Software Platform for Development of Multimodular Robotic Systems with Asynchronous Multithreaded Control

Arseniy Ivin, Daniil Mikhalchenko
St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences,
ITMO University
St. Petersburg, Russia
ocnechlahim@mail.ru, arssivka@yandex.ru

Abstract—**Modern robotic complexes (RC) are equipped with lots of integrated hardware and software tools, the purposes of which are: environment analysis, communications, control of executional mechanisms and other specific features. In most cases, existing software platforms for developing RC are oriented to a very specific class of tasks, or in fact, these platforms are too "heavy" for a quick adaptation to an application task. In this article, a software platform for development of Multimodular robotic systems with asynchronous multithreaded control is proposed. The main feature of the platform is high performance of communications between modules of a robotic system, which was confirmed by experiments. For most practical robotic systems with 10 modules and transmission of 100 messages per cycle the proposed platform deals with such a load in less than 1 ms. This is significantly faster than the speed of program interaction with hardware of the robot. For instance, the average frequency of program interaction with hardware of a popular mobile robot Darwin OP was about 14 ms.**

I. Introduction

Robotics is an interdisciplinary science, which requires decisions, connected with math modeling of controlling complex processes, software implementations of digital signal processing, hardware implementation of a sensor system, execution system and others [1]. Interaction of onboard computers, controllers, sensors, input/output devices, power supply devices, etc., must be asynchronous and multithreaded considering onboard computing and network equipment. Furthermore, in some cases communication between autonomous RC must be provided [2]. In most cases, developers of RC have to integrate various heterogeneous software and hardware components, which causes a number of problems related to the unification of communicational protocols and simultaneous control of distributed equipment.

The problem of asynchronous multithreaded control is increased in swarm robotics with a growing number of communication units with simplest computing, sensor and built-in actuators, as well as limited resources of homogeneous swarm robots [3],[4]. In the area of swarm robotics, multi-agent technologies are used to simulate the interaction of large groups of simple homogeneous robots. The limited resources of individual robots significantly affect the configuration and capabilities of the whole system; however, due to the distributed swarm intelligence based on data retrieved during the mass of pair interactions of robots, the existence of a swarm and solving them required tasks is solved [5],[6].

At solving a task by a system of robots a range of emerging tasks depends on three main aspects [7]: 1) the robot simultaneously performs one task or a multitude of tasks; 2) the task is executed by one robot or a multitude of robots; 3) the problem is solved immediately when appointed or there is a plan of tasks requiring execution. Based on the proposed taxonomy of problems in [8] an approach to forming coalitions of robots for problem solving in real time is proposed, providing a prediction of the execution time each robot needs to perform a specific task.

The paper [9] investigates the problem of dividing a swarm of mobile robots into balanced subgroups and provides control algorithms of the position and orientation of groups of robots when forming a certain spatial structure. In the proposed algorithms, the control of robot models is performed based on two main parameters: the synchronization and orientation of individual robots. At that three types of synchronization of the robots are considered: 1) fully-synchronous (FSYNC) model, when all robots operate according to the same time and cycles, and perform any action type in every cycle; 2) semi-synchronous (SSYNC) model, when all robots operate according to the same cycles, but not all robots are necessarily active in all cycles; 3) asynchronous (ASYNC) model, when the robots operate on independent in terms of duration cycles

Proposed software platform for creating multimodal asynchronous multithreaded robotic systems will provide communication of software modules, whose functionality is limited by the C++ language standard and by the platform on which robotic system is running. In addition, a module with high resource efficiency for controlling robots' hardware can be done with the use of the proposed robotic platform. Robotic Platform satisfies the requirement of the architecture flexibility for developers using this platform. It will provide execution of required functionality using modern methods of multithreaded programming.

This research mainly deals with three tasks:

1) Developing platform architecture that will provide registration and removing modules of the robotic systems, execution of functional part of modules using three different modes: execution with thread-

pull, fully multithreaded execution and execution using single thread. In addition, the use of third-party libraries should be minimized.

2) Developing methods of execution of functionality of robotic system using modern multithreaded programming techniques.
3) Developing effective serialization and data transfer method using RTTI – run time type identification.

The paper presents analysis of the existing frameworks in Section II. After that platform structure, its components and development approaches are described in Section III. Testing and an example of application of the platform are presented in Section IV. Section V gives a conclusion. Future directions of platform development are stated in Section VI.

II. RELATIVE WORK

Multimodular robotic systems with asynchronous multithreaded control are suitable for various purposes, and Multimodular approach extends their usefulness and allows rebuilding the system, depending on specific tasks. There is a wide application area of Multimodular robotic systems: manned space exploration [10], geological exploration [11], disaster relief [12], etc. An analysis of the existing frameworks for development of multi-agent robotic systems, such as ROS, YARP, OROCOS, ORCA, Open-RTM, and Open-RDK, is presented in [13]. The authors highlight the main aspects for multi-agent robotic systems software development and identify certain characteristics of framework which provide a wide range of tools for the developers of robotic systems. An overview of the mentioned frameworks is presented in Table I.

The authors of the paper [14] introduce an orchestrated data mapping service, based on Service Oriented Computing (SOC), which maps the information present in a virtual scenario and that it is used by a multi-robot system. An overview of Multimodular platform for robotic systems named ROS is presented in the paper [15]. Nowadays this is the most popular robotic software platform among robotic developers. ROS possesses a multiprocess architecture. Such type of architecture makes it effective only on UNIX-like systems. One should pay attention to some disadvantages of ROS: it is made without using modern standards of language, is too "heavy" and depends on lots of third-party libraries. It is said that this platform is mainly oriented to working with big robotic complexes. Although ROS is capable to work with almost any robot, it requires too many resources to work on small mobile robots. More often, it is used for research purposes.

An overview of multithreaded architecture of robotic platform PX4, developed with the use of "Publisher-Subscriber" pattern, is presented in the paper [16]. However, this platform is based on "Posix" interface and is mainly oriented to work with microcontrollers. The platform can communicate with ROS and can be easily controlled in Unix-shell style. However, it cannot be considered as universal due to the fact that it is oriented only to work with microcontrollers.

An overview of the system named ROCOS is presented in the paper [17]. It is a platform for real-time programming of robotic systems. This platform allows integrating not complicated automates in run-time. Memory-allocator and garbage collector are developed to optimize the usage of memory in this platform. It uses script language LUA. The main feature of this platform is a possibility to develop modules in run-time without recompilation. It does not possess any other noticeable features.

TABLE I. FRAMEWORKS OVERVIEW

Platform	**Noticeable features**	**Constraints**	**Shortcomings**
ROS	multimodality, a lot of useful tools from simulators support to different visualizers	effective only on UNIX-like systems	multiprocess architecture, a lot of dependencies on third-party libs
PX4	multimodality, unix-shell style controlling, communications with ROS	only for microcontrollers	usage of "Posix"
ROCOS	multimodality, Real-time programming, optimized memory usage	only research application	
OPRoS	multimodality, visual programming tool	only research application	low-performance
Urbi	multimodality, new robotic-specific programming language, cross-platform, prototype-based programming	requires to learn new programming language with non-standard paradigm	non-traditional multithreading method, non-standard programming language

Multimodular platform for developing robotic systems named OPRoS is presented in the paper [18]. It is based on multiprocess architecture. Its components are integrated as network modules. XML technology is widely used. The platform possesses a visual programming tool. This platform is similar to the already mentioned ROS. It is used for research purposes and is mainly oriented to speed and comfortability of development. However, this platform lacks high-performance feature, due to this it is not suitable for real robotic systems.

A platform named Urbi is presented in the paper [19]. A new programming language that considers specific aspects of robotic developments is proposed. However, this feature can make development process more complex because one needs to learn a new specific language. In addition, multithreading implemented in this platform is based on a non-traditional method. This fact can be another complexity in a development process. A new specific programming language cannot be considered as standard. Due to that fact, it cannot be considered as universal.

The authors of overviewed platforms mainly concentrated on comfortability and speed of development. As can be seen, the main common feature of such platforms is multi-modularity. The platform, proposed in this article, seeks to be universal and possess high performance feature. This platform can be applied not only to robotic systems, but to any project that requires a light-weight high-performance multimodular system. Furthermore, it is in line with mobile and embedded devices of cyber-physical systems [20].

III. Platform architecture

The 11-th standard of C++ programming language is used for developing the platform [21]. This standard has a lot of integrated tools, which allow avoiding the use of a third-party library “Boost”. “Boost” is a widely-used library that extends possibilities of C++ language. “Boost” contents a lot of implementations of different programming tools such as algorithms or metaprogramming tools. Avoiding this library allows one to liquidate big dependency and make platform more light-weight. In addition, it makes installation of platform easier. C++ (standard 11) has integrated support of multithreading that allows to implement required algorithms, such as thread-safe containers, control of multithreaded execution, synchronization of transferring data and others. In addition, C++ (standard 11) possesses another useful feature – rvalues references. This feature allows using move constructors instead of copy constructors, which is good for optimizing the work with memory.

Lock-free implementations of thread-safe containers are considered to be used in the platform. As one can see from the name, such thread-safe containers are implemented without using mutexes – mutual exclusions – which are needed to provide synchronization while running in multithreaded mode. This method allows increasing performance of program.

Such containers are well suitable for the suggested architecture because situation MPMC (Multiple Producer Multiple consumer) is possible. This is a situation when data can be written to the container and read from the container from different threads simultaneously. Using containers with locks in such situations can dramatically decrease performance. The possibility to use such implementations without third-party libraries is provided by the tools of C++ (standard 11).

In addition, one of the main features of this library is the maximum of flexibility and extensibility. The platform architecture is made considering this desirable feature. Abstraction called *Mechanism* is widely used in this platform. This abstraction can be literally considered as a mechanism, which is responsible for a certain task of synchronization within the platform. A developer, using this platform, can use ready mechanisms provided by library, but if this does not suit the required purposes, a developer can implement his own specific mechanisms to extend the functionality of the platform.

An example to illustrate the flexibility of the platform: there is a mechanism responsible for the start and execution of all modules within the system; there are few variants of implementation of this mechanism: single threaded, multithreaded and thread pool. The platform can be used on different CPU’s. CPU’s can be single core or multi core. It is reasonable to use specific implementation of this mechanism to suit the possibilities of CPU to avoid a waste of resources to provide synchronization when it is not needed and to provide maximum performance. For example, there is no point of using multithreaded mode on an old single core CPU, while there is no point of avoiding using possibilities of multi core CPU’s. Flexibility and extensibility of the proposed platform allows configuring it for the specific environment. Fig. 1 illustrates a prototype of platform in its basic configuration: core and a basic set of modules.

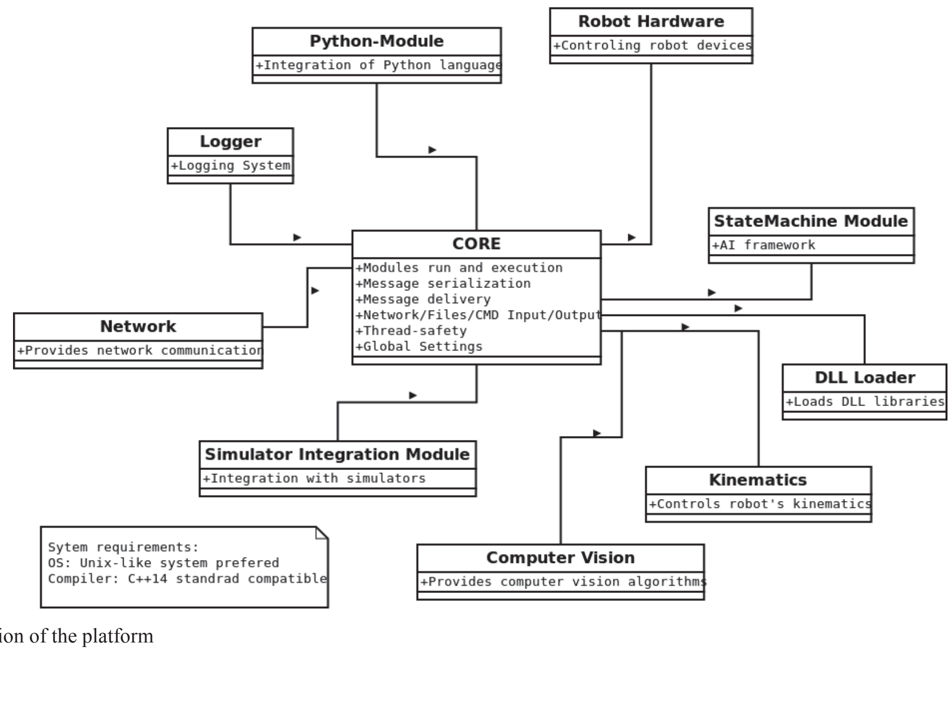
Python-Module
+Integration of Python language
Robot Hardware
+Controling robot devices
Logger
+Logging System
StateMachine Module
+AI framework
CORE
+Modules run and execution
+Message serialization
+Message delivery
+Network/Files/CMD Input/Output
+Thread-safety
+Global Settings
Network
+Provides network communication
DLL Loader
+Loads DLL libraries
Simulator Integration Module
+Integration with simulators
Kinematics
+Controls robot's kinematics
Sytem requirements:
OS: Unix-like system prefered
Compiler: C++14 standrad compatible
Computer Vision
+Provides computer vision algorithms

Fig. 1. Basic configuration of the platform

In addition, Fig. 1 clearly describes functions of the platform. Core is the main component of the system that provides all required mechanisms for running and creating modules. All of core components are weakly connected, which is good when a developer wants to change core's behavior according to his purposes. Launcher is a core's component, which is responsible for an order of processing components of the system. Its implementation delivers within the core. Furthermore, the core possesses a set of mechanisms to provide interaction of system components.

Basic functions of the core:

1) Registration of the required modules.
2) Removal of modules from the registered modules list.
3) Start and execution of modules.
4) Control of modules execution.
5) Transferring messages between modules.
6) Serialization of transferring data.
7) Providing thread-safety.
8) Providing input/output for the files/network/terminal.
9) Keeping and transferring settings.

Planned basic set of modules:

1) DLL Loader – module providing dynamic load of DLL libraries.
2) Python-module – module providing the possibility of using modules written in Python language.
3) Logger – module providing writing of logged data.
4) State-machine-module – module providing possibility to work with framework allowing creation of decision automata.
5) Network module – module providing network communications.
6) Simulator Integration Module – module providing interaction with simulators such as Gazebo.
7) Robot hardware module – module providing control of robot's hardware.
8) Kinematics module – module providing algorithms for calculating robot's kinematics.
9) Computer Vision module – module providing computer vision algorithms.

Let us present a detailed description of the platform components and how it functions. Fig. 2 illustrates a generalized architecture of the core of the platform. This diagram illustrates the main core components. These components can describe main features of the platform's core. It should be noticed that, in fact, the core consists of a larger number of components. This architecture is oriented towards flexibility and extensibility. Extensibility is provided by the possibility to create new implementations of each abstraction. In addition, provided implementations allow configuring the system for the specific runtime environment. It should be noted that this architecture is not final. Some of the components are not presented in this article; other components are still under development.

Abstract Launcher is an interface of the component, which is responsible for order and mode of launching, execution and synchronization of system's components. Existing of this abstraction allows creating different implementations of this mechanism.

Linear Launcher is implementation of the Abstract Launcher interface, which provides a running of the whole system in a single threaded mode.

Thread Pool is implementation of the abstract launcher interface, which provides a running of the whole system using thread pool – mechanism, which allows the system easing multithreaded processing. A programmer chooses certain code fragments, which can be executed in parallel. Runtime environment optimizes execution of this code fragments using working threads from thread pool.

Abstract Queue Adapter is an interface of the component, which is responsible for providing deffered synchronization. It transfers all the synchronization tasks to the certain queues. STL Queue Adapter is an implementation of Abstract Queue Adapter Interface, which provides the possibility to work with the queue from the C++ standard STL library.

Lock Free Adapter is an implementation of Abstract Queue Adapter Interface, which provides the possibility to work with the lock-free thread-safe queue.

Ring Queue Adapter is an implementation of Abstract Queue Adapter Interface, which provides the possibility to work with the ring queue.

Using a mechanism of deffered synchronization allows avoiding problems of cycling when a certain module can send messages to itself. In addition, this allows choosing specific implementation for the specific situation.

Abstract Node is an interface which allows creating independent components of the system – nodes aka modules. Each of these modules might be responsible for a specific functionality. Modules do not "know" about each other's existence and are fully independent from each other. Their interaction and execution are provided by the core of the system. Such a method of modules creation allows re-using ready modules in different configurations of the robotic platform.

Abstract Mechanism is an interface providing some kind of interaction between modules. This abstraction itself provides versioning of the mechanisms.

Basic Abstract Mechanism creates the needed number of lock-free thread-safe queues to provide regulation of execution of requests. For example, firstly, the set of services will be changed in the Service Mechanism and only after that requests will be sent. In addition, it creates an object of the basic non-thread safe class that must be thread safe. The derived class of mechanism creates functions-wrappers, which encapsulate every call of the needed class to the queue of deffered execution that will be processed in the synchronization phase.

Launcher Mechanism is an implementation of the Abstract Mechanism interface providing connection with a certain implementation of the Abstract Launcher interface. It allows adding and removing modules as well as starting and stopping execution of the system.

Messaging Mechanism is an implementation of the Abstract Mechanism interface providing possibility of communication and data exchange between modules of the system. Services Mechanism is an implementation of the Abstract Mechanism interface providing the possibility to execute specific methods using their names. Such behavior is commonly known under the name RPC – remote procedure call. Messaging and Service mechanisms are two main mechanisms provided in the platform. These mechanisms transfer buffers of symbols between modules. Such an approach allows changing the serialization and deserializations methods without changing the core. This is made in terms of flexibility. It is required that all modules of the platform use exactly the same serialization and deserialization method.

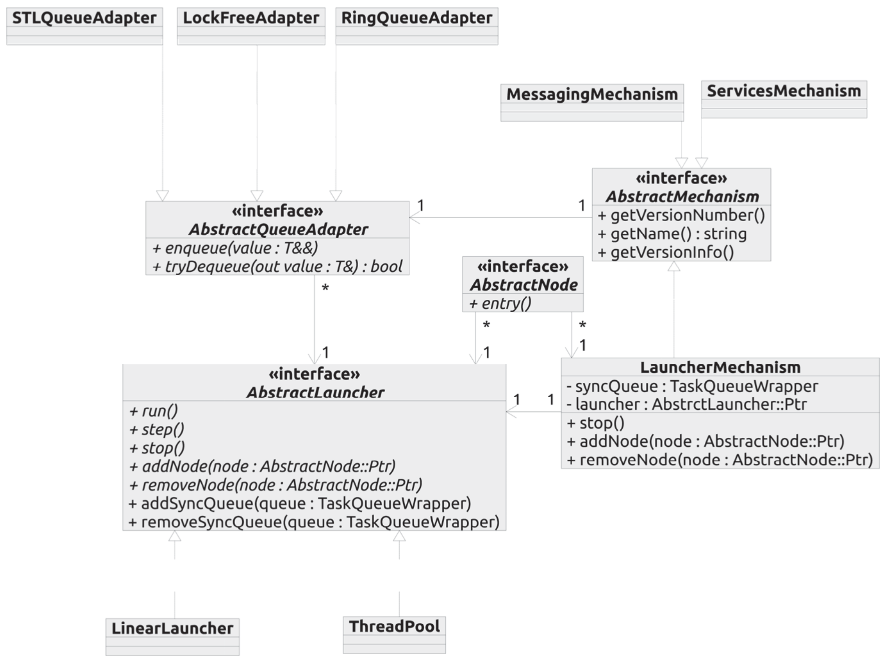

Fig. 2. Generalized core architecture

As it was mentioned, the platform is based on the asynchronous approach. It is made using deffered call of the “callbacks”. “Callbacks” are pieces of code that are passed as arguments to other executable pieces of code. Launcher, the basics of which were described above, encapsulates two queues. One queue is a queue of synchronization tasks. Other queue is a queue of “user” tasks. “User” tasks are tasks that must be done by modules. Launcher contains a scheduler to schedule execution of tasks to a certain time. This is made to avoid inappropriate waste of resources. There is also a class called Core that is not shown in Fig. 2. But it is important to mention its existence, because it is a sort of a layer between Launcher and mechanisms which facilitates the work with the queues of the launcher for mechanisms and modules as well as addition of new mechanisms. Fig. 3 illustrates the sequence diagram of the deffered call. It is an example of the interaction between components of the system. First of all, Launcher runs the processing of all modules. At the same time, synchronization mechanisms, can only asynchronically accept requests from modules. When modules process ends, processing of synchronization mechanisms is started by the Launcher. This processing is needed to apply all the planned changes to the modules. One can see from the diagram that such a system allows facilitating the synchronization process reducing it to the use of thread-safe queues, in particular, lock-free or wait-free implementations of thread-safe queues, which obviously provides better performance than critical sections.

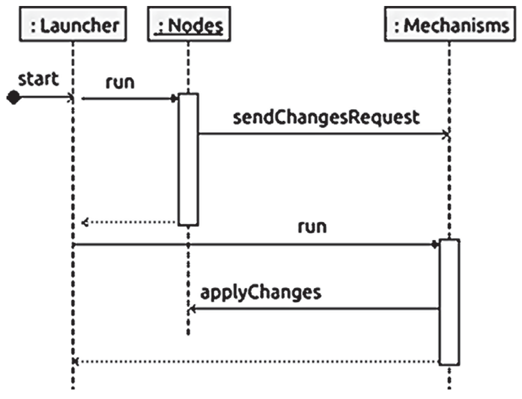

Fig. 3. Deffered call diagram

In terms of the proposed platform, the Module conception is abstract. There is no need to create the real object of the Module. The Module conception consists of a set of callbacks, which are responsible for specific functionality. The creation of the real object of the Module can be useful when dynamic registration or deletion of module is needed to provide the possibility for registered modules to automatically add or remove callbacks from the core with the full memory freeing. To increase the performance messages are transferred through the constant shared pointer, which allows avoiding copying and additional reallocation of buffers. Mechanisms use templates for the message type. This allows changing the transferring and keeping approach with minimal changes to the core. Type of message is set in the class of the Core that interacts with classes of modules.

At the moment, core prototype based on using Google Protocol Buffers Library is implemented. This library is used to provide data serialization. Current prototype provides basic functionality. However, this implementation possesses few disadvantages. At the moment, the methods, which will allow avoiding the following disadvantages, are being developed.

These disadvantages are:

1) Dependency on the third-party library – this forces user to install additional library and makes installation of the system more complex.
2) Serialization of data provided by Google Protocol Buffers can be considered as relatively slow due to the high degree of compression.
3) Using Google Protocol Buffers library forces to use an external tool for creating classes of messages. Furthermore, this tool must be built for every specific platform.

IV. Application and Testing

Experiment scenario was implemented to test performance of the developed system. In this scenario, a certain relatively big number of modules were created. These modules are subscribed to receiving certain message. During the execution of scenario a big number of messages are sent to these modules. Table II presents the results of the tests.

TABLE II. Tests results

	Number of messages			
Number of modules	**100**	**1000**	**10000**	**100000**
10	0 ms	3 ms	37 ms	346 ms
100	2 ms	15 ms	158 ms	1568 ms
1000	18 ms	129 ms	1251 ms	13508 ms

At the moment, the average frequency of program interaction with hardware of the mobile robot is about 14 ms. This frequency was achieved on the popular mobile robot Darwin OP. Most probably the number of modules in practical use of such systems would be similar to the numbers presented in the first test, which is about 10 modules and 100 messages. Proposed robotic platform deals with such a load in less than 1 ms. It is significantly faster than the speed of program interaction with hardware of the robot. The worst case presented in test table is 1000 modules and 100000 messages. One can see that platform deals with such a load in 13 seconds. In real time situations, it is not an accessible speed. However, it is hard to imagine mobile autonomous robot that requires such a number of modules. Nowadays, there are no such autonomous robotic systems. To prove the chosen numbers of modules and messages in test, a diagram of possible use of the platform is proposed in Fig.4.

It is a possible set of modules for the robot-football player. Rectangles are for modules, ellipses are for messages. Arrows can be considered as channels for the message transfer. As one can see, there will be 14 messages transferred between 8 modules. Even if there will be 2 times more modules than presented, the platform's performance still will show great results as one can judge from the tests presented in Table II. From the diagram it is seen how modules interact with each other through the messages. Gait module gets sensors data from the Hardware module, after that execution determines needed positions for robot's limbs, and sends it to kinematics module. Kinematics module receives needed positions for limbs, determines angles for servos and sends it to Hardware module, so it can be applied. Computer vision module gets an image from the Hardware module, processes it and sends messages with coordinates of founded robots, lines and corners on the field and the ball. These messages are received by the Localization module, which can determine a robot's position on the field. A lot of messages are received by the State-Machine module. It is used to make a decision about what to do: e.g., determine where to go and send corresponding parameters to the Gait module, or decide to kick a ball and send corresponding parameters to Kicking by the Leg module, which after that will send needed limb positions to the kinematics module etc.

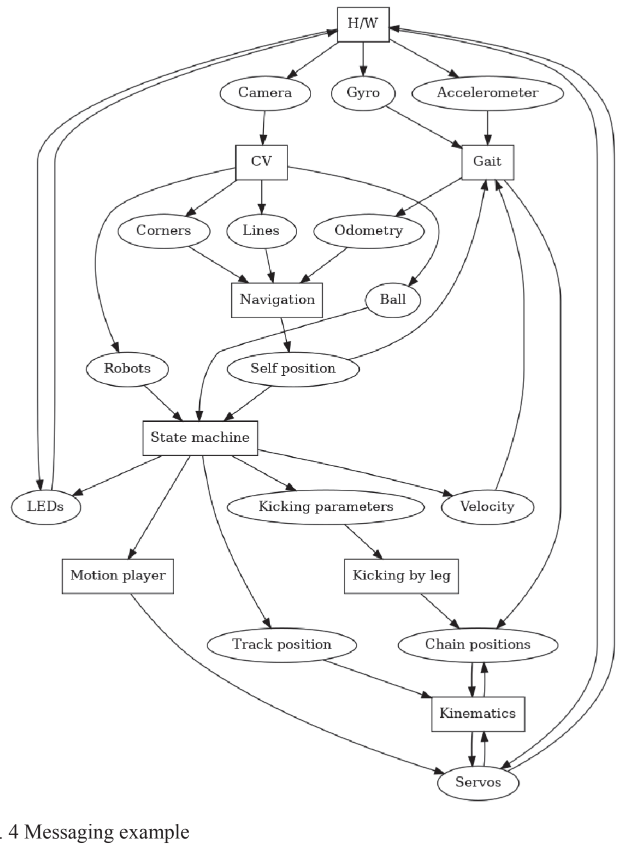

Fig. 4 Messaging example

V. Conclusion

The conducted research shows that the complexity of architectures of robotic complexes is growing significantly. This is due to the application of a wide set of different integrated software and hardware tools providing analysis of environment, connection, controlling executional mechanisms and other specific functions. Modern software platforms for developing Robotic Complexes are oriented to the implementation of a very specific class of tasks needed to be done by the robot. Other platforms, oriented to the universality, lack high performance. There are platforms that are oriented towards the comfort and speed of development. Development of all these platforms requires a lot of adaptation to the specific task. The proposed software platform with the asynchronous multithreaded control possesses the opportunity of high performance of communication between modules of the robotic system. This is confirmed by the results of experiments. The developed software will be used in humanoid robotic systems [22], [23] and other mobile platforms [24].

VI. Future directions of the platform development

After the field-testing some changes may be done to the core architecture, but without changing the overall concept. Another serialization/deserialization library is planned to be integrated into the system for testing the library google flatbuffers. It is expected to be more efficient than protocol buffers because of its zero-copy deserialization algorithm and overall performance. Furthermore, the platform needs to be more user-friendly, so one of the most important directions of the platform development is a development of useful user-friendly tools, e.g. some clients with graphical user interfaces, etc. One can notice that concepts of the proposed platform are suitable for the "Internet of things" developments, so the possibility of applying this platform to the "Internet of things" will be considered.

Acknowledgment

This work is supported by state research № 0073-2015-0003.

References

[1] A. Smirnov, A. Kashevnik, N. Teslya, S. Mikhailov, A Shabaev, "Smart-M3-Based Robots Self-Organization in Pick-and-Place System", *Proceedings of the 17th Conference of the Open Innovations Association FRUCT*, Yaroslavl, Russia, on 20-24 April 2015, pp. 210 – 215.

[2] A. Kashevnik, N. Teslya, B. Padun, K. Kipriyanov, and V. Arckhipov, "Industrial Cyber-Physical System for Lenses Assembly: Configuration Workstation Scenario", *Proceedings of the 17th Conference of the Open Innovations Association FRUCT*, Yaroslavl, Russia, on 20-24 April 2015, pp. 62-67.

[3] M. Gauci, J. Chen, W. Li, T.J. Dodd, R. Groß. "Self-organized aggregation without computation". *International Journal of Robotics Research.* 2014, 33(8), pp. 1145-1161.

[4] I. Vatamaniuk, G. Panina, A. Saveliev, A. Ronzhin. "Convex Shape Generation by Robotic Swarm". *IEEE 2016 International Conference on Autonomous Robot Systems and Competitions*, 2016, pp. 306-310.

[5] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari. "Auto Mo De: a novel approach to the automatic design of control software for robot swarms". *Swarm Intell.* 2014, 8(2), pp. 89–112.

[6] A. Ronzhin, I. Vatamaniuk, N. Pavliuk. "Automatic Control of Robotic Swarm during Convex Shape Generation". *2016 International conference and exposition on electrical and power engineering*, Romania, Iasi, October 20-22, 2016, index 926.

[7] B. Gerkey, M. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, 23(9), 2004, pp. 939–954.

[8] J. Guerrero, G. Oliver, "Multi-Robot Coalition Formation in Real-Time Scenarios", *Robotics and Autonomous Systems*, 60, 2012, pp. 1295–1307.

[9] A. Efrima, D. Peleg, "Distributed algorithms For Partitioning a Swarm of Autonomous Mobile Robots", *Theoretical Computer Science*, 410, 2009, pp. 1355–1368.

[10] B.I. Kryuchkov, A.A. Karpov and V.M. Usov, "Promising Approaches for the Use of Service Robots in the Domain of Manned Space Exploration", *SPIIRAS Proceedings*, 2014, 32(0), pp. 125-151.

[11] A.V. Vasiliev, A.S. Kondratyev, A.A. Gradovtsev and I.Yu. Dalyaev "Research and Development of Design Shape of a Mobile Robotic System for Geological Exploration on the Moon's Surface", *SPIIRAS Proceedings*, 2016, 2(45), pp. 141-156.

[12] V.F. Petrov, A.I. Terentev, S.B. Simonov, D.N. Korolkov, V.I. Komchenkov and A.V. Arkhipkin, "Problems of Group Control of Robots in the Robotic Complex of Fire Extinguishing", *SPIIRAS Proceedings*, 2016, 2(45), pp. 116-129.

[13] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development", *Robotics and Autonomous Systems*, 2012 60(6), pp. 803-821.

[14] A. Matta-Gómez, J. Del Cerro and A. Barrientos, "Multi-robot data mapping simulation by using microsoft robotics developer studio", *Simulation Modelling Practice and Theory*, 2014, 49, pp. 305-319.

[15] M. Quigley, "ROS: an open-source Robot Operating System", *ICRA workshop on open source software*, 2009.

[16] L. Meier, D. Honegge and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms", *Robotics and Automation*, 2015, pp. 6235–6240.

[17] M. Klotzbüche, P. Soetens and H. Bruyninckx, "Orocos rtt-lua: an execution environment for building real-time robotic domain specific languages", *Dynamic languages for Robotic and Sensors*, 2010, pp. 284–289.

[18] C. Jang, "OPRoS: A new component-based robot software platform", *ETRI journal*, 2010, pp. 646–656.

[19] J. C. Baillie, "The Urbi universal platform for robotics", *First International Workshop on Standards and Common Platform for Robotics,* 2008.

[20] A. Kashevnik, A. Ponomarev and S. Savosin, "Hybrid Systems Control Based on Smart Space Technology", SPIIRAS Proceedings, 2014, 35(4), pp. 212-226.

[21] ISO/IEC, 2011, Standard for programming language C++, Web: http: //www.pen-std. org/jtc1/sc22/wg21.

[22] A. Denisov, V. Budkov and D. Mikhalchenko, "Designing Simulation Model of Humanoid Robot to Study Servo Control System Interactive Collaborative Robotics", *First International Conference ICR 2016*. Budapest, Hungary, Aug. 2016, pp. 70-79.

[23] N. Pavluk, A. Ivin, V. Budkov, A. Kodyakov and A. Ronzhin, "Mechanical Leg Design of the Anthropomorphic Robot Antares Interactive Collaborative Robotics", *First International Conference ICR 2016*. Budapest, Hungary, Aug. 2016, pp. 113-123.

[24] A.I. Motienko, A.G. Tarasov, I.V. Dorozhko and O.O. Basov, "Proactive Control of Robotic Systems for Rescue Operations", *SPIIRAS Proceedings,* 2016, 3(46), pp. 174–195.