applied sciences

MDPI

Review

Engineering Challenges Ahead for Robot Teamwork in Dynamic Environments

Kurt Geihs [1,2]

[1] University of Kassel, Electrical Engineering & Computer Science, 34121 Kassel, Germany; geihs@uni-kassel.de

[2] Universidad Carlos III de Madrid, Ingenieria Telematica, 28911 Leganes (Madrid), Spain

Received: 29 November 2019; Accepted: 12 February 2020; Published: 18 February 2020

Abstract: The increasing number of robots around us creates 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 was studied extensively during the last two decades and commercial multi-agent applications were built based on the theoretical foundations, the steadily increasing use of autonomous robots in many application domains gave the topic new significance and shifted the focus more toward engineering concerns for multi-robot systems. From a distributed systems perspective, we discuss general engineering challenges that apply to robot teamwork in dynamic application domains and review state-of-the-art solution approaches for these challenges. This leads us to 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. Clearly, autonomous robots will play a crucial role in more and more application domains. 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.

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 [1–5], to name just a few out of many (see the survey in [6] for more information on early research). However, the commercial adoption of agent-based solutions was low during the subsequent decade. According to [7], among the main hindrances for agent–based applications were limited awareness about the potentials, limited publicity of successful industrial projects, misunderstandings and over-expectations, and lack of mature enough design and development tools. While the subject since then never vanished from the research agenda, one may notice a recent increased interest in robot teamwork. This is due to substantial progress in robotic hardware and software. In the realm of software, which is our focus in this paper, advances in artificial intelligence techniques make autonomous robots fit for applications in

private, industrial, and public environments. Thus, we are observing a shift of research focus from theoretical multi-agent models to practical software and hardware engineering issues. Moreover, the general acceptance of robotic helpers is increasing steadily in society, as demonstrated, for example, in the private sector by the increasing number of autonomous robots for lawn mowing, vacuum cleaning, window cleaning, and even for caretaking and medical applications.

Our emphasis in this paper is on the engineering challenges that arise when autonomous robots collaborate as a team to solve a joint task. In particular, we view these systems from a distributed systems perspective. In general, teamwork in multi-robot systems exhibits potential benefits and complexities as any distributed computing system. However, robot teamwork introduces a number of additional new challenges that we discuss in this paper. For each challenge, we present state-of-the-art solution approaches. This leads us to research questions that future research needs to tackle.

Our own experience with multi-robot systems originated from our participation in the Middle-Size League of international RoboCup tournaments where teams of custom-build soccer robots compete against each other. Our research focus was mainly on a software framework for the development of teamwork applications in adverse and dynamic environments, as they prevail in Robocup tournaments. Later, we evolved the framework and showed successfully that it was also fit for other robotic application domains, such as collaborative exploration, autonomous driving, and service robotics.

In particular, in this paper we are interested in robotic teamwork in dynamic and adverse application environments where adaptation and reconfiguration may be necessary due to a continuously changing runtime context. This is a far-reaching assumption that includes fewer demanding requirements. Our focus is on the software for robot teamwork. Neither the variety of theoretical models for multi-agent systems nor the mechatronic and hardware design issues are subjects of this paper. For these issues, we refer the reader to surveys such as [8,9].

In Section 2, we start with a brief clarification of terminology for collaborative robots. This is necessary because there is no general agreement on the terminology in the wider robotic community. In Section 3, we discuss engineering challenges for robot teamwork, and we point to existing solution approaches in order to explicate dimensions of the design space. Section 4 presents a summary of open research questions. Section 5 concludes the paper.

2. Terminology

Let us first briefly define the basic terminology used in this paper. A *robot* is a programmable machine capable of carrying out a complex series of actions automatically (The Oxford English Dictionary, Oxford University Press). 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 agent is considered a more general term, i.e., a robot is a special kind of agent that (mostly) 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 different tasks than regular traffic participants.

According to Farinelli et al. [6], 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 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 [6].

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 an MRT according to their capabilities and current situation [10]. This allocation was formally modeled and analyzed as an optimization problem using various optimization techniques [11].

From a conceptual modeling perspective, an MRT is a multi-agent system that has a physical representation with specific properties determined by the mechatronic nature of the agents. 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 [6,12], 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. Here, we have to point out that, even with existing standards such as the Foundation for Intelligent Physical Agents (FIPA, http://www.fipa.org/), there is no common agreement on the definitions of these terms, i.e., different authors use different connotations. We refer to Parker [13] who differentiates between four types of interaction styles as follows:

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.

We refer the reader to [13] for more information and examples. Here, it should suffice to note that our focus with 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 an 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 hereafter view the two terms as synonyms (as a side remark: The Merriam-Webster dictionary lists both words as synonyms; see https://www.merriam-webster.com) and do not differentiate between the two styles, but combine and denote them as *collaborative* interaction. It is worthwhile to note here that a collaborative interaction style does not imply a particular choice of system architecture, teamwork programming paradigm, communication protocol, decision-making technique, agreement protocol, etc.

3. Teamwork Challenges

In this section, we discuss key engineering challenges that apply to multi-robot teams in dynamic application scenarios from a software developer's point of view. It is not our intention to present a complete review of the broad spectrum of design aspects for multi-robot teamwork. Instead, we focus on those engineering challenges that are related specifically to dynamic environments. For each challenge, we present a brief look at initial approaches as examples for possible solutions. The order of the sections below does not imply any kind of priority.

3.1. Dynamic Coalitions

In open robot teams where the team members are not known a priori at design time, 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 in autonomous driving scenarios is an example of a short-lived coalition with continuous team reconfiguration, while Industry 4.0 scenarios would likely 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 about their membership. Moreover, it may require security measures to protect the interactions of a team. Key challenges are as follows:

- How are team members discovered and identified?
- Who manages team membership?
- How do team members learn about the team composition?
- How does the team protect itself against malicious intruders?

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 [14], or named channels in channel-based binary communication [15]. On the other hand, broadcast communication [16] 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 [17].

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 in environments where robots may move out of reach temporarily or break down completely.

One solution is based on attribute-based interaction. It is a variant of publish/subscribe communication, and it was proposed in [17,18] as a paradigm to address collectives of possibly anonymous agents. 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*.

For example, in an Industry 4.0 scenario, one might ask for "components that need to be delivered within the next 15 min" or, in autonomous driving, one might want to address "vehicles that are capable of autonomous driving and are closer than 50 m to the intersection". Thus, attribute-based interaction is 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. The drawback is the need for a powerful, rather heavyweight distribution infrastructure. Attribute-based interaction is integrated into the syntax of several programming languages, such as Erlang [19] and Google Go [20].

Other well-known protocols for open coalitions include the JXTA peer-to-peer protocols that target overlay-based communication of peers across public networks, where security issues such as firewall traversal are of utmost importance [21]. An application of JXTA for the control of robots was reported in [22]. However, the JXTA project is no longer officially supported. The FIPA recommendations (http://www.fipa.org/) offer a conceptual framework for communication and management in multi-agent teams. Their focus is on intelligent (software) agents, not on mechatronic robots. The FIPA standardization is currently not active.

3.2. Heterogeneity

Heterogeneity in an 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 each other. Thus, not only must a common communication architecture be in place, but team-wide understood application level protocols for information exchange, coordination, and decision-making are also needed. On the one hand, the heterogeneity of robots certainly contributes to the complexity of the teamwork application design, particularly since, so far, there are no dominating standards for masking the heterogeneity through transparency solutions. On the other hand, from an application perspective, heterogeneity may be beneficial if a team is able to make use of specific capabilities of team members. Then, the whole can truly be more than its parts.

The key challenge is as follows:

- Will it be possible to agree on a small set of standardized hardware and software interfaces for the wide range of diverse robot hardware components and software services?

The situation is similar to the evolution of computing hardware and software where commonly agreed interface standards—de facto or official ones—made it possible to mix and match components from different manufacturers to prevent vendor lock-in and to respond to the variety of user requirements. If the emergence of multi-robot applications continues at the speed that we are witnessing today, more standardization is needed in order to enable a flexible combination of heterogeneous robots for application specific robot teamwork. Consequently, more practical experience and applied research are needed to fuel such a standardization.

3.3. Middleware Support

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, and it improves software maintenance and evolution, as well as software re-use across multiple development efforts, thus reducing application development costs.

Taking up experiences with agent reference models (e.g., FIPA and its offspring NAWG (FIPA P2P Nomadic Agents WG (P2PNA WG6), http://www.fipa.org/subgroups/P2PNA-WG.html)) and agent platform implementations (see the FIPA web page at http://www.fipa.org/resources/livesystems.html for an (outdated) list of publicly available FIPA compliant implementations of agent platforms), as well as with popular general-purpose middleware systems, many middleware architectures specifically for multi-robot systems were proposed in the literature. We point the reader to surveys such as [23,24].

Key challenges are as follows:

- Do we need specialized middleware for robot teamwork, or can we build on existing standards for general middleware architectures?
- Are the robot devices capable of running a heavyweight middleware in terms of processing and storage capacity?

- How important are quality of service guarantees for the application at hand?

There are a variety of models underlying middleware for multi-agent coordination. Middleware frameworks such as Orocos [25], CLARAty [26], and MIRO [27] 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 rely on heavyweight architectures, i.e., CORBA [28] and Ice [29], 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 [30], Spica [31], and ROS [32]. RoboFrame and Spica were designed explicitly for distributed computing in multi-robot systems. They are capable of dealing with unreliable communication, as for example encountered in RoboCup soccer tournaments 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) (Data Distribution Service, OMG, https://www.omg.org/spec/DDS/).

Nevertheless, there are several open issues related to middleware support for robot teamwork that need more research. A truly flexible middleware toolbox would be needed that enables the designer of a teamwork application to configure the middleware by selecting the specific set of components that matches best the application requirements. Operating systems research faced basically the same problem several decades ago when more and more electronic devices became available with very different hardware architectures and application requirements.

Another open issue is the concern for security in distributed collaboration scenarios; most existing middleware solutions for multi-robot systems assume that applications, if needed, can make use of transport-level security mechanisms. Not all applications would need such security. For example, in RoboCup tournaments, there is no real need to secure the communications between the robots (and no time to do so anyway). Likewise, in the collaborative exploration of unknown territories, e.g., on another planet as part of a space mission, secure communication between the explorer robots is not required. However, when it comes to collaborative autonomous vehicles on public roads, potential vulnerabilities are a crucial concern. Moreover, in Section 3.1, we already mentioned the need for security in managing dynamic team membership. More research specifically on adaptive security for teams of autonomous robots is needed. This seems to be a research area on its own.

3.4. Organizational Structure and Decision-Making

Teams need to take team decisions. For example, vehicles need to agree on the speed and direction of a particular vehicle or soccer robots on the location of the ball on the field. Obviously, application requirements are very different. In an autonomous driving scenario, decisions on the locations and intentions of unequipped traffic participants need to be taken very fast in very dynamic short-lived coalitions. Since safety concerns are paramount, consensus is required for most decisions. On the other hand, in robot soccer, for most decisions, we tolerate a relaxed consensus level but demand swift reactions due to the high dynamics of the game situation.

Decision-making (note that our notion of decision-making is different from Reference [9] where the term is viewed as a synonym for planning and control of a multi-agent system) in a team can be organized according to three basic structural principles, i.e., centralized, hierarchical, and distributed [33]. In a *centralized* structure, decisions are made by a central leader or controller. This 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 *decentralized* structure, all team members autonomously perceive their own situation and the state of the surrounding execution environment. Team members decide about their actions by themselves according to a given team plan. Team decisions that tolerate a relaxed consensus level can be taken using different kinds of voting schemes, auctions, games, and more.

Decisions are made based on the given team plan and observations about the current context. The application developer will need to evaluate and judge the required level of agreement for team actions, as well as the affordable coordination overhead. A decentralized decision structure is an obvious choice if we are concerned about the reliability of the individual robots and the communication network. Likewise, if we deal with temporary coalitions in highly dynamic environments where swift decisions are required, such as in robot soccer, there is no time for the execution of a time-consuming leader election algorithm or any other costly algorithm for establishing an organizational structure. The reader is referred to [33–35] for detailed discussions of organizational structures and decision-making in multi-agent systems.

The key questions are as follows:

- What kind of organizational structure follows from the application requirements?
- What kind of consensus level is required for team-wide decisions?
- Which decision-making protocols are appropriate considering the trade-off between consensus level and protocol overhead?

Let us look in more detail at a decentralized team organization. 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.

For replication and conflict resolution, there is a choice of well-known protocols depending on the application needs. Hence, a teamwork middleware should offer flexible support for decision-making that is tunable to different application requirements. The core functionality of such a middleware function is to support the team decision-making process with respect to the current values of specified decision variables. Below, we present one concrete example for such a middleware.

The middleware PROViDE [36] is part of a multi-agent framework called ALICA [37]; ALICA teams have a decentralized team structure. PROViDE offers a choice of replication and agreement protocols for common decision variables. If a team decision about the value of an environment variable needs to be made, all team members broadcast their opinion to their teammates. The developer may choose the level of replica consistency depending 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 receives 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 accept different values of the decision variable as their own individual "view of the world". Now, 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, timestamps, or other criteria.

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

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, based on the PROViDE middleware, the application developer can tune the middleware by choosing from a set of provided protocols and, thus, can adapt the quality and overhead of decision-making to diverse application requirements. However, this raises another question. One might argue that such an abundant choice of strategies shifts the complexity onto choosing the right combination of strategies. This argument cannot be ignored. One solution might be to identify reusable typical patterns of strategy combinations for specific application scenarios. This can be addressed in future work.

3.5. Programming

The complexity of teamwork in multi-robot systems in dynamic and adverse environments requires software architectures and integrated toolchains that ease the 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 engineering point of view, MDE demands a toolchain that not only automates the required model transformations, but also includes tools for examining the models through simulation (e.g., using Gazebo (http://gazebosim.org/)) or model checking (e.g., using UPPAAL [38]).

In order to ease the modeling and implementation of executable plans for robot applications, domain-specific languages (DSL) were proposed. A DSL is a *computer programming language of limited expressiveness focused on a particular domain* [39]. 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., [3,40–45]), only a few DSLs explicitly address teamwork for multi-robot systems (e.g., [46,47]) (see [48] for a detailed review of robot DSLs). We claim that the complexity of teamwork in dynamic environments makes such a high-level abstraction a necessity, i.e., a DSL that enables the developer to concentrate on the teamwork behavior of the distributed robot system.

Dynamic environments typically imply a dynamic allocation of tasks to individual team members. A good example is robot soccer. A soccer team 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 in a decentralized formation is a team decision where all team members should agree on their current duties. In contrast, in an Industry 4.0 scenario, allocation of tasks to robots will typically be static.

General research questions are as follows:

- Do we need different teamwork DSLs for different application domains? 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.
- Does the modeling and execution environment support a dynamic task allocation to team members instead of fixed allocations?
- How can we efficiently integrate simulation and automated verification into the application development environment in order to examine the models for desired MRT properties, such as safety, fairness, freedom from deadlocks and livelocks, no starvation, etc.?

Let us look at three examples for high-level modeling languages for robot teamwork, i.e., STEAM, BITE, and ALICA.

Shell for Teamwork (STEAM) [5] is a modeling approach for teamwork. STEAM builds on two well-known teamwork theories, i.e., joint intentions theory [3] and shared plans theory [1]. It 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 [49] divides the team modeling into three structures. A tree-like structure, similar to hierarchical task networks [50], represents the global execution plan of the team. Another 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.

Let us look at one framework in more detail to make the descriptions more concrete. ALICA (A Language for Interactive Collaborative Agents) is a language and execution environment for developing teamwork applications. ALICA provides a formally defined modeling language, tool support for development, and an execution engine for highly adaptive multi-agent team behavior [37,46]. The design of ALICA targets dynamic environments with fast changing situations, imperfect network communication, and possibly diverging sensor data from team members. It supports the known design blocks of multi-robot systems [51], i.e., task decomposition, team formation, and task allocation, as well as task execution and control. 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, autonomous driving, and service robotics [52].

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 a 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 [37].

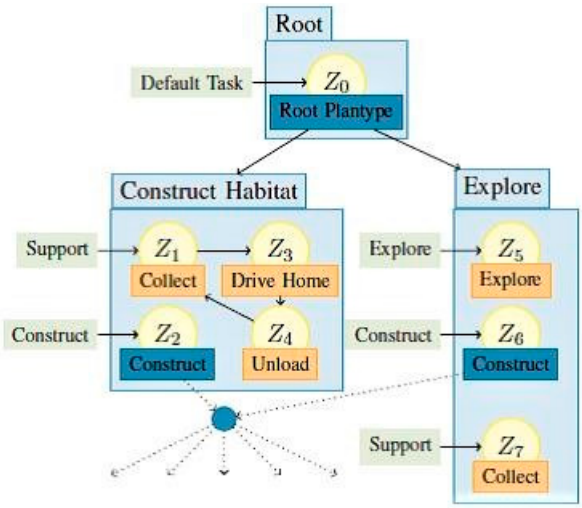

Figure 1. Example of ALICA (A Language for Interactive Collaborative Agents) program for an exploration.

ALICA features a strictly decentralized team organization. Team decisions, e.g., about task allocation, result from individual decisions of the team members according to the given team plan, their observations of the environment, and the information exchange 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 [53]. The developer of the respective ALICA plans must decide what kind of agreement is appropriate for an application in a certain situation. To facilitate this choice, ALICA contains a specific decision-support middleware (see Section 3.4 above).

Other examples of languages suitable for the specification of MRT behavior are Buzz [54], ISPL [55], and SCEL [56]. 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 (see [19] for a detailed comparison of the three languages).

3.6. Shared Knowledge

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 awareness often, but not necessarily, is based on the provision of a shared global knowledge base for the entire team. The knowledge base, which typically would be implemented as a distributed replicated knowledge store, contains the concepts, objects, and relations known to the robots, as well as the fused perceptions of the state of the execution environment. In many works, the individual local view of a robot is called the *local world model*, while the shared team knowledge base is called the *shared world model* [57].

Potentially, such a shared knowledge base with frequently consulted thousands of objects and relations is a very resource-consuming and a performance-critical element of the teamwork. This creates several challenges, as listed below.

- How do we formally define the shared knowledge such that reasoning at run time satisfies performance requirements?
- How do we equip robots with commonsense knowledge that human beings would have implicitly about the environment?
- How can we integrate individual, heterogeneous knowledge representations of heterogeneous robots into a shared world model?
- What kind of consistency guarantees are necessary and feasible for the distributed, replicated storage of the (dynamic) contents of the knowledge base?
- How robust and scalable is the common knowledge base in view of, e.g., unreliable communication connections, imprecise sensor data, and predefined time barriers?
- Can the knowledge base structure be adapted and extended at run time?

While there is a large set of publications focusing on knowledge representation techniques for robotic applications (e.g., [58–60]), little was published specifically on distributed knowledge bases for multi-robot systems. One thread of research—in particular for service robots—looked at offloading the knowledge base to the cloud [61]. The viability of such an approach clearly depends on the application requirements in terms of access performance and availability. Other approaches exploit a decentralized storage of knowledge [57] leading to well-known questions of consistency in data replication.

The spectrum of diverse requirements in multi-robot applications seems to be so large, that a harmonization of techniques for knowledge representation and storage is out of reach. On the other hand, a vast body of know-how is available for knowledge representation, reasoning, distributed data

storage systems, and replication and consensus protocols. In this situation, it is worth considering the question of whether the variety of existing solutions could be narrowed down to a few solution patterns, which would satisfy the majority of application scenarios.

3.7. Robustness and Dynamic Adaptation

Robustness is the ability of a system to cope with errors during execution and with erroneous input (https://en.wikipedia.org/wiki/Robustness_(computer_science)). A 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 communication protocols cannot provide a guaranteed error-free service. Individual robots may move out of communication range and be temporarily disconnected from the team. 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. 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. Moreover, run time execution errors may be caused by situations that were not foreseen at design time. Robustness in this case would mean that the system is able to perform an unanticipated adaptation. 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 software adaptations, which were not planned by the developer at design time, are a challenging problem. Only a few attempts on a general solution for unanticipated on-the-fly adaptation appeared in the literature [62,63]. Most adaptive systems assume that the adaptation state space is known completely at design time [64].

Related research questions are as follows:

- How does the MRT cope with diverging sensor readings?
- Can the MRT tolerate temporarily impaired communications?
- Is the MRT capable of evolving its plans on-the-fly in order to integrate a learned or otherwise derived behavior?
- What are appropriate strategies for unanticipated adaptation?

In teamwork scenarios, the arrival of new team members with new capabilities or the departure of team members with 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 in an MRT is based on semantic annotations of team plans using a declarative logic programming language such as answer set programming (ASP) [65,66]. ASP adheres to a similar programming model as Prolog [67]. A number of projects showed 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 [68], dynamic information stream configuration in crisis management scenarios [57], and service robotics [59]. 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 [10,69]. Nevertheless, this is still largely unchartered territory in respect to the applicability to different robot scenarios. More research and practical experience are needed on the scope, expressiveness, and performance implications of different paradigms for unanticipated adaptation.

3.8. Socio-Technical Concerns

Even if a team of robots is able to operate autonomously and perform application tasks without human intervention, experience with self-adaptive applications shows that the human user does not always appreciate being out of the loop [70]. 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 importantly, 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 [70], is known since automated control systems took over tasks that were previously carried out by human operators. Psychologists identified human contribution in automated systems as not less but more important. A more advanced automated system denotes a more demanding interaction with the human user. In cases of failures or irregular conditions, humans should still have a chance to intervene. At all times, humans need to be protected against harm caused by the robot behavior. Clearly, this general insight related to automation applies also to the engineering of an MRT, especially if the MRT may self-adapt its plans to situations that the designer did not anticipate.

In addition to the *human in the loop* aspect, concerns about the social embedding of a robotic application solution arise when a team of robots operates in a dynamic environment where users and robots interact. 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. Surely, these concerns play a crucial role for the user acceptance of any technical system and particularly in safety-critical applications. They apply to single robots, as well as multi-robot systems. However, one question remains unanswered so far in the literature:

- Will the envisaged teamwork of robots, in comparison to a single robot application, create more complex or even additional challenges in respect to socio-technical design concerns?

4. Summary

The wide spectrum of applications that require teamwork of robots poses the following question: can we discuss engineering concerns at all from a general, all-encompassing point of view? Application domains such as autonomous driving, Industry 4.0, and search and rescue clearly have very different requirements. Nevertheless, our answer is positive, looking at a comparison of robot teamwork with the evolution of distributed systems technologies where models, architectures, and techniques emerged that provide a strong foundation for practical implementations.

In contrast to classical distributed systems technologies, we assume that robot teamwork happens in dynamic environments; robots are mobile, robots use unreliable wireless communications, robots move out of communication range, new team members appear, robots sense the state of the runtime environment and reason about appropriate reactions, specific components of robots fail without rendering the whole robot useless, the team encounters unforeseen situations, and more. Below, we summarize from a general, systems-oriented perspective the discussions in the previous chapter about engineering challenges for robot teamwork in dynamic environments. Thus, we point to research areas that need to be tackled in future work.

4.1. Dynamic Coalitions

The dynamic environment, as described above, implies a need for a highly flexible team organization and collaboration infrastructure. Team membership must be managed to cope with varying team membership. The capabilities of the team as a whole may change if robots leave, join, or experience a breakdown of components or the whole robot. We need a semantic description of the capabilities that are currently available in the team; consequently, we need reasoning about the appropriate dynamic allocation of tasks to team members and the modification of execution plans. All of this should be supported by the teamwork collaboration infrastructure such that the developer is relieved as much as possible from the nitty-gritty details. The individual building blocks for such an infrastructure are known. However, one difficult engineering question remains: how can the different independently developed pieces be put together?

4.2. Platform Harmonization and Configurability

In order to facilitate the reuse of software components, portability of applications, and exchange of know-how, a harmonization—if not standardization—of robot platforms and their application programming interfaces is desirable. The diversity of robot application domains creates a need for a flexibly configurable and customizable collaboration platform architecture that reflects the different computational capacities of robots. Thus, in analogy to operating systems for embedded systems, we need highly configurable, component-based teamwork collaboration platforms (i.e., middleware) that can be tailored to specific application needs and properties of the involved robots. Such a toolbox is missing.

4.3. Knowledge Base

The ability to share knowledge is a crucial prerequisite for teamwork. As discussed in Section 3.6, there are various approaches for building a shared knowledge base in multi-robot systems. Some standardization would also be helpful here. Important research questions to ask in this realm concern the integration of heterogeneous knowledge representations, the implementation of the knowledge base in a distributed system with largely diverging agent capabilities, the satisfaction of stringent application performance requirements, the extensibility of the knowledge base structure at run-time, i.e., adding new concepts and relations, as well as removing invalid facts, and the inclusion of "common-sense knowledge" that humans would have implicitly, but which needs to be provided explicitly to a robot team.

For all of these aspects, the state of the art provides individual solutions. However, how to forge these solutions into a shared knowledge base that satisfies the specific requirements of an application domain is an open problem. Moreover, we need to explore whether it is feasible to reduce the large number of approaches to a few consolidated ones.

4.4. Methodology and Tools

Like other software, the development of robot teamwork applications should be supported by an effective development methodology as well as corresponding development tools. Many proposals for domain-specific languages for robotic applications exist, as mentioned above in Section 3.5. How to filter out a few approaches that would serve a larger number of application domains is an open question. The formal verification of teamwork plans for dynamic environments with respect to correctness and properties such as liveness and freedom from deadlocks is not well developed so far and requires more research. Moreover, facing the large variety of protocols for agreement, data consistency, synchronization, etc., the identification of agreed-upon reusable best-practice design patterns for teamwork applications would greatly facilitate the software development process. Ideally, all of this should be integrated into a robot teamwork development environment built around a powerful DSL.

4.5. Edge and Cloud Integration

Offloading computation-intensive tasks or large data quantities from robots to edge or cloud computing resources is an attractive option for resource-scarce robots. However, the challenges and open questions that arise in such a scenario are manifold. Which part of an application can/should be offloaded to improve the performance or to save battery capacity taking into account the communication overhead and latencies? When and under what conditions should it be done? How can the state of the robot's execution context be provided to the offloaded computation? These questions were solved already for mobile cloud computing scenarios on mobile devices, primarily looking at offloading computation and data from a single device [68]. For teamwork scenarios in multi-robot systems with a high degree of agent cooperation and coupling, the viability and effectiveness of these solutions have to be re-examined.

4.6. Human in the Loop and Other Sociotechnical Concerns

In many application environments, robots interact with humans. For example, humans may use the services of a robot team. Alternatively, humans may augment the capabilities of the team, effectively making them a member of the team, or humans may give instructions to a robot team to control the execution. Research in social robotics delivered various means of interacting with robots, e.g., based on voice or gestures. However, interaction with and control of a whole team of robots received little attention so far. Some general questions remain. How would the team and the individual team members be addressed? How would an "emergency button" be implemented that immediately stops the execution of the whole MRT? How would the possible actions of a human be modeled in the team plan? How would the MRT react to unanticipated actions of the human?

Sociotechnical concerns for technical innovations, as presented in Section 3.8 above, received increasing attention in society recently. Clearly, these concerns also apply to robot teamwork. As in other technical domains, the big question is as follows: how do we translate abstract sociotechnical requirements that are mostly specified in natural language into concrete engineering artefacts for multi-robot applications? For example, liability issues for a team of heterogeneous robots from different manufacturers could be difficult to decide. Likewise, explanations for team decisions and actions may be even more difficult to understand for a user who demands transparency for MRT activities; this will undermine the user´s trust in the technology and may lead to a lack of acceptance. Experts from different disciplines must work together to solve these interdisciplinary puzzles.

4.7. General Remarks

As for any distributed system, the engineering of robot teamwork raises questions about concerns such as scalability, fault tolerance, performance, software evolution, and the like. It is important that developers of teamwork applications respond to these concerns. We do not discuss these concerns here because they do not create specific questions for the engineering of robot teamwork.

5. Conclusions

The proliferation of robotics is likened often to the evolution of the personal computer (PC). 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 collaborate to achieve a common goal. Due to the manifold technical, contextual, and situational dependencies, these teams will often 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 focused particularly on the engineering of teamwork for multi-robot systems that operate in dynamically changing environments. We tried to raise the awareness for crucial issues

in the realization of such teamwork, and we pointed out exemplary solution approaches. Clearly, the diversity of application requirements is huge, and the design space is vast. This will keep the research and development community busy in the coming years.

Funding: This research received no external funding.

Acknowledgments: Parts of this paper were written while the author was a guest scientist at IMT Lucca (Italy). The author sincerely thanks 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 discussions with researchers from Universidad Carlos III de Madrid (UC3M) and IMDEA Networks, as well as the constructive comments by the anonymous reviewers.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Grosz, B.; Kraus, S. Collaborative plans for complex group action. *Artif. Intell.* **1996**, *86*, 269–357. [CrossRef]
2. Jennings, N. Commitments and conventions: The foundation of coordination in multi-agent systems. *Knowl. Eng. Rev.* **1993**, *8*, 223–250. [CrossRef]
3. Levesque Hector, J.; Cohen Philip, R.; Nunes José, H.T. On Acting Together. In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), Boston, MA, USA, 29 July–3 August 1990; pp. 94–99.
4. Parker, L.E. Current state of the art in multi-robot teams. In *Distributed Autonomous Robotic Systems*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 3–12.
5. Tambe, M. Towards flexible teamwork. *J. Artif. Intell. Res.* **1997**, *7*, 83–124. [CrossRef]
6. Farinelli, A.; Iocchi, L.; Nardi, D. Multi-Robot Systems: A classification focused on coordination. *IEEE Trans. Syst. Man Cybern. B* **2004**, *34*, 2015–2028. [CrossRef] [PubMed]
7. Pěchouček, M.; Mařik, V. Industrial deployment of multi-agent technologies: review and selected case studies. *J. Auton. Agents Multi Agent Syst.* **2008**, *17*, 397–431. [CrossRef]
8. Chennareddy, S.; Agrawal, A.; Anupama, K.R. Modular Self-Reconfigurable Robotic Systems: A Survey on Hardware Architectures. *J. Robot.* **2017**, *2017*, 1–19.
9. Rizk, Y.; Awad, M.; Tunstel, E.W. Cooperative Heterogeneous Multi-Robot Systems: A Survey. *ACM Comput. Surv.* **2019**, *52*, 29–31. [CrossRef]
10. Gerkey, B.P.; Mataric, M.J. A formal analysis and taxonomy of task allocation in multi-robot systems. *Int. J. Robot. Res.* **2004**, *23*, 939–954. [CrossRef]
11. Mosteo, A.R.; Montano, L. *A Survey of Multi-Robot Task Allocation*; Technical Report AMI-009-10-TEC; University of Zaragoza: Zaragoza, Spain, 2010.
12. Doran, J.; Franklin, S.; Jennings, N.; Norman, T. On cooperation in multi-agent systems. *Knowl. Eng. Rev.* **1997**, *12*, 309–314. [CrossRef]
13. Parker, L.E. Distributed intelligence: Overview of the field and its application in multi-robot systems. *J. Phys. Agents* **2008**, *2*, 5–14. [CrossRef]
14. Agha, G. *Actors: A Model of Concurrent Computation in Distributed Systems*; MIT Press: Cambridge, MA, USA, 1986.
15. Sangiorgi, D.; Walker, D. *The Pi-Calculus: A Theory of Mobile Processes*; Cambridge University Press: Cambridge, UK, 2003.
16. Prasad, K. A calculus of broadcasting systems. In *TAPSOFT'91*; Springer: Berlin/Heidelberg, Germany, 1991; pp. 338–358.
17. De Nicola, R.; Di Stefano, L.; Inverso, O. Towards formal models and languages for verifiable Multi-Robot Systems. *Front. Comput. Sci.* **2018**, *5*. [CrossRef]
18. Alrahman, Y.A.; De Nicola, R.; Loreti, M. On the Power of Attribute-Based Communication. In Proceedings of the 36th IFIP WG 6.1 International Conference, FORTE 2016, Heraklion, Greece, 6–9 June 2016; pp. 1–18.
19. De Nicola, R.; Duong, T.; Inverso, O.; Trubiani, C. *AErlang: Empowering Erlang with Attribute-Based Communication*; Jacquet, J.-M., Massink, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 21–39.
20. Alrahman, Y.A.; De Nicola, R.; Garbi, G. GoAt: Attribute-based Interaction in Google Go. *Comput. Sci.* **2018**. [CrossRef]

21. Sun Microsystems, JXTA Java Standard Edition V2.5: Programmers Guide. 2007. Available online: https://www.tamps.cinvestav.mx/~{}vjsosa/clases/redes/JXTA_SE_ProgGuide_v2.5.pdf (accessed on 18 February 2020).
22. Ogata, Y.; Spaho, E.; Matsuo, K.; Barolli, L.; Moreno, J.; Xhafa, F. JXTA-Overlay P2P Platform and Its Application for Robot Control. In Proceedings of the 13th International Conference on Network-Based Information Systems (NBiS 2010), Takayama, Gifu, Japan, 14–16 September 2010; pp. 133–138.
23. Elkady, A.; Sobh, T. Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. *J. Robot.* **2012**, *2012*, 1–15. [CrossRef]
24. Mohamed, N.; Al-Jaroodi, J.; Jawhar, I. Middleware for Robotics: A Survey. In Proceedings of the 2008 IEEE Conference on Robotics, Automation and Mechatronics, Chengdu, China, 21–24 September 2008; pp. 736–742.
25. Bruyninckx, H.; Soetens, P.; Koninckx, B. The Real-Time Motion Control Core of the Orocos Project. In Proceedings of the 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, 14–19 September 2003; pp. 2766–2771.
26. Volpe, R.; Nesnas, I.A.D.; Estlin, T.; Mutz, D.; Petras, R.; Das, H. CLARAty: Coupled Layer Architecture for Robotic Autonomy. Tech. rep. NASA Jet Propulsion Laboratory: Pasadena, CA, USA, 2000.
27. Utz, H.; Sablatnog, S.; Enderle, S.; Kraetzschmar, G. Miro—Middleware for mobile robot applications. *IEEE Trans. Robot. Autom.* **2002**, *18*, 493–497. [CrossRef]
28. Object Management Group (OMG). *The Common Object Request Broker: Architecture and Specification (CORBA 3.3)*; Object Management Group: Needham, MA, USA, 2012.
29. Shumko, S. Ice Middleware in the New Solar Telescope's Telescope Control System. In *Astronomical Data Analysis Software and Systems XVIII*; Astronomical Society of the Pacific: Orem, UT, USA, 2009; Volume 411.
30. Petters, S.; Thomas, D.; von Stryk, O. RoboFrame—A Modular Software Framework for Lightweight Autonomous Robots. In Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware, IEEE/RSJ IROS, San Diego, CA, USA, 29 October – 2 November 2007.
31. Baer, P.A. Platform-Independent Development of Robot Communication Software. Ph.D. Thesis, University of Kassel, Kassel, Germany, 2008.
32. Robot Operating System. Available online: https://index.ros.org/ (accessed on 10 February 2020).
33. Abbas, H.A.; Shaheen, S.I.; Amin, M.H. Organization of Multi-Agent Systems: An Overview. *Int. J. Intell. Inf. Syst.* **2015**, *4*, 46–57.
34. Bulling, N. *A Survey of Multi-Agent Decision-Making, KI - Künstliche Intelligenz*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 147–158.
35. Grossi, D.; Dignum, F.; Dastani, D.; Royakkers, L. Foundations of Organizational Structures in Multiagent Systems. In Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), Utrecht, The Netherlands, 25–29 July 2005; pp. 690–697.
36. Geihs, K.; Witsch, A. Decentralized decision-making in adaptive multi-robot teams. *Inf. Technol.* **2018**, *60*, 239–248. [CrossRef]
37. Skubch, H. Modelling and Controlling Behaviour of Cooperative Autonomous Mobile Robots. Ph.D. Thesis, Universität Kassel, Kassel, Germany, 2012.
38. Nguyen Van, T.; Fredivianus, N.; Tran Huu, T.; Geihs, K.; Binh Huynh, T. Formal Verification of ALICA Multi-agent Plans Using Model Checking. In Proceedings of the 9th Int. Symposium on Information and Communication Technology, Danang, Vietnam, 6–7 December 2018.
39. Fowler, M. *Domain-Specific Languages*; Addison-Wesley: Boston, MA, USA, 2010.
40. Verma, V.; Estlin, T.; Jonsson, A.; Pasareanu, C.; Simmons, R.; Sing Tso, K. Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences. In Proceedings of the Internatational Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS), Munich, Germany, 5–8 September 2005.
41. Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Dolan, J.; Duggins, D.; Ferguson, D.; Galatali, T.; Geyer, C.; et al. *Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge*; CMU TR Urmson-2007-9708; Carnegie Mellon University: Pittsburgh, PA, USA, 2007.
42. Kammel, S.; Ziegler, J.; Pitzer, B.; Werling, M.; Gindele, T.; Jagszent, D.; Schroder, J.; Thuy, M.; Goebl, M.; Hundelshausen, F.; et al. Team AnnieWAY's Autonomous System for the 2007 DARPA Urban Challenge. *J. Field Robot.* **2008**, *25*, 615–639. [CrossRef]

43. Dhouib, S.; Kchir, S.; Stinckwich, S.; Ziadi, T.; Ziane, M. *Robotml, A Domain-Specific Language to Design, Simulate and Deploy Robotic Applications*; Noda, I., Ando, N., Brugali, D., Kuffner, J.J., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 149–160.
44. Arda, K.; Ümit, O. Hierarchical Finite State Machines for Autonomous Mobile Systems. *Control Eng. Pract.* **2013**, *21*, 184–194.
45. The Eclipse Foundation: Papyrus. Available online: https://www.eclipse.org/papyrus-rt (accessed on 2 April 2019).
46. Skubch, H.; Wagner, M.; Reichle, R.; Geihs, K. A Modelling Language for Cooperative Plans in Highly Dynamic Domains. *Mechatronics* **2011**, *21*, 423–433. [CrossRef]
47. Zweigle, O.; Lafrenz, R.; Buchheim, T.; Käppeler, U.P.; Rajaie, H.; Schreiber, F.; Levi, P. Cooperative Agent Behavior Based on Special Interaction Nets. In Proceedings of the 9th International Conference on Intelligent Autonomous Systems—IAS, Tokyo, Japan, 7–9 March 2006; pp. 651–659.
48. Nordmann, A.; Hochgeschwender, N.; Wigand, D.; Wrede, S. A Survey on Domain-Specific Modeling and Languages in Robotics. *J. Softw. Eng. Robot.* **2016**, *7*, 75–99.
49. Kaminka, G.A.; Frenkel, I. *Flexible Teamwork in Behavior-Based Robots*; Veloso, M.M., Kambhampati, S., Eds.; The MIT Press: Cambridge, MA, USA, 2005; pp. 108–113.
50. Tate, A. Generating Project Networks. In Proceedings of the 5th International Joint Conference on Artificial Intelligence IJCAI'77, Cambridge, MA, USA, 22–25 August 1977; Morgan Kaufmann Publishers Inc.: Burlington, MA, USA, 1977; Volume 2, pp. 888–893.
51. Kiener, J.; Von Stryk, O. Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. *Robot. Auton. Syst.* **2010**, *58*, 921–929. [CrossRef]
52. Opfer, S.; Jakob, S.; Jahl, A.; Geihs, K. ALICA 2.0—Domain-Independent Teamwork. In Proceedings of the 42nd German Conference on Artificial Intelligence (KI2019), Kassel, Germany, 23–26 September 2019.
53. Lamport, L. The Part-time Parliament. *ACM Trans. Comput. Syst.* **1998**, *16*, 133–169. [CrossRef]
54. Pinciroli, C.; Beltrame, G. Buzz: An Extensible Programming Language for Heterogeneous Swarm Robotics. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 3794–3800.
55. Lomuscio, A.; Qu, H.; Raimondi, F. MCMAS: An open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.* **2017**, *19*, 9–30. [CrossRef]
56. De Nicola, R.; Loreti, M.; Pugliese, R.; Tiezzi, F. A Formal Approach to Autonomic Systems Programming. *ACM Trans. Auton. Adapt. Syst.* **2014**, *9*, 1–29. [CrossRef]
57. Niemczyk, S.; Opfer, S.; Fredivianus, N.; Geihs, K. ICE: Self-Configuration of Information Processing in Heterogeneous Agent Teams. In Proceedings of the Symposium on Applied Computing 2017, Marakesh, Marocco, 4–6 April 2017; pp. 417–423.
58. Opfer, S.; Jakob, S.; Geihs, K. Reasoning for Autonomous Agents in Dynamic Domains: Towards Automatic Satisfaction of the Module Property. In *Agents and Artificial Intelligence*, 1st ed.; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 22–47. ISBN 978-3-319-93581-2.
59. Opfer, S.; Jakob, S.; Geihs, K. Teaching Commonsense and Dynamic Knowledge to Service Robots. In Proceedings of the 11th Conference on Social Robotics (ICSR2019), Madrid, Spain, 26–29 November 2019.
60. Paulius, D.; Sun, Y. A Survey of Knowledge Representation in Service Robotics. *Robot. Auton. Syst.* **2019**, *118*, 13–30. [CrossRef]
61. Riazuelo, L.; Tenorth, M.; Di Marco, D.; Salas, M.; Gálvez-López, D.; Mösenlechner, L.; Kunze, L.; Beetz, M.; Tardos, J.D.; Montano, L.; et al. Roboearth semantic mapping: A cloud enabled knowledge-based approach. *IEEE Trans. Autom. Sci. Eng.* **2015**, *12*, 432–443. [CrossRef]
62. Keeney, J. Completely Unanticipated Dynamic Adaptation of Software. Ph.D. Thesis, The University of Dublin, Dublin, Ireland, 2004.
63. Khan, M.U. Unanticipated Dynamic Adaptation of Mobile Applications. Ph.D. Thesis, University of Kassel, Kassel, Germany, 2010.
64. Floch, J.; Fra, C.; Fricke, R.; Geihs, K.; Wagner, M.; Lorenzo, J.; Cantero, E.S.; Mehlhase, S.; Paspallis, N.; Rahnama, H.; et al. Playing MUSIC—Building context-aware and self-adaptive mobile applications. *Softw. Pract. Exp.* **2013**, *43*, 359–388. [CrossRef]
65. Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T. *Answer Set Solving in Practice*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2012; Volume 6.

66. Gelfond, M.; Kahl, Y. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*; Cambridge University Press: Cambridge, UK, 2014.
67. Clocksin, W.F.; Mellish, C.S. *Programming in PROLOG*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2003.
68. Baraki, H.; Schwarzbach, C.; Jakob, S.; Jahl, A.; Geihs, K. SAM: A Semantic-Aware Middleware for Mobile Cloud Computing. In Proceedings of the 11th IEEE International Conference On Cloud Computing (IEEE CLOUD 2018), San Francisco, CA, USA, 2–7 July 2018.
69. Scioni, E. Online Coordination and Composition of Robotic Skills: Formal Models for Context-aware Task Scheduling. Ph.D. Thesis, KU Leuven, Leuven, Belgium, 2016.
70. Geihs, K.; Evers, C. User Intervention in Self-Adaptive Context-Aware Applications. In Proceedings of the 17th Australasian User Interface Conference (AUIC), Canberra, Australia, 2–5 February 2016.