2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)

Towards Systematic Engineering of Collaborative Heterogeneous Robotic Systems

Simos Gerasimou, Nicholas Matragkas, Radu Calinescu
Department of Computer Science, University of York, UK
{simos.gerasimou; nicholas.matragkas; radu.calinescu}@york.ac.uk

***Abstract*—Collaborative heterogeneous robotic systems are distributed and interconnected multi-robot systems whose members can have different capabilities and can perform specialised tasks. Existing engineering processes and tools facilitate the development of various robotic aspects including kinematics, sensing and, architecture, through high-level design and low-level code generation. Although there are several frameworks and middleware, providing infrastructure for the development of single-robot and homogeneous multi-robot applications, there is lack of systematic methods and tools supporting the engineering of heterogeneous multi-robot systems and the analysis of collaborative intelligence. In this paper, we present our vision for a framework that supports the specification of collaborative heterogeneous robotic systems, generation of platform-specific code, and efficient exploration and exercise of collective intelligence algorithms.**

***Index Terms*—software engineering, robotic systems, collective intelligence, model-driven engineering, domain-specific language**

I. Introduction

Collaborative autonomous robotic systems (CARS) are distributed and interconnected multi-robot systems typically deployed to perform missions whose complexity and/or cost is too high for a single robot to accomplish on its own [1]. Examples of such missions include environmental data collection, surveillance and reconnaissance, and the discovery of natural resources. The intrinsic characteristics of these missions, i.e., distributed sensing and action, uncertain operating environment, and the need for endurance and robust behaviour, necessitates the use of CARS instead of single-robot solutions [2]. CARS can successfully undertake such missions through collective intelligence, i.e., sense, learn, share knowledge, reason and act based on a combined contributed self-awareness from team members. Thus, using CARS brings additional benefits including improved performance (missions can be performed more efficiently through parallelism if they are decomposable), mission enablement (executing missions beyond the capabilities of individuals, e.g., collective transport) and increased robustness and reliability through redundancy.

Depending on the *capabilities* exhibited by CARS members, the team can be either *homogeneous* or *heterogeneous* [2]. These capabilities correspond to properties of individual robotic team members including operating behaviour, computational capacity, size, perception abilities, or the type of terrain they can traverse [2]. Accordingly, a homogeneous robotic team comprises robots with identical capabilities that could be completely interchangeable (although, their physical structure might be different). In contrast, heterogeneous robotic teams include robots whose capabilities are different between other team members allowing them to perform specialised tasks.

The engineering of heterogeneous multi-robot systems has several benefits compared to their homogeneous counterparts. Although homogeneity improves a system's resiliency due to redundancy, heterogeneity enables to distribute the services needed to execute a given mission across team members. This enables service-based specialization within the team, thus, eliminating the need to build multiple clones of large monolithic robots. Given the technical challenges to engineer robots capable of fulfilling all mission requirements, this unique characteristic of heterogeneous multi-robot systems is equally important from an engineering perspective. For instance, incorporating all necessary mission-specific sensing, computational and actuator abilities into the closed-loop control (e.g., MAPE-K [3]) of a state-of-the-art robot could result in a large, inflexible and expensive robotic team. Heterogeneity is also crucial from a safety point of view as it reduces the risk for common hazardous behaviour, and consequently failures, due to common causes (e.g., a malfunction affecting a core controller component used in all robotic team members).

Despite the heterogeneity-induced benefits, the engineering of heterogeneous multi-robot systems remains a challenging and significantly more complex process when compared to homogeneous multi-robot systems [4]. Most of recent research and practice focuses on providing frameworks, middleware and libraries to support the engineering of single-robot and homogeneous multi-robot applications, from high-level design to low-level platform-specific code generation [1], [5]. The lack of methods and frameworks for the systematic engineering of heterogeneous multi-robot applications limits the ability to achieve the complexity required for real-world multi-robot applications [4]. This problem becomes bigger when heterogeneous robotic team members are developed using different frameworks and deployed on different robotic platforms (Figure 1). In this setting, the typical engineering process is more convoluted as it involves designing, developing and validating the robotic team from the very beginning and implementing additional software to facilitate communication between team members. This is an important limitation of existing frameworks that increases significantly the effort required for exploring the tradeoffs between adaptation strategies driven by different collective intelligence algorithms. For instance, specifying and analysing the behaviour of a CARS

978-1-7281-2249-6/19/$31.00 ©2019 IEEE
DOI 10.1109/RoSE.2019.00012

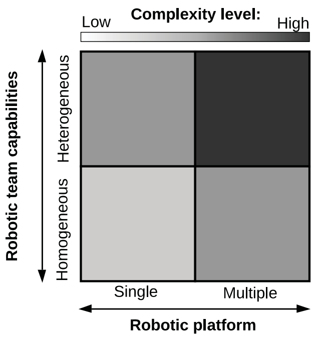

Fig. 1. Comparison between robotic platform and robot capabilities

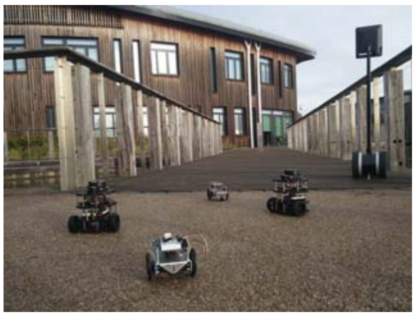

Fig. 2. A heterogeneous team of two Activity Bots, two Turtlebot 3 Burger robots and a Double 2 telepresence robot that could be used for a data collection task

that includes some robots running on ROS [6] and others on MOOS requires not only extensive knowledge of the principles and architecture of those robotic frameworks but also the development of specialised software components to enable communication and knowledge sharing between the team.

In this paper, we introduce a framework and its supporting architecture and implementation artifacts that addresses the gap in existing research and practice. We derive requirements of an effective such framework using an application scenario in Section II, and overview research on CARS in Section III. Based on those findings, we introduce in Section IV our framework that supports the specification of collaborative heterogeneous robotic systems, generation of platform-specific code, and efficient exploration and exercise of collective intelligence algorithms.

II. Motivating Application Scenario

We motivate our work using an example application scenario of a heterogeneous multi-robot team deployed to carry out an air quality data collection task in urban environments. Undoubtedly, improving air quality is an increasingly important issue, especially in densely populated areas. National and international regulatory bodies have made efforts in this direction by establishing air quality plans and low emission strategies (e.g., http://jorair.co.uk) to reduce the impact of air emissions on public health and the wider environment. To this end, the objective of the robotic team is to monitor a large area of a city and measure the air quality over a period of time. Compared to the current practice, which involves having a limited number of low-cost but imprecise nitrogen dioxide diffusion tubes mounted at fixed locations across a city, the robotic team will provide deeper insights and more accurate and timely identification of heavily polluted areas.

Figure 2 shows the robotic team which comprises a Parallax Activity Bot, an Arduino Activity Bot, two Turtlebot 3 Burger robots and a Double 2 telepresence robot. All robots are available in our lab. Accordingly, the robots differ significantly in their physical properties and computational abilities, e.g., size, travelling speed range, power resources, power consumption, and type of sensors they could be equipped with. In this scenario, the inexpensive but with considerably lower computational abilities Activity Bots could be equipped with less accurate sensors and deployed in less condensed city areas, in contrast to the more powerful and more expensive Turtlebots. The deployment of the Double 2 robot in populated areas could provide runtime notifications to citizens informing them about pollution levels in those areas. Alternatively, the Double 2 robot could use its high resolution and 150 degree wide-angle camera to detect obstacles along the path of its peers and help them to adapt their path early, thus, reducing the use of the expensive obstacle avoidance and re-planning strategies. Furthermore, enhancing each robot with abilities to undertake the data collection task requires to instantiate its closed-loop control. Since some robots use different robotic platforms, this activity involves developing the collective intelligence algorithm of the robotic team in all employed robotic platforms and additional platform-specific software components to facilitate communication and knowledge sharing/acquisition.

Considering the air quality monitoring scenario, a framework supporting the engineering of heterogeneous multi-robot applications should meet the following key requirements:

R1: **Definition of heterogeneous multi-robot applications**
Supporting the specification of robots with different capabilities enables team members to carry out specialized tasks.

R2: **Support applications for multiple robotic platforms**
This will reduce the effort for using multiple platform-specific frameworks to define the characteristic of the robotic team.

R3: **Specification of component-based robotic architectures**
Since many robotics libraries are component-based, the framework should enable specifying compositional architectures.

R4: **Definition of collective intelligence algorithms**
The systematic engineering of CARS must support tradeoff analysis between different collective intelligence algorithms.

R5: **Separation of concerns**
Low coupling between framework components improves extensibility and allows easy changes to the team's specification.

R6: **Ease of use**
The learning curve should be gentle and the framework should be easy to use both to software engineers and robotics experts.

III. Brief Overview of CARS Research

The design and implementation of software that facilitates the engineering of robotic systems has been a longstanding research strand [2], [7]–[9]. Recent research introduced frameworks, middleware, and code libraries that can support the development of various robotic application aspects including communication (e.g. ROS [6]), abstract access to sensors and actuators (e.g. Player [10]), and robot simulation (e.g. Gazebo [11]). Beyond this research, which focuses on prototyping and developing low-level robot functionalities, several model-based and domain-specific approaches have been proposed to facilitate system-level design and analysis Most of these approaches use domain-specific or general-purpose modelling languages to specify different aspects of a robot such as kinematics, dynamics, sensing, and architecture, and employ dedicated code generators to generate low-level robotic-platform specific code. For a comprehensive survey on domain-specific languages for robotics, please see [1].

Despite the existing research on robotic prototyping and tool support for low-level robot development, recent robotics surveys [1], [2], [5] emphasise that limited research exists on providing tailored tools and techniques for the systematic development and maintenance of CARS (especially heterogeneous robotic teams). An interesting research work on multi-robot systems is *NaoText* [12] which introduces a Domain-Specific Language (DSL) for specifying the collaborative behaviour of multi-robot systems. Engineers can use *NaoText* to specify the different entities of a multi-robot system, their roles, and how they ate at a high level of abstraction. Although *NaoText* permits the specification of different roles in a robotic team, it does not allow the specification of robot capabilities, thus limiting its applicability to scenarios involving homogeneous robotic teams. Following a similar direction, *FLYAQ* [13] is a family of DSLs that allows the specification of civilian missions for multi-robot systems such as scientific research and environmental protection. Although *FLYAQ* supports the specification of behaviours for robotic team members, its expressive power is limited to low-level single-robot actions (e.g. start, stop, move to a specific coordinate), and does not consider the overall coordination of a robotic team. Finally, the research in [14] introduces the *CAStlE* language for designing collective adaptive systems with a focus on robot adaptation. Although *CAStlE* supports to some extent the specification of heterogeneous multi-robot system concepts (e.g., roles), it does not provide any mechanisms for specifying different robot capabilities. Moreover, it does not provide any mechanism for evaluating alternative team compositions for specific tasks.

The lack of systematic CARS engineering methods is exacerbated by the plethora of coordination and collaboration algorithms [15]. How can engineers choose between different algorithms that can meet the same functional requirements but have different implications on non-functional system requirements? To this end, CARS engineers must choose between:

- **Static algorithms**, which provide a coordination plan before robots start a task [7], versus **dynamic algorithms**, which coordinate the robots during the task execution [8].
- Dynamic coordination can be further divided into: **explicit coordination**, in which team members use communication mechanisms for their coordination [16], and **implicit coordination**, in which robots exchange information by observing the behaviour of other robots in the environment instead of directly exchanging messages [9].
- Multiple functionally-equivalent algorithms for **task allocation** [5], **motion planning** [17], and **decision making** (centralised [18] and decentralised [19]).

Deciding on which algorithms to use for robot coordination is not only important for achieving the functional requirements of the specified tasks, but it can also affect the non-functional requirements of the robotic team, i.e., safety, reliability, energy efficiency and performance. However, it is not only the choice for the collective intelligence algorithms that can affect non-functional requirements of a CARS. Other factors including the physical design of the individual robots can affect the overall cost, ease of development, reliability, etc. Designing an optimal robot team for a given mission requires significant analysis and consideration of trade-offs. To the best of our knowledge, currently there is no systematic process and dedicated tool support for performing the required analysis and for facilitating this kind of decision-making activities.

IV. Towards A Heterogeneous CARS Framework

A framework supporting the systematic engineering of CARS should enable experimentation and tradeoff analysis from the early stages of the development process. We present our vision for such a framework and define an architecture that meets the requirements identified in Section II.

Figure 3 depicts the architecture of the proposed framework. At its core lies a family of DSLs which enables the specification of a CARS at a high-level of abstraction. The first DSL of this family facilitates the specification of a CARS mission including mission goals, tasks, and constraints as in [20].

The second language of the DSL family enables the specification of individual robots which comprise the robot team. Engineers can use this DSL to specify the robot type (e.g., UAV, UUV, USV), its architecture, its internal behaviours, and its capabilities (e.g. availability of high-resolution cameras or other bespoke sensors). Accordingly, a CARS is considered as a hierarchical composition of components which interact with the environment. Every robot of a team has a set of inputs and outputs, and can be decomposed into components such as sensors, actuators, and control units. Individual robots communicate with each other via message passing, and the team coordinates itself by using collective intelligence algorithms. Finally, engineers can use the collective intelligence DSL to specify different coordination strategies for the robots of a team. The novelty of these DSLs will lie in the set of concepts they will provide. These concepts will enable engineers to specify: functional and non-functional requirements for the team of robots; coordination strategies that the robots can use to accomplish a mission; and traceability between the requirements, the individual robots, and the team's strategy.

The proposed DSLs will be supported by a code generation engine and will target popular robotic frameworks and their simulation engines such as Gazebo/ROS [11] and

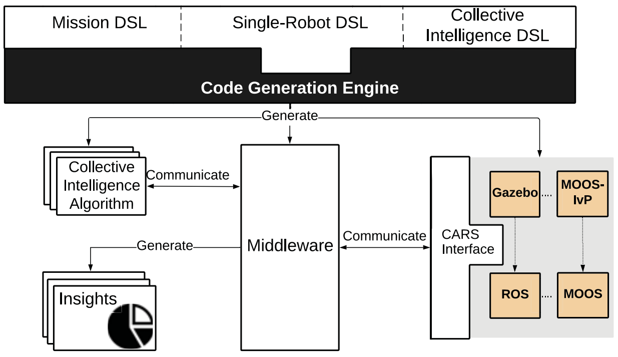

Fig. 3. Envisioned architecture of proposed framework

MOOS/MOOS-IvP [21]. At the crux of the proposed framework is a middleware which will be responsible for handling the communication between the system's components, the coordination of the individual robots, and the monitoring of a simulation. Since the exact implementation of the middleware will depend on the composition of a robotic team and the collective intelligence algorithms used, the middlewaere will also be generated from the high-level system specification. Also, a logical interface between the simulation engines and the middleware will be generated, whose role will be to interact directly with a running simulation, to exchange data with it, and to monitor the state of the simulation at any given point in time, so that it can produce detailed logs (which could be used for further analysis). Finally, concrete coordination strategies for the robotic team will be generated from the specification. These strategies will be used by the robotic team (via the middleware) to accomplish a mission. The reason that the coordination strategies are a different component of the framework, and are not embedded in the simulation code, is twofold. First, we want to achieve separation of concerns, and second we want to be able to interchange strategies easily for experimentation and analysis purposes.

The components of the proposed architecture can be directly traced back to the requirements identified in Section II. The family of DSLs will enable the specification of heterogeneous, component-based multi-robot applications (**R1**, **R3**) and the specification of collective intelligence algorithms (**R4**). Since the DSL will use high-level concepts familiar to roboticists, it will be accessible both to software engineers and robotic experts (**R6**). Using transformation technologies (e.g., model-to-text transformation for code generation) will enable quick redeployment to different robotic platforms (**R2**). The framework's component-based architecture makes it extensible (**R5**) and enables experimentation (**R4**), since components such as collective intelligence algorithms can be swapped easily.

V. Steps Forward

In this paper, first we identified the lack of software engineering tools and processes to drove the systematic development of heterogeneous multi-robot applications, and second we proposed a framework to address this gap. Currently, we are designing the family of DSLs using the process from [22]. The framework implementation will be based on EMF (https://www.eclipse.org/modeling/emf) and Epsilon [23]

References

[1] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede, "A survey on domain-specific modeling and languages in robotics," *Journal of Software Engineering in Robotics*, vol. 7, no. 1, pp. 75–99, 2016.
[2] L. E. Parker, "Multiple mobile robot systems," in *Handbook of Robotics*. Springer, 2008, pp. 921–941.
[3] R. De Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 1–32.
[4] M. Dorigo *et al.*, "A novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics & Automation Magazine*, vol. 1070, no. 9932/13, 2013.
[5] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.
[6] 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, 2009, p. 5.
[7] E. Todt, G. Rausch, and R. Suárez, "Analysis and classification of multiple robot coordination methods," in *International Conference on Robotics and Automation*, vol. 4. IEEE, 2000, pp. 3158–3163.
[8] L. Iocchi, D. Nardi, and M. Salerno, "Reactivity and deliberation: a survey on multi-robot systems," in *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, 2000, pp. 9–32.
[9] E. Pagello *et al.*, "Cooperative behaviors in multi-robot systems through implicit communication," *Robotics and Autonomous Systems*, vol. 29, no. 1, pp. 65–77, 1999.
[10] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *11th International Conference on Advanced Robotics*, vol. 1, 2003, pp. 317–323.
[11] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator." in*IROS*, vol.4, 2004, pp. 2149–2154.
[12] S. Götz *et al.*, "A role-based language for collaborative robot applications," in *Leveraging Applications of Formal Methods, Verification, and Validation*. Springer, 2012, pp. 1–15.
[13] D. Di Ruscio, I. Malavolta, and P. Pelliccione, "A family of domain-specificlanguagesfor specifying civilian missions of multi-robot systems," in *First Workshop on Model-Driven Robot Software Engineering*, 2014.
[14] A. Bucchiarone, A. Cicchetti, and M. De Sanctis, "Towards a domain specific language for engineering collective adaptive systems," in *2nd International Workshops on Foundations and Applications of Self* Systems*. IEEE, 2017, pp. 19–26.
[15] F. Rossi, S. Bandyopadhyay, M. Wolf, and M. Pavone, "Review of multi-agent algorithms for collective behavior: a structural taxonomy," *arXiv preprint arXiv:1803.05464*, 2018.
[16] B. Gerkey and M. J. Mataric, "Are (explicit) multi-robot coordination and multi-agent coordination really so different," in *AAAI Spring Symposium on Bridging the Multi-agent and Multi-robotic Research Gap*, 2004, pp. 1–3.
[17] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, p. 65, 2010.
[18] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape, and M. Yim, "Indoor automation with many mobile robots," in *International Conference on Intelligent Robots and Systems*. IEEE, 1990, pp. 67–72.
[19] R. Calinescu, S. Gerasimou, and A. Banks, "Self-adaptive software with decentralised control loops," in *FASE'15*, 2015, pp. 235–251.
[20] S. Gerasimou, R. Calinescu, S. Shevtsov, and D. Weyns, "UNDERSEA: an exemplar for engineering self-adaptive unmanned underwater vehicles," in *SEAMS'17*, 2017, pp. 83–89.
[21] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.
[22] M. Felderer and F. Jeschko, "A process for evidence-based engineering of domain-specific languages," in *22nd Intl. Conf. on Evaluation and Assessment in Software Engineering*. ACM, 2018, pp. 169–174.
[23] D. Kolovos, L. Rose, R. Paige, and A. Garcıa-Domınguez, "The epsilon book," *Structure*, vol. 178, pp. 1–10, 2010.