 doi:10.20944/preprints201911.0385.v1

Teamwork for Multi-Robot Systems in Dynamic Environments

Requirements and Solutions

Kurt Geihs
EECS Department, University of Kassel, Kassel, Germany
geihs@uni-kassel.de

Abstract: The increasing number of robots around us will soon create a demand for connecting these robots in order to achieve goal-driven teamwork in heterogeneous multi-robot systems. In this paper, we focus on robot teamwork specifically in dynamic environments. While the conceptual modeling of multi-agent teamwork has been studied extensively during the last two decades, related engineering concerns have not received the same degree of attention. Therefore, this paper makes two contributions. The analysis part discusses general design challenges that apply to robot teamwork in dynamic application domains. The constructive part presents a review of existing engineering approaches for challenges that arise with dynamically changing runtime conditions. An exhaustive survey of robot teamwork aspects would be beyond the scope of this paper. Instead, we aim at creating awareness for the manifold dimensions of the design space and highlight state-of-the-art technical solutions for dynamically adaptive teamwork, thus pointing at open research questions that need to be tackled in future work.

Keywords: Autonomous robots, Multi-robot systems, Teamwork, Coordination, Dynamic environments

1. Introduction and Motivation

Autonomous robots pervade our daily lives. Single autonomous robots for particular tasks are already accepted and used in private, business, and public environments, for example in application domains such as warehouse and transportation logistics, search and rescue, smart factories, space exploration, healthcare, smart public transportation, precision farming, and domestic services. The list of application domains is almost endless where robots play already or will soon play a major role.

It is an obvious thought that these robots around us will have to talk to each other and work collaboratively as a team – a trend that one can compare with the evolution of distributed computing by connecting stand-alone computers. Teams can be more than the sum of their parts. A multi-robot system is able to perform tasks that exceed the capabilities of a single robot, not only due to workload sharing but also in terms of functionality. Just like a team of human beings can achieve more than a single individual, the teamwork of autonomous robots provides opportunities for robots to accomplish tasks that a single robot cannot do alone. In general, teamwork in multi-robot systems exhibits potential benefits and complexities as any distributed computing system.

Conceptual teamwork models for multi-agent systems were a subject of intense research approximately 20 years ago. Among the seminal papers at this time were [35], [60], [49], to name just a few out of many. See the survey in [24] for more information on early research. While the subject since then has never vanished from the research agenda, lately one notices an increased interest in robot teamwork. This is due to three concurrent evolution threads. 1) Robot hardware including processing and communication capacity as well as sensing and actuating technology has reached a level of maturity that enables the use of autonomous robots in application domains such as Industry 4.0, autonomous driving, transport logistics, and many more. 2) Algorithmic advances in artificial intelligence, data science and machine learning facilitate data-oriented applications and thus autonomously acting devices. 3) Software engineering methodology has become available for building dynamically adaptable and evolvable software for distributed computing applications including advanced validation and verification techniques such as model checking for improving the reliability and robustness of ICT systems. Last but not least, one should not forget that the general acceptance of robotic helpers is increasing slowly but steadily in society, as demonstrated for example in the private sector by the

increasing number of partially autonomous robots for lawn mowing, vacuum cleaning, window cleaning, and even for caretaking and medical applications.

Achieving effective multi-robot teamwork raises big research challenges. Our particular focus in this paper is on the software that enables adaptive teamwork. Comprehensive support for designing the interaction, coordination and decision-making is essential to exploit the potential of multi-robot teams. For example, consider a large-scale emergency scenario where robots of different rescue organizations come together and collaborate spontaneously to search and rescue victims. Clearly, in the first place, we need communication channels between the robots – just like for the human rescue forces. Based upon the basic communication capabilities coordination of task assignment, execution, and monitoring is required. In dynamic, error-prone application environments support for adaptive task allocation and decentralized decision-making is crucial to determine who does what and when in a team of autonomous robots. Heterogeneity of robot hardware and software may further add to the complexity. When engineering a software framework for the teamwork of autonomous robots all of these challenges, and more, have to be addressed and solved.

The goal of this paper is to explore the design space for the engineering of robot teamwork and to present solution approaches for dynamic application scenarios. We highlight the spectrum of application requirements and identify challenges for the engineering of teamwork solutions in multi-robot systems. This leads us to research questions that future research needs to tackle.

The analysis part discusses general design challenges that apply to robot teamwork in dynamic application domains. The constructive part presents a survey of existing engineering approaches for challenges that arise with the dynamically changing runtime conditions.

This paper makes two main contributions: (1) Based on requirements originating from different application domains, we explore the many dimensions of the design space for teamwork engineering models, methods, and techniques, considering in particular dynamic, adverse runtime environments including noisy sensors and unreliable communication channels. (2) We review existing engineering approaches for teamwork focusing on dynamically changing runtime conditions. Thus, we elicit open research challenges that need to be solved in order to support the systematic development of effective teamwork in multi-robot systems.

Our emphasis in this paper is on software for robot teamwork. We focus neither on the variety of theoretical models for multi-agent systems nor on mechatronic and hardware design issues. Section 2 presents definitions and thus clarifies the used terminology. This is necessary because there is no general agreement on the terminology in the wider robotic community. In Section 3 we present two typical scenarios that show the breadth of requirements for robot teamwork. Section 4 discusses general design challenges, while Section 5 presents potential solutions and open research questions. Section 6 concludes the paper.

2. Definitions

Let us first briefly define the basic terminology used in this paper. This will help to elaborate on the design dimensions for robot teamwork.

A *robot* is a programmable machine capable of carrying out a complex series of actions automatically[1]. An *autonomous* robot is capable of perceiving its environment through sensors, reasoning about the gained information, making decisions accordingly, and acting upon its environment through actuators, all without human intervention. These capabilities are also commonly associated with the term *agent* whereby an agent is considered a more general term, i.e., a robot is a special kind of agent that is realized as a mechatronic construct. A robot may adopt a certain *role* based on its capabilities. It executes *tasks* that are described in a *task plan*. For example, in an autonomous driving traffic scenario, an emergency vehicle has a role that is different from regular vehicles. It has specific capabilities and rights and executes tasks such as “Switch on the siren and notify all regular vehicles at an intersection to make way.”

According to Farinelli et al. [24], a *multi-robot system* is a group of robots operating in the same environment. The authors point out that there are many different kinds of multi-robot systems. Their taxonomy

[1] The Oxford English Dictionary, Oxford University Press

is based on the two general dimensions *Coordination* and *System*. The Coordination dimension is subdivided into *Cooperation* (Do the robots cooperate to solve a problem?), *Knowledge* (How much knowledge do the robots have about each other?), *Coordination* (How much coordination is enforced?), and *Organization* (What kind of decision structure does the multi-robot system employ?). The System dimension consists of *Communication* (What kind of communication mechanisms and protocols do the robots use?), *Team Composition* (Are the robots homogeneous or heterogeneous?), *System Architecture* (Does the collective as a whole deliberately cope with an unanticipated problem or just the directly affected robots), and *Team Size* (How scalable is the system in terms of number of robots?). For a more detailed discussion, we refer the reader to the original publication [24].

Specifically, our emphasis in this paper is on multi-robot systems consisting of autonomous robots that *collaborate* in order to achieve a common *global goal*, perhaps in addition to their own local goals. We call such a collective of collaborating robots a *multi-robot team (MRT)* or multi-robot coalition. In dynamic and unpredictable environments, roles and tasks are allocated dynamically to the members of a MRT according to their capabilities and current situation [32]. This allocation has been formally modeled and analyzed as an optimization problem using various optimization techniques [44].

From a conceptual modeling perspective, a MRT is a multi-agent system that has a physical representation with specific properties determined by the mechatronic nature of the agents. A swarm of robots [11] is an extreme example of teamwork with very specific characteristics. Typically, the swarm members are rather simple devices with only basic sensing and actuation capabilities and very limited autonomy. Swarms are not considered explicitly in this paper. The scenarios described in the following section will make it clear what kind of robotic applications and teams of autonomous robots we are aiming at primarily.

The main focus of our own research is achieving adaptive goal-driven *teamwork* in a group of autonomous robots. Thus, according to the taxonomies in [24] and [21], we are concerned only with *cooperative* MRTs, consisting of robots that are aware of their teammates. How cooperation and awareness are achieved may differ.

Since teamwork is the main subject of this paper, we should briefly discuss the related terminology for characterizing the type of interaction that the robots employ to achieve teamwork. Unfortunately, we have to point out that there is neither standardization nor common agreement on the definitions of these terms, i.e. different authors use different connotations. Parker [50] differentiates between four types of interaction styles:

Collective	Entities are not aware of other entities on the team, yet they do share goals, and their actions are beneficial to their teammates.
Cooperative	Entities are aware of other entities, they share goals, and their actions are beneficial to their teammates.
Collaborative	Robots have individual goals, they are aware of their teammates, and their actions do help advance the goals of others.
Coordinative	Entities are aware of each other, but they do not share a common goal, and their actions are not helpful to other team members.

For more information and examples, we refer the reader to the original paper [50]. Here it should suffice to note that our focus in respect to goal-driven teamwork is on the two interaction styles *cooperative* and *collaborative*. We rule out the other two because they lack properties that we consider essential for teamwork in a MRT: *collective* lacks awareness for other teammates, and *coordinative* lacks a common team goal and robots do not act together as a team.

Clearly, as stated by Parker, there is no sharp boundary between the two interaction styles *cooperative* and *collaborative*. For the sake of clarity and simplicity, we view the two terms as synonyms[2] in the following and do not differentiate between the two styles, but combine and denote them as *collaborative* interaction. It is important to note here that a collaborative interaction style does not determine immediately the choice of system architecture, teamwork programming paradigm, communication protocol, decision-making and agreement protocol, etc. We will come back to this in Section 5.

[2] As a side remark: The Merriam-Webster dictionary lists both words as synonyms, see https://www.merriam-webster.com.

 doi:10.20944/preprints201911.0385.v1

We already mentioned that our viewpoint of robot teamwork is closely linked to mutual awareness in the robots, i.e. teammates have – in addition to their local knowledge – some knowledge about their colleagues in the team. This kind of awareness often, but not necessarily, implies the provision of a shared global knowledge base for the entire team. The global knowledge base, which could be implemented as a distributed replicated knowledge store, contains the contributed individual and then fused perceptions of the state of the execution environment. In many works, the individual local view of a robot is called *local world model*, while the shared team knowledge base is called a *shared world model* [46].

3. Scenarios

In this section, we present two application scenarios that are characteristic for teamwork in dynamic MRTs. The scenarios are taken from two application domains that currently are receiving increased attention in the general public. We will use them as running examples to elicit typical design requirements and to illustrate our presentation throughout the paper. Besides these two scenarios, we occasionally refer to robot soccer as an example of teamwork in a highly dynamic environment. This is based on our experiences with the RoboCup competition. We have participated successfully with a team of soccer robots in tournaments of the RoboCup Middle Size League[3]. Our team is called *Carpe Noctem Cassel*. The robot hardware and software have been designed and implemented completely by ourselves. Robot soccer is not only a lot of fun but poses challenging problems for teamwork from a research perspective.

3.1. Industry 4.0

A popular definition says[4]: “Industry 4.0 refers to the fourth industrial revolution … <where> the Internet of Things and Services is becoming an integral part of manufacturing”. This implies that not only autonomous robots and intelligent machines share information and self-organize their collaborative production work but also workpieces, components and products will make available their properties, requirements, and status to the production facilities and enterprise IT systems.

Let us consider the following scenario: An autonomous transport vehicle in an intelligent factory breaks down and its load has to be shifted automatically to one or more (autonomous) replacement vehicles that will finish the transportation job. Obviously, there is a variable number of agents involved, i.e. the broken transport vehicle, several transported items, a mobile robot with an arm to move items, and one or more replacement vehicle(s). Let us assume that all these participants have some processing and communication capabilities. The transported items know where to go and the replacement vehicles will take them to their destination. Thus, the participants in this scenario form a temporary team, also called coalition [50], whereby the size of the team, i.e. the number of participants, is unknown at design time. From the perspective of the application designer, this is called an *open team* or *open coalition*. The duration of the coalition in this specific example is the time it takes from the breakdown of the vehicle to the delivery of the items at their intended destinations.

A few more constraints are typical for Industry 4.0 scenarios. We can assume that all autonomous robots, as well as the ‘intelligent’ transport items, are homogeneous from the viewpoint of the software design, i.e. their coordination software is based on a common software framework that is the foundation for the teamwork application. Moreover, all team members are familiar with the specific execution environment and aware of the location, i.e. they have some kind of shared understanding of a map of the factory and – perhaps with the exception of the transported items – means to determine their location within this map. Also, they are familiar with the types of agents they might encounter in their environment. This might include human workers and operators. Their integration into the automated production scenarios is a research field on its own since it poses many difficult man-machine interface challenges whereby the safety of the “human in the loop” is of paramount importance.

[3] http://www.robocup.org/leagues/6

[4] https://english.bdi.eu/article/news/what-is-industry-40

3.2. Autonomous driving

An autonomous car may not be called *robot* by most people, but actually it satisfies the definition given above, i.e. it is capable of perceiving its environment through sensors, interpreting the gained information, making navigation decisions accordingly, and acting upon its environment through actuators, all without human intervention. Generally, we also assume that an autonomous vehicle is able to communicate with other equipped cars (car-to-car) and with the traffic infrastructure (car-to-infrastructure). Although the general acceptance and adoption of autonomous driving is still unclear and probably several years away, many believe that it has the potential to be a disruptive innovation with major implications for society[5].

Let us consider the following scenario: Autonomous cars coordinate their passage through a road intersection without traffic signs or traffic lights (i.e. a right-before-left intersection) while an (autonomous) ambulance car with priority approaches the intersection. As in the previous scenario, the number of participants in the coordination coalition is not known at the design time of the software. In fact, it is difficult to determine the size of the coalition because not only the cars that are at the intersection but also the cars behind them, the approaching ambulance vehicle, other unequipped cars and motorcycles, as well as bicycles and pedestrians are potential members of such an open coalition.

Likewise, the duration of the coalition is unclear and raises the question when does the membership in such a dynamic coalition begin and when does it end. Certainly, this question relates to the time that it takes to pass the intersection and leave it behind. Consequently, team membership depends on the position and driving direction of agents. Agents that have passed the intersection are out of the game.

The assumptions on the communication properties in such a dynamic environment differ from those in the previous example and raise questions such as: Car-to-car broadcast may be available but does it reach all involved members of the coalition and how reliable are the communication channels? How would we address the temporary team as a whole? How would we address individual team members whose identity is a priori unknown? Most vehicles will have a license plate whose data could perhaps be converted to some kind of unique address. And how do we – if at all – integrate into an open coalition unequipped traffic participants that do not have the same kind of communication capabilities, or more precisely, that will have only visual and acoustical communication facilities as used by drivers of unequipped cars and motorbikes, riders of bicycles, and pedestrians. Thus, the heterogeneity of the team is inherent, members may have different properties and capabilities, and they don't know each other in advance. As to the knowledge about the environment, participants can be assumed to have some understanding of the location and own position, but otherwise and contrary to the first scenario the environment may be unknown. All of these requirements need particular consideration.

A special role in this scenario plays the ambulance vehicle. Its appearance at the intersection completely changes the coordination situation and requires an immediate behavior adaptation of all other agents in the coalition. It would be very helpful if the ambulance could broadcast its route across the intersection to the other participants.

As before, the safety of all involved agents is of paramount importance. In the first place, this demands correctness and robustness of the team coordination plans and their execution. This is a tough challenge considering potentially impaired communications and unreliable, distorted sensor input. Clearly, autonomous cars will need additional emergency-avoiding mechanisms on top of and independent from their autonomous driving features, such as collision avoidance and automatic brake systems. Moreover, issues such as the privacy of driver-related data may add to the complexity of the solution.

[5] https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/self-driving-car-technology-when-will-the-robots-hit-the-road

 doi:10.20944/preprints201911.0385.v1

4. Design challenges

In this section, we discuss the spectrum of design concerns that apply to dynamic multi-robot teams in different application scenarios. The order of the following subsections is more or less arbitrary and does not imply any kind of priority.

Dynamic coalitions: In open MRTs in dynamic environments where the team members are not known a priori at design time, such as the ones described in the examples above, we need support for establishing a temporary team membership. Participants form temporary coalitions in order to solve a problem and achieve a common goal. A traffic intersection is an example of a short-lived coalition with continuous team reconfiguration, while the Industry 4.0 example above would probably imply a longer lasting coalition. Only members of a coalition should be involved in the teamwork interactions and agents outside of the coalition should not disturb it. This requires that all agents know precisely about their membership. Moreover, it may require security measures to protect the interactions of a team.

Organizational structure: The decision-making in a MRT can be organized according to three basic structural principles, i.e. centralized, hierarchical, and distributed. In a *centralized* structure, decisions are made by a central leader or controller. Clearly, such a structure suffers from the vulnerability of a central point of failure and the potential performance bottleneck. In a *hierarchical* structure, decisions are made at different levels by a hierarchy of leaders that have decision authority according to their rank. Such a structure is more robust than a centralized one because it can potentially react faster to "lower level" events and tolerate partial failures. Its drawback is the incurred high organizational overhead. In a *distributed* structure, there is a range of techniques for team decision-making. We assume that all team members autonomously perceive their own situation and the state of the surrounding execution environment. Moreover, we assume that team members are able to communicate and exchange information with their teammates. Then team decisions can be taken based on different kinds of voting schemes, auctions, games, and more. The reader is referred to [33], [13] and [1] for detailed discussions of organizational structures and decision-making in multi-agent systems.

A distributed decision structure is an obvious choice if we are concerned about the reliability of the individual robots and the reliability of the communication network. Likewise, if we deal with temporary coalitions in highly dynamic environments where swift decisions are required, such as in the autonomous driving scenario, there will be no time for the execution of a time-consuming leader election algorithm or any other costly algorithm for building an organizational structure. Thus, a distributed structure is most appropriate in this case.

Distributed decision-making: Teams need to take team decisions. For example, team members need to agree on the position of some object or on the allocation of tasks to team members. Decisions are made based on the given team plan and based on observations about the current context. Generally, decision-making happens in five steps:

1. agents collect relevant data by observing the environment and their own status;
2. agents form their own opinion based on the outcome of step 1;
3. agents propose their own opinion by replicating it to all team members;
4. the team discusses and resolves conflicting opinions;
5. the team takes a joint decision.

In terms of replication and coordination protocols, there is a choice of protocols depending on the application needs. This is a concern in particular if the communication overhead counts or communication is unreliable. The application developer will need to evaluate and judge the required level of agreement for the decision result as well as the affordable coordination overhead and time.

Both examples in Section 3 involve team decisions. Clearly, the autonomous driving scenario is more demanding because decisions need to be taken very fast in a dynamic short-lived coalition. Since safety concerns are paramount, a strict consensus is required for the team decisions. This implies the need for strict consistency in the collective perception of the locations and intentions of unequipped traffic participants.

Adaptive task allocation: Dynamic environments may not only require dynamic teambuilding but also the dynamic allocation of tasks to individual team members. A good example is robot soccer. A soccer team

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 November 2019 doi:10.20944/preprints201911.0385.v1

continuously needs to be aware of the game situation which may change instantaneously. Thus, tasks such as defending, attacking the ball, dribbling, blocking an opponent, etc. need to be assigned dynamically based on conditions such as whether the team possesses the ball, proximity of robots to the goal, position of the ball, distance to opponents, etc. Clearly, dynamic task allocation is a team decision where all team members should agree on their current duties. However, in the Industry 4.0 scenario task allocation will typically be static.

Robustness: The lack of robustness in dynamic team coordination may be due to various technical causes. Communication links may be unreliable, i.e. different communication technologies and conditions in the runtime environment may lead to message loss and network partitions such that standard network and transport layer protocols cannot provide a guaranteed error-free service. Individual robots may move out of communication range and be temporarily unreachable. This has implications for the design of the application level protocols. Centralized configurations are not appropriate in this case since a single point of failure and performance bottleneck can severely hinder the teamwork and make the whole MRT useless.

Likewise, sensors of team members may deliver different values for the same environment variable. This raises the question of what level of agreement the application requires for collective perceptions and whether the fusion of different types of sensor information can help in such a situation to improve the quality of the information in the shared world model. The amount of overhead for achieving reliable sensor information and consensus building may be prohibitive in very dynamic environments where swift decisions are more important than lengthy computation and communication activities.

Unanticipated adaptation: A key ingredient in the robust coordination of a MRT in dynamic environments is the adaptation of the agent behavior to unforeseen situations. Thus, the team as a whole should be able to evolve its team plan as well as the plans of the individual team members based on e.g. input from other agents or machine learning techniques. In general, unanticipated adaptation is a challenging problem [38], [39], and most adaptive systems assume that the adaptation state space is known completely at design time [25].

For example, if a new product is to be produced in a smart factory, new behaviors may be required for the robots involved in the production process. Therefore, it must be possible that a product, e.g. using a smart tag, will tell the driverless autonomous transport system and the manufacturing station a new behavior at runtime. The new behaviors would then be integrated into the existing behavior plans of the robots. This requires a common language with clearly defined semantics as well as some form of shared knowledge base in order to make such an unanticipated adaptation happen. Besides, processing and communication overheads of the plan evolution must be acceptable with respect to the capacities of the robots and the timing requirements of the application.

Reactivity: Timing constraints can have a substantial effect on team interactions. RoboCup soccer is a very good example. The game situation in the Middle Size League changes very quickly when robots kick or intercept the ball or when the ball bounces back from the goalpost. Therefore, there is no time for extensive computations and coordination protocols. Swift reactions within seconds are much more important for successful teamwork than precision. This has direct influence on the software and protocol design.

Heterogeneity: Heterogeneity in a MRT may refer to the hardware and software features of the individual team members. Different application domains require different robot functionalities. Thus, capabilities related to robot mobility (e.g. static, wheels, legs, aerial), sensors (e.g. optical, acoustical, temperature, air quality parameters, laser, lidar, infrared, etc.) and actuators (e.g. arm, drill, kicker, extension rails, etc.), compute power, storage capacity, operating system software, communication type and range (e.g. WiFi, Bluetooth, LoRaWAN), access to cloud computing resources, and many more will be different for different robot types. In a smart factory, the degree of heterogeneity will probably be limited and known in advance at design time, while in autonomous driving scenarios we cannot anticipate completely what kind of traffic participants appear and what their specific properties and capabilities are.

From the perspective of teamwork, robots in a team need to be capable of interacting with other robots. Not only a common communication architecture has to be in place, but also we need team-wide understood application level protocols for information exchange, coordination, and decision-making.

Programming: In order to ease the modeling and implementation of executable plans for robot activities Domain Specific Languages (DSL) have been proposed. A DSL is a *computer programming language of limited expressiveness focused on a particular domain [26]*. The "limited" in this definition should not be seen as a negative point; instead, it signals that a DSL is targeted at a specific application domain. Typically, a DSL for developing plans for robots consists of two parts, i.e. a modeling language and an associated execution engine. While there are a number of DSLs available for programming single robots (e.g. [64], [41], [62], [37], [20], [6], [22]), only a few DSLs explicitly address teamwork for multi-robot systems (e.g. [66], [59]). We claim that the complexity of teamwork in dynamic environments requires a high-level abstraction, i.e. a DSL that enables the developer to concentrate on the teamwork behavior of the distributed robot system. Whether we need different teamwork DSLs for different application domains is an open question. Ideally, a single DSL would be suitable for programming a wide spectrum of teamwork scenarios in order to enable reuse of models and development know-how.

Moreover, MRT plans specified in a DSL should lend themselves to automated verification of desired MRT properties, such as safety, fairness, freedom from deadlocks and livelocks, no starvation, etc. Preferably, the verification component should be part of an integrated development environment for teamwork plans.

Human in the loop: Even if a team of robots is able to operate autonomously and perform application tasks without human intervention, experience with self-adaptive applications has shown that the human user does not always appreciate being out of the loop [29]. Self-adaptive systems may fail to meet user expectations, and autonomous actions may be inappropriate in certain user situations. In other words, the user wants to stay in control in certain situations, or even more important, in safety critical application domains such as autonomous driving the user must be able to override automatic decisions.

This automation paradox, also called the irony of automation [29], has been known since automated control systems took over tasks that were previously carried out by human operators. Psychologists identified human contribution in automated systems not less but more important. The more advanced the automated system is the more demanding is the interaction with the human user. In case of failures or irregular conditions, humans should still have a chance to intervene. Clearly, this general insight related to automation applies also to teams of autonomous robots, especially if the MRT may self-adapt its plans to situations that the designer did not anticipate.

More socio-technical aspects: In addition to the *Human in the loop* aspect, concerns about the social embedding of a MRT application solution arise when a MRT operates in a dynamic environment where users and a MRT may interact or interfere. Most of the concerns are of a general nature for adaptive systems, such as transparency of decisions, trust in technology, fairness, privacy of context information, liability, and more. A team of autonomous robots is a collective adaptive (distributed) system which makes it inherently more difficult to find the right answers to these socio-technical design questions. While such concerns are less (or even not at all) relevant in the presented Industry 4.0 and robot soccer scenarios, they play a crucial role in the acceptance of autonomous driving technology.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 November 2019 doi:10.20944/preprints201911.0385.v1

5. Engineering Viewpoint

In the previous chapter, we discussed general requirements and design concerns for teamwork in multi-robot systems with a focus on dynamic environments. In this chapter, we concentrate on the key engineering challenges from a software developer's point of view, and we review solutions that have been presented in the literature. The reader should note that we do not intend to present a complete review of the vast spectrum of engineering challenges for multi-robot teamwork. Instead, we focus on those aspects that are specifically related to dynamic environments where team composition, task allocation, and networking conditions are all subject to continuous changes. The key question here is: What kind of mechanisms are suitable for multi-robot teams in order to cope with these changes?

5.1. Model-Driven Engineering

The complexity of teamwork in multi-robot systems in dynamic and adverse environments requires software architectures and integrated toolchains that support and ease the entire development process. Model-driven engineering (MDE) allows developers to shift their focus from implementation to modeling in the domain knowledge space. MDE is expected to promote separation of concerns, efficiency, flexibility, and evolution in application development. From a practical point of view, MDE demands a toolchain that not only automates the required model transformations, but should also include simulation as well as formal validation and verification tools. Depending on the application domain the MDE toolchain for robots could embrace available development tools and system infrastructures, such as the Robot Operating System[6] and the Gazebo simulator[7].

At the level of the platform-independent model, an MDE approach for MRT development requires an appropriate high-level modeling language. Typically, this language is a domain-specific language specifically targeted at MRT in particular application environments. Let us look at three examples, i.e. STEAM, BITE and ALICA, whereby we explain ALICA in greater detail in order to provide the reader with a more concrete understanding about the nature of such models.

Shell for TEAMwork (STEAM) [60] is a modeling approach for implementing teamwork. STEAM builds on two well-known teamwork theories, i.e. Joint Intentions Theory [41] and Shared Plans Theory [34], and tries to combine their benefits in order to achieve a coordinated behavior of the team members. In particular, STEAM assigns sub-teams of agents to a hierarchical shared plan structure. Agents need to establish a joint intention before acting together. This makes the teamwork susceptible to degraded or failed communication links.

The *Bar Ilan Teamwork Engine* (BITE) by Kaminka and Frenkel [36] divides the team modeling into three structures: Firstly, a tree-like structure similar to hierarchical task networks [61], which represents the global execution plan of the team. A second structure describes the organizational hierarchy of sub-team memberships. This results in a hierarchical task structure that provides a team-wide allocation of robots and sub-teams to behaviors. The third structure describes the social interaction behaviors, i.e., explicit communication and coordination activities between agents. A major drawback of BITE is the fact that it requires a successful negotiation before any physical action can take place. As a result, BITE is not appropriate for domains that require swift reactive behavior.

We have developed a language and execution environment called ALICA (A Language for Interactive Collaborative Agents) for designing teamwork collaboration. ALICA provides a formally defined modeling language, tool support for development, and an execution engine for highly adaptive multi-agent team behavior [58], [59]. The design of ALICA targets dynamic environments with fast changing situations, imperfect network communication, and possibly diverging sensor data from team members. ALICA was developed and used originally for robot soccer and then evolved and applied to other application domains such as collaborative exploration of unknown territories, service robotics, and autonomous driving.

[6] http://www.ros.org/
[7] http://gazebosim.org/

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 November 2019 doi:10.20944/preprints201911.0385.v1

The team behavior is specified from a global perspective in a single ALICA program which is deployed to all team members and executed without central control. ALICA uses hierarchically arranged annotated state machines to model robot tasks. Figure 1 shows an example where agents collaborate to explore territory, collect objects, and assemble some structure. Note that this plan is not complete; the figure only shows the highest specification level. A characteristic feature of ALICA is that task allocation to the individual robots is not static but adaptive to the current context and capabilities of the involved robots. State transitions depend on the situation at-hand as perceived by a robot. For further information on the syntax and semantics of ALICA, the reader is referred to [58].

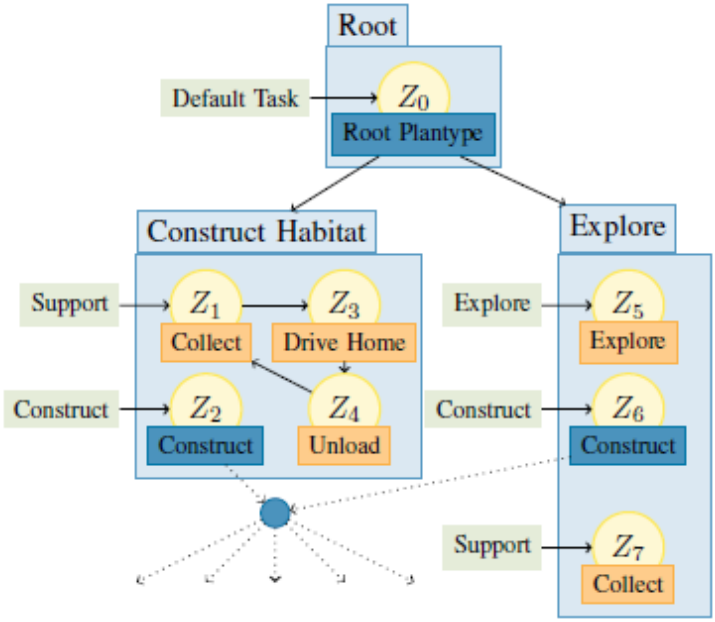

Figure 1. Example ALICA program for an exploration scenario

ALICA adopts the general principle that team decisions, e.g. about task allocation, are taken in a decentralized fashion. They result from individual decisions of the team members that follow the joint team plan, observe their environment, and exchange their views with their teammates. Nevertheless, there may be application situations where the team has to agree unanimously on the value of a certain decision variable. This might lead to decision conflicts that have to be resolved. For example, to execute a ball passing plan in robot soccer at least the pass executor and pass receiver have to agree on their own positions, the position of the ball and the opponents' positions. Thus, these positions represent joint decision variables. Note that "agree" in this context may mean different levels of agreement on a spectrum from simple broadcasts of opinions to strict consensus, as defined in the literature [40]. The developer of the respective ALICA plans must decide what kind of agreement is appropriate for an application. To facilitate this choice, we provide a specific decision-making middleware for ALICA. This will be discussed in the following subsection.

Other examples of languages suitable for the specification of MRT behavior are Buzz [53], ISPL [43] and SCEL [17]. These languages differ in many properties according to their specific application focus and design paradigms. SCEL is the only language that supports open teams using attribute-based interaction (explained in Section 5.3 below). See [18] for a detailed comparison of the three languages.

5.2. Middleware support for team decisions

A Multi-Robot System, as any other distributed system, benefits from middleware that hides the complexities of distributed computing in heterogeneous environments and thus eases the job of the developer of a distributed application. In general, middleware for robotic applications needs to satisfy the same basic requirements as any middleware in a distributed computing system, i.e. to simplify the application design by making

transparent the low-level details of the underlying hardware, software, communication, sensing, and actuating activities. Moreover, middleware facilitates the integration of new technologies, improves software maintenance and evolution, as well as software re-use across multiple development efforts, thus reducing application development costs [23]. Many middleware architectures for multi-robot systems have been proposed in the literature. We point the reader to surveys such as [18] and [45]. In this paper we present a brief summary of some well-known approaches, and we go into detail of one approach to make the presentation more concrete and informative.

There is a variety of models underlying middleware for multi-agent coordination. Frameworks such as Orocos [12], CLARAty [65], and MIRO [63] use event-based behavior coordination. The events are triggered by either communication or timer events that are mostly realized as remote procedure calls. This results in an insufficient decoupling between the initiator and receiver of an event. Orocos and MIRO are relying on the heavy-weight architectures CORBA [47] and ICE [57], respectively. In contrast, CLARAty which was developed for communication of NASA rovers, explicitly handles unreliable communication and can operate in either centralized or decentralized mode.

The most common communication concept of robot middleware is Publish-Subscribe due to its higher degree of decoupling. Examples are RoboFrame [51], Spica [7], and ROS [54]. RoboFrame and Spica have been designed explicitly for distributed computing. They are capable to deal with unreliable communication, as encountered in RoboCup events where standard WiFi communication channels often suffer from bandwidth limitations and packet losses due to interferences among the many WLANs at the competition site. While the robot software framework ROS 1 had limited support for distributed multi-robot applications, the new ROS 2 includes a middleware based on the popular Data Distribution Service (DDS) [15].

There is one specific issue in teamwork middleware that is not present in general middleware architectures, i.e. support for decentralized decision-making among autonomously acting agents. As robots are autonomous they will be capable of making their own decisions based on their own perception of the environment as well as on their local knowledge about their own status and assumptions on the team status. However, teamwork often requires agreement among team members in order to resolve conflicting value proposals. For example, vehicles need to agree on the speed and direction of a particular vehicle in an autonomous driving scenario, or soccer robots need to agree on the location of the ball on the field. Actually, agreement can mean different things in different application scenarios. Some application scenarios may require strict consensus among the involved robots, e.g. autonomous cars approaching an intersection from different directions must agree on the trajectory of an unequipped vehicle, while in robot soccer, due to the highly dynamic nature of the game, it does not pay to spend too much time on finding agreement about the position of an opponent robot because within seconds it may have moved somewhere else.

Hence, in teams of autonomous robots, we need middleware support for decision-making that is tunable to different application requirements. The core functionality of such middleware is to support the team decision-making process in respect to defined decision variables shared by all team members. In the following we discuss one concrete example for such a middleware.

We integrated the middleware PROViDE [30] into the ALICA framework to support different ways of decentralized team decision-making, as appropriate for different application scenarios. PROViDE offers a choice of replication protocols for common decision variables. Thus, all team members receive the opinions of their teammates. The level of replica consistency depends on the specific application requirements in the face of unreliable communications, temporarily disconnected robots, and diverging sensor readings by the robots. The replicated values of a decision variable can lead to a situation where a robot has received several divergent observations from its teammates in addition to its own observed value. Thus, after the replication phase, a robot needs to decide which value from the set of available opinions it will accept locally as its own value. This may lead to a situation where the individual team members have accepted different values of the decision variable as their own individual “view of the world”. Hence, we need a third coordination phase where the robots agree on a single joint value. Such a decision could be based on majority voting, priorities, time-stamps or other criteria.

 doi:10.20944/preprints201911.0385.v1

In summary, there are three distinct phases in team decision-making that resemble the typical process of decision-making in human teams (added in parenthesis):

1. Replication of individually perceived values of the decision variable to teammates. (Team members learn about diverse opinions in the team.)
2. Team members locally commit to a value. (Team members determine their own opinion.)
3. If needed, conflicting choices are resolved by a specified conflict resolution protocol. (The team consolidates diverging opinions and arrives at a joint decision.)

In summary, the PROViDE middleware offers a flexibly configurable middleware support for decision-making. The application developer can choose from a set of protocols depending on the application scenario and thus can adapt the quality and overhead of decision-making to the diverse application requirements.

5.3. Attribute-based interaction for open teams

Many communication paradigms achieve interaction among distributed components based on the identities of the components. Examples are the classical Client/Server model, the Actor model [2], or Named Channels in channel-based binary communication [55]. On the other hand, broadcast communication [52] may not require identities depending on the capabilities of the underlying communication system, but loses the ability to address a selection of individual agents. However, in open teams in dynamic environments the identity of robots may not be known at design-time, if robots may join and leave a team at run-time. Thus, the concept of identity is not easy to establish and may even be irrelevant [19]. In such environments we need different ways to determine team membership and to address team members. Note that a single central team manager that monitors and controls team membership is out of the question here because we need to avoid a single point of failure which would be contrary to our requirement that robots may be out of reach temporarily or break down completely.

Let us look at how dynamic coalitions are handled in human teams. If a specific subgroup of a crowd of people shall be addressed, group membership often is determined based on some property, such as “persons older than 65” or “owners of Diesel cars”. Similar concepts are needed for dynamic robot coalitions. In our Industry 4.0 scenario, we might ask for “components that need to be delivered within the next 15 minutes” or in autonomous driving we might want to address “vehicles that are capable of autonomous driving and are closer than 50 meters to the intersection”.

Attribute-based interaction, a variant of publish/subscribe communication, was proposed in [3] and [18] as a paradigm to address collectives of possibly anonymous agents. It appears to be a viable solution for anonymous interaction in open coalitions. In attribute-based interaction, robots of a multi-robot system explicitly expose a set of attributes that are relevant for the application at hand. Interaction between robots is based on groupcast communication whereby sending and receiving messages is determined by predicates over the specified attributes. A *Send* command expresses the intention to deliver a message to all robots satisfying the *Send Predicate*. Likewise, a *Receive* command signals willingness to receive messages from team members according to the specified *Receive Predicate*. Thus, attribute-based interaction is a special kind of publish-subscribe communication with a more fine-grained content-based selection of possible receivers and senders. Potentially, it allows a more efficient filtering of messages by the distribution infrastructure and reduces the communication overhead.

Attribute-based interaction has been integrated into the syntax of several programming languages, such as Erlang [19] and Google Go [5]. It requires a powerful distribution infrastructure, as demonstrated in these papers.

5.4. Unanticipated adaptation of team plans

Unanticipated dynamic adaptation of software systems at run-time, which was not foreseen by the developer at design time, generally is a very difficult challenge. Only a few attempts on a general solution for unanticipated on-the-fly adaptation have appeared in the literature [38], [39].

In teamwork scenarios, a dynamically changing runtime environment can lead to a similar technical challenge, i.e. on-the-fly adaptation of team plans at runtime. The arrival of new team members with new capabilities or the departure of team members with specific individual capabilities might require changes in the team plans. Likewise, the evolution of global team goals and/or individual robot goals might demand a re-planning. Note that we are not concerned about the generation of the new plans. This may be done manually by a human developer or automatically by machine learning techniques and planning algorithms. Our emphasis is on the implications of openness of teams, and thus on the capability for dynamic evolution and interchange of team plans.

A possible approach to unanticipated adaptation is based on semantic annotations of team plans using a declarative logic programming language such as Answer Set Programming (ASP) [28], [27]. ASP is a declarative non-monotonic logic programming language, adhering to a similar programming model as Prolog [14]. A number of projects have shown that ASP meets the requirements for semantic specifications in a wide range of application areas in terms of expressiveness, efficiency, dynamic extensibility, and scalability. Examples are semantic service adaptation [9], dynamic information stream configuration in crisis management scenarios [46], and service robotics [48]. Thus, by adding semantic annotations to team plans the developer lays the foundation for re-planning at runtime based on the specified properties and constraints for the robots and their relationships. The semantic compatibility of annotated team plans can be checked using established techniques for semantic matching and adaptation [56], [31]. Nevertheless, this still is mostly unchartered territory where more research and practical experience are needed on the scope and expressiveness of different paradigms for unanticipated adaptation.

6. Conclusions

The proliferation of robotics is likened often to the introduction of the Personal Computer. Many expect that - like the PC - autonomous robots, in whatever form, will become everyday assistants that will surround and support us in all kinds of application domains. Naturally, over the years the increasing number of robots will lead to “distributed robot systems” where autonomous robots form (temporary) teams and interact to achieve a common goal. Due to the manifold technical, contextual and situational dependencies, often these teams will act under dynamically changing conditions, and not all teamwork can be planned and implemented at design time. Hence, dynamic team building and adaptive team behavior will become important concerns.

In this paper, we have focused particularly on the analysis and engineering of teamwork for multi-robot systems that operate in dynamically changing environments. Thus, we raised the awareness for crucial issues for such teamwork, and we reviewed solutions for these issues. Clearly, the diversity of application requirements is huge and the design space is vast. It seems that the number of open research questions is (almost) unlimited.

Acknowledgments

Parts of this paper were written while the author was a guest scientist at IMT Lucca (Italy). Many thanks to Rocco de Nicola and the members of his group for insightful discussions and contributions. The author gratefully acknowledges the support from the Banco Santander Chairs of Excellence program and the insightful collaborations with researchers from Universidad Carlos III de Madrid (UC3M) and IMDEA Networks.

References

[1] Abbas, H. A., Shaheen, S. I., Amin, M. H. (2015). Organization of Multi-Agent Systems: An Overview, International Journal of Intelligent Information Systems. Vol. 4, No. 3, p. 46-57.

[2] Agha, G. (1986). Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge, MA, USA.

[3] Alrahman, Y. A., De Nicola, R., Loreti, M. (2016). On the Power of Attribute-Based Communication, 36th IFIP WG 6.1 Intern. Conf. FORTE 2016, Springer, LNCS 9688, p. 1-18.

[4] Alrahman, Y. A., De Nicola, R., Loreti, M. (2016). Programming of CAS systems by relying on attribute-based communication. In: Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th Int. Symp. ISoLA, Part I, pp. 539–553. Springer (2016)

[5] Alrahman, Y. A., De Nicola, R., Garbi, G. (2018). GoAt: Attribute-based Interaction in Google Go. Int. Conference ISOLA 2018, Nicosia/Cyprus.

[6] Arda, K. et al. (2013). Hierarchical Finite State Machines for Autonomous Mobile Systems. In: Control Engineering Practice 21.2, p. 184–194.

[7] Baer, P.A. (2008). Platform-Independent Development of Robot Communication Software. PhD Thesis, Computer Science. Kassel: University of Kassel. ISBN: 978-3-89958-644-2.

[8] Baraki, H., Geihs, K., Voigtmann, C., Hoffmann, A., Kniewel, R., Macek, B., Zirfas, J. (2015). Interdisciplinary Design Patterns for Socially Aware Computing, 37th International Conference on Software Engineering (ICSE), Software Engineering in Society (SEIS) track, ACM/IEEE

[9] Baraki, H. et al. (2018). SAM: A Semantic-Aware Middleware for Mobile Cloud Computing. 11th IEEE International Conference On Cloud Computing (IEEE CLOUD 2018), San Francisco.

[10] Baldoni, R., De Nicola, R., Prinetto, P. (2018). White Book: The Future of Cybersecurity in Italy: Strategic project areas, Laboratorio Nazionale di Cybersecurity, https://www.consorzio-cini.it/images/Libro-Bianco-2018-en.pdf.

[11] Bonabeau, E., Dorigo, M., Theraulaz, G. (1999). Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press.

[12] Bruyninckx, H., Soetens, P., Koninckx, B. (2003). The Real-Time Motion Control Core of the Orocos Project. In: IEEE International Conference on Robotics and Automation. 2003, pp. 2766–2771.

[13] Bulling, N. (2014). A Survey of Multi-Agent Decision-making, KI - Künstliche Intelligenz, Springer, 28-3: 147-158.

[14] Clocksin, W. F. et al. (2003). Programming in PROLOG. Springer Science & Business Media.

[15] Data Distribution Service, OMG, https://www.omg.org/spec/DDS/

[16] David, K. et al. (2014). Socio-technical Design of Ubiquitous Computing Systems. Springer.

[17] De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F. (2014). A Formal Approach to Autonomic Systems Programming. ACM Transactions on Autonomous and Adaptive Systems 9: 1–29.

[18] De Nicola, R., Duong, T., Inverso, O., Trubiani, C. (2017). AErlang: Empowering Erlang with Attribute-Based Communication, J.-M. Jacquet, M. Massink (Eds.): COORDINATION 2017, LNCS 10319, p. 21–39.

[19] De Nicola, R., Di Stefano, L., Inverso, O. (2018). Towards formal models and languages for verifiable Multi-Robot Systems, Frontiers of Computer Science, Vol. 5, Springer.

[20] Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M. (2012). Robotml, a domain-specific language to design, simulate and deploy robotic applications. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR, Springer Berlin Heidelberg, p. 149–160.

[21] Doran, J., Franklin, S., Jennings, N., Norman, T. (1997). On cooperation in multi-agent systems. The Knowledge Engineering Review 12, p. 309-314

[22] The Eclipse Foundation: Papyrus. https://www.eclipse.org/papyrus-rt (2018), accessed on 19-04-02

[23] Elkady, A., Sobh, T. (2012). Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography, Journal of Robotics, Volume 2012.

[24] Farinelli, A. , Iocchi, L., Nardi, D. (2004). Multi-Robot Systems: A classification focused on coordination; IEEETrans. on System, Man and Cybernetics, part B, p. 2015-2028.
[25] Floch J. et al. (2013). Playing MUSIC—building context-aware and self-adaptive mobile applications. In: Software: Practice and Experience 43-3: 359–388.
[26] Fowler, M. (2010). Domain-Specific Languages, Addison-Wesley.
[27] Gebser M. et al. (2012). Answer Set Solving in Practice. Vol. 6. Morgan & Claypool Publishers.
[28] Gelfond, M. et al. (2014). Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press.
[29] Geihs, K., Evers, C. (2016). User Intervention in Self-Adaptive Context-Aware Applications, 17th Australasian User Interface Conference (AUIC), Canberra/Australia.
[30] Geihs, K., Witsch, A. (2018). Decentralized decision-making in adaptive multi-robot teams, it – Information Technology, vol. 60, No. 4, p. 239-248, de Gruyter.
[31] Giunchiglia, F. et al. (2007). Semantic Matching: Algorithms and implementation. In: Journal on data semantics IX. Springer, 2007: 1–38.
[32] Gerkey, B. P., Mataric, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems, The International Journal of Robotics Research, 23-9: 939-954.
[33] Grossi D. et al. (2005). Foundations of Organizational Structures in Multiagent Systems, Proceedings of AAMAS'05, ACM, p. 690–697.
[34] Grosz B., Kraus S. (1996). Collaborative plans for complex group action. Artificial Intelligence 1996; 86(2):269–357.
[35] Jennings, N. (1993). Commitments and conventions: The foundation of coordination in multi-agent systems. The Knowledge Engineering Review 8 (1993): 223-250
[36] Kaminka, G. A., Frenkel, I. (2005). Flexible Teamwork in Behavior-Based Robots. In: AAAI. Ed. by M. M. Veloso and S. Kambhampati. AAAI Press / The MIT Press, pp. 108–113.
[37] Kammel, S. et al. (2008). Team AnnieWAY's Autonomous System for the 2007 DARPA Urban Challenge. In: Journal of Field Robotics 25-9: 615–639.
[38] Keeney, J. (2004). Completely unanticipated dynamic adaptation of software. Computer Science, Trinity College Dublin, Ph.D. Thesis.
[39] Khan, M., U. (2010). Unanticipated Dynamic Adaptation of Mobile Applications. University of Kassel, Ph.D. Thesis, Kassel University Press.
[40] Lamport, L. (1998). The Part-time Parliament. ACM Trans. Computer Systems 16-2: 133–169.
[41] Levesque Hector J, Cohen Philip R, Nunes José HT. On acting together. In: Proceedings of AAAI-90, Boston (MA); 1990. p. 94–99.
[42] Lötzsch M. et al. (2006). XABSL - A Pragmatic Approach to Behavior Engineering. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Beijing, China.
[43] Lomuscio, A., Qu, H., Raimondi, F. (2017). MCMAS: An open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer 19: 9–30.
[44] Mosteo, A. R., Montano, L. (2010). A survey of multi-robot task allocation, University of Zaragoza, Technical Report AMI-009-10-TEC.
[45] Mohamed, N., Al-Jaroodi, J., Jawhar, I. (2008). Middleware for robotics: A survey, IEEE Conference on Robotics, Automation and Mechatronics, 2008, p. 736-742.
[46] Niemczyk, S. et al. (2017). ICE: Self-Configuration of Information Processing in Heterogeneous Agent Teams. In: Proceedings of the Symposium on Applied Computing 2017, ACM. p. 417–423.
[47] Object Management Group (OMG). (2012) The Common Object Request Broker: Architecture and Specification (CORBA 3.3). Object Management Group.
[48] Opfer S. et al. (2017). Reasoning for Autonomous Agents in Dynamic Domains. In: ICAART (2) 2017, p. 340–351.
[49] Parker, L. E. (2000). Current state of the art in multi-robot teams, Distributed Autonomous Robotic Systems, Springer, No. 4, p. 3-12.

[50] Parker, L. E. (2008). Distributed intelligence: Overview of the field and its application in multi-robot systems, J. of Physical Agents 2 (2008), 5-14.
[51] Petters, S., Thomas, D., von Stryk, O. (2007). „RoboFrame - A Modular Software Framework for Lightweight Autonomous Robots“. In: Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware, IEEE/RSJ IROS. San Diego, CA, USA.
[52] Prasad, K. (1991). A calculus of broadcasting systems. In: TAPSOFT'91. pp. 338–358. Springer.
[53] Pinciroli, C. Beltrame, G. (2016). Buzz: An extensible programming language for heterogeneous swarm robotics. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, p. 3794–3800.
[54] Robot Operating System, https://index.ros.org/
[55] Sangiorgi, D., Walker, D. (2003). The pi-calculus: a Theory of Mobile Processes. Cambridge University Press.
[56] Scioni, E. (2016). Online Coordination and Composition of Robotic Skills: Formal Models for Context-aware Task Scheduling. KU Leuven, Ph.D. Thesis.
[57] Shumko, S. (2009). Ice Middleware in the New Solar Telescope's Telescope Control System. In: Astronomical Data Analysis Software and Systems XVIII. Vol. 411.
[58] Skubch, H. (2012). Modelling and Controlling Behaviour of Cooperative Autonomous Mobile Robots. Ph.D. Thesis, Universität Kassel, Springer Vieweg.
[59] Skubch, H., Wagner, M., Reichle, R., Geihs, K. (2011). A Modelling Language for Cooperative Plans in Highly Dynamic Domains. Mechatronics 21-2: 423–433.
[60] Tambe, M. (1997). Towards flexible teamwork, Journal of Artificial Intelligence Research (JAIR), vol. 7: 83-124.
[61] Tate, A. (1977). Generating Project Networks. In: Proc. of the 5th Intern. Joint Conf. on Artificial Intelligence – Vol.2, IJCAI'77. Cambridge, USA: Morgan Kaufmann Publishers Inc., pp. 888–893.
[62] Urmson C. et al. (2007). Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge. CMU TR Urmson-2007-9708.
[63] Utz, H., Sablatnog, S., Enderle, S., Kraetzschmar, G. (2002). Miro - middleware for mobile robot applications. IEEE Transactions on Robotics and Automation, 18.4, pp. 493–497.
[64] Verma V. et al. (2005). Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences. In: Internat. Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS).
[65] Volpe, R., Nesnas, I. A. D., Estlin, T., Mutz, D., Petras, R., Das, H. (2000). CLARAty: Coupled Layer Architecture for Robotic Autonomy. Tech. rep. NASA Jet Propulsion Laboratory.
[66] Zweigle O. et al. (2006). Cooperative Agent Behavior Based on Special Interaction Nets. In: Proceedings of the 9th International Conference on Intelligent Autonomous Systems - IAS, Tokyo, Japan, IOS Press, p. 651–659.