding complexity quantitatively. The number of keywords is shown at the top of the table. Since Buzz and Dolphin are extensions of existing languages, they have a relatively large number of keywords. On the other hand, new language Protoswarm and our mission script have fewer keywords than them. We also compare the number of lines when representing the same example in each language. Example 1, 2, and 3 are simple scenarios taken from [30]. In Example 1, there are two teams, a red team and a blue team. The red team proceeds to the blue team, and then the blue team runs away when they find the incoming red team. The second example is about deployment, and the third example is more complicated than other scenarios. One team patrols to find another team. When they detect, they are scattered. Since Buzz is based on the bottom-up approach, the line of code is the longest. Example 4 is expressing a specific service in the Buzz example. Since our approach may use abstracted services, the number of lines can be significantly reduced. In summary, the proposed methodology is no harder to develop robot software than the related work.

VI. CONCLUSIONS

In this paper, we propose a novel software development framework to support distributed robot systems, swarm robots, and their hybrid. The proposed framework is based on

the SeMo framework that supports distributed robot systems only. It separates the high-level mission specification and low-level robot programming. To improve the robustness, scalability, and flexibility of robot collaboration, we extend the high-level mission specification by adding new features such as team hierarchy, group service, and one-to-many communication. The mission specification is refined into extended dataflow graphs, one for each robot, with the help of a strategy description file. Two types of information sharing, global information shared and local knowledge sharing, are supported for robot collaboration in the dataflow graph. The actual robot code per robot is automatically generated from the associated task graph, which minimizes the human efforts in low-level robot programming. The viability of the proposed methodology is verified with preliminary experiments with two cooperative mission scenarios.

REFERENCES

[1] H. Hong, H. Jung, K. Park, and S. Ha, "Semo: Service-oriented and model-based software framework for cooperating robots," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2952–2963, 2018.

[2] A. M. Hoover, S. Burden, X.-Y. Fu, S. S. Sastry, and R. S. Fearing, "Bio-inspired design and dynamic maneuverability of a minimally actuated six-legged robot," in *2010 3rd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics*. IEEE, 2010, pp. 869–876.

[3] M. Kovac, M. Fuchs, A. Guignard, J.-C. Zufferey, and D. Floreano, "A miniature 7g jumping robot," in *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 373–378.

[4] R. J. Wood, "The first takeoff of a biologically inspired at-scale robotic insect," *IEEE transactions on robotics*, vol. 24, no. 2, pp. 341–347, 2008.

[5] ROBOTIS. (2020) Roboplus manager. [Online]. Available: http://emanual.robotis.com/docs/en/software/rplus1/manager/

[6] S. H. M. E. F. B. P. Jeroen Janssen, Kare Halvorsen. (2018) Capers2. [Online]. Available: https://github.com/Toglefritz/Capers_II

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[8] L. Garber, "Robot os: A new day for robot design," *Computer*, vol. 46, no. 12, pp. 16–20, 2013.

[9] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi, "Building a ros-based testbed for realistic multi-robot simulation: taking the exploration as an example," *Robotics*, vol. 6, no. 3, p. 21, 2017.

[10] R. Téllez, A. Ezquerro, and M. A. Rodriguez, "Ros navigation in 5 days: Entirely practical robot operating system training," 2017.

[11] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[12] P. Walker, S. A. Amraii, M. Lewis, N. Chakraborty, and K. Sycara, "Control of swarms with multiple leader agents," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2014, pp. 3567–3572.

[13] C. Pinciroli and G. Beltrame, "Buzz: An extensible programming language for heterogeneous swarm robotics," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3794–3800.

[14] E. Fernández Perdomo, J. Cabrera Gámez, A. C. Domínguez Brito, and D. Hernández Sosa, "Mission specification in underwater robotics," 2010.

[15] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[16] J. Fredslund and M. J. Mataric, "A general algorithm for robot formations using local sensing and minimal communication," *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 837–846, 2002.

[17] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[18] H.-w. Park, H. Jung, H. Oh, and S. Ha, "Library support in an actor-based parallel programming platform," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 340–353, 2011.

[19] J. Kim, M. Kim, M.-O. Stehr, H. Oh, and S. Ha, "A parallel and distributed meta-heuristic framework based on partially ordered knowledge sharing," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 564–578, 2012.

[20] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1321–1326.

[21] iRobot. (2019) Create 2 open interface spec. [Online]. Available: https://www.irobot.com/about-irobot/stem/create-2/projects

[22] ROBOTIS. (2019) Robotis turtlebot3 e-manual. [Online]. Available: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[23] ev3dev. (2019) Ev3 programming. [Online]. Available: https://www.ev3dev.org/

[24] ROBOTIS. (2019) Robotis opencr 1.0 e-manual. [Online]. Available: http://emanual.robotis.com/docs/en/parts/controller/opencr10/

[25] S. Park and G. Lee, "Mapping and localization of cooperative robots by ros and slam in unknown working area," in *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. IEEE, 2017, pp. 858–861.

[26] S. H. Juan and F. H. Cotarelo, "Multi-master ros systems," *Institut de Robotics and Industrial Informatics*, pp. 1–18, 2015.

[27] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh, "Programming micro-aerial vehicle swarms with karma," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2011, pp. 121–134.

[28] K. Lima, E. R. Marques, J. Pinto, and J. B. Sousa, "Dolphin: a task orchestration language for autonomous vehicle networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 603–610.

[29] J. Bachrach, J. Beal, and J. McLurkin, "Composable continuous-space programs for robotic swarms," *Neural Computing and Applications*, vol. 19, no. 6, pp. 825–847, 2010.

[30] J. Beal, K. Usbeck, and B. Krisler, "Lightweight simulation scripting with proto," *2012) colocated with AAMAS (W21)*, p. 1, 2012.

[31] D. Pianini, M. Viroli, and J. Beal, "Protelis: practical aggregate programming," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1846–1853.

[32] J. Bachrach, J. McLurkin, and A. Grue, "Protoswarm: a language for programming multi-robot systems using the amorphous medium abstraction," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1175–1178.