Modelling of data flow in component-based robot perception systems

Piotr Trojanek, Maciej Stefańczyk, Tomasz Kornuta
Institute of Control and Computation Engineering, Warsaw University of Technology

Abstract: The paper presents results of research on the development of robot perception systems. On the basis of the selected exemplary system, typical problems occurring during design of such systems were indicated. Those problems motivated the work set out to formalize the data flow by the use of metamodelling – the key technology of model-driven engineering. The obtained metamodel facilitates development of such systems and enables further creation of tools for models' editing, validation and automatic generation of relevant source code skeletons. Additionally, requirements for a robot perception systems runtime environment were identified and compared with existing component-based robot software frameworks.

Keywords: robot, control system, perception, data flow, metamodel, model-driven engineering, component-based systems

Robots, similarly to humans, need to perceive the environment to perform their tasks. This involves acquiring, processing and interpreting data from sensors, such as cameras or microphones. Typically, data come from several sources, they flow through diverse successive processing steps, and arrive at a unit responsible for taking decision about what to do next. The underlying structure of the data flow in perception systems resembles more a tree (with leaves and root corresponding respectively to sensors and the main decisional module) than a sequence.

Data flows are difficult to model with general-purpose programming languages, since (in most cases) their syntaxes are inspired by a natural language – they are designed to specify successive operations in imperative style. Trees (or graphs in general) are easier to model with graphical languages, which enable us to express the design in a form that is very similar to the way we think about these structures – such representation is easier to comprehend, verify and debug. Unfortunately, experimental robotics lacks tools that facilitate graphical, graph-oriented design of perception systems and their implementation.

We begin by introducing an exemplary perception system that enables the robot to recognize objects based on data from a pair of RGB- (image) and D- (depth) cameras. We use this example to motivate sections which follow, where we formalize the data flow of perception systems and identify challenges of their design and implementation. Our aim is to create a foundation for software tools that support the perception systems development process.

1. Motivating example

As a relatively complex example of perception systems – one that includes several flows and operations on data – we take multi-modal segmentation of dense depth maps with associated color information (as described in [1]). Typically, we describe such systems with use of data flow graphs (fig. 2), where solid boxes represent data processing nodes and dashed boxes group them together into units that could be reused also in other systems. Our algorithm combines color and depth information to identify independent objects in the environment of a robot (fig. 1).

Short description of the whole process is as follows. At first, we acquire color and depth data from Kinect sensor. Then, we convert depth data to a cloud of 3D-points and estimate normal vector for each of them. As a result, we have three, supplementary, sets of data: color, depth and normal vectors. In the next step, for each of them we calculate differences of values between each pixel and its neighbours (e.g. angle difference calculated for normal vectors or euclidean distance for depth) and accumulate those differences (weighted by specified factors for each modality) using either sum or maximum. After accumulation, we have all the information integrated into one image and segment it with the region growing algorithm.

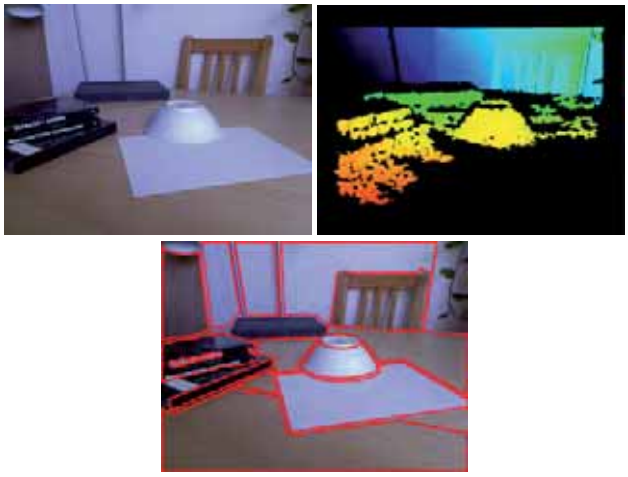

Fig. 1. Result of operation of the multi-modal segmentation. From top-left: color image, depth map, resulting segments
Rys. 1. Wynik działania segmentacji wielomodalnej. Od lewego górnego rogu: obraz kolorowy, mapa głębi, segmenty wynikowe

We can clearly identify some of the commonalities in diagrams like the one for the perception system described above. Processing nodes (or components) possess inputs

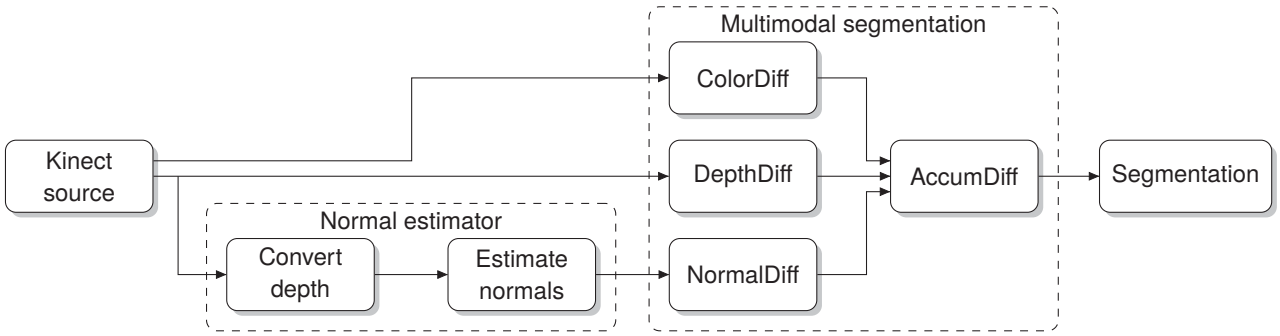

Fig. 2. The structure of data flow realising the multi-modal segmentation
Rys. 2. Struktura potoku przetwarzania dla zadania wielomodalnej segmentacji

and outputs, but their exact numbers and types are specific to particular nodes. Flows link outputs with inputs; one output may be linked to more than one input (but not the opposite). Those links should not be hardcoded in the implementation of a node – we want to reuse nodes in different contexts, hence we want them to be as independent from each other as possible (i.e., loosely coupled).

Taking into account this independence and reusability, another problem arises – how big (or how small) our components should be. On the one hand, we expect them to be as small as possible, which increases the overall system flexibility. But on the other, it also might be convenient to reuse the whole fragments of the existing data flow network. Hence, we demand a mechanism enabling such a composition.

2. Formalization of the data flow

Perception (in general) is a complex process, which involves flow of data from sensors to a decisional unit. Data flow within such a process can be represented as a directed graph $G = (V, E)$ with nodes V corresponding to individual operations (e.g., processing, combining and interpreting data) and edges $E \subseteq V \times V$ corresponding to flows of results of one operation to another.

Formalization with mathematical notation enables us to reason about certain properties of the data flow that are essential for scheduling operations of the perception process, e.g., presence of cycles. However, the above description misses one of the primary aspects of software implementation of the perception systems, i.e., reusability of both the individual data processing nodes and subgraphs of the data flow graph.

Individual nodes of the perception data flow and the perception system as a whole intuitively map onto software components and component-based software system, respectively.

Following the component-based approach, we would like each processing node to contractually define its interfaces, in particular, the types of data consumed and produced [2]. The description of perception systems solely in terms of nodes and edges of a data flow graph arguably fails to capture such important features of the design. Moreover, mathematical notation is impractical for automatic processing by the computer as a part of the software development process. Clearly, we need some other formalism that better fits the data flow modelling.

2.1. Domain-specific modelling languages

Domain-specific modelling languages are designed to formalize the application structure, behaviour, and requirements within particular domains [3]. Formally, a domain-specific modelling language L is typically defined as a 5-tuple of concrete syntax C, abstract syntax A, semantic domain S, and semantic and syntactic mappings (M_S and M_C): $L = \langle C, A, S, M_S, M_C \rangle$ [4].

In this paper we focus on the abstract syntax, which defines the *concepts*, *relationships* and *integrity constraints* of a modelling language dedicated to design of robot perception systems. We postpone definition of the concrete syntax, i.e., the specific notation used to express the designs, and syntactic mapping $M_C \colon A \to C$, which assigns notation symbols to the elements of the abstract syntax, until further research on the concepts related to scheduling data flow for concurrent execution. Similarly, we only briefly address the semantic domain, which is used to explain the meaning of the designs expressed in the modelling language, and the semantic mapping $M_S \colon A \to S$, which relates syntactic concepts to those of the semantic domain. Rather, we prefer to rely on an existing software framework as the semantic domain and express semantic mapping as relationship between the modelling language and the software framework.

2.2. Introduction to metamodelling

The first step in the development of a domain-specific language is to identify abstractions used to solve the problem at hand [5]. This includes distinguishing the concepts of the problem domain and their relationships. The next step is to formalize the identified abstractions. Metamodelling is a formalism, which is particularly well suited to this purpose. It is more expressive and flexible than alternative formalisms, e.g., BNF grammars or UML profiles [6]. Moreover, the model-driven engineering technology makes it much easier to develop tools that facilitate the use of the modelling language, if its abstract syntax is defined as a metamodel [3].

The basic concepts of metamodelling include *system*, *model* and *metamodel*. In this paper we adopt the definitions of these terms as found in [7] and refer to a *system* as a part of the world that is the subject of communication or reasoning. The *model* is a description of a system (or

a part of a system) written in a well-defined language.[1] The *metamodel* is a definition of a well-defined language, which is used to describe models. It is convenient to represent the above concepts using the *representedBy* and *conformsTo* relationships (fig. 3).

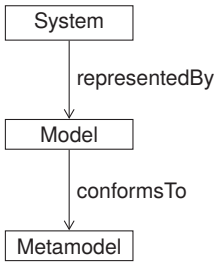

Fig. 3. Basic concepts of metamodelling [8]
Rys. 3. Podstawowe pojęcia metamodelowania [8]

The choice of a meta-metamodel, i.e., a formalism for definition of the modelling language, depends mainly on the availability of tools supporting the development of a complete modelling solution. MetaEdit+ was among the first commercially available domain-specific development environments [5]. The potential of the metamodelling approach was evidenced by the release of Domain-Specific Language Tools from Microsoft [9]. The Eclipse Modeling Framework (EMF) is the leading open-source competitor in the field [10, 11].

We motivate our choice of EMF as the modelling environment mainly by its popularity in the research community, which favours open-source over proprietary solutions. EMF provides advanced support for both graphical and textual notation of modelling languages, as well as model-to-model and model-to-text transformations. Individual tools of the framework follow standards of the Object Management Group (OMG).

2.3. Metamodel of perception systems

Metamodels of EMF are typically described with graphical notation that resembles UML class diagrams, which captures concepts, their attributes and relationships. Integrity constraints that cannot be expressed on a diagram are usually specified as formal annotations in the Object Constraint Language (OCL) [12]. OCL annotations declaratively express invariants of the constructs of the modelling language; EMF ensures that models conforming to a given metamodel satisfy such constraints.

The root concept of the robot perception system is an abstract *Component*[2] (fig. 4). The name attribute identifies an instance[3], while description serves only the purpose of documentation. These general attributes are common to several concepts of the domain; they are used to generate names and comments in software artefacts derived from the model, e.g., code skeletons and documentation.

[1] A *well-defined language* is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer [7].

[2] We denote names of concepts, attributes and relationships of the metamodel by the use of sans serif font; names in italics denote abstract concepts.

[3] EMF provides an additional feature that enforces uniqueness of the name identifiers. By convention, this feature is omitted from metamodel diagrams.

```
context LeafComponent
-- leaf component needs to accept or produce data
inv: inputs->notEmpty() or outputs->notEmpty()

context ContainerComponent
-- external ports routes data to/from internal components
inv: contains.inputs->includes(outputs.propagatesTo)
inv: contains.outputs->includes(inputs.delegatesFrom)
```

Listing 1: OCL invariants imposed on the data flow metamodel

There are two kinds of concrete components: LeafComponents, which represent atomic computations on data, and ContainerComponents, which represent reusable subgraphs of data flow and recursively group other *Component*s. An abstract concept of *Property* parametrizes LeafComponents; it enables the developer to change parameters of data processing at runtime. NumericProperty represents a range of values, e.g., threshold of image binarization, with the minValue and maxValue limits; the designer specifies the default setting as defaultValue. EnumProperty represents a fixed, non-empty list of possible values; the head of the list is the default setting.

Data flow interface of a *Component* is defined by its *InputPort*s and *OutputPort*s, for which the common dataType attribute (inherited from the abstract *Port* concept) specifies the type of consumed and produced data, respectively. LeafComponents contain PrimitiveInputPorts and PrimitiveOutputPorts, as specified by the inputs and outputs relationships. We distinguish separate concepts of the ports of LeafComponents, which directly map onto storage for data in the memory of a runtime system. Such components require at least one input or output port (listing 1).

The ContainerComponent (a composite component) recursively groups other *Component*s with the contains relationship. We adopt the terminology of OMG: input ports of the container *delegate* data to the input ports of its internal components, while its output ports *promote* data produced by the internal components. We distinguish separate concepts of the ContainerInputPort and ContainerOutputPort, which represent routing of data rather than locations in the memory. The delegatesTo and propagatesFrom relationships specify destinations and origins of data flow between the externally visible ports and the ports of the internal parts of a ContainerComponent, respectively. OCL invariants limit scope of these relationships to the internal components of the container (listing 1).

Data flow from output to input port is represented by the sink and source bi-directional relationships. We intentionally forbid directing data from more than one output to a single input port – we have found such designs seldom required and difficult to debug.[4]

[4] For brevity, we omit the invariants that limit the scope of the sink/source relationships to a single ContainerComponent; their specification in OCL involves a complex navigation across the metamodel.

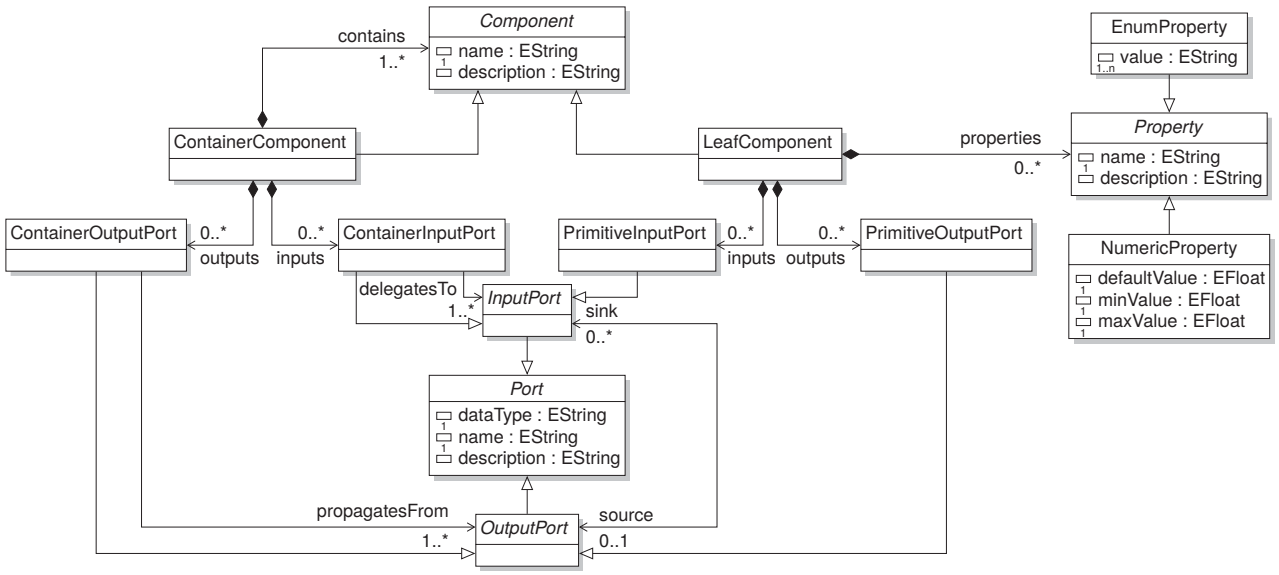

Fig. 4. Metamodel of robot perception system
Rys. 4. Metamodel układu percepcji robota

3. Requirements of the robot perception software

Data flow specified by the presented metamodel matches requirements of diverse domains – not only robot perception systems. However, implementations of such systems require certain features, which distinguish them from other applications.

The robot perception system, as the part of a robot control system, needs to execute in *real time*. Effector devices (indirectly, i.e., by the decisional unit) impose deadlines for data processing. The rate of data delivery from the perception system is typically limited by one (or several) of sensory inputs or data processing subroutines. However, performance of the robot control requires the implementation of the perception system to prevent delays, which are typical in business-oriented data flow applications. In particular, all the resources, such as memory or threads, required for executing and passing data between the components of a perception data flow need to be pre-allocated before staring the operation.

Data inputs – and so the perception systems – are *distributed*. Data from sensors located in different places often require pre-processing to reduce their volume before transmitting (e.g. over wireless) to other nodes of the data flow. Also, resource-intensive computations often require processing power that exceeds capabilities of a single computer. Thus, the runtime system must deploy and configure components on different nodes of LAN.

Typical workstation PCs often lack the performance required for computing-intensive operations, e.g., image recognition. Recently, it became possible to accelerate some operations on data within perception systems by executing them on graphic processing units (GPU). Such approach extends the requirement of *portability* toward new kinds of specialized hardware devices, which need to be integrated with components implemented in general-purpose programming language like C++.

The performance of data flow programming heavily depends on the scheduling of individual operations on data. The sequential execution of subroutines of the components – the easiest one to implement – offers the worst processing time. Alternatives, which promise gain in efficiency, include two kinds of concurrency: *parallelizing*, i.e., running several operations in one data processing step, and *pipelining*, i.e., running a single operation in several data processing steps.

Our survey, which we briefly summarize in section 5, revealed that none of the popular robotics software frameworks satisfies all of the listed requirements. Thus, in next section we describe our ongoing work on mapping of the concepts of the presented metamodel onto a custom framework dedicated to processing sensory information in robot control systems.

4. DisCODe

Robot perception systems, which typically involve several steps of data processing, readily map onto component-based software frameworks. One of such frameworks is DisCODe (*Distributed Component Oriented Data Processing*) [13]. With its core implemented in C++, the design of DisCODe follows three paradigms: component-based programming [14], which provides measures for modularization and reusability, reflexive programming [15], which enables the user to introspect the system at runtime, and generic programming [16], which instantiate the software templates with data types and subroutines provided by the developer. Supported platforms currently include Linux and Windows.

DisCODe provides a number of ready-to-use components dedicated to robot perception, e.g., acquiring data from cameras, and debugging of sensory data processing, e.g., loading and saving images from/to files. Individual components are implemented as dynamically loaded libraries (*SO*s or *DLL*s, depending on the operating system), which are organized in *DCL*s (DisCODe Component Libraries). For most of the tasks, the developer creates

a number of custom components and combines them with those provided by the framework.

Components in DisCODe communicate by data flows; they contain input and output buffers parametrized by data types and (optionally) also variables that need to be preserved between the calls of their data processing subroutines. The design follows the *inversion of control* pattern – the framework triggers the operation of the components, and in this way also schedules the execution of the data flow. At the time of writing, the schedule in statically determined by a configuration file with a mapping of components to threads of the underlying operating system. Thus, the framework exploits the performance offered by modern multi-core workstation PCs.

DisCODe provides graphical interface, which enables the user to configure the components in runtime. The number and types of the properties of a given component are determined automatically by the use of reflection in the code. Properties with a range of values are accessed with sliders, and those with a list of values, with radio buttons.

5. Related work

Software frameworks facilitate the definition of processing pipelines in various ways. General-purpose environments oriented on data flow programming, e.g., Simulink and LabVIEW, enable the developer to easily build an application from a set of predefined building blocks. However, they are difficult to customize and integrate with existing software libraries. Moreover, they miss the requirement of distributed processing and the use of concurrency is hidden from the programmer.

Ecto – a lightweight, open-source framework dedicated to implementation of data flows in robot control systems – automatically schedules directed acyclic graphs on multi-core processors[5]. However, it neither supports distributed operation nor modification of parameters in runtime. ROS, a full-featured predecessor of Ecto, offers these features, but is not a real-time system [17]. Moreover, Ecto and ROS, as software frameworks, do not formalize the design, but only provide guidelines for the developer, who implements the data flow directly in C++.

Orocos is a real-time, component-based framework dedicated to robot control, which can be also used as a runtime environment for robot perception system [18]. There are preliminary reports on formalizing Orocos component model by the use of the EMF metamodels.[6] However, the formalized subset enables only the design of plain, i.e., non-composite data flows.

6. Conclusions

We have reported on our ongoing work on the development of robot perception systems. Based on our experience and a motivating example presented in section 1, we identified some commonalities in our earlier works and formalized them by the use of metamodeling. The metamodels form central elements of a domain-specific modelling language and enable rapid development of complete modelling solutions, which include graphical environments for editing and validation of models as well as their automatic transformation into other software artefacts, e.g., source code skeletons.

We have also identified the requirements for the runtime of robot perception systems and matched them with our component-based software framework DisCODe. The framework still lacks support for distributed processing and enables only static scheduling of data flow. However, we have already successfully used it for implementation of several interesting perception systems, e.g., for a system able to localize dices and determine the game state, being a key element in the robotic system playing a game of dice.

Our next steps will be to develop a graphical modelling solution based on the presented metamodel and the Eclipse Modeling Framework and integrate it with model-to-code transformation, which generates skeletons of source code, configuration files and documentation from a single design.

Besides, our work pointed to several interesting fields worth further research. The first one is asynchronous arrival of data to different input buffers of a given component. Being aware that in the majority of complex robotic applications developers write special switches enabling the execution of a given handler only in the case when the required data is present, we plan to develop special mechanisms facilitating such solutions. We are going to use the model-driven engineering to solve this problem. Another problem is extending the metamodel with constructs for specification of automatic scheduling of data flow computations – most of the so far investigated component-based systems use the round-robin-like algorithms, which belong to the simplest scheduling mechanisms. Taking into account the real-time requirements of robot perception systems, we identified this as an important issue.

[5] http://ecto.willowgarage.com/

[6] http://www.best-of-robotics.org/bride/bcm.html

Acknowledgements

We acknowledge the support of the Faculty of Electronics and Information Technology Dean's grant no. 504M 1031 0016. We also would like to thank Konrad Banachowicz for his interest and thoughtful comments.

References

1. Stefańczyk M., Kasprzak W., *Multimodal segmentation of dense depth maps and associated color information*, [in:] Bolc L., Tadeusiewicz R., Chmielewski L., Wojciechowski K. (eds.), Proceedings of the International Conference on Computer Vision and Graphics, Springer Berlin/Heidelberg, 2012, 626–632.
2. Szyperski C., *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley and ACM Press, 1998.
3. Schmidt D., *Model-driven engineering*, "IEEE Computer", Vol. 39, No. 2, 2006, 25–31.
4. Karsai G., Sztipanovits J., Ledeczi A., Bapty T., *Model-integrated development of embedded software*, "Proceedings of the IEEE", Vol. 91, No. 1, 2003, 145–164.
5. Kelly S., Tolvanen J.-P., *Domain-Specific Modeling: Enabling full code generation*, Wiley-IEEE Computer Society Press, 2008.
6. Kleppe A., *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*, Addison-Wesley, 2009.

7. Kleppe A., Warmer J., Bast W., *MDA Explained: The Model Driven Architecture^{TM}: Practice and Promise*, Addison-Wesley Professional, 2003.
8. Bézivin J., *On the Unification Power of Models*, "Software and Systems Modeling", Vol. 4, No. 2, 2005, 171–188.
9. Cook S., Jones G., Kent S., Wills A.C., *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley Professional, 2007.
10. Steinberg D., Budinsky F., Paternostro M., Merks E., *EMF: Eclipse Modeling Framework*, Addison-Wesley Professional, 2nd edition, 2009.
11. Gronback R.C., *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley Professional, 2009.
12. Warmer J., Kleppe A., *The Object Constraint Language: Getting Your Models Ready for MDA*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.
13. Kornuta T., Stefańczyk M., *DisCODe: komponentowa struktura ramowa do przetwarzania danych sensorycznych*, "Pomiary Automatyka Robotyka", 7-8/2012, 76–85.
14. Szyperski C., Gruntz D., Murer S., *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Professional, 2nd edition, 2002.
15. Sobel J.M., Friedman D.P., *An Introduction to Reflection-Oriented Programming*, 1996.
16. Alexandrescu A., *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley Professional, 2001.
17. 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), 2009.
18. Bruyninckx H., *The Real-Time Motion Control Core of the OROCOS Project*, [in:] Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2003, 2766–2771.

Modelowanie przepływu danych w komponentowych układach percepcji robotów

Streszczenie: W artykule przedstawiono wyniki prac poświęconych budowie układów percepcji systemów robotycznych. Na podstawie wybranego przykładu złożonego systemu percepcji wyróżniono typowe problemy występujące podczas projektowania takich systemów. W celu ich rozwiązania dokonano formalizacji modelu przepływu danych przy użyciu metamodelu – kluczowego elementu w inżynierii opartej na modelach. Opracowany metamodel ułatwia rozwój takich systemów oraz umożliwia stworzenie graficznych narzędzi do edycji modeli konkretnych systemów percepcji, ich walidacji, a ostatecznie do automatycznego generowania szkieletów kodu. Ponadto, zidentyfikowano wymagania odnośnie środowiska wykonawczego omawianej klasy układów oraz porównano je z istniejącymi ramowymi strukturami programowymi opartymi na podejściu komponentowym.

Słowa kluczowe: robot, układ sterowania, percepcja, przepływ danych, metamodel, inżynieria oparta na modelach, systemy komponentowe

Piotr Trojanek, MSc Eng

He is a PhD student and works as the design engineer at the Institute of Control and Computation Engineering of Warsaw University of Technology. He gained experience working on control systems for mobile robots. Currently conducts research concerning multi-agent systems and applications of software engineering methods in robotics. For many years he has been associated with the WUT science group Bionik.
e-mail: piotr.trojanek@gmail.com

Maciej Stefańczyk, MSc Eng

He is a graduate of the Faculty of Electronics and Information Technology of Warsaw University of Technology. In 2010 he received the Eng title, in 2011, MSc Eng title, both with honours. In 2011 started work on doctorate concerning application of active vision together with knowledge base systems in a robot control system. His main scientific interests cover applications of visual information both in robotics and in computer entertainment systems.
e-mail: stefanczyk.maciek@gmail.com

Tomasz Kornuta, MSc Eng

He is a graduate of the Faculty of Electronics and Information Technology of Warsaw University of Technology. In 2003 he received the Eng title, in 2005, MSc Eng title. Since 2008 he is employed as Assistant at the Institute of Control and Computation Engineering, where he carries out the didactic work, and since 2009 heads the Laboratory of the Foundations of Robotics. Since 2005, as a part of a doctorate, he conducts research associated with designing the robotic systems which exploit the paradigm of active sensing for analysis of environment. His main scientific interests concern applications of visual information in robotics.
e-mail: tkornuta@ia.pw.edu.pl