A FIPA-based communication infrastructure for a reconfigurable multi-robot system

Firstname Lastname[1] and Firstname Lastname[1]

Institute, City, Country
firstname.lastname@institute.org

Abstract. This paper presents a high-level communication infrastructure to deal with dynamically changing reconfigurable multi-robot systems. The infrastructure builds upon official standards of the Foundation for Intelligent Physical Agents (FIPA). FIPA standards have been successfully applied in a variety of multi-agent frameworks, but they have found little application in the domain of robotics. This paper introduces an implementation that can complement existing robotic communication frameworks and allows the robotics community to take better advantage of multi-agent research efforts. We present the essential components of the infrastructure and show its interoperability using the widely known multi-agent framework JADE.

1 Introduction

Robustness and reliability are key aspects for the design of robots that have to operate autonomously in remote places.

Single robotic systems are widely applied, but reliability can often be achieved only by increasing redundancy. In contrast, multi-robot systems (MRSs) inherently offer a higher level of redundancy and are thus naturally suited for application scenarios where systems are exposed to unforeseen risks of outage. This holds even more when looking at reconfigurable multi-robot systems (RMRSs), e.g., [20] presents a RMRS which consists of multiple robotic systems which can be dynamically extended using modular so-called payload-items (cf. Figure 2). The modularity of the system is achieved by introducing a standardized electro-mechanical interface (EMI) that allows to connect two systems. Payload-items serve as generic containers which can host sensors; they comprise two EMIs.

Due to its modularity an RMRS offers more flexibility to change its morphology and to cope with dynamically arising challenges. In order to exploit the redundancy of a RMRS an effective coordination mechanism has to be put in place, e.g., to dynamically build a payload-stack from a number of payload-items. Additionally, individual systems can appear or disappear in the communication infrastructure either as part of a nominal operation, e.g., as result of a merge of two systems, or as part of a non-nominal outage. At the foundation of organizing the RMRS presented in [20] lies a communication infrastructure that

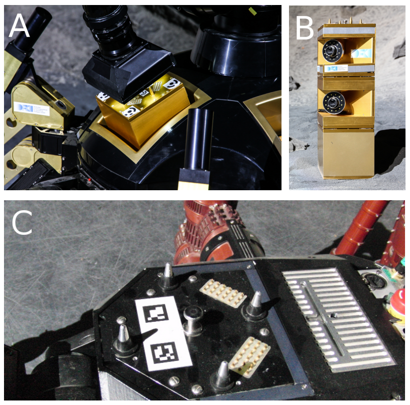


Fig. 2: A modular, reconfigurable multi-robot system [20], A: Sherpa robot's manipulator approaching a payload-item, B: stack of two camera and one power payload-item, C: electro-mechanical interface on the back of the legged robot CREX

accounts for dynamically changing robotic systems which establishes a peer-2-peer communication network. Compared to [20] we have significantly revised the infrastructure's components to improve the applicability. This paper details the revised infrastructure and its benefits for robotic cooperation. The following Section 2 presents the state of the art and background information to motivate the approach. Section 3 provides a detailed presentation of the essential components. Subsequently, we present an evaluation in Section 4 and illustrate a typical application and interoperability. We conclude with a discussion of the benefits and limitations of the presented approach.

2 Background

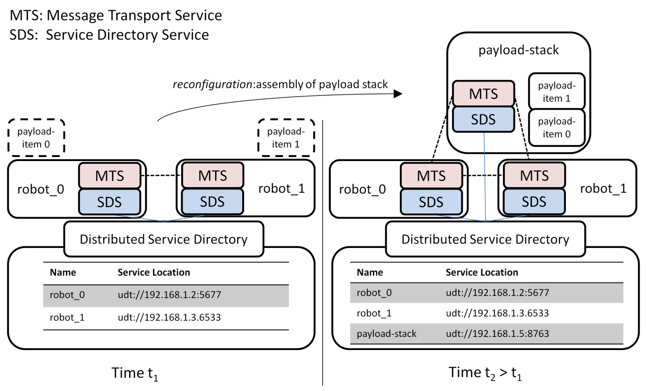
A core activity of robotics is the integration of hardware and software, and managing communication within a robotic system takes a significant share of this work. Nowadays, robotic middlewares or frameworks such as Robot Operating System (ROS) [17] or Robot Construction Kit (Rock) [11] are key to effectively implementing robots. These frameworks typically rely on specialized and modular components that are inter-connected to form an information processing network designed to solve a given problem. The mentioned frameworks allow the construction of distributed systems, but focus on setting up single robotic systems. ROS and Rock, for example, depend on the availability of a centralized naming service to access and connect components; while the former relies on an instance of the so-called ROS Master, the later requires CORBA [13] to provide the naming service. Several strategies have been applied by these frameworks to mitigate these effects, but support for fully distributed systems with no single point of failure has not been achieved. MRSs use communication patterns such as broadcasting [14], blackboards [1] or cloud-based [8] communication, mixed with publish-subscribe mechanisms [12] or peer-to-peer solutions.

Though these communication patterns have their individual benefits, a main challenge for completely decoupled robots resides in setting up and maintaining the inter-communication channel. Additionally, distributed agent-based systems use auctions as a common interaction pattern to solve the resource allocation problem [21]. However, communication in all the robotic frameworks mentioned above is based on stateless message exchange and does not naturally extend to auctions, which have a state-based interaction pattern. In order to facilitate the development of physical agents the Foundation for Intelligent Physical Agents (FIPA) [6] has devised a set of standards, defining – among other things – an abstract agent architecture including message formats and interaction protocols. The abstract architecture consists of a set of infrastructure services such as an agent directory with *white pages* and *yellow pages* functionality, i.e., a component that manages the information about all known agents. Though FIPA is now a standards committee of IEEE, the application of these standards in the domain of MRSs is rarely seen. In contrast, a broad application is found in the area of multi-agent systems (MASs), e.g., Java Agent DEvelopment Framework (JADE) [3] offers a reference implementation of FIPA standards and it is widely used in the multi-agent community. The best known implementations of FIPA standards, such as FIPA-OS [16] and JADE [3] are Java-based – a factor that might hinder a broader application in robotics where C/C++ implementations are often predominant. Mobile-C [4] offers a C-based implementation, but sets its focus on a different use case: the transition of software-agents between multiple machines by relying on a C/C++ interpreter. Our communication architecture builds upon FIPA standards as an outcome of the MAS community. At the same time we close the aforementioned implementation gap.

3 A FIPA-based communication infrastructure

Reconfiguration in the context of MRS refers to a change of morphology, i.e., two or more previously separated systems are physically merged to form a so-called *coalition*. More formally, a RMRS consists of a set of physical agents $A = \{a_1, a_2, \ldots, a_n\}$, which can form coalitions $C \subseteq A$. Each coalition again represents a single physical actor. A distinct set of coalitions is typically denoted a *coalition structure* CS, and each agent can be interpreted as a coalition C of size $|C| = 1$ [24]. The overall flexibility of the RMRS directly depends on the modularity of the system and the fact that these systems can be (almost) arbitrarily combined.

This paper (cf. Figure 3) introduces a high-level communication layer that can complement existing frameworks such as ROS and Rock to build fully distributed systems and support the operations of an RMRS and its dynamics. The main requirements are: (i) transparent handling of coalitions that join or leave the communication infrastructure, (ii) interoperability and standardization of high-level communication, (iii) communication protocols for coordination of reconfiguration, and (iv) applicability to ARM-based devices such as payload-items [20].


MTS: Message Transport Service
SDS: Service Directory Service
reconfiguration:assembly of payload stack
payload-item 0
payload-item 1
payload-stack
MTS
SDS
robot_0
robot_1
Distributed Service Directory
Time t_1
Time $t_2 > t_1$

Name	Service Location
robot_0	udt://192.168.1.2:5677
robot_1	udt://192.168.1.3.6533

Name	Service Location
robot_0	udt://192.168.1.2:5677
robot_1	udt://192.168.1.3.6533
payload-stack	udt://192.168.1.5:8763

Fig. 3: Service infrastructure of a reconfigurable system, before and after reconfiguration, i.e., assembling two initially inactive payload-items to one active payload-stack

In a typical MRS the coalition structure is fixed and a single robot leaving the infrastructure can be considered a non-nominal event. In contrast, a RMRS takes advantage of a flexible coalition structure so that agents joining and leaving the system are part of nominal operation. Furthermore, one of the key features of the RMRS is extensibility and due to a standardized interface new functionality for the overall system can be independently designed. However, to guarantee interoperability communication has to be standardized as well; this does account for the inter-robot communication and does not extend to the internal communication of a single robot. We assume that the high-level communication layer acts as a control channel and allows for coordination of multiple systems, e.g., to perform reconfiguration.

We take the FIPA [6] as guidelinesince it applies to a fully distributed scenario and is therefore well suited as a guideline to our design inter-robot communication. In particular, we reuse the concept of a message-transport service (MTS) including the related Agent Communication Language (ACL), and the service-directory service (SDS) in order to establish a robust agent communication channel. FIPA-based communication accounts for interaction protocols as patterns of message flows between multiple robots based on so-called 'performatives' such as *request* or *query-when*. This facilitates the design of coordination protocols involving operators or autonomous agents, and in the context of RMRS serves as basis for automated negotiation for reconfiguration. The essential backbone of this communication infrastructure comprises the SDS and the MTS, which are described in the following and are implemented in our software library *fipa_services* [19].

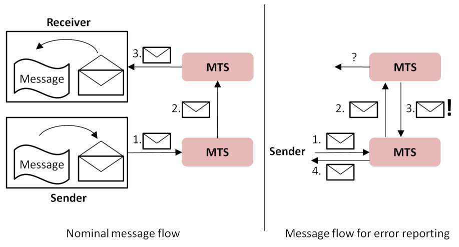
Compared to [20] the following changes have been made to increase applicability of the infrastructure: (i) encapsulation of main services into a C++ library, (ii) support for message envelopes (including bit-efficient and XML encoding), (iii) support for additional representation types for messages (support was limited to *bit-efficient*), and (iv) support for selecting transports.

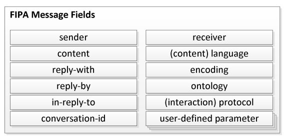
3.1 Service Directory Service

An SDS is a mandatory element of the FIPA Abstract Architecture [6] and all robots (including payload-items) in our infrastructure (cf. Figure 3) run an SDS, which overall forms a decentralized Distributed Service Directory (DSD). The DSD decouples the modular systems in the RMRS, since robots can register and deregister to this DSD dynamically by attaching or detaching to an MTS. The usage of a DSD eliminates the single-point of failure that comes with most existing robotic middlewares, but instead the DSD becomes a critical service overall for an RMRS. The current implementation builds upon Avahi [15] – a component for zeroconf [10] and service discovery that is standard to most Linux-based systems.

3.2 Message Transport Service

Along with the SDS the FIPA specification requires an MTS. The single purpose of the MTS is to deliver messages (wrapped in envelopes) to attached clients or to another MTS which can forward this message to one of its connected clients. Figure 3 shows the overall communication infrastructure. The MTS retrieves a receiver's transport address from the DSD and a receiver's name is interpreted as a regular expression for this search. This allows for a natural support of broadcasting and multicasting using wildcards in the receiver name. Each MTS stamps a handled messages to prevent looping of messages. Connections are established in a lazy fashion; the MTS tries to connect to the remote MTS only if a message is directed to a remote client. Each MTS can draw from a list of supported transports to transfer messages in-between two MTS – currently, *fipa_services* supports TCP and UDT [7].





FIPA Message Fields

sender	receiver
content	(content) language
reply-with	encoding
reply-by	ontology
in-reply-to	(interaction) protocol
conversation-id	user-defined parameter

Fig. 5: (left) Message handling for a nominal and an error scenario, (right) FIPA Message structure

Notification about failed delivery is part of the MTS functionality and all available transports are tried in order to deliver a message. If a receiver cannot be found in the DSD or the delivery fails at whatever MTS, an error response is triggered. This error message is propagated back to the sender (if it is still available), using the reverse communication path. Relay is not part of this infrastructure, but achieved by applying dedicated meshing protocols.

Figure 5 outlines the message flow for the nominal message delivery and a failed delivery. The error scenario considers that delivery fails at the MTS of the (expected) receiving client.

Messages comprise a set of standard fields (cf. Figure 5) and can be serialized using different representation types. Table 1 lists the currently supported representation types.

Table 1: Supported FIPA representations (evaluated in Section 4)

Element	**Representation types**
message	bit-efficient, XML, string
envelope	bit-efficient, XML

A message is put into an envelope which is then forwarded to an MTS; the envelope contains the serialized message as its payload. This allows the MTS to operate in a content-agnostic way, and decoding of the messages and its content only needs to be done by the final receiver. The results in Section 4 will illustrate this characteristic.

3.3 Conversation Monitor

FIPA also describes the application of interaction protocols, which make use of performatives defined in a message. An interaction protocol describes a message flow using a state transition system $\sum = (S, P, R, \gamma)$, where $S = \{s_0, s_1, \ldots\}$ is the set of conversation states, $P = \{$*accept-proposal, agree, ...*$\}$ is the set of performatives, and $R = \{r_1, r_2, \ldots\}$ is the set of roles. The set of roles typically consists only of a sender and receiver label. The state transition function is $\gamma : S \times P \times R \rightarrow 2^S$.

To take advantage of interaction protocols, we implemented a conversation monitor to validate state transitions that are triggered by incoming and outgoing messages. Validation is done with respect to a single agent which is involved in the conversation. To model interaction protocols we use a custom extension of State Chart XML (SCXML) [23] and embed default transitions for error handling and cancellation of interaction protocols to reduce the modeling effort for users. Figure 6 illustrates how the accounting for default states affects a simple request protocol.

This functionality allows to outline and validate inter-robot coordination, especially for reconfiguration and cooperation between multiple robotic systems.

4 Evaluation of fipa_services

In this section we evaluate the performance of our *fipa_services* library regarding potential effects on bandwidth and computation.. The evaluation shows the

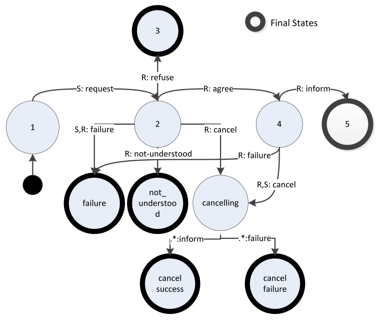


Fig. 6: Statemachine for the request protocol [6] including default states and transitions as well as roles: receiver R, sender S and wildcard $.*$ representing any other client. Transitions are labeled with roles (disjunctive) and performatives required to match.

impact of the selection of the representation type (cf. Table 1) and the applicability to embedded systems. The representation-dependent overhead for messages wrapped into an envelope is illustrated in Figure 7, and shows significant differences between the different representation types.

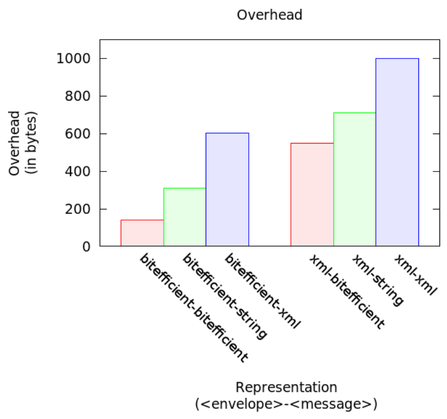


Fig. 7: Net overhead for a message with content size of 1 byte and the header containing 99 byte information filling all message fields (cf. Figure 5)

Figure 9 illustrates the message encoding and decoding performance on a PC with an Intel CORE i7 2.1 GHz, 12 GB RAM and a Gumstix with ARM Cortex-A7 CPU 720 MHz, 256 MB RAM [1]; the evaluation is based on 1000 encoding and decoding cycles that are measured using software timers.

This evaluation shows that bit-efficient message encoding is best suited for environments with limited bandwidth. It comes with a slight performance draw-

[1] All modular payload-items host a Gumstix Overo Fire.

back for small messages compared to the XML representation, but performs better for messages with large content. The string representation shows the worst performance for large messages on the Gumstix, while it remains ahead on XML on the PC.

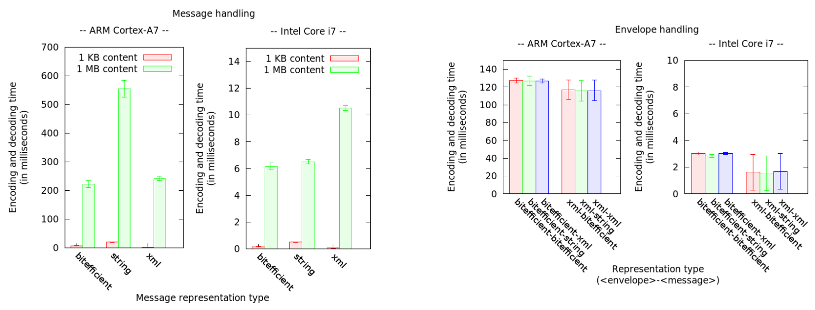


Fig. 9: (left) Message handling performance for 1000 cycles of encoding and decoding and 1 KB and 1 MB sized contents, (right) Evaluation of envelope handling for 1000 cycles of encoding and decoding and each envelope containing a message with 1 MB content

Figure 9 presents the performance for different combinations of envelope and message representations. The performance on the ARM-based system illustrates the achieved effect that an envelope's payload does not need to be decoded for transport on an MTS. Bit-efficient encoding for envelopes comes with a small performance drawback, but is also exhibits a significantly smaller standard deviation.

This implementation [19] fully supports bit-efficient encoding and is to our best knowledge the only publicly available implementation of FIPA's final standard; an implementation for the experimental standard exists in the form of a JADE plugin, but is limited to the message representation [9]. A fair comparison against the reference framework JADE could not be made due to the warm-up behavior of the Java Virtual Machine (JVM), i.e., code optimization is done at runtime. The influence of this feature in an application scenario will have to be investigated. However, initial findings indicate that after warm-up JADE will show a superior performance compared to *fipa_services*.

The benefits of using FIPA-based communication can be attributed to the partial standardization of coordination. All components in *fipa_services* have been developed to allow for simple integration into various robotic frameworks. We have embedded *fipa_services* into Rock and illustrate the application within this robotic framework for coordination of robotic reconfiguration and its interoperability in the following.

4.1 Application

To illustrate the application of the infrastructure we take two examples of robotic coordination: a mulit-robot reconfiguration process and distributed mutual exclusion.

Multi-robot reconfiguration High-level coordination is required to perform physical reconfiguration, i.e. merging, of two or more robots. To apply coordination the definition of a content language can introduce decoupling which allow for a more generic application. Here, we complemented our communication architecture using a content language to control a robot's actions; the content language is text-based and human-readable to permit interpretation on any system and facilitate debugging. The content language also allows querying available robot actions, along with the status of running actions. The content language enables robots to control each other and has been successfully applied as part of the unrevised infrastructure for the docking process in [20] in order to merge two systems: one master robot guiding another client robot to connect to one of the master's electro-mechanical interface (EMI) (cf. Figure 2). We successfully applied our revised infrastructure to perform the visually guided docking of the Sherpa robot's manipulator to the CREX robot [20]. Sherpa's manipulator was moved to a fixed position, and the CREX robot was remotely guided by the Sherpa robot.

Distributed mutual exclusion A main feature of an RMRS is the exchange of resources. But to perform automated reconfiguration as mentioned in the previous application scenario, the participating resources should be exclusively reserved beforehand. We implemented Ricart-Agrawala's [18] non-token-based and Suzuki-Kasami's [22] token-based algorithm using *fipa_services*. Both algorithms cannot deal with a dynamically changing set of agents or agent-failure. To overcome this limitation the detection of resource owners and agent-failure have been added to the implementation. Detection of the resource owner relies on a query message and the owner's response that is sent as broadcast. The detection of agent-failure is based upon sending heartbeat messages at low-frequency (0.2 Hz) in-between participants that depend upon each other, i.e., the (physical) owner of a resource, the lock holder and system waiting to lock the resource. This allows to mark a given resource to be unreachable and trigger further error handling. This approach leads to three overlapping interactions between multiple agents: (1) probing, (2) discovery of the owner of a resource, and (3) performing the actual locking. The three types of interactions are modeled with corresponding protocols. Figure 10 illustrates the protocol for the (extended) Ricart-Agrawla's algorithm. When an agent acquires or releases the lock it communicates this information to the resource owner using the performatives *confirm* and *disconfirm*. In addition to using interaction protocols, the use of a content language, e.g., for Suzuki-Kasami's algorithm to exchange the token between different agents, provides a further mean to separate and validate the flow of messages.

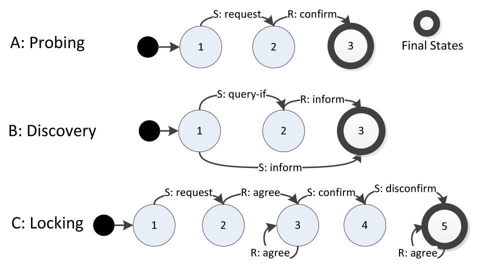


Fig. 10: Three separate interaction protocols that are part of the (extended) Ricart-Agrawala's algorithm; A: probing of inter-dependent agents, B: discovery of the resource owner (owner information is sent as broadcast), C: the message flow when performing the actual locking algorithm. Failure handling is embedded into default transitions (which are not depicted) and S denotes the conversation initiator's role and R the recipient's role.

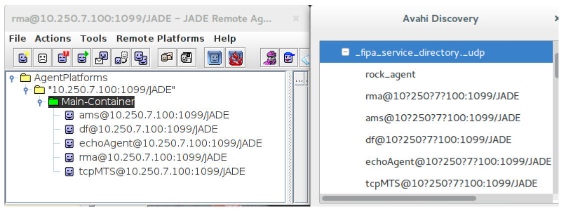
Modeling interaction protocols allows verification of the message flow at run-time and facilitates debugging. Protocol compliance of all participating agents can be enforced by using the conversation monitor (cf. Section 3.3).

4.2 Interoperability

Our motivation to use standards is to achieve extensibility and interoperability, e.g., due to the standardization of messages and infrastructure we can connect Rock to JADE using the existing extension mechanisms [2] and connecting JADE to the DSD. Since the DSD is based on Avahi it uses multi-cast DNS for communication. Registering services within JADE's infrastructure can thus be achieved by using JmDNS [5]. A problem was encountered upon registration of JADE agents since dots in service names cannot be handled by JmDNS; dots in agent names had to be replaced by the question mark wildcard for registration (cf. Figure 11). However, this change is transparent to users and agents, which can refer to other agents only by their real name.

Agents from both ecosystems communicate via XML encoded envelopes and messages. The bit-efficient encoding implementations of the two systems turned out to be incompatible – as mentioned, the only corresponding JADE plugin implements the experimental standard, while *fipa_services* implements the latest standard.

To validate the integration we placed an EchoAgent in the JADE infrastructure and attached a client *rock_agent* to an MTS in the Rock infrastructure; all components resided on the same local machine (Intel CORE i7 2.1 GHz). Figure 11 illustrates the resulting services registered in the DSD. The average round-trip time measured for a sequence of 10000 conversations and a message content size of 100 bytes was: 24±5.6 ms. For future applications, this integration allows to use JADE's infrastructure and tooling for high-level coordination in parallel to Rock's infrastructure and tools.



```
AgentPlatforms
  "10.250.7.100:1099/JADE"
    Main-Container
      ams@10.250.7.100:1099/JADE
      df@10.250.7.100:1099/JADE
      echoAgent@10.250.7.100:1099/JADE
      rma@10.250.7.100:1099/JADE
      tcpMTS@10.250.7.100:1099/JADE
```

```
_fipa_service_directory._udp
    rock_agent
    rma@10?250?7?100:1099/JADE
    ams@10?250?7?100:1099/JADE
    df@10?250?7?100:1099/JADE
    echoAgent@10?250?7?100:1099/JADE
    tcpMTS@10?250?7?100:1099/JADE
```

Fig. 11: Reuse of existing tooling: JADE's management GUI and the default avahi-discovery GUI to browse visible services, i.e., the set of registered services

5 Conclusion

An RMRS requires coordinating reconfiguration at various levels: (i) finding available resources, (ii) allocating resources, (iii) performing required reconfiguration maneuvers, and (iv) handling errors. This demands a structured approach and tool support in order to reach practical and scalable solutions. The presented infrastructure provides a basis using standardized components and allows verification of robot communication by relying on FIPA standards. The implemented libraries are open-source [19] and hosted along with Rock.

The criticism of [24] points out the semiformality of FIPA, but still acknowledges its practicality. Our experience confirms this standpoint and with *fipa_services* we provide the robotics community with easier access to these benefits with the intention to facilitate the application of multi-robot systems. In addition, while FIPA provides an architectural template, the individual services such as the DSD can remain completely out of the FIPA context.

Obviously, specific interaction scenarios can be easily solved by creating special message types and the presented infrastructure adds overhead for abstraction and standardization. However, in order to maintain a truly extensible and fully distributed RMRS, we advocate the presented approach.

Acknowledgments This work was supported by the German Space Agency (DLR) and conducted under grant agreement 50RA1301 (BMWi).

References

1. BBN Technologies: Cougaar Architecture Document (2004). URL `http://cougaar.org/doc/11_4/online/CAD_11_4.pdf`
2. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. John Wiley &amp; Sons, Ltd., Chichester, West Sussex, England (2007). DOI 10.1002/9780470058411
3. Bellifemine, F., Poggi, A., Rimassa, G.: JADE-A FIPA-compliant agent framework. In: Proceedings of the 4th International Conference on Practical Application of Agents and Multi-Agent Technology (PAAM), pp. 97–108 (1999)
4. Chen, B., Cheng, H.H., Palen, J.: Mobile-C: a mobile agent platform for mobile C-C++ agents. Software: Practice and Experience (Software Pract Ex) **36**(15), 1711–1733 (2006). DOI 10.1002/spe.v36:15

5. Cheshire, S.: JmDNS (2011). URL `http://jmdns.sourceforge.net/`
6. Foundation of Intelligent Physical Agents: FIPA Abstract Architecture Specification (2011). URL `http://www.fipa.org/specs/fipa00001/SC00001L.pdf`
7. Gu, Y., Grossman, R.L.: UDT: UDP-based data transfer for high-speed wide area networks. Computer Networks **51**, 1777–1799 (2007). DOI 10.1016/j.comnet.2006.11.009
8. Hartanto, R., Eich, M.: Reliable, cloud-based communication for multi-robot systems. In: The 6th Annual IEEE International Conference on Technologies for Practical Robot Applications (TePRA-2014). IEEE (2014)
9. Heikki, H., Laukkanen, M.: How to use the Bit-efficient ACL (BE-ACL) encoding with JADE (2015). URL `http://jade.tilab.com/doc/tutorials/BEFipaMessage.html`
10. IETF Zeroconf Working Group: Zero Configuration Networking ({Zeroconf}) (2011). URL `http://zeroconf.org/`
11. Joyeux, S., Schwendner, J., Roehr, T.M.: Modular Software for an Autonomous Space Rover. In: The 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014) (2014)
12. Object Management Group: Distributed Data Service (DDS) (2014). URL `http://www.omg.org/spec/DDS/`
13. Object Management Group: Common Object Request Broker Architecture (2015). URL `http://www.corba.org/`
14. Parker, L.E.: ALLIANCE: An architecture for fault tolerant multirobot cooperation. IEEE Transactions on Robotics and Automation **14**, 220–240 (1998). DOI 10.1109/70.681242
15. Poettering, L., Lloyd, T., Estienne, S.: Avahi (2012). URL `http://www.avahi.org`
16. Poslad, S., Buckle, P., Hadingham, R.: The FIPA-OS agent platform: Open source for open standards. 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (2000). URL `http://fipa-os.sourceforge.net`
17. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T.B., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software (2009)
18. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM **24**, 9–17 (1981). DOI 10.1145/358527.358537
19. Roehr, T.M.: Rock Multiagent (2015). URL `https://github.com/rock-multiagent`
20. Roehr, T.M., Cordes, F., Kirchner, F.: Reconfigurable Integrated Multirobot Exploration System (RIMRES): Heterogeneous Modular Reconfigurable Robots for Space Exploration. Journal of Field Robotics **31**(1), 3–34 (2014). DOI 10.1002/rob.21477
21. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, New York, NY, USA (2009)
22. Suzuki, I., Kasami, T.: A distributed mutual exclusion algorithm. ACM Transactions on Computer Systems **3**(4), 344–349 (1985). DOI 10.1145/6110.214406
23. W3C: State Chart XML (SCXML): State Machine Notation for Control Abstraction (2015). URL `http://www.w3.org/TR/scxml`
24. Weiss, G. (ed.): Multiagent Systems, 2nd edn. MIT Press (2009)