energies

Article

Communication-Focused Top-Down Design of Robotic Systems Based on Binary Decomposition

Piotr Pałka , Cezary Zieliński , Wojciech Dudek , Dawid Seredyński and Wojciech Szynkiewicz *

Institute of Control and Automation Engineering, Faculty of Electronics and Information Technology, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
* Correspondence: wojciech.szynkiewicz@pw.edu.pl; Tel.: +48-22-234-7119

Abstract: This article proposes a formal method of designing robotic systems focusing on communication between components, as well as standardization of the messages between those components. The objective is to design a robotic system controller in a systematic way, focusing on communication at an abstract agent level. Communication, thus organized, and its properly defined specification facilitate the system's further development. The method uses a standard message structure, based on IEEE FIPA standards, for communication within robotic systems composed of agents. Communication-focused top-down design of robotic systems based on binary decomposition is proposed, and used to design a companion robot working in the kitchen environment. The implemented robotic system is verified based on whether or not the specification conforms to the specified requirements. The characteristics of the designed communication are evaluated. The obtained results prove that the proposed method of designing robotic systems is formally correct, it facilitates the implementation of agents, and separates specification of the system from its implementation. The method of designing robotic systems is correct and useful. The proposed formal notation facilitates understanding of how the system operates and organizes the design process. It puts the communication between system components at the forefront. The resulting system specification facilitates the implementation. The tools for experimental evaluation of its characteristics enable the confirmation that it fulfills the requirements, and that the communication between the system components is correct.

Keywords: robot design; system design; communication approach; robotic agent; robot controller

check for updates

Citation: Pałka, P.; Zieliński, C.; Dudek, W.; Seredyński, D.; Szynkiewicz, W. Communication-Focused Top-Down Design of Robotic Systems Based on Binary Decomposition. *Energies* **2022**, *15*, 7983. https://doi.org/10.3390/en15217983

Academic Editors: Dariusz Pazderski and Slawomir Jan Stepien

Received: 29 September 2022
Accepted: 22 October 2022
Published: 27 October 2022

1. Introduction

The energy consumption of electromechanical systems, including robots, depends on the quality of control, and that is influenced by the control system architecture. This article delves into the problem of producing the architecture of the control system by analysing the task, or the category of tasks, that the system has to accomplish. Those tasks provide guidance for the system decomposition. Systems are decomposed into modules (agents in this case); however, those modules need to communicate to produce an integrated system. All these aspects influence the optimality of energy consumption of the overall system, thus it is necessary to study them. The presented design method addresses the pre-specification design phase, which is the basis for the creation of system specification, which in turn is the foundation of its implementation.

The spectrum of approaches to robotic system design stretches from purely implementation-based to specification-based. Implementation-based approaches encompass: (1) coding the system software in a universal programming language (e.g., C, C++, Python) targeting a specific implementation platform, (2) composing the system out of library modules (e.g., functions, procedures, software objects) coded in a universal programming language (e.g., RCCL [1], PASRO [2]), (3) utilising a programming framework, which is composed of a library of modules supplemented by use patterns (e.g., Player [3], MRROC++ [4]), (4) composing the system out of components [5] (usually software black-boxes, or less frequently white-boxes, providing services through well defined interfaces), which form a programming framework (e.g., OROCOS [6], ORCA [7], GenoM [8], ROS [9]). In all implementation-based cases, the system specification is formulated as a general architecture defined in terms of boxes and arrows diagrams. At the other end of this spectrum, systems are built using model driven approaches [10]. The system specification is provided as a system model that can be used for either manual or automatic code generation. In the manual mode, the model is a blueprint for the developers writing the system code. In the automatic mode, a compiler transforms the model into code automatically, usually in many transformation steps—tool-chains are used for that purpose [11]. This article addresses the problem of providing adequate guidance to the designer by proposing a specification method rationalising the design decisions, at the same time facilitating the system architecture design.

This article aspires to provide means for reasoning about the composition of the designed system, both from the point of view of constituent elements and the point of view of the communication between them. The problem tackled in this article is further aggravated by the fact that, although many robotic systems have been created, there is no single method of describing their architecture, e.g., [12–19]. References [20,21] suggest the decomposition of the architecture presentation into structure and activities (style). Nevertheless, Reference [20] acknowledges that it is difficult to distinguish between those two elements in the presentation of the majority of systems. Formal descriptions of robotic systems also exist, e.g., [22–28]; however, this approach is not popular in the robotics community. This is unfortunate as, besides a clear presentation, they enable formal verification of some system properties at the specification stage of their design, e.g., [29,30]. As most robotics systems have been designed using implementation-based methods, the composition of modules or components dominates, i.e., bottom-up design prevails.

However, the method postulated here uses a decomposition-based top-down approach. In the bottom-up method, the system's task emerges from the composition of modules, while in the top-down method, the system's task is the design-guiding principle. Instead of using an implementation-guided approach, the task-guided approach is favoured here. This approach focuses on the system-specification stage.

It should be noted that even if formal system specification methods are employed, the communication aspect is treated as an implementation detail, e.g., [31,32].

As robotic systems are usually inherently complex, the first problem the designer faces is how to tackle this complexity. System engineering provides a hint—use decomposition. Decomposition implies a division of the system into modules or components, allowing the designer to subsequently focus on those elements. However, this subdivision also implies that communication between them will have to be established. Although this communication aspect of system design is of paramount importance, it tends to be hidden behind the computational aspects of individual system modules. The services that the modules or components provide are at the forefront. Thus, this inter-module communication should gain a prominent position in the design process, but usually is treated as subordinate. Communication has to be discussed from two perspectives: (1) who communicates with whom, (2) what is being communicated, i.e., the contents of the communication. The design methodology proposed here takes into account all of these aspects of system design. The novelty of the proposed design methodology lies in the rationalisation of the decomposition process and putting inter-module communication on equal terms with computational aspects of modules.

Reference [20] points out that, for the majority of created systems, it is difficult to determine both their structure and activities. Usually architectures are presented by using informal textual descriptions and block diagrams with a very varied level of detail. This usually leads to problems during implementation and documentation of such systems. This is the consequence of the fact that the robotic system controller design lacks well-established design methodology. Sometimes the design relies on the software specification methods based on general software engineering practice, e.g., the specification is expressed in terms of UML diagrams [33]. However, there is no hint as to how to create the system architecture and how the task influences it. Knowing the hardware system composition and the tasks that it will be put to, there is no standard procedure leading to the specification of the structure and description of activities of the designed system. This paper tries to fill this gap and formulate such a methodology, simultaneously taking into account the abundant communication that appears in complex systems. The focus on communication is its unique feature.

The theses of the article are as follows. The proposed design method leads to an organised analysis of the communication at an abstract agent-level. The so-organised communication and its properly defined specification facilitate further system development. The proposed method uses a standard message structure for communication within robotic systems composed of agents. It is based on the IEEE FIPA standard, as utilised by multi-agent systems. The work contributes to the design of robotic systems in terms of: (i) standardization of the message structure for communication within robotic systems composed of agents applying IEEE FIPA standards (Foundation for Intelligent Physical Agents, brings together the creators of the agent technology); (ii) top-down, binary, communication focused design method for robotic systems. However, our work does not concern either the distribution of the system on multiple physical platforms [34,35] or the security of communication [36–39].

The structure of the article is as follows. Section 3 introduces the notation used in the article and presents the formal approach to the communication within robotic systems composed of embodied agents. Section 4 describes the proposed communication-focused design of robotics systems. Section 5 provides an example of the use of this approach. Section 6 draws the conclusions.

2. Related Work

Many architectures of robotic systems have been proposed (e.g., reactive [40], behaviour-based [12,29] subsumption [41]; however, layered architectures dominate. The decomposition into layers is conducted either based on the frequency of subsystem behaviour repetition or on the subdivision into subtasks the system is to execute [20]. Three-layered structures dominate, e.g., Sense–Plan–Act (SPA), subsumption [42], hybrid planning–reactive [43], hierarchic [44,45]), biologically inspired [22,23,46], using belief–desire– intension (BDI) approach [47,48]. Two-layered structures also have been proposed [49]. However, all of them provide guidance to the designer only as templates that can be used for the purpose of imitation.

Ahmad and Babar [50] presented a systematic classification of software architectural solutions for robotic systems. They suggest that architectural solutions support operations enabling information and resource distribution and development such as modelling, designing and programming robotic software.

Among component-based approaches, the Robot Operating System (ROS) is becoming the standard programming framework for developing robotic applications [9,51]. The ROS-based software architecture is composed of components and connectors between them, that are partially specified in the code and created at run-time. However, there are no generally accepted rules and guidelines for developing applications to perform the required task. Moreover, static information about the architecture during system configuration is limited, and consistency checking during development is usually very difficult or even impossible. The quality of the created software depends primarily on the skills and experience of the developers and is mostly a matter of trial and error [52].

The service-driven architecture model or Service-Oriented Architecture (SOA) for robotics relies on loosely connected, heterogeneous and dynamically composed services [53], which are typically located in the cloud. SOA involves using Internet technologies to define services through which robots can be accessed or robots can access the resources of other machines. Service-based robotics enables robot operations by utilizing powerful computing capacities, virtually unlimited storage, and communication resources available from cloud-based infrastructures.

Model Driven Engineering (MDE) is a formal methodology that can be used for the design of robotics software, which involves a transition from code-based software development to model-based development [52]. MDE for robotics provides an opportunity to systematically and simultaneously bring together different levels of abstraction at which developers can work to improve the quality of the system in terms of security, reliability and reusability. The model is defined in terms of formal meta-models (or generic models). Software languages for the specification of system models play a similar role to that of meta-models (e.g., [54]). In rare cases, meta-models are provided, guiding the developers in the design of the system model (e.g., [55]). The problem is that the above mentioned approaches must strike a balance between too much freedom of choice and too little—this is termed proper freedom from choice [56,57]. Unfortunately, there is no one generally accepted definition of a meta-model. Another major problem among existing model-based robotic system design methodologies is their complexity. The software development process is multi-stage, and the number of models required is significant. In consequence, multiple domain languages and tools are needed to support the development process of robotic systems.

3. Inter-Agent Communication

3.1. Notational Convention

Formalisation of the discussion of any subject requires adequate notation. The numerosity of discussed abstract objects necessitates an introduction of a naming convention to facilitate the decryption of the symbols used. The article discusses both the system structure and its activities. The system structure is described in terms of agents a and their subsystems s. Each agent has its own name, which, in general, is represented by j used as a right subscript of the central symbol, i.e., a_j, or its control subsystem c_j. Similarly, a subsystem of a_j may have its own name v; thus, to distinguish it $s_{j,v}$ is used. Subsystems interact with each other using input (left subscript x) and output (left subscript y) buffers, i.e., ${}_{x}s_{j,v}$, ${}_{y}s_{j,v}$. Usually the type of subsystem s' from which the information is received or to which it is dispatched is denoted by a left superscript of a buffer symbol, i.e., ${}^{s'}_{x}s_{j,v}$, ${}^{s'}_{y}s_{j,v}$, where $s \in \{c, e, r\}$ and $s' \in \{c, e, r, E, R, T\}$, i.e., control subsystem c, virtual effector e, virtual receptor r, real effector E, real receptor R or an inter-agent transmission buffer T. The internal memory of a subsystem is denoted by ${}^{s}s_{j,v}$. As it is not connected to other subsystems, no leading subscript x or y is used. As the values contained in the buffers change in time, a discrete time index has to be introduced for each of the subsystems separately. If the time counter is i, the contents of a specific buffer are labeled with a right superscript, e.g., ${}_{x}s^{i}_{j,v}$ or ${}_{y}s^{i+1}_{j,v}$, to denote that the contents are considered at the current instant i or the next one $i+1$, respectively.

The activities of an agent result from the activities of its subsystems and their interaction through the abovementioned buffers. The activities of a subsystem $s_{j,v}$ of an agent a_j are described hierarchically by using the following concepts. At the lowest level appear transition functions ${}^{s}f_{j,v,\omega}$, terminal conditions ${}^{s}f^{\tau}_{j,v,\omega}$, error conditions ${}^{s}f^{\varepsilon}_{j,v,\omega}$, where ω is the name of a particular function. At the intermediate level, behaviours ${}^{s}\mathcal{B}_{j,v,\gamma}$ appear, where γ is the name of a particular behaviour. At the highest level, a finite state machine (FSM) ${}^{s}\mathcal{F}_{j,v}$ appears. ${}^{s}\mathcal{F}_{j,v}$ is represented by a graph containing nodes/states ${}^{s}\mathcal{S}^{\nu}_{j,v}$ (where ν is a particular state designator) and directed arcs labeled by initial conditions ${}^{s}f^{\sigma}_{j,v,\delta}$, where δ is its name. In general all functions are symbolised by f. Terminal, error and initial conditions are predicates and are distinguished by the right superscript τ, ε and σ, respectively. Such a superscript is missing in the case of a transition function.

A group of agents ${}^{a}g_q$, where q is the group name, aggregates one or more agents or subgroups. A group of agents is used as a component of a robotic system when the exact internal structure is unknown or when this component will be decomposed. Communication between groups ${}^{a}g_q$ and ${}^{a}g_{q'}$ is carried out through the communication channel $\mathcal{C}^k_{q,q'}$, where k is its name, and q and q' are the names of the communicating groups of agents. One or more communication channels may exist between a pair of agent groups. The communication channels $\mathcal{C}^k_{q,q'}$ and $\mathcal{C}^k_{q',q}$ are identical.

3.2. Influence of Communication on System Activities

An embodied agent a is subdivided into subsystems, namely: a single control subsystem c and zero or more virtual and real effectors (e and E, respectively) and zero or more virtual and real receptors (r and R, respectively) (Figure 1) [27,28,58–60]. As only the control subsystem of an agent has the capability of interacting with other agents (precisely speaking with their control subsystems) this subsystem will be the focus of our attention. The primary activity of the control subsystem c_j of an agent a_j, is the computation of the control subsystem transition function ${}^{c}f_{j,m}$ (m is the name of the particular function), which in the decomposed form can be presented as:

$$\begin{cases} {}^{c}c^{i+1}_{j} & := {}^{c,c}f_{j,m}({}^{c}c^{i}_{j}, {}^{e}_{x}c^{i}_{j}, {}^{r}_{x}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \\ {}^{e}_{y}c^{i+1}_{j} & := {}^{c,e}f_{j,m}({}^{c}c^{i}_{j}, {}^{e}_{x}c^{i}_{j}, {}^{r}_{x}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \\ {}^{r}_{y}c^{i+1}_{j} & := {}^{c,r}f_{j,m}({}^{c}c^{i}_{j}, {}^{e}_{x}c^{i}_{j}, {}^{r}_{x}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \\ {}^{T}_{y}c^{i+1}_{j} & := {}^{c,T}f_{j,m}({}^{c}c^{i}_{j}, {}^{e}_{x}c^{i}_{j}, {}^{r}_{x}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \end{cases} \tag{1}$$

From the point of view of inter-agent communication, the utilisation of the transmission input buffer ${}^{T}_{x}c_j$ and output buffer ${}^{T}_{y}c_j$ is most important. Whenever ${}^{c}f_{j,m}$ is used, ${}^{c,T}f_{j,m}$ is the only partial function producing the contents of the output buffer ${}^{T}_{y}c_j$. Thus, only the partial transition function ${}^{c,T}f_{j,m}$ produces the response of the agent a_j to the messages inserted into ${}^{T}_{x}c_j$ by the other agents. However, all partial transition functions use ${}^{T}_{x}c_j$ to compute the commands for other subsystems of the agent a_j; thus, the obtained message influences all the activities of the agent. The transition function (1) takes in its arguments at discrete time i and computes the values dispatched to the other subsystems at time $i+1$. This defines the subsystem sampling time. Hence the elementary action of a subsystem is defined for a single sampling period and consists of:

1. Computation of the transition function ${}^{c}f_{j,m}$ and assigning the computed values to appropriate components of the output buffer ${}_{y}c_j = [{}^{e}_{y}c_j, {}^{r}_{y}c_j, {}^{T}_{y}c_j]$ and internal memory ${}^{c}c_j$,
2. Dispatching the contents of ${}_{y}c_j$ to the associated subsystems,
3. Incrementation of the discrete time counter i,
4. Acquiring into ${}_{x}c_j = [{}^{e}_{x}c_j, {}^{r}_{x}c_j, {}^{T}_{x}c_j]$ new input values from the cooperating subsystems.

This elementary action is iterated until a predicate ${}^{c}f^{\tau}_{j,m}$, called the terminal condition, or ${}^{c}f^{\varepsilon}_{j,m}$, called an error condition, becomes true. The process of iterating the elementary action is called a behaviour ${}^{c}\mathcal{B}_{j,m}$ and can be represented by a graph (Figure 2). As behaviour depends on the transition function, terminal condition and error condition, it is parameterised by all of them: ${}^{c}\mathcal{B}_{j,m}$ $({}^{c}f_{j,m}, {}^{c}f^{\tau}_{j,m}, {}^{c}f^{\varepsilon}_{j,m})$; however, for brevity the dependence on ${}^{c}f^{\varepsilon}_{j,m}$ will be neglected. Once behaviour ${}^{c}\mathcal{B}_{j,m}$ terminates its actions, another behaviour has to be chosen. This choice is based on a predicate ${}^{c}f^{\sigma}_{j,v,m,m'}$, called the initial condition (m and m' are the designators of the nodes connected by the directed arc labeled by ${}^{c}f^{\sigma}_{j,v,m,m'}$). Initial conditions label the directed arcs connecting the nodes of a graph representing a Finite State Machine (FSM) ${}^{c}\mathcal{F}_j$. With each of its nodes (states) ${}^{c}\mathcal{S}^{m}_{j}$, a behaviour ${}^{c}\mathcal{B}_{j,m}$ is associated; thus, switching states switches behaviours. As initial conditions are parameterised by the contents of input buffers, here again the acquired message influences the activities of the agent. Hence the messages introduced into the input buffer ${}^{T}_{x}c_{j}$ can influence the activities of an agent in three ways:

- In each iteration of the behaviour through the influence on the values computed by the iterated transition function (1),
- By terminating the behaviour, i.e., by causing the terminal condition ${}^{c}f^{\tau}_{j,m}$ to be satisfied,
- By selecting a new behaviour, i.e., by causing an adequate initial condition ${}^{c}f^{\sigma}_{j,v,m,m'}$ to become true.

All these ways of influencing the activities of an agent result from the assumed model of an embodied agent. However, the designer of the system can also exert influence on the structure of the FSM ${}^{c}\mathcal{F}_{j}$. Some of its states can be distinguished as communication states (a single such state is quite common), in which the behaviour of the agent waits for a message. Once a message is obtained, the FSM reacts to it by directing the execution to an appropriate section of the FSM, which is responsible for the execution of a set of behaviours that compose a service—a service-oriented architecture (SOA) results. At each of the four mentioned communication levels, FIPA messages can be utilised.

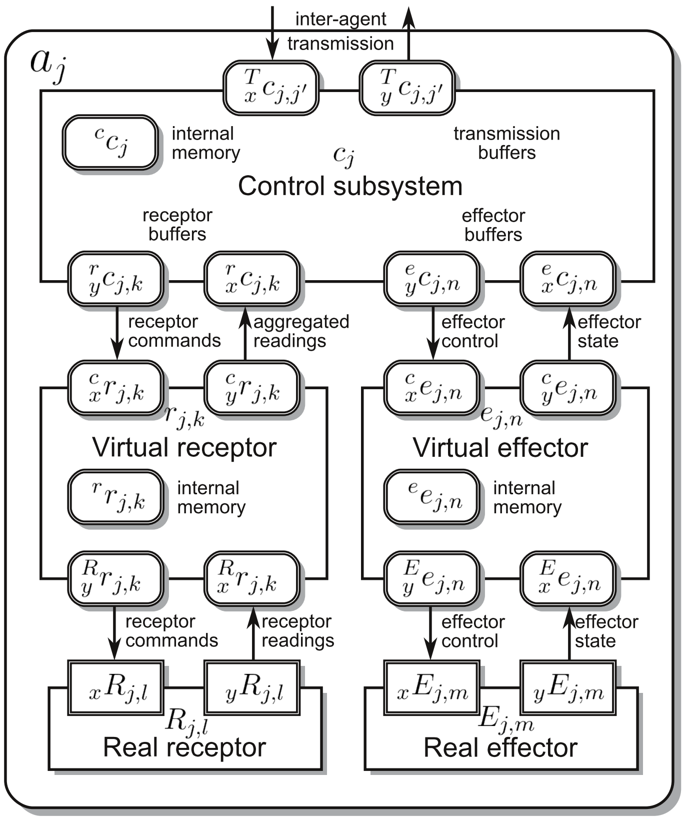


Figure 1. Internal structure of an agent a_j.

It should be noted that this propagation of the received message through the four levels of the agent execution structure is beneficial from the point of view of the agent's reactivity to outside stimuli, i.e., incoming messages. Each message can be accessed at the control subsystem sampling rate. If the currently executed service has to be interrupted, e.g., due to the request for a higher priority service, this subsystem, and hence the agent, will detect this through the evaluation of the terminal condition associated with the currently executed behaviour. The satisfaction of the terminal condition will cause the current behaviour to be aborted, and that in turn will initiate the evaluation of the initial condition, what will result in switching the FSM state, e.g., the state responsible for invoking the newly requested service. Such an instantaneous switch from one service to the other might not be possible, e.g., if the current service consists in the delivery of a cup of tea to the user, this cup cannot be simply dropped to start the execution of the higher priority service. In this case the FSM will need to pass through an intermediate state, in which it will execute a behaviour of placing the cup in a safe location, and only then proceeding to the execution of a new service.

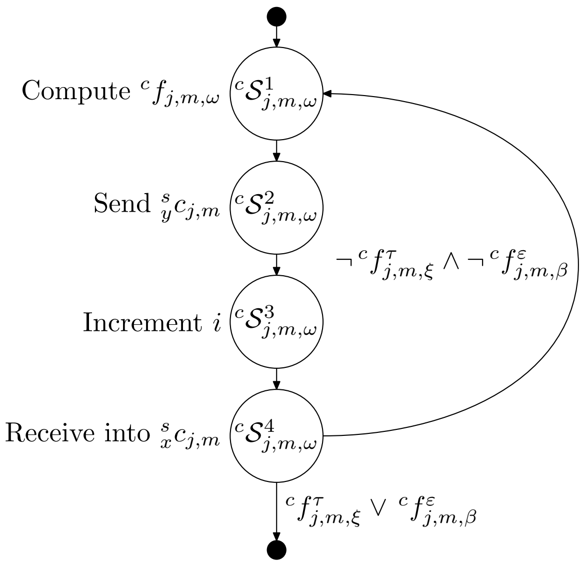

Compute ${}^{c}f_{j,m,\omega}$
${}^{c}\mathcal{S}^{1}_{j,m,\omega}$
Send ${}^{s}_{y}c_{j,m}$
${}^{c}\mathcal{S}^{2}_{j,m,\omega}$
$\neg\,{}^{c}f^{\tau}_{j,m,\xi} \wedge \neg\,{}^{c}f^{\varepsilon}_{j,m,\beta}$
Increment i
${}^{c}\mathcal{S}^{3}_{j,m,\omega}$
Receive into ${}^{s}_{x}c_{j,m}$
${}^{c}\mathcal{S}^{4}_{j,m,\omega}$
${}^{c}f^{\tau}_{j,m,\xi} \vee {}^{c}f^{\varepsilon}_{j,m,\beta}$

Figure 2. The c_j FSM ${}^{c}\mathcal{F}_j$ executing any behaviour.

It is worth noticing that in the case of complex robotic systems, e.g., cognitive robots, their control systems contain many CT type agents [28,61], i.e., computational agents, having the capability of communicating with other agents, but lacking effectors and receptors. In this case (1) becomes:

$$\begin{cases} {}^{c}c^{i+1}_{j} & := & {}^{c,c}f_{j,m}({}^{c}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \\ {}^{T}_{y}c^{i+1}_{j} & := & {}^{c,T}f_{j,m}({}^{c}c^{i}_{j}, {}^{T}_{x}c^{i}_{j}) \end{cases} \tag{2}$$

In other words , the activities of the FSM of such an agent (its control subsystem, to be precise) mainly rely on the inter-agent communication, i.e., the contents of the messages transferred between such agents. Thus, ${}^{c,T}f_{j,m}$ is of paramount importance.

3.3. Structure of a Communicate

To take into account all the necessary possibilities of message contents, we propose that the contents of the output and input buffers—a communicate—be set in accordance with the IEEE FIPA standards. The partial transition function ${}^{c,T}f_{j,m}$ should produce the content of the output buffer ${}^{T}_{y}c_j$ as a message $\mathcal{M}^{i}_{j,j'}$ (where j and j' are the designators of the communicating agents) and accept the content of the input buffer ${}^{T}_{x}c_j$ in the format of $\mathcal{M}^{i}_{j',j}$ (see Figure 3). A communicate transmitted from agent a_j to agent $a_{j'}$ at instant i is a tuple:

$$\mathcal{M}^{i}_{j,j'} = \langle \mathcal{V}, \mathcal{Z}, \mathcal{L}, \mathcal{O}, \mathcal{I}, t_b, m_w, m_t, m_c \rangle \tag{3}$$

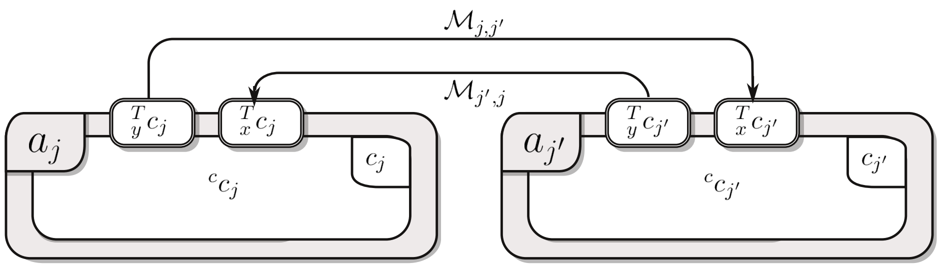


Figure 3. Messages $\mathcal{M}_{j,j'}$ and $\mathcal{M}_{j',j}$ transmitted between a_j and $a_{j'}$.

The tuple $\mathcal{M}^i_{j,j'}$, according to FIPA ACL Message Structure Specification standard [62], contains the following items:

- Performative verb $\mathcal{V} \in \hat{\mathcal{V}}$, where $\hat{\mathcal{V}}$ is a set of possible performative verbs defined by [62]: $\hat{\mathcal{V}} = \{$accept-proposal, agree, cancel, cfp, confirm, disconfirm, failure, inform, inform-if, inform-ref, not-understood, propagate, propose, proxy, query-if, query-ref, refuse, reject-proposal, request, request-when, request-whenever, subscribe$\}$. It defines the intention of sending the communicate (request, inquiry, statement, etc.), and how the recipient of the message should interpret it.
- Content $\mathcal{Z}$ of the communicate. If the language $\mathcal{L}$ is provided, the content should conform to the syntax and semantics of that language.
- Name of language $\mathcal{L} \in \hat{\mathcal{L}}$ used to formulate the content, where $\hat{\mathcal{L}}$ is a set of languages used for the formulation of contents of the communicates. Proper indication of the language expressing the content of each message helps the sender of the message to correctly formulate the content of the message, and the recipient to correctly understand this content. The FIPA standard does not require a language to be defined (no syntax and semantics are required). Different languages can be assigned to different communicative acts between pairs of agents.
- Name of the ontology $\mathcal{O} \in \hat{\mathcal{O}}$, where $\hat{\mathcal{O}}$ is a set of ontologies. An ontology defines the meaning and relations between symbols used in the message content. An ontology is needed when it is not enough to define the language and performative verb to understand the message content correctly, and also when the agent does not have the common knowledge required for its operation. The ontology determines for the agent how to understand the message in the context of knowledge that the ontology specifies.
- Name of the interaction protocol $\mathcal{I} \in \hat{\mathcal{I}}$, where $\hat{\mathcal{I}}$ is a set of possible interaction protocols defined by [62]: $\hat{\mathcal{I}} = \{$query, request, subscribe, request-when, cfp, icfp, propose, ...$\}$. It defines the pattern of interaction governing the conversation. The protocol organizes the exchange of messages and determines their order. Conversation between agents is usually bi-directional, and the goal is not only to pass an information item, but also to request and obtain a piece of information, to query the other agent, subscribe to a specific piece of information, etc. Thus, pairs of input and output buffers are considered $({}^{T}_{x}c_{j}, {}^{T}_{y}c_{j})$; and for the sake of brevity, the pair is denoted as ${}^{T}_{x,y}c_{j}$.
- Reply-by $t_b \in \hat{\mathcal{T}}$, is a timestamp indicating the latest time by which the reply to this communicate should be produced.
- Reply-with identifier $m_w \in \hat{M}$ and in-reply-to identifier $m_t \in \hat{M}$. The pair of identifiers is used to combine messages into short-term conversations. If the sender of the message wants the recipient's reply to match the message it sent, it should set the m_w identifier value. In turn, the recipient who wants to refer to the message with the m_w identifier should set this value in m_t.
- Conversation identifier $m_c \in \hat{M}$ identifies the ongoing sequence of communicative acts that together form a conversation.

The figure $\hat{\mathcal{T}}$ is a set of timestamps, and $\hat{M}$ is a set of identifiers of communicates sent. While $\hat{\mathcal{M}}$ is a set of all communicates. The symbol $f^M : \hat{\mathcal{M}} \to \hat{M}$ is a function mapping a message to its identifier. The set of all messages that can be sent by the agent a_j is denoted as ${}^{a}_{y}\hat{\mathcal{M}}_j$. The set of all messages that can be received by the agent a_j is denoted as ${}^{a}_{x}\hat{\mathcal{M}}_j$.

4. Robotic System Design Method

The design method assumes that the designer begins with a general robotic system characterized by a set of generic tasks for the robot. Next, he/she specifies the tree of requirements, making the requirements ever more detailed and thus expanding the tree. Based on that, the robotic system is divided into smaller parts (agent groups), and as a result, the communication channels appear. The decomposition is stopped when the requirements are fully addressed, and the resulting agent groups are within the mental grasp of the designer, so that they can be substituted by agents. During decomposition, adequate specification of communication has to be produced. Finally, the specification of agents' interfaces is done. The top-down approach based on system decomposition implies that system properties follow directly from the requirements. This is in contrast to the bottom-up compositional approach, where system properties emerge during the design process. Detailed specification of communication is completed when the agents substitute the groups. The process is depicted in Figure 4 and consists of the following steps.

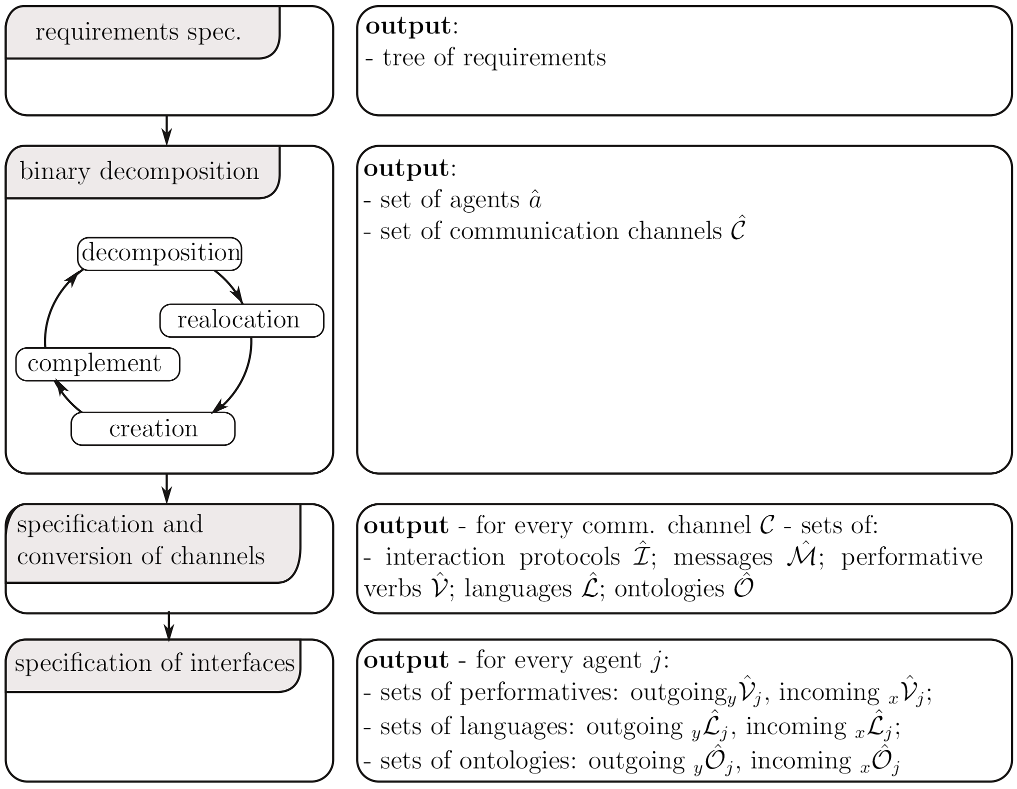

requirements spec.
output:
- tree of requirements
binary decomposition
decomposition
realocation
creation
complement
output:
- set of agents $\hat{a}$
- set of communication channels $\hat{\mathcal{C}}$
specification and conversion of channels
output - for every comm. channel $\mathcal{C}$ - sets of:
- interaction protocols $\hat{\mathcal{I}}$; messages $\hat{\mathcal{M}}$; performative verbs $\hat{\mathcal{V}}$; languages $\hat{\mathcal{L}}$; ontologies $\hat{\mathcal{O}}$
specification of interfaces
output - for every agent j:
- sets of performatives: outgoing$_y\hat{\mathcal{V}}_j$, incoming $_x\hat{\mathcal{V}}_j$;
- sets of languages: outgoing $_y\hat{\mathcal{L}}_j$, incoming $_x\hat{\mathcal{L}}_j$;
- sets of ontologies: outgoing $_y\hat{\mathcal{O}}_j$, incoming $_x\hat{\mathcal{O}}_j$

Figure 4. Binary decomposition process.

1. **Specification of system requirements.** The designer formulates a set of tasks that the robot should perform. On this basis, a set of requirements is formulated for the robotic system. Each general requirement is specified as a set of more detailed requirements; thus, a hierarchy of requirements is formulated. It can be represented as a requirement tree. This tree will serve as a basis for system decomposition.
   The output of this step is a hierarchy of requirements, specified as a tree.
2. **Binary decomposition of a robotic system.** The designer divides the system into smaller elements, assuring proper assignment of tasks (requirements) to each of them. Each decomposition step has to produce elements that have clearly defined and coherent tasks. The elements that are the result of decomposition are groups of agents ${}^{a}g_q$. This step considers an iterative process consisting of the following stages concerning the analysis of successive requirements from the tree in the top-down order.
   (a) **Decomposition of a group.** One group of agents is divided into two groups, thus the name—binary decomposition. When ${}^{a}g_q$ is decomposed into ${}^{a}g_p$ and ${}^{a}g_r$, the following notation is used: ${}^{a}g_q := \langle {}^{a}g_p, {}^{a}g_r \rangle$.
   (b) **Creation of new channels.** The need for creation of one to N new communication channels might arise. In this case, it is necessary to correctly determine which component needs an action to be realized (e.g., answer in the case of query interaction protocol, command execution in the case of request interaction protocol, new information in the case of subscribe interaction protocol).

(c) **Reallocation of existing channels.** After each decomposition step, the designer must decide on the proper allocation of the existing communication channels, associated with the decomposed group, to the newly created groups. When group ${}^{a}g_{q}$ is decomposed into ${}^{a}g_{p}$ and ${}^{a}g_{r}$, and $\mathcal{C}^{k}_{q,q'}$ is reallocated to ${}^{a}g_{\mathrm{p}}$, the following notation is used: $\mathcal{C}^{k}_{q,q'} \mapsto \mathcal{C}^{k}_{p,q'}$. When both decomposed groups ${}^{a}g_{\mathrm{p}}$ and ${}^{a}g_{\mathrm{r}}$ need to communicate with the non-decomposed group ${}^{a}g_{\mathrm{q}'}$ the channel is split into two: $\mathcal{C}^{k}_{q,q'} \mapsto \mathcal{C}^{m}_{p,q'}, \mathcal{C}^{n}_{r,q'}$.

(d) **Complementing communication channels.** Due to the analysis of new requirements, there may be a need to supplement the existing communication channels, without need for creating new ones.

Stop condition. The decomposition process stops when all of the requirements are processed. When the designer concludes that the agent groups have clearly defined and coherent tasks that are comprehensively understood by the programmer he/she should substitute each of the existing agent groups by single agents. Otherwise, the analysis of the requirements has to be repeated, making them more detailed.

The output of binary decomposition are sets of agents $\hat{a}$, and of communication channels $\hat{\mathcal{C}}$. Any group of agents can be composed of a single agent. Specific agents have strictly assigned functionalities.

3. **Specification and conversion of the communication channels.** The next step is to specify the communication exchanged between the pairs of agents a_j and $a_{j'}$, substituting groups ${}^{a}g_{\mathrm{q}}$ and ${}^{a}g_{\mathrm{q}'}$, between which the communication channel $\mathcal{C}^{k}_{\mathrm{q,q'}}$ was created initially. There is a need to properly convert the communication channel into the specification of input and output buffers, and to define the patterns of communication between the agents a_j and $a_{j'}$. This results in the creation of four buffers for agents a_j and $a_{j'}$: two input buffers: ${}^{T}_{x}c_{j}$, ${}^{T}_{x}c_{j'}$ and two output buffers: ${}^{T}_{y}c_{j}$, ${}^{T}_{y}c_{j'}$. Buffers ${}^{T}_{y}c_{j}$ and ${}^{T}_{y}c_{j'}$ are used to send, while ${}^{T}_{x}c_{j}$ and ${}^{T}_{x}c_{j'}$ to receive the communicates. Those communicates are specified by a number of interaction protocols. The interaction protocol $\mathcal{I}^{p}_{\mathrm{j,j'}}$ (or $\mathcal{I}^{p}_{\mathrm{j',j}}$—depending on who initiates the communication) specifies the pattern of communication between the agents. A conversation is an exchange of messages within a specific communication channel. A number of interaction protocols can be associated with one communication channel.

The definition of the interaction protocol determines the performative verbs for the specific messages. Moreover, every message contains the communication language, and possibly the ontology used.

While specifying new communication channels, an analysis targeted at communication deadlocks is mandatory. In our approach the analysis is focused on the newly created channels, therefore, it is easier to conduct. This article does not delve into deadlock analysis, as it focuses on the problem of system decomposition. However, in [31], communication analysis of Embodied Agent-based systems is provided, and in [59], a Petri-Net-based description of Embodied Agents is presented. The formal specification and verification of FIPA interaction protocols by means of Colored Petri Nets is proposed in [63]. Therefore, the system designers utilising this approach can check deadlocks with the known tools for Petri-Net deadlock analysis, and the proposed decomposition method enables concentration of the analysis on a part of a complex system, making the analysis simpler.

The output of this step is the specification of each of the defined communication channels from the set $\hat{\mathcal{C}}$. Each channel $\mathcal{C}^{k}_{\mathrm{j,j'}}$ specifies the sets of interaction protocols $\hat{\mathcal{I}}_{j,j'}$. Each interaction protocol $\mathcal{I}^{p}_{\mathrm{j,j'}} \in \hat{\mathcal{I}}_{j,j'}$ defines a set of messages that are sent from the agent a_j to the agent $a_{j'}$: $\hat{\mathcal{M}}_{j,j'}$, and back from the agent $a_{j'}$ to the agent a_j: $\hat{\mathcal{M}}_{j',j}$. The messages should contain the performative verb $\mathcal{V}$, interaction protocol $\mathcal{I}$, and optionally the language $\mathcal{L}$ and ontology $\mathcal{O}$.

4. **Specification of agent interfaces.** The previous steps concentrated on the communication channels, while the current focuses on the agents (or groups of them). This step specifies the following elements: sets of performative verbs ${}_y\hat{\mathcal{V}}_j$ and ${}_x\hat{\mathcal{V}}_j$ that agents send and receive, respectively, in the communicates, and the sets of languages ${}_y\hat{\mathcal{L}}_j$ and ${}_x\hat{\mathcal{L}}_j$ used in those communicates, as well as sets of ontologies ${}_y\hat{\mathcal{O}}_j$ and ${}_x\hat{\mathcal{O}}_j$ employed by those communicates. The specification of the interfaces organizes the project, and enables the detection of inconsistencies. Moreover, it helps to organize the handling of errors that occur when an unsupported message is received. However, this is not discussed in this article.

The design process should be iterative as, after the initial system design, it may be necessary to relapse and re-run through it several times.

5. Illustrative Example: Robot Companion

The herein presented illustrative example shows how to apply the proposed design method. It is purposefully kept simple to let the reader concentrate on the method, however the resulting system is of utility. The purpose of a robot companion is to assist users (usually elderly) at their homes. A survey of robot tasks that the elderly person might request is presented in [64]. A robot companion, as pointed out in [65,66], should exhibit the following capabilities: (1) moving around in the user's home; (2) being safe for humans, household appliances and furniture; (3) being aware of the dynamic environment; (4) having a simple and intuitive human-robot interface (e.g., voice, display, gestures, tactile); (5) possessing manipulation capability; (6) having a friendly look; (7) having its own task management. A robot companion should be versatile and address as many requests of the elderly as possible. However, there are still technology limitations that force robot specialization. The mentioned range of tasks for a companion robot is wide, therefore an analysis of the requirements set for a specific robot should be carried out and a selection has to be made.

5.1. Formulation of Requirements

For brevity, we shall focus on the Robot Companion helping in a kitchen. The above general requirements should be analysed from the point of view of the chosen task. The most general requirement for the Robot Companion is that it should be able to acquire and subsequently execute human commands. Next, the robot should decide upon the necessary activity, i.e., interpret the command. To be able to perform the activity, there is a need for creating and managing a plan and afterwards executing it. As the Robot Companion interacts with an elderly person, who can change his/her mind, it is crucial for it to be able to interrupt the action currently performed and schedule the next one. To plan in a kitchen environment, the Robot Companion must be able to model this environment. The environment in which it works is dynamic, so in order to correctly plan its activities, the Robot Companion should deduce what activities it should perform. Moreover, it must identify and understand relationships (including spatial ones) between objects, both existing currently and those that will result from the execution of the plan. To execute the plan, it has to be able to sense and influence the environment using appropriate sensors and actuators. The environment is dynamic, because the robot and other agents change the positions and orientations of objects, thus, the Robot Companion should be able to update the environment model. The hierarchy of proposed requirements is presented in Figure 5.

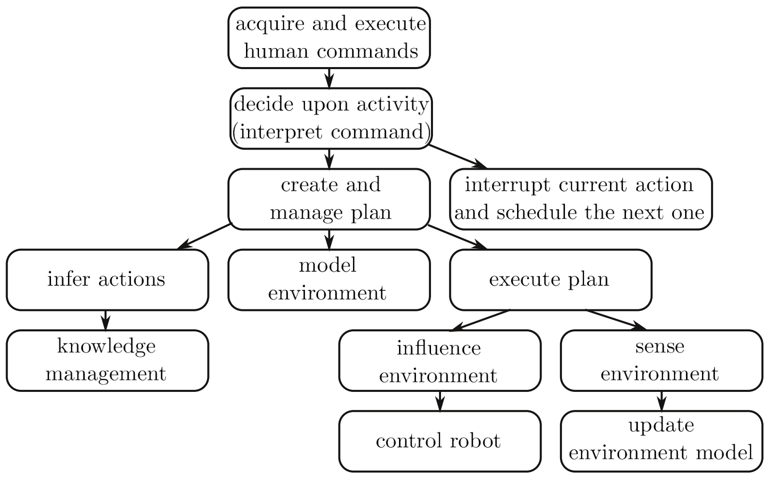


Figure 5. Requirement hierarchy for the Robot Companion.

5.2. Decomposition of the Robot Companion

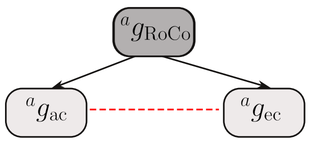
The Robot Companion can be treated as a monolithic system represented as a group of agents ${}^{a}g_{\mathrm{RoCo}}$. For brevity, only two decomposition steps (the first and fourth) are described in detail, the remaining are only outlined.

The first decomposition is due to the requirement at the root of the tree (see Figure 5) that says that a robotic system should acquire and execute human commands. The division ${}^{a}g_{\mathrm{RoCo}} := \langle {}^{a}g_{\mathrm{ac}}, {}^{a}g_{\mathrm{ec}} \rangle$ results in two agent groups:

- ${}^{a}g_{\mathrm{ac}}$ (ac for acquire commands)—group responsible for acquiring the commands from a human
- ${}^{a}g_{\mathrm{ec}}$ (ec for execute commands)—group responsible for execution of the commands.

These two groups communicate by sending commands and responses. In general, communication channels $\mathcal{C}^{k}_{j,j'}$ created in the successive steps of the decomposition process are marked by consecutive integers k. Those numbers remain unchanged during the design process. Moreover, those channels are also marked with the symbols of the groups, however, those will evolve. The communication channel $\mathcal{C}^{1}_{\mathrm{ac,ec}}$ is created between groups ${}^{a}g_{\mathrm{ac}}$ and ${}^{a}g_{\mathrm{ec}}$ (Table 1).

Table 1. Summary of the first step of ${}^{a}g_{\mathrm{RoCo}}$ decomposition process. The arrows show which groups are the result of the decomposition of the initial group. Dashed lines show emerging communication channels. The light coloured groups form the decomposition tree fringe, which shows which groups exist at the current step—the dark ones disappear, because they were decomposed.

Requirements	acquire and execute human commands
Decomposition	${}^{a}g_{\mathrm{RoCo}} := \langle {}^{a}g_{\mathrm{ac}}, {}^{a}g_{\mathrm{ec}} \rangle$
${}^{a}g_{\mathrm{ac}}$ tasks	acquiring commands from a human
${}^{a}g_{\mathrm{ec}}$ tasks	executing commands
Channel reallocation	—
New channels	$\mathcal{C}^{1}_{\mathrm{ac,ec}}$
$\mathcal{C}^{1}_{\mathrm{ac,ec}}$ transmits	commands to be executed
	

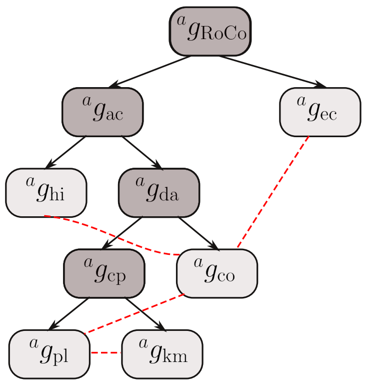
The second division results from the requirement that the robotic system interprets the commands obtained from a human, and decides about the command to be executed first. The group ${}^{a}g_{\mathrm{ac}}$ is decomposed into: ${}^{a}g_{\mathrm{hi}}$ (human-machine interface), and ${}^{a}g_{\mathrm{da}}$ (decides upon command to be executed). It results in reallocation of the communication channel $\mathcal{C}^1_{\mathrm{ac,ec}} \mapsto \mathcal{C}^1_{\mathrm{da,ec}}$, and creation of the new one: $\mathcal{C}^2_{\mathrm{ic,hi}}$, that is used to pass interpreted human commands.

The third division results from the requirements that demand the creation and management of a plan, and allow the interruption of the current action and scheduling the next one. The group ${}^{a}g_{\mathrm{da}}$ is divided into ${}^{a}g_{\mathrm{co}}$ (coordinates command execution and scheduling), and ${}^{a}g_{\mathrm{cp}}$ (creates and manages a plan). Two channels are reallocated: $\mathcal{C}^1_{\mathrm{da,ec}} \mapsto \mathcal{C}^1_{\mathrm{co,ec}}$, $\mathcal{C}^2_{\mathrm{da,hi}} \mapsto \mathcal{C}^2_{\mathrm{co,hi}}$. Also, a new one is created: $\mathcal{C}^3_{\mathrm{co,cp}}$, that is used to request for plan generation.

In the fourth step the designer takes into account that the Robot Companion should infer actions, model the environment and manage the knowledge (see Figure 5). Inference of actions requires a component responsible for creating abstract plans: ${}^{a}g_{\mathrm{pl}}$ (pl for planner). In turn, the group responsible for knowledge management will provide knowledge about the objects and relationships between them: ${}^{a}g_{\mathrm{km}}$ (km for knowledge manager). Thus, the group ${}^{a}g_{\mathrm{cp}}$ is split into two: ${}^{a}g_{\mathrm{cp}} := \langle {}^{a}g_{\mathrm{pl}}, {}^{a}g_{\mathrm{km}} \rangle$ (see Table 2). There is a need to reallocate the channel $\mathcal{C}^3_{\mathrm{co,cp}}$ and it is obvious that it will be handled by the group responsible for creating the plan: $\mathcal{C}^3_{\mathrm{co,cp}} \mapsto \mathcal{C}^3_{\mathrm{co,pl}}$. A new channel $\mathcal{C}^4_{\mathrm{pl,km}}$ must be created between the new groups to enable the transfer of knowledge extracted from the knowledge base.

Table 2. Summary of the fourth step—${}^{a}g_{\mathrm{cp}}$ decomposition.

Requirements	infer actions, model environment, knowledge management
Decomposition	${}^{a}g_{\mathrm{cp}} := \langle {}^{a}g_{\mathrm{pl}}, {}^{a}g_{\mathrm{km}} \rangle$
${}^{a}g_{\mathrm{pl}}$ tasks	planning
${}^{a}g_{\mathrm{km}}$ tasks	knowledge management
Chann. reallocation	$\mathcal{C}^3_{\mathrm{co,cp}} \mapsto \mathcal{C}^3_{\mathrm{co,pl}}$
New channels	$\mathcal{C}^4_{\mathrm{pl,km}}$
$\mathcal{C}^1_{\mathrm{co,ec}}$ transmits	commands to be executed
$\mathcal{C}^2_{\mathrm{co,hi}}$ transmits	interpreted human commands
$\mathcal{C}^3_{\mathrm{co,pl}}$ transmits	generated plan
$\mathcal{C}^4_{\mathrm{pl,km}}$ transmits	knowledge needed by planner



In the fifth step, we take into account that the robotic system must execute the plan, influence and sense the environment by controlling the robot. Thus, ${}^{a}g_{\mathrm{ec}}$ is decomposed into: ${}^{a}g_{\mathrm{ex}}$ (responsible for influencing and sensing the environment by operating the hardware) and ${}^{a}g_{\mathrm{ta}}$ (responsible for executing the plan). $\mathcal{C}^1_{\mathrm{co,ec}}$ is reallocated: $\mathcal{C}^1_{\mathrm{co,ec}} \mapsto \mathcal{C}^1_{\mathrm{co,ta}}$. Two communication channels are created: $\mathcal{C}^5_{\mathrm{ex,ta}}$, where the low-level commands for the hardware, i.e., acquiring sensor readings and controlling the actuators, are transmitted, and $\mathcal{C}^6_{\mathrm{ta,km}}$ passing the knowledge needed by a_{ta}.

The last step takes into account that the robotic system should update the environment model. As there already exists a group responsible for managing the environment model (${}^{a}g_{\mathrm{km}}$), and sensor data are provided by ${}^{a}g_{\mathrm{ex}}$, and a communication channel $\mathcal{C}^{6}_{\mathrm{ta,km}}$ exists already; thus, only the content transmitted by this channel has to be supplemented.

After exhausting the requirement tree, the decomposition process is complete. Six groups of agents emerged (see Figure 6). Again for brevity the description of decomposition of ${}^{a}g_{\mathrm{ex}}$, that controls the robot hardware, has been omitted here. Hardware configuration is described below. Five groups (${}^{a}g_{\mathrm{ta}}$, ${}^{a}g_{\mathrm{pl}}$, ${}^{a}g_{\mathrm{km}}$, ${}^{a}g_{\mathrm{co}}$, ${}^{a}g_{\mathrm{hi}}$) are transformed into respective agents: $\hat{a} = \{a_{\mathrm{ta}}, a_{\mathrm{pl}}, a_{\mathrm{km}}, a_{\mathrm{co}}, a_{\mathrm{hi}}\}$. Those agents have clearly defined and coherent tasks. a_{pl} generates a plan based on problem definition; a_{km} manages the knowledge database; a_{co} coordinates a set of commands received from a human, orders plans from a_{pl}, and schedules their execution; a_{ta} executes the plan and dispatches low-level instructions to ${}^{a}g_{\mathrm{ex}}$; a_{hi} acts as a human-machine interface. Further decomposition would result in too much fragmentation and too much communication effort. Moreover, during decomposition, six communication channels were identified: $\hat{\mathcal{C}} = \{\mathcal{C}^{1}_{\mathrm{co,ta}}, \mathcal{C}^{2}_{\mathrm{co,hi}}, \mathcal{C}^{3}_{\mathrm{co,pl}}, \mathcal{C}^{4}_{\mathrm{pl,km}}, \mathcal{C}^{5}_{\mathrm{ex,ta}}, \mathcal{C}^{6}_{\mathrm{ta,km}}\}$.

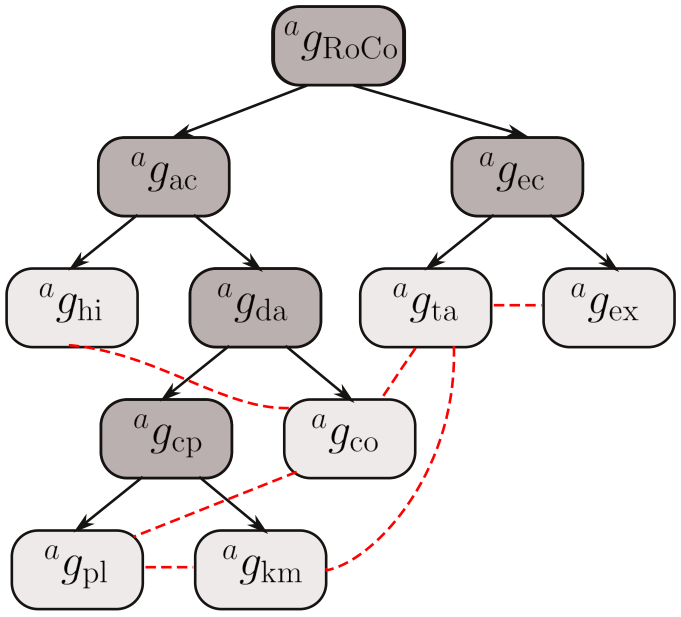


Figure 6. The decomposition after six steps of the process.

5.3. Hardware Configuration

The Velma robot (Figure 7) has adequate capabilities to satisfy ${}^{a}g_{\mathrm{ex}}$ requirements. Thus, it became the hardware part of the Robot Companion [67]. It is the torso of a humanoid robot with two KUKA LWR arms and additional F/T sensors embedded in the wrists to which three-fingered BarrettHand grippers are attached. The torso is mounted on a 1 DOF revolute column which also supports a 2 DOF neck equipped with cameras and Kinect sensor (the head). Arms are torque controlled, employing impedance control [67,68]. The control laws offer two basic modes of operation: Cartesian and configuration space [69]. The neck and grippers are position-controlled. The control system is implemented using the FABRIC programming framework [70].

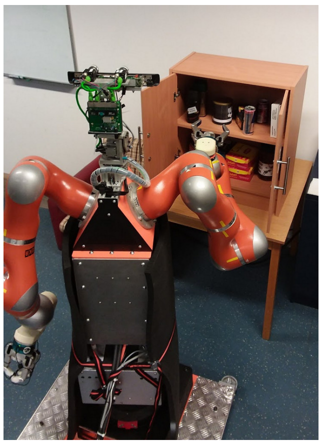


Figure 7. Velma robot operating in the environment.

5.4. Specification of Communication Channels

At this stage of design, the messages sent between pairs of agents need to be refined, leading to the specification of the interaction protocols. Each communication channel should have a specified set of interaction protocols according to which the conversations take place. Moreover, every interaction protocol should contain: performative verbs, languages that specify the content of the messages, and possibly the ontologies. Due to the limited space, only two communication channels are described. However, the creation of the remaining follows the same pattern.

The communication channel $\mathcal{C}^1_{\mathrm{co,ta}}$ is used to send plans to be executed. The first message deals with requesting from a_{ta} to execute the plan. The plan should be understandable to the agents; therefore, the task notation using HFSM (HFSM for Hierarchical Finite State Machine) is chosen. As a_{co} requests from a_{ta}, the performative verb is $\mathcal{V}_{\mathrm{request}}$, and the content $\mathcal{Z}$ is expressed using $\mathcal{L}_{\mathrm{HFSM}}$. $\mathcal{L}_{\mathrm{HFSM}}$ is used for requesting the execution of the generated plan; thus, a typical message is up to dozens of kB.

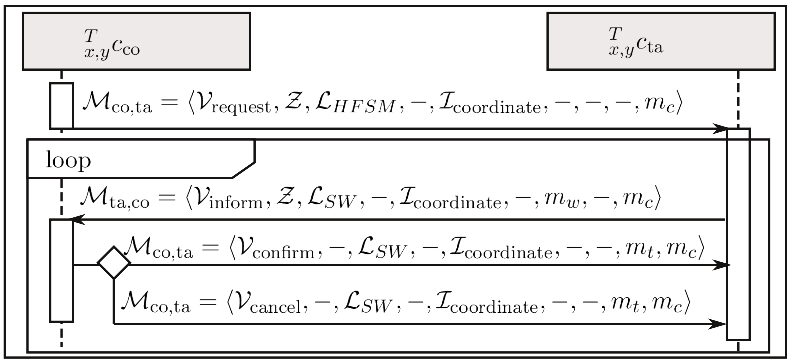
While executing the plan, a_{ta} sends its status (name of its current state). Together with the performative verb $\mathcal{V}_{\mathrm{inform}}$ it sends $\mathcal{Z}$ from a specified set of states: $\mathcal{Z}$ ∈{Idle, Run, Error, Terminating, Finished, Failed, Replan}. The language $\mathcal{L}_{\mathrm{SW}}$ (SW for Supervised Worker) was proposed to organize the list, thus, $\mathcal{Z}$ should be formulated using $\mathcal{L}_{\mathrm{SW}}$. The detailed description of $\mathcal{Z}$ is as follows. **Idle**—awaiting for the next plan. **Running**—executing a plan. **Error**—the plan execution ended with an error. **Terminating**—the plan execution is being terminated. **Succeeded**—the plan is finished with success. **Preempted**—the plan is preempted due to an interruption from a_{co}. **Replan**—the plan cannot be further executed and a new plan is required.

The agent a_{co} confirms each message it receives from a_{ta} by a message with the performative verb $\mathcal{V}_{\mathrm{confirm}}$ and empty content, as the intention provides complete information.

In the case when the plan has to be interrupted, a_{co} sends an interrupting message to a_{ta}. The performative verb is $\mathcal{V}_{\mathrm{cancel}}$, and again sending any content is not necessary. The messages sent with $\mathcal{L}_{\mathrm{SW}}$ have no content, so their size is 100 to 200 bytes, and are sent every time a_{ta} modifies its internal state. The detailed specification is presented as follows.

$$\begin{aligned} \mathcal{V} \in \{confirm, cancel\} \quad &\Rightarrow \quad (\mathcal{L} = \mathcal{L}_{\mathrm{SW}}) \wedge (\mathcal{Z} = -) \\ \mathcal{V} \in \{request\} \quad &\Rightarrow \quad (\mathcal{L} = \mathcal{L}_{\mathrm{HFSM}}) \wedge (\mathcal{Z} = plan) \end{aligned} \tag{4}$$

These four messages make up the interaction protocol $\mathcal{I}^{\mathrm{coordinate}}_{\mathrm{co,ta}}$. The interaction protocol managing the conversation between a_{co} and a_{ta}, together with the detailed formulation of the messages is shown in Figure 8.

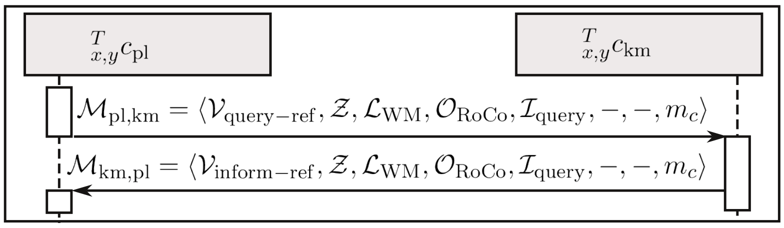


${}^{T}_{x,y}c_{\mathrm{co}}$
${}^{T}_{x,y}c_{\mathrm{ta}}$
$\mathcal{M}_{\mathrm{co,ta}} = \langle \mathcal{V}_{\mathrm{request}}, \mathcal{Z}, \mathcal{L}_{HFSM}, -, \mathcal{I}_{\mathrm{coordinate}}, -, -, -, m_c \rangle$
loop
$\mathcal{M}_{\mathrm{ta,co}} = \langle \mathcal{V}_{\mathrm{inform}}, \mathcal{Z}, \mathcal{L}_{SW}, -, \mathcal{I}_{\mathrm{coordinate}}, -, m_w, -, m_c \rangle$
$\mathcal{M}_{\mathrm{co,ta}} = \langle \mathcal{V}_{\mathrm{confirm}}, -, \mathcal{L}_{SW}, -, \mathcal{I}_{\mathrm{coordinate}}, -, -, m_t, m_c \rangle$
$\mathcal{M}_{\mathrm{co,ta}} = \langle \mathcal{V}_{\mathrm{cancel}}, -, \mathcal{L}_{SW}, -, \mathcal{I}_{\mathrm{coordinate}}, -, -, m_t, m_c \rangle$

Figure 8. $\mathcal{I}^{\mathrm{coordinate}}_{\mathrm{co,ta}}$ interaction protocol. The initiator agent a_{co}, that contains the pair of buffers ${}^{T}_{x,y}c_{\mathrm{co}}$, starts the interaction, while the participating agent a_{ta}, that contains the pair of buffers ${}^{T}_{x,y}c_{\mathrm{ta}}$ takes part in it. The dashed lines are lifelines, arrows represent particular communicates. A diamond indicates an alternative in messaging. The sub-diagram 'loop' means the repetition of its contents.

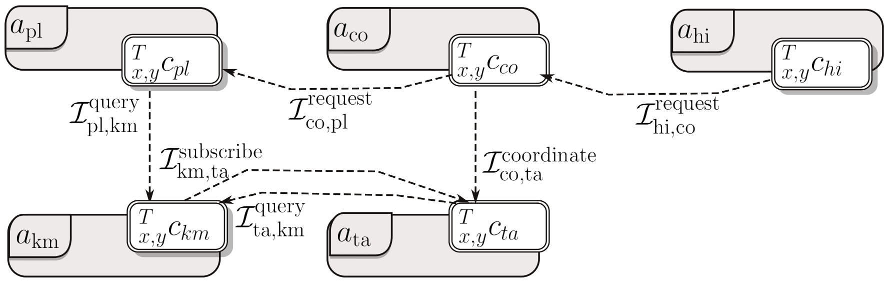
The communication channel $\mathcal{C}^{4}_{\mathrm{pl,km}}$ is used to transmit knowledge, in the form of queries produced by a_{pl}, and answers formulated and sent by a_{km}. For both agents, to understand the queries and knowledge, the language $\mathcal{L}_{\mathrm{WM}}$ (WM for World Model) is proposed. The agent initiating the conversation is a_{pl}, asking for knowledge using $\mathcal{Z}$ expressed using $\mathcal{L}_{\mathrm{WM}}$. The ontology $\mathcal{O}_{\mathrm{RoCo}}$ specifies the knowledge about the robot's environment. It is built from scratch in OWL to keep it as simple as possible. The ontology was inspired by the KnowRob2 project [71]. It was used to implement a simple knowledge base. The performative verb for the message is $\mathcal{V}_{\mathrm{query-ref}}$. Agent a_{km} formulates the answer to the query, and informs the inquirer; thus, the performative verb is $\mathcal{V}_{\mathrm{inform-ref}}$.

The interaction protocol $\mathcal{I}^{\mathrm{query}}_{\mathrm{pl,km}}$, together with the detailed formulation of the messages is presented in Figure 9. After designing the six communication channels, the system shown in Figure 10 results.



${}^{T}_{x,y}c_{\mathrm{pl}}$
${}^{T}_{x,y}c_{\mathrm{km}}$
$\mathcal{M}_{\mathrm{pl,km}} = \langle \mathcal{V}_{\mathrm{query-ref}}, \mathcal{Z}, \mathcal{L}_{\mathrm{WM}}, \mathcal{O}_{\mathrm{RoCo}}, \mathcal{I}_{\mathrm{query}}, -, -, m_c \rangle$
$\mathcal{M}_{\mathrm{km,pl}} = \langle \mathcal{V}_{\mathrm{inform-ref}}, \mathcal{Z}, \mathcal{L}_{\mathrm{WM}}, \mathcal{O}_{\mathrm{RoCo}}, \mathcal{I}_{\mathrm{query}}, -, -, m_c \rangle$

Figure 9. Interaction protocol $\mathcal{I}^{\mathrm{query}}_{\mathrm{pl,km}}$.



a_{pl}
${}^{T}_{x,y}c_{pl}$
a_{co}
${}^{T}_{x,y}c_{co}$
a_{hi}
${}^{T}_{x,y}c_{hi}$
$\mathcal{I}^{\mathrm{query}}_{\mathrm{pl,km}}$
$\mathcal{I}^{\mathrm{request}}_{\mathrm{co,pl}}$
$\mathcal{I}^{\mathrm{request}}_{\mathrm{hi,co}}$
$\mathcal{I}^{\mathrm{subscribe}}_{\mathrm{km,ta}}$
$\mathcal{I}^{\mathrm{coordinate}}_{\mathrm{co,ta}}$
$\mathcal{I}^{\mathrm{query}}_{\mathrm{ta,km}}$
a_{km}
${}^{T}_{x,y}c_{km}$
a_{ta}
${}^{T}_{x,y}c_{ta}$

Figure 10. The considered part of Robot Companion realised as five agents: a_{ta}, a_{km}, a_{co}, a_{hi}, and a_{pl}, including interaction protocols. The group ${}^{a}g_{\mathrm{ex}}$ is not shown.

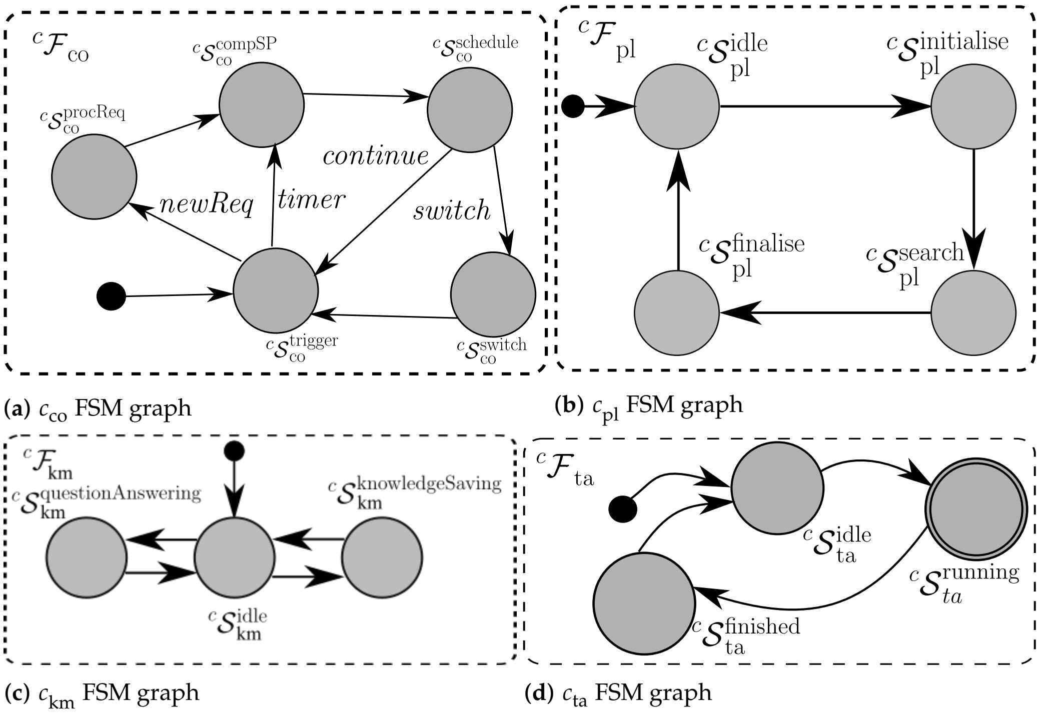
The size of the content formulated using $\mathcal{L}_{\mathrm{WM}}$ depends on the purpose of the message, e.g., requests are up to 250 bytes, but the size of answers depends on particular knowledge—from 250 bytes (in the case of, e.g., state of the gripper) up to 1 MB (in the case of complex or accurate information about the environment). The messages are sent every time the knowledge is needed, i.e., mainly during plan generation.

5.5. Agents in the Robot Companion System

Having defined the communication between agents, we can proceed to the specification of the embodied agents themselves. A crude specification of the agents arises from the system decomposition, however, a more detailed description is required. Thus, a brief description of key behaviours and specification of the FSMs governing them, for each agent, is presented below (Figure 11). All of the agents considered here (Figure 10) contain only their control subsystem—they are purely computational agents.

The objective of the coordinator agent a_{co} is to select one task out of all the tasks requested and coordinate its execution. The problem of task execution coordination and its solution for service robots is published in [72,73].

Based on configurable parameters and the scheduling algorithm, a_{co} decides when to switch the plan being executed by a_{ta} to another one. The agent manages the procedure of suspending the current and starting the next plan execution. The FSM ${}^{c}\mathcal{F}_{\text{co}}$ of the control subsystem c_{co} of the agent a_{co} is tailored to fit the Robot Companion requirements. The FSM has 5 states (Figure 11a). A brief description of the behaviours executed in each of them is shown in Table 3. The agent receives the communicates from a_{hi} in state ${}^{c}\mathcal{S}_{\text{co}}^{\text{trigger}}$, from a_{pl} in state ${}^{c}\mathcal{S}_{\text{co}}^{\text{compSP}}$, and from a_{ta} in all states.


${}^{c}\mathcal{F}_{\text{co}}$
${}^{c}\mathcal{S}_{\text{co}}^{\text{compSP}}$
${}^{c}\mathcal{S}_{\text{co}}^{\text{schedule}}$
${}^{c}\mathcal{S}_{\text{co}}^{\text{procReq}}$
continue
newReq
timer
switch
${}^{c}\mathcal{S}_{\text{co}}^{\text{trigger}}$
${}^{c}\mathcal{S}_{\text{co}}^{\text{switch}}$
${}^{c}\mathcal{F}_{\text{pl}}$
${}^{c}\mathcal{S}_{\text{pl}}^{\text{idle}}$
${}^{c}\mathcal{S}_{\text{pl}}^{\text{initialise}}$
${}^{c}\mathcal{S}_{\text{pl}}^{\text{finalise}}$
${}^{c}\mathcal{S}_{\text{pl}}^{\text{search}}$
${}^{c}\mathcal{F}_{\text{km}}$
${}^{c}\mathcal{S}_{\text{km}}^{\text{questionAnswering}}$
${}^{c}\mathcal{S}_{\text{km}}^{\text{knowledgeSaving}}$
${}^{c}\mathcal{S}_{\text{km}}^{\text{idle}}$
${}^{c}\mathcal{F}_{\text{ta}}$
${}^{c}\mathcal{S}_{\text{ta}}^{\text{idle}}$
${}^{c}\mathcal{S}_{ta}^{\text{running}}$
${}^{c}\mathcal{S}_{\text{ta}}^{\text{finished}}$

(a) c_{co} FSM graph **(b)** c_{pl} FSM graph

(c) c_{km} FSM graph **(d)** c_{ta} FSM graph

Figure 11. FSMs governing the behaviours of the agents, i.e., their control subsystems. The arc labels specify conditions; missing labels are always *True*. The conditions are checked as soon as a behaviour terminates.

Table 3. Mapping of the states of c_{co} FSM to a_{co} behaviours.

State	Behaviour	Description
${}^{c}\mathcal{S}_{\mathrm{co}}^{\mathrm{trigger}}$	${}^{c}\mathcal{B}_{\mathrm{co,trigger}}$	a_{co} is idle, it receives a new command by $\mathcal{I}_{\mathrm{hi,co}}^{\mathrm{request}}$ and a_{ta} state by $\mathcal{I}_{\mathrm{co,ta}}^{\mathrm{coordinate}}$
${}^{c}\mathcal{S}_{\mathrm{co}}^{\mathrm{procReq}}$	${}^{c}\mathcal{B}_{\mathrm{co,procReq}}$	a_{co} stores the task it received from a_{hi} and calculates its deadline and start time
${}^{c}\mathcal{S}_{\mathrm{co}}^{\mathrm{compSP}}$	${}^{c}\mathcal{B}_{\mathrm{co,compSP}}$	a_{co} requests a_{pl} to update plans (initiates $\mathcal{I}_{\mathrm{co,pl}}^{\mathrm{request}}$) of all tasks awaiting execution and, based on them, determines the schedule parameters
${}^{c}\mathcal{S}_{\mathrm{co}}^{\mathrm{schedule}}$	${}^{c}\mathcal{B}_{\mathrm{co,schedule}}$	a_{co} executes the scheduling algorithm that produces a decision. The decision is either to switch the plan being executed to another one, or to continue with the current one. A transition labelled (*continue* or *switch*) is satisfied if the decision (*continue* or *switch*) of the schedule algorithm is made.
${}^{c}\mathcal{S}_{\mathrm{co}}^{\mathrm{switch}}$	${}^{c}\mathcal{B}_{\mathrm{co,switch}}$	a_{co} executes the plan switch procedure. It initiates the $\mathcal{I}_{\mathrm{co,ta}}^{\mathrm{coordinate}}$ protocol.

The purpose of a_{pl} is to generate the plan on demand of a_{co}, in response to $\mathcal{I}_{\mathrm{co,pl}}^{\mathrm{request}}$. During plan generation it may need additional knowledge, so it queries a_{km} using $\mathcal{I}_{\mathrm{pl,km}}^{\mathrm{query}}$. The activities of the control subsystem c_{pl} of the agent a_{pl} are governed by the FSM shown in Figure 11b. In each state of the FSM, an associated behaviour is executed (Table 4).

Table 4. Mapping of the states of c_{pl} FSM to a_{pl} behaviours.

State	Behaviour	Description
${}^{c}\mathcal{S}_{\mathrm{pl}}^{\mathrm{idle}}$	${}^{c}\mathcal{B}_{\mathrm{pl,idle}}$	a_{pl} waits for a planning problem to be transmitted through $\mathcal{I}_{\mathrm{co,pl}}^{\mathrm{request}}$
${}^{c}\mathcal{S}_{\mathrm{pl}}^{\mathrm{initialize}}$	${}^{c}\mathcal{B}_{\mathrm{pl,initialize}}$	a_{pl} initializes the planning problem, multiple possible initiations of $\mathcal{I}_{\mathrm{pl,km}}^{\mathrm{query}}$
${}^{c}\mathcal{S}_{\mathrm{pl}}^{\mathrm{search}}$	${}^{c}\mathcal{B}_{\mathrm{pl,search}}$	a_{pl} searches for the solution
${}^{c}\mathcal{S}_{\mathrm{pl}}^{\mathrm{finalize}}$	${}^{c}\mathcal{B}_{\mathrm{pl,finalize}}$	a_{pl} generates the plan

The agent a_{km} is responsible for query answering and for updating the knowledge database. At initialization the agent starts the $\mathcal{I}_{\mathrm{km,ta}}^{\mathrm{subscribe}}$ protocol, which results in receiving updates on the environment states. Its activity is governed by the FSM shown in Figure 11c. In each state of the FSM, a corresponding behaviour is executed (Table 5).

Table 5. Mapping of the states of c_{km} FSM to a_{km} behaviours.

State	Behaviour	Description
${}^{c}\mathcal{S}_{\mathrm{km}}^{\mathrm{idle}}$	${}^{c}\mathcal{B}_{\mathrm{km,idle}}$	a_{km} waits for three kinds of messages: a query passed by using $\mathcal{I}_{\mathrm{pl,km}}^{\mathrm{query}}$, $\mathcal{I}_{\mathrm{ta,km}}^{\mathrm{query}}$ or a new observation of the environment. Based on that it prepares a switch of state.
${}^{c}\mathcal{S}_{\mathrm{km}}^{\mathrm{questionAnswering}}$	${}^{c}\mathcal{B}_{\mathrm{km,questionAnswering}}$	a_{km} reasons using the knowledge database to answer the received query, and formulates an appropriate message.
${}^{c}\mathcal{S}_{\mathrm{km}}^{\mathrm{knowledgeSaving}}$	${}^{c}\mathcal{B}_{\mathrm{km,knowledgeSaving}}$	a_{km} updates the knowledge database with the received information.

The agent a_{ta} communicates with a_{co}, a_{km}, and ${}^{a}g_{\mathrm{ex}}$. It receives plans and plan execution interruption messages from a_{co} by means of $\mathcal{I}_{\mathrm{co,ta}}^{\mathrm{coordinate}}$. In reaction to the plan execution and interruption requests the a_{ta} states are switched. For the sake of brevity the communication with ${}^{a}g_{\mathrm{ex}}$ is not presented. The FSM that governs the activity of a_{ta} is shown in Figure 11d. In each state of the FSM, a corresponding behaviour is executed (Table 6).

Table 6. Mapping of the states of c_{ta} to a_{ta} behaviours.

State	Behaviour	Description
${}^{c}\mathcal{S}_{\text{ta}}^{\text{idle}}$	${}^{c}\mathcal{B}_{\text{ta,idle}}$	a_{ta} waits for a plan execution request, transmitted by communicates $\mathcal{M}_{\text{co,ta}} = \langle \mathcal{V}_{\text{request}}, \mathcal{Z}, \mathcal{L}_{\text{HFSM}}, -, \mathcal{I}_{\text{coordinate}}, -, m_w, -, m_c \rangle$ within $\mathcal{I}_{\text{co,ta}}^{\text{coordinate}}$ interaction protocol.
${}^{c}\mathcal{S}_{\text{ta}}^{\text{running}}$	${}^{c}\mathcal{B}_{\text{ta,running}}$	a_{ta} executes the plan specified by a_{co}. The plan is defined by a Hierarchical Finite State Machine (HFSM); therefore, ${}^{c}\mathcal{S}_{ta}^{Running}$ is a super state defined by the HFSM. It constantly reports on the state of execution of the plan by sending $\mathcal{M}_{\text{ta,co}} = \langle \mathcal{V}_{\text{inform}}, \mathcal{Z}, \mathcal{L}_{\text{SW}}, -, \mathcal{I}_{\text{coordinate}}, -, m_w, -, m_c \rangle$. When the HFSM terminates or a_{ta} receives an interruption message ($\mathcal{M}_{\text{co,ta}} = \langle \mathcal{V}_{\text{cancel}}, \mathcal{Z}, \mathcal{L}_{\text{HFSM}}, -, \mathcal{I}_{\text{coordinate}}, -, m_w, -, m_c \rangle$) from a_{co} (via $\mathcal{I}_{\text{co,ta}}^{\text{coordinate}}$) the behaviour is terminated. In the case when it needs a piece of knowledge, it initiates $\mathcal{I}_{\text{ta,km}}^{\text{request}}$ interaction protocol to query the a_{km} for it.
${}^{c}\mathcal{S}_{\text{ta}}^{\text{finished}}$	${}^{c}\mathcal{B}_{\text{ta,finished}}$	a_{ta} waits for an acknowledgment from a_{co} regarding the plan termination

5.6. Implementation

The system is implemented using ROS 1. The message exchange was performed using the Remote Procedure Call (RPC) mechanism. Agents are implemented as ROS nodes.

The standardized message format, proposed by IEEE FIPA standards, and specified in this article, is treated as a design pattern from the system implementation point of view. It is implemented as an ROS service, and used for all agents. The interaction protocols of the agents are specified with the `ClassInterfaceInfo` class. Thus, communicates of any agent are specified by filling elements of IEEE FIPA standard communicate. For example, the interaction protocol $\mathcal{I}_{\text{co,pl}}^{\text{request}}$ is specified as:

```
ClassInterfaceInfo( [('request',␣'generatePlan', [GroundTaskNetwork], HierarchicalPlan)]),
```

where 'request' is the performative, 'generatePlan' is the query name, '[GroundTaskNetwork]' is the request content class expressed in $\mathcal{L}_{\text{HFSM}}$, 'HierarchicalPlan' is the response content class given in $\mathcal{L}_{\text{HFSM}}$. Besides enforcing IEEE FIPA communicate standard, the `ClassInterfaceInfo` class verifies if the content follows the specified language. The part of the code responsible for communication has been separated, and used as programming modules. The modules implement, e.g., the `ClassInterfaceInfo` class, all interfaces for all agents and variety of world model objects being transmitted between the agents.

The standardization of interaction protocols enabled verification of the implementation against the specification. It made it easier to verify the implementation of agents by checking that all kinds of queries are supported at the agent code level and that the arguments and returned value comply with the specification. Due to structured messages and well-organised communication at the abstract agent-level, it is straightforward to implement query checks in the agents and analyse inter-agent communication abstracting from the implementation details. The approach presented in this article requires the definition of six communication languages used in the system. Due to that the organization of data structures into a hierarchy of classes emerged.

5.7. Experiment

In the previous sections, the proposed method was used to design a robotic system, then its implementation was described. The experiment presented herein verifies that the thus-produced specification conforms to the requirements. Moreover, the characteristics of the designed communication are assessed. The mentioned characteristics are as follows: the size of communicates, number of them, and frequency of their occurrence in specific timeframes.

The sizes of the messages were estimated in Section 5.4, but their true sizes, their number and frequency of occurrence, depend on particular tasks that the robotic system executes. Therefore, in order to determine whether the communication between the individual components of the system is efficient, these characteristics should be verified in simulations, and then in the real robotic system. The approach for testing the communication is also sketched. Due to limited space, only one test case is described.

The designed robotic system was run under the following conditions. The group ${}^{a}g_{\mathrm{ex}}$ is built of robot hardware components and software running on a machine with 4 core CPU and Linux OS with RT (real time) kernel. The software is semi-automatically generated from the specification using FABRIC [70] and it meets hard RT requirements. The remaining agents, i.e., a_{co}, a_{ta}, a_{hi}, a_{pl}, a_{km} do not require hard RT, and they are deployed on another machine with Linux OS and ROS 1. Both machines are connected by a LAN using an Ethernet cable.

In the presented experiment, a complex task was performed that employs interleaved planning and execution. The experimental task was to pour content of a jar into a bowl. Both objects were initially located in an open cabinet. A video of the experiment is available at the link : https://vimeo.com/720200335 (accessed on 21 October 2022). The provided video, as well as Table 7, show that the requirements imposed on the system, as stated in Section 5.1, are fulfilled.

During the experiment the occurrence of each message was logged along with the metadata (size, sender, recipient, performative, and interaction protocol) and the time at which it was sent. Based on this information, the detailed analysis presented below was carried out.

The execution of a given task took 778 s. and required 1856 messages to be exchanged between the agents. A timetable summarizing the experiment is shown in Table 7.

Table 7. Communicates and robot actions in the experiment.

Time [s]	Event	$\mathcal{C}$ Used
0–37	a_{km} receives an initial knowledge about the environment	$\mathcal{C}^{5}_{\mathrm{ex,ta}}$, $\mathcal{C}^{6}_{\mathrm{ta,km}}$
83	a_{hi} receives a new command, and requests a_{co} to coordinate its execution	$\mathcal{C}^{2}_{\mathrm{co,hi}}$
97	a_{co} requests a_{pl} for the plan #1	$\mathcal{C}^{3}_{\mathrm{co,pl}}$
97–152	a_{pl} generates the plan #1, and queries a_{km}	$\mathcal{C}^{4}_{\mathrm{pl,km}}$
153	a_{co} receives the generated plan #1 from a_{pl}	$\mathcal{C}^{3}_{\mathrm{co,pl}}$
153–681	a_{ta} executes the plan, reports its state to a_{co}	$\mathcal{C}^{1}_{\mathrm{co,ta}}$
293–315, 674–680	a_{km} receives information about the modified environment from a_{ta}	$\mathcal{C}^{5}_{\mathrm{ex,ta}}$, $\mathcal{C}^{6}_{\mathrm{ta,km}}$
300	the jar is grasped with the left gripper	—
315	the jar is pulled out of the cabinet	—
495	the bowl is grasped with the right gripper	—
508	the bowl is pulled out of the cabinet	—
620	the bowl is put down on the table and released	—
680	the bowl is localized	—
681	a_{co} requests a_{pl} for the plan #2	$\mathcal{C}^{3}_{\mathrm{co,pl}}$
681–682	a_{pl} generates the plan #2, and querries a_{km}	$\mathcal{C}^{4}_{\mathrm{pl,km}}$
682	a_{co} receives the generated plan #2 from a_{pl}	$\mathcal{C}^{3}_{\mathrm{co,pl}}$
682–778	a_{ta} executes the plan, reports its state to a_{co}	$\mathcal{C}^{1}_{\mathrm{co,ta}}$
674–682	a_{km} receives information about the environment from a_{ta}	$\mathcal{C}^{5}_{\mathrm{ex,ta}}$, $\mathcal{C}^{6}_{\mathrm{ta,km}}$
763	the content of the jar is poured into the bowl	—
778	the task is finished	—

Agents a_{co}, a_{ta}, a_{hi}, a_{pl} and a_{km} share the same codes for inter-agent communication. The intensity of communication in the whole system is presented in Figure 12. There is intense communication from 97 up to 152 s. (planning #1), and a few other communication episodes. There are two planning processes. In the first one (#1) the robot plans how to take the jar and bowl out of the cabinet, and put the bowl on the table. In the second one (#2) the robot plans how to pour the contents of the jar into the bowl.

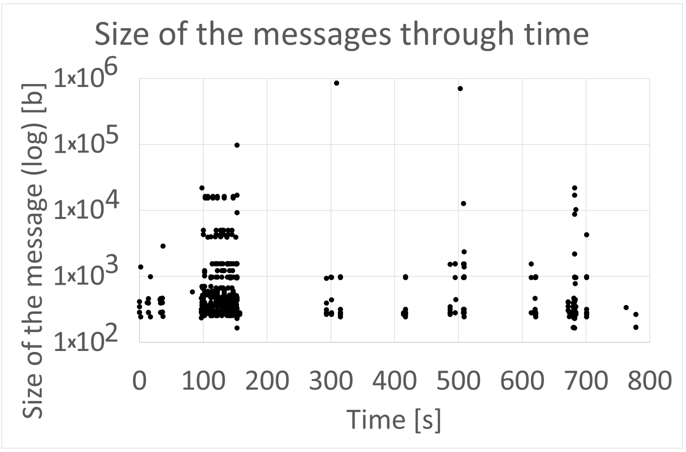


Figure 12. Size (in bytes) of messages sent at specific times.

86% of the sent messages is below 0.5 kB, but there are larger ones too, going up to 852 kB. This shows a great variety of the size of communicates. During execution of a command, agents apply specific communication, depending on the current situation (see Table 7). The difference in the size of the communicates is in line with the estimates of the various types of them made in Section 5.4.

The number of messages sent and received by particular agents is presented in Table 8. Most of the communication is carried out by agents a_{km} and a_{pl} through the communication channel $\mathcal{C}^4_{\mathrm{pl,km}}$. For the whole planning process #1 the required bandwidth is on average 32 kbps, but the peak (150 kbps) is at the end of the first planning process (sec. 153) (see Figure 13). The number of communicates sent in planning #1 is equal to 1598. Later on, the channel is not used (154–680 s), up to 681–682 s. where the planning #2 is done. 57 communicates are exchanged, with average bandwidth equal to 32 kbps. In total, the minimum size of the communicate is 231 b, the maximum is equal to 98 kb, and the average size is 654 b. It shows the accuracy of the predictions made during the construction of the communication channels (see Section 5.4). The distribution of messages over time is difficult to estimate accurately. The size of the message querying a_{km} is small, as it contains only the specification of the knowledge needed. However, the size of the reply from a_{km} depends on the specific piece of knowledge. The communicate size can be small (up to 1 kb), e.g., information on the gripper state, or creation of the hypothetical state of the knowledge base; large (dozens of kb), e.g., sequence of positions of the bowl held by the gripper, possible grasps of the jar; or even very large (up to 100 kb), e.g., the current state of the whole environment.

Table 8. Number of messages sent and received by agents.

	a_{co}	a_{hi}	a_{km}	a_{pl}	a_{ta}	$^{a}g_{\mathrm{ex}}$	Received
a_{co}		1		2	8		11
a_{km}				925	83	24	1032
a_{pl}	2		730				732
a_{ta}	4		71				75
$^{a}g_{\mathrm{ex}}$			14		1		15
send	6	1	815	927	92	24	1865

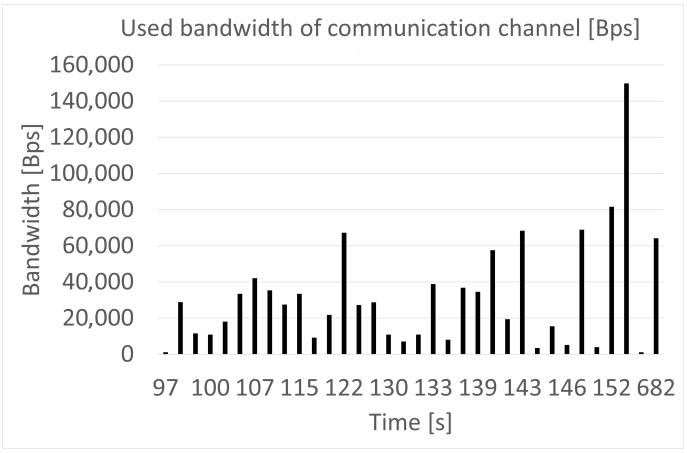


Figure 13. Bandwidth used by $C^{4}_{\mathrm{pl,km}}$ in every second of robotic system operation. The plot contains only the time slots in which the communication takes place.

6. Conclusions

The article presents a method of designing robotic systems, which puts the communication between system components at the forefront. This systematic design method facilitates system implementation and enables experimental evaluation of its characteristics, thus confirming that it fulfills the requirements, and that the communication between the system components is correct. This strengthens the authors' claim that the communication-focused top-down robotic system design method based on binary decomposition is correct and useful. Decomposition based top-down design methods, if applied systematically, assure that the resulting system fulfils the requirements, as opposed to bottom-up synthesis techniques, which produce emergent system properties, which have to be deduced and subsequently tested against the requirements. Moreover, it uses the standardization of the message structure for communication within robotic systems composed of agents based on the IEEE FIPA standard, known from multi-agent systems. In contrast to a commonly used bottom-up approach, where designers take components from a library or a framework and connect them complying with their interfaces, the proposed method is strictly top-down. It separates the specification of the system from its implementation. It shows how to systematically decompose groups and connect them by communication channels.

The communication focused system decomposition enables the analysis of the communication on the abstract agent-level. Organized communication and its properly defined specification facilitates the system's further development. System-specific messages and language definitions are grouped and separated from the implementation. The implementation of the software that handles the communication process is generic. Therefore, there is no need to test the software multiple times. Only verification against the system specifications is required.

Highlighting communication in the design process shows the critical aspects of its operation. The bandwidth of communication channels and communication intensity can be estimated in the development. On their basis, it is possible to properly define or redefine the architecture of the entire robotic system and avoid bottlenecks. Application of the communication focused system decomposition method, with adequately defined requirements, allows for designing a robotic system with optimal control and energy consumption.

Author Contributions: Conceptualization, C.Z.; methodology, P.P. and C.Z.; software, D.S. and W.D.; validation, D.S. and P.P.; writing—original draft preparation, C.Z, P.P. and W.S.; writing—review and editing, C.Z., P.P., W.S., D.S. and W.D.; supervision, W.S.; project administration, P.P. and W.S.; funding acquisition, W.S. All authors have read and agreed to the published version of the manuscript.

Funding: Research was funded by the Centre for Priority Research Area Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme.

Institutional Review Board Statement: Not applicable

Data Availability Statement: Not applicable

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

References

1. Hayward, V.; Paul, R. Robot Manipulator Control Under Unix RCCL: A Robot Control C Library. *Int. J. Robot. Res.* **1986**, *5*, 94–111. [CrossRef]
2. Blume, C.; Jakob, W. *PASRO: Pascal for Robots*; Springer: Berlin/Heidelberg, Germany, 1985.
3. Collett, T.; MacDonald, B.; Gerkey, B. Player 2.0: Toward a Practical Robot Programming Framework. In Proceedings of the Australasian Conference on Robotics and Automation (ACRA), Sydney, Australia, 5–7 December 2005.
4. Zieliński, C.; Winiarski, T. Motion Generation in the MRROC++ Robot Programming Framework. *Int. J. Robot. Res.* **2010**, *29*, 386–413. [CrossRef]
5. Brugali, D.; Scandurra, P. Component-based Robotic Engineering. Part I: Reusable building blocks. *IEEE Robot. Autom. Mag.* **2009**, *16*, 84–96. [CrossRef]
6. Bruyninckx, H. OROCOS: Design and implementation of a robot control software framework. In Proceedings of the IEEE International Conference on Robotics and Automation, Washington, DC, USA, 11–15 May 2002.
7. Brooks, A.; Kaupp, T.; Makarenko, A.; Williams, S.; Orebäck, A. Orca: A component model and repository. In *Software Engineering for Experimental Robotics*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 231–251.
8. Mallet, A.; Pasteur, C.; Herrb, M.; Lemaignan, S.; Ingrand, F. GenoM3: Building middleware-independent robotic components. In Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010; pp. 4627–4632.
9. Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; Ng, A. ROS: An open-source Robot Operating System. In Proceedings of the Open-Source Software Workshop at the International Conference on Robotics and Automation (ICRA), Kobe, Japan, 12–17 May 2009.
10. Brambilla, M.; Cabot, J.; Wimmer, M. Model-Driven Software Engineering in Practice. *Synth. Lect. Softw. Eng.* **2017**, *3*, 1–207.
11. MacDonald, B.; Biggs, G.; Collett, T., Software Environments for Robot Programming. In *Software Engineering for Experimental Robotics*; Brugali, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 107–124._6. [CrossRef]
12. Matarić, M.J.; Michaud, F. The Handbook of Robotics. In *The Handbook of Robotics*; Siciliano, B., Khatib, O., Eds.; Chapter Behavior-Based Systems; Springer: Berlin/Heidelberg, Germany, 2008; pp. 891–909.
13. Bulter, Z.; Rizzi, A. Distributed and Cellular Robots. In *Springer Handbook of Robotics*; Khatib, O., Siciliano, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 911–920.
14. Yim, M.; Shen, W.M.; Salemi, B.; Rus, D.; Moll, M.; Lipson, H.; Klavins, E.; Chirikjian, G.S. Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]. *IEEE Robot. Autom. Mag.* **2007**, *14*, 43–52. [CrossRef]
15. Farinelli, A.; Iocchi, L.; Nardi, D. Multirobot systems: A classification focused on coordination. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2004**, *34*, 2015–2028. [CrossRef]
16. Dudek, G.; Jenkin, M.R.M.; Milios, E.; Wilkes, D. A taxonomy for multi-agent robotics. *Auton. Robot.* **1996**, *3*, 375–397. [CrossRef]
17. Doriya, R.; Mishra, S.; Gupta, S. A brief survey and analysis of multi-robot communication and coordination. In Proceedings of the 2015 International Conference on Computing, Communication & Automation, Noida, India, 15–16 May 2015; pp. 1014–1021. [CrossRef]
18. Chibani, A.; Amirat, Y.; Mohammed, S.; Matson, E.; Hagita, N.; Barreto, M. Ubiquitous robotics: Recent challenges and future trends. *Robot. Auton. Syst.* **2013**, *61*, 1162–1172. [CrossRef]
19. Campusano, M.; Fabry, J.; Bergel, A. Live programming in practice: A controlled experiment on state machines for robotic behaviors. *Inf. Softw. Technol.* **2019**, *108*, 99–114. [CrossRef]
20. Kortenkamp, D.; Simmons, R.; Brugali, D. Robotic Systems Architectures and Programming. In *Springer Handbook of Robotics*, 2nd ed.; Siciliano, B., Khatib, O., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 283–306.
21. Coste-Maniere, E.; Simmons, R. Architecture, the backbone of robotic systems. In Proceedings of the IEEE International Conference on Robotics and Automation ICRA '00, San Francisco, CA, USA, 24–28 April 2000; Volume 1, pp. 67–72. [CrossRef]
22. Lyons, D.M.; Arbib, M.A. A Formal Model of Computation for Sensory-Based Robotics. *IEEE Trans. Robot. Autom.* **1989**, *5*, 280–293. [CrossRef]
23. Lyons, D.M. Adaptive behavior and intelligent systems without symbols and logic. In *Studies in Cognitive Systems*; Chapter A Schema-Theory Approach to Specifying and Analysing the Behavior of Robotic Systems; Kluwer Academic: Alphen aan den Rijn, The Netherlands, 2001; Volume 2, pp. 51–70.

24. Zieliński, C.; Kornuta, T. Diagnostic Requirements in Multi-robot Systems. In *Intelligent Systems in Technical and Medical Diagnostics*; Korbicz, J., Kowal, M., Eds.; Advances in Intelligent Systems and Computing; Springer: Berlin/Heidelberg, Germany, 2014; Volume 230, pp. 345–356. [CrossRef]
25. Armbrust, C.; Kiekbusch, L.; Ropertz, T.; Berns, K. Soft Robot Control with a Behaviour-Based Architecture. In *Soft Robotics*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 81–91.
26. Figat, M.; Zieliński, C. Methodology of Designing Multi-agent Robot Control Systems Utilising Hierarchical Petri Nets. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 3363–3369.
27. Kornuta, T.; Zieliński, C.; Winiarski, T. A universal architectural pattern and specification method for robot control system design. *Bull. Pol. Acad. Sci. Technol. Sci.* **2020**, *68*, 3–29. [CrossRef]
28. Zieliński, C. Robotic System Design Methodology Utilising Embodied Agents. In *Automatic Control, Robotics and Information Processing*; Kulczycki, P., Korbicz, J., Kacprzyk, J., Eds.; Advances in Intelligent Systems and Computing; Springer: Berlin/Heidelberg, Germany, 2021; Volume 296, pp. 523–561. [CrossRef]
29. Kiekbusch, L.; Armbrust, C.; Berns, K. Formal verification of behaviour networks including sensor failures. *Robot. Auton. Syst.* **2015**, *74*, 331–339. [CrossRef]
30. Kiekbusch, L.; Armbrust, C.; Berns, K., Formal Verification of Behaviour Networks Including Hardware Failures. In *Proceedings of the International Conference Intelligent Autonomous Systems IAS 13*; Advances in Intelligent Systems and Computing; Springer: Berlin/Heidelberg, Germany, 2016; Volume 302, pp. 1571–1582.
31. Zieliński, C.; Figat, M.; Hexel, R. Communication Within Multi-FSM Based Robotic Systems. *J. Intell. Robot. Syst.* **2019**, *93*, 787–805. [CrossRef]
32. Hussain, R.; Zielinska, T.; Hexel, R. Finite state automaton based control system for walking machines. *Int. J. Adv. Robot. Syst.* **2019**, *16*, 1729881419853182. [CrossRef]
33. Pilone, D.; Pitman, N. *UML 2.0 in a Nutshell*; O'Reilly: Sebastopol, CA, USA, 2005.
34. Kaptein, F.; Kiefer, B.; Cully, A.; Celiktutan, O.; Bierman, B.; Rijgersberg-peters, R.; Broekens, J.; Van Vught, W.; Van Bekkum, M.; Demiris, Y.; et al. A Cloud-Based Robot System for Long-Term Interaction: Principles, Implementation, Lessons Learned. *J. Hum. Robot Interact.* **2021**, *11*, 1–27. [CrossRef]
35. Dudek, W.; Szynkiewicz, W.; Winiarski, T. Cloud computing support for the multi-agent robot navigation system. *J. Autom. Mob. Robot. Intell. Syst.* **2017**, *11*, 67–74. [CrossRef]
36. Dieber, B.; Breiling, B.; Taurer, S.; Kacianka, S.; Rass, S.; Schartner, P. Security for the Robot Operating System. *Robot. Auton. Syst.* **2017**, *98*, 192–203. [CrossRef]
37. DiLuoffo, V.; Michalson, W.R.; Sunar, B. Robot Operating System 2: The need for a holistic security approach to robotic architectures. *Int. J. Adv. Robot. Syst.* **2018**, *15*, 1729881418770011. [CrossRef]
38. Dudek, W.; Szynkiewicz, W. Cyber-security for Mobile Service Robots—Challenges for Cyber-physical System Safety. *J. Telecommun. Inf. Technol.* **2019**, *2*, 29–36. [CrossRef]
39. Krzykowska-Piotrowska, K.; Dudek, E.; Siergiejczyk, M.; Rosiński, A.; Wawrzyński, W. Is Secure Communication in the R2I (Robot-to-Infrastructure) Model Possible? Identification of Threats. *Energies* **2021**, *14*, 4702. [CrossRef]
40. Baklouti, E.; Amor, N.B.; Jallouli, M. Reactive control architecture for mobile robot autonomous navigation. *Robot. Auton. Syst.* **2017**, *89*, 9–14. [CrossRef]
41. Brooks, R.A. Intelligence Without Representation. *Artif. Intell.* **1991**, *47*, 139–159. [CrossRef]
42. Brooks, R.A. A robust layered control system for a mobile robot. *IEEE J. Robot. Autom.* **1986**, *2*, 14–23. [CrossRef]
43. Arkin, R.C. *Behavior-Based Robotics*; MIT Press: Cambridge, MA, USA: 1998.
44. Gat, E. On Three-Layer Architectures. In *Artificial Intelligence and Mobile Robots*; Kortenkamp, D., Bonnasso, R.P., Murphy, R., Eds.; AAAI Press: Cambridge, MA, USA, 1998; pp. 195–210.
45. Alami, R.; Chatila, R.; Fleury, S.; M. Ghallab, M.; Ingrand, F. An Architecture for Autonomy. *Int. J. Robot. Res.* **1998**, *17*, 315–337. [CrossRef]
46. Tang, F.; Parker, L. A Complete Methodology for Generating Multi-Robot Task Solutions using ASyMTRe-D and Market-Based Task Allocation. In Proceedings of the 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 10–14 April 2007; pp. 3351–3358.
47. Shoham, Y. Agent-Oriented Programming. *Artif. Intell.* **1993**, *60*, 51–92. [CrossRef]
48. Padgham, L.; Winikoff, M. *Developing Intelligent Agent Systems: A Practical Guide*; John Wiley & Sons: Hoboken, NJ, USA, 2004.
49. Nesnas, I. The CLARAty Project: Coping with Hardware and Software Heterogenity. In *Software Engineering for Experimental Robotics*; Brugali, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 31–70.
50. Ahmad, A.; Babar, M.A. Software architectures for robotic systems: A systematic mapping study. *J. Syst. Softw.* **2016**, *122*, 16–39. [CrossRef]
51. Malavolta, I.; Lewis, G.A.; Schmerl, B.; Lago, P.; Garlan, D. Mining guidelines for architecting robotics software. *J. Syst. Softw.* **2021**, *178*, 110969. [CrossRef]
52. de Araújo Silva, E.; Valentin, E.; Carvalho, J.R.H.; da Silva Barreto, R. A survey of Model Driven Engineering in robotics. *J. Comput. Lang.* **2021**, *62*, 101021. [CrossRef]
53. Remy, S.; Blake, M. Distributed Service-Oriented Robotics. *IEEE Internet Comput.* **2011**, *15*, 70–74. [CrossRef]

54. Kleppe, A. The Field of Software Language Engineering. In *Software Language Engineering*; Gašević, D., Lämmel, R., Wyk, E., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–7.
55. Figat, M.; Zieliński, C. Parameterised robotic system meta-model expressed by Hierarchical Petri nets. *Robot. Auton. Syst.* **2022**, *150*, 103987. [CrossRef]
56. Cisternino, A.; Colombo, D.; Ambriola, V.; Combetto, M. Increasing Decoupling in the Robotics4.NET Framework. In *Software Engineering for Experimental Robotics*; Brugali, D., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 307–324.
57. Stampfer, D.; Lotz, A.; Lutz, M.; Schlegel, C. The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software. *J. Softw. Eng. Robot.* **2016**, *7*, 3–19.
58. Zieliński, C. General Robotic System Software Design Methodology. In Proceedings of the Advances in Mechanism and Machine Science, 15th IFToMM World Congress, Kraków, Poland, 30 June–4 July 2019; Uhl, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2019; Volume 73, pp. 2779–2788.
59. Figat, M.; Zieliński, C. Robotic System Specification Methodology Based on Hierarchical Petri Nets. *IEEE Access* **2020**, *8*, 71617–71627. [CrossRef]
60. Janiak, M.; Zieliński, C. Control System Architecture for the Investigation of Motion Control Algorithms on an Example of the Mobile Platform Rex. *Bull. Pol. Acad. Sci. Technol. Sci.* **2015**, *63*, 667–678. [CrossRef]
61. Zieliński, C.; Trojanek, P. Stigmergic cooperation of autonomous robots. *J. Mech. Mach. Theory* **2009**, *44*, 656–670. [CrossRef]
62. Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. 2000. Available online: http://www.fipa.org/specs/fipa00061/SC00061G.html (accessed on October 25, 2022) .
63. Mazouzi, H.; Seghrouchni, A.E.F.; Haddad, S. Open protocol design for complex interactions in multi-agent systems. In *First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*; Association for Computing Machinery: New York, NY, USA, 2002; pp. 517–526.
64. Bedaf, S. The Future is Now: The Potential of Service Robots in Elderly Care. Ph.D. Thesis, Datawyse/Universitaire Pers Maastricht, Maastricht, The Netherlands, 2017. [CrossRef]
65. Salichs, M.A.; Encinar, I.P.; Salichs, E.; Castro-González, Á.; Malfaz, M. Study of scenarios and technical requirements of a social assistive robot for Alzheimer's disease patients and their caregivers. *Int. J. Soc. Robot.* **2016**, *8*, 85–102. [CrossRef]
66. Frennert, S.; Eftring, H.; Östlund, B. What older people expect of robots: A mixed methods approach. In *International Conference on Social Robotics*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 19–29.
67. Winiarski, T.; Jarocki, S.; Seredyński, D. Grasped Object Weight Compensation in Reference to Impedance Controlled Robots. *Energies* **2021**, *14*, 6693. [CrossRef]
68. Seredyński, D.; Banachowicz, K.; Winiarski, T. Graph–based potential field for the end–effector control within the torque–based task hierarchy. In Proceedings of the 21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR'2016, Miedzyzdroje, Poland, 29 August–1 September 2016; pp. 645–650. [CrossRef]
69. Winiarski, T.; Banachowicz, K.; Seredyński, D. Two mode impedance control of Velma service robot redundant arm. In *Progress in Automation, Robotics and Measuring Techniques*; Szewczyk, R., Zieliński, C., Kaliczyńska, M., Eds.; Advances in Intelligent Systems and Computing (AISC); Springer: Berlin/Heidelberg, Germany, 2015; Volume 351, pp. 319–328. [CrossRef]
70. Seredyński, D.; Winiarski, T.; Zieliński, C. FABRIC: Framework for Agent-Based Robot Control Systems. In Proceedings of the 12th International Workshop on Robot Motion and Control (RoMoCo), Poznań, Poland, 8–10 July 2019; pp. 215–222. [CrossRef]
71. Beetz, M.; Beßler, D.; Haidu, A.; Pomarlan, M.; Bozcuoğlu, A.K.; Bartels, G. Knowrob 2.0—A 2nd generation knowledge processing framework for cognition-enabled robotic agents. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 512–519.
72. Dudek, W.; Winiarski, T. Scheduling of a Robot's Tasks With the TaskER Framework. *IEEE Access* **2020**, *8*, 161449–161471. [CrossRef]
73. Dudek, W.; Węgierek, M.; Karwowski, J.; Szynkiewicz, W.; Winiarski, T. Task harmonisation for a single–task robot controller. In Proceedings of the 12th International Workshop on Robot Motion and Control (RoMoCo), Poznań, Poland, 8–10 July 2019; pp. 86–91. [CrossRef]