European Robotics Forum

March 13, 2015 - Vienna (Austria)

Proceedings of the

1st Workshop on Model-Driven Knowledge Engineering for Improved Software Modularity in Robotics and Automation (MDKE) 2015

Klas Nilsson
Bernhard Rumpe
Ulrike Thomas
Andreas Wortmann
(Eds.)

Published May 2015

[MDKE15] K. Nilsson, B. Rumpe, U. Thomas, A. Wortmann (Eds.):
1st Workshop on Model-Driven Knowledge Engineering for Improved
Software Modularity in Robotics and Automation (MDKE).
In: RWTH Aachen, vol. RWTH-2015-01968, Online Proceedings. Vienna, Austria, March 2015.
www.se-rwth.de/publications

Organizers

Klas Nilsson	Lund University, Sweden
Bernhard Rumpe	RWTH Aachen University, Germany
Ulrike Thomas	German Aerospace Center (DLR e.V.), Germany
Andreas Wortmann	RWTH Aachen University, Germany

Program Committee

Anders Billesø Beck	Danish Technological Institute, Denmark
Matthias Lutz	University of Applied Sciences Ulm, Germany
Jacek Malec	Lund University, Sweden
Martin Naumann	Fraunhofer IPA, Germany
Jan Oliver Ringert	Tel Aviv University, Israel

Table of Contents

1 Preface

Robotics is a growing discipline where developers with different backgrounds and focuses work together. Nowadays, robotic systems usually consist of particular hardware platforms with specific software architectures. Knowledge about sensor interaction, constraints, and further information is coded within the architecture or certain modules. ROS is one step towards interfacing different robotic algorithms, software, etc. with a unique interface. Nevertheless, information about how to handle things is still coded in hardly re-usable solutions without harmonized interfaces and model descriptions.

Model-Driven Engineering already has huge impact on other fields. Currently, there are various approaches to the model-driven engineering of robotics applications, but widely applicable and thus accepted approaches have yet to emerge. Improving reusability and modularity of robotics applications requires to model the implicit knowledge encapsulated in current robotics modules and models explicitly. Applying knowledge engineering to model-driven robotics development will ease reuse and enable more efficient robotics software engineering.

The goal of this workshop was to bring together researchers from different fields: on the one hand languages and tools for model-driven engineering have been developed, on the other hand robotics systems consist of an increasing amount of heterogeneous software which contains new exploitable knowledge about their properties and composition. Demands on robotics software regarding reusability, reliability, expandability, and efficiency are very high, hence suitable modeling techniques are required to achieve high quality software products. To this end, the MDKE workshop provided a platform for presentation of novel approaches to tackle these challenges by means of model-driven engineering and knowledge engineering and how they reduce development cost and time. The scope of this workshop included, but was not limited to:

- Integration of knowledge engineering with software architecture and deployment modeling;
- Composition of software modules and components with the help of knowledge engineering;
- Modeling languages for knowledge engineering;
- Toolchains for the knowledge-aware modeling of robotic applications;
- Applications of knowledge engineering to models at run-time and self-*;
- Knowledge-Driven model transformation between different languages and frameworks;

The workshop was co-located with the European Robotics Forum 2015, which represents a forum for practitioners and researchers in robotics. We received five papers out of which three papers were selected for inclusion in the proceedings. The accepted papers covered from model-driven composition with explicated knowledge on communication, scheduling, and computation, to knowledge engineering techniques for abstracting low-level details, to abstraction of robot capabilities for high-level plan composition.

This was the first edition of the MDKE workshop and the attention received in terms of submissions and participants demonstrates that the topics are relevant to current robotics software engineering. Thus, we would like to thank the authors and the program committee for their hard and precious work.

April 2015

Klas Nilsson, Bernhard Rumpe, Ulrike Thomas, and Andreas Wortmann

The Composition Pattern for model-driven robot application development

Dominick Vanthienen and Herman Bruyninckx

KU Leuven, Dept. of Mechanical Engineering, PMA Division
Celestijnenlaan 300B, 3001 Leuven, Belgium
dominick.vanthienen@kuleuven.be **

Robots evolve to more autonomous systems that can operate in human-populated environments. Moreover, these new environments and related applications expect physical and cognitive interaction capabilities of the robot. This evolution requires the integration of many different fields of research to be carried out by a multitude of experts. In order to bring this integration challenge to a good end, a systematic and model-driven approach to software is needed, which should result in flexible, reusable, and adaptable software. This paper describes the Composition Pattern [3] as such a systematic approach to model robot applications. It is an architectural pattern to systematically structure, i.e. to contain and connect, types of behavior. It builds on and provides a constructive way to apply the 5C's approach to separation of concerns [2].

The pattern can be applied to analyse, model and implement applications. Earlier work introduced the Composition Pattern as software architectural pattern [4]. The focus of this paper and presentation will be on the modeling aspect.

1 The Composition Pattern

The Composition Pattern builds on four concepts, i.e. metamodeling, composition, hierarchy, and semantic context, explained in following paragraphs.

The first concept is **metamodeling** [1], an approach from Model Driven Engineering (MDE) which separates domain knowledge from its software implementation, and formalizes this knowledge in a meta-model. The meta-model forms a Domain Specific (modeling) Language (DSL) to describe specific models. The conformance of a model to one or more meta-models can be verified. From a model, an implementation in the software framework of choice can be hand-coded or generated.

The second concept is **composition**. The Composition Pattern defines a structure to compose *entities* (i.e. *models* in the context of this paper). Each of these entities is of one of the following types (of behavior).

– *Functional Entities* model continuous time and space behavior, i.e. 'data processing' or 'computations'. A Functional Entity can be a *composite (Func-*

** All authors gratefully acknowledge the financial support by the Flemish FWO project G040410N, KU Leuven's Concerted Research Action GOA/2010/011, and European FP7 project Factory-in-a-Day (grant agreement no. 609206).

tional Entity), composing other Functional Entities and 'support entities'. The latter consist of entities of the types listed below.

- *Monitors* model conditions to verify on data and the events to raise based on these conditions.
- A *Coordinator* models the actions to command from the other entities within a composite. It gives the composite the autonomy to handle certain situations locally. It is a pure 'event processor', it receives events from the other entities and triggers (commands) other entities with events.
- A *Scheduler* models the resource access and timing constraints on the different entities within a composite.
- *Configurators* model different sets of settings, i.e. data and parameters to apply to an entity when triggered by the Coordinator. In that sense it 'translates' the event to the parameters to apply. A Configurator forms the point where knowledge from a knowledge base can be introduced.
- A *Composer* models how the entities within a composite are connected.
- *Communicators* model constraints on how entities exchange data and events over these connections. Data and event communication is not limited to the boundaries of a composite.

Figures 1 and 2 detail how these entity types (behavior) is structured by the Composition Pattern.

The third concept is **hierarchy**. Since each Functional Entity can be (replaced by) a composite Functional Entity, a tree of entities emerges with a recurring structure. The tree depth level does not have to be identical for all branched of the tree. The composition is hierarchical, however data and event communication is not hindered by the hierarchy: events are broadcast and data is communicated through the boundary of a composite.

The fourth concept is **semantic context**. The entities within a composite need to use a shared vocabulary, i.e. its semantic context, to be able to interact. Making this context explicit is important to apply knowledge driven approaches. The support entities handle the translation from the context of a composite to its child functional entities. The composite and its semantic context forms a boundary to what the support entities have to 'know' and manage. However, this boundary does not imply information hiding; child entities can be introspected and hence reasoned about.

2 Use cases and discussion

In following example we model the 'reaching task' part of a robotic pick and place application using the Composition Pattern. This reaching task is a *Functional Entity* which is part of an application composite. It is in itself a composite Functional Entity, composing a controller (composite) Functional Entity, a trajectory planner (composite) Functional Entity, and 'support entities'. For example following support entities and example interactions of their implementations: A *Monitor* monitors the control error and signals when it reaches a certain limit. The *Coordinator* reacts on this event and sends out an event to adapt the gains

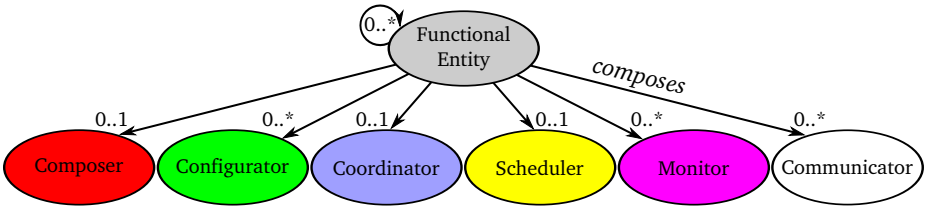

Fig. 1: Entity types within the Composition Pattern. A Functional Entity can compose a number of entities of each type, indicated by the arrow and the cardinality. A Functional Entity can be a composite Functional Entity, composing other Functional Entities, as indicated by the reflective arrow.

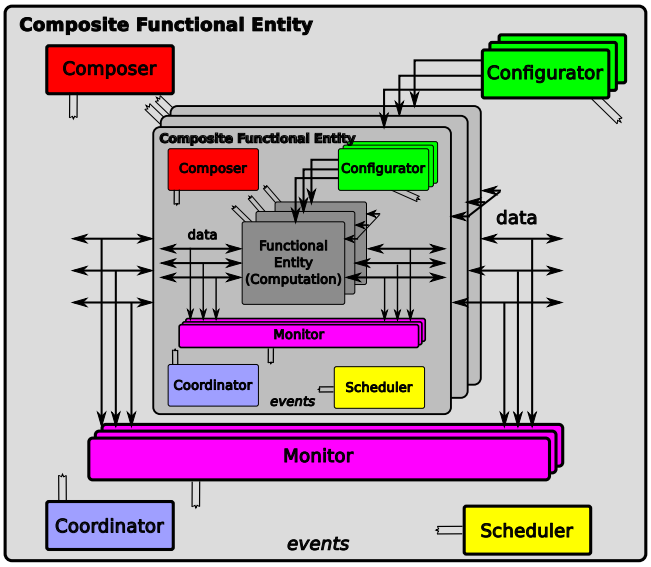

Fig. 2: Structure of the Composition Pattern. Blocks indicate entities, block colors indicate entity types, and darker shades of grey indicate deeper levels in the hierarchy. Arrows indicate data communication and double lines indicate event broadcasting over a 'bus system' (only partially drawn).

of the controller. The *Configurator* on its turn reacts to the event of the coordinator by setting a new control gain. The *Scheduler* of the reaching task composite first triggers the planner to generate a new setpoint or a complete trajectory, before it triggers the controller to track the setpoint or trajectory. The *Composer* models that the planner setpoint should be communicated to the controller. The *Communicator* models that the setpoint of the planner needs to be communicated in real-time (assuming online trajectory generation).

The presentation will detail this and other examples of the application of the Composition Pattern and its advantages over classical (e.g. layered) architectures to address the integration challenge in robotics. The Composition Pattern has

been applied to model robot applications that make use of constraint-based programming, for which a DSL is made available [3].

References

1. Bézivin, J.: On the unification power of models. Software and Systems Modeling 4(2), 171–188 (2005)
2. Bruyninckx, H., Klotzbücher, M., Hochgeschwender, N., Kraetzschmar, G., Gherardi, L., Brugali, D.: The BRICS Component Model: A model-based development paradigm for complex robotics software systems. In: 28th ACM Symposium On Applied Computing. pp. 1758–1764 (2013)
3. Vanthienen, D.: Composition Pattern for Constraint-based Programming with Application to Force-sensorless Robot Tasks. Ph.D. thesis, Dept. Mech. Eng., Katholieke Univ. Leuven, Belgium (January 2015)
4. Vanthienen, D., Klotzbücher, M., Bruyninckx, H.: The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming. J. Softw. Eng. in Robotics 5(1), 17–35 (2014)

Abstracting Away Low-Level Details in Service Robotics with Fuzzy Fluents

Stefan Schiffer[1], Alexander Ferrein[2], and Gerhard Lakemeyer[1]

[1] Knowledge Based Systems Group (KBSG)
RWTH Aachen University, Aachen, Germany
{schiffer,gerhard}@cs.rwth-aachen.de

[2] Mobile Autonomous Systems & Cognitive Robotics Institute (MASCOR)
FH Aachen University of Applied Sciences, Aachen, Germany
ferrein@fh-aachen.de

Abstract. In domestic service robotic applications, complex tasks have to be fulfilled in close collaboration with humans. We try to integrate qualitative reasoning and human-robot interaction by bridging the gap in human and robot representations and by enabling the seamless integration of human notions in the robot's high-level control. The developed methods can also be used to abstract away low-level details of specific robot platforms. These low-level details often pose a problem in re-using software components and applying the same programs and methods in different contexts. When combined with methods for self-maintenance developed earlier these abstractions also allow for seamlessly increasing the robustness and resilience of different robotic systems with only little effort.

1 Introduction

In our previous work we focused on the design of intelligent behaviours in domestic service robotics applications. We developed the logic-based high-level robot programming and plan language Readylog [2, 1], which allows for specifying complex behaviours in a quite intuitive fashion. Readylog supports, for instance, decision-theoretic planning, online-passive sensing and is able to deal with uncertainty. It is a member of the GOLOG language family and is based on the Situation Calculus [5, 6]. We successfully applied Readylog in dynamic real-time domains such as robotic soccer and also on a domestic service robot platform; in particular we participated successfully at RoboCup@Home competitions in the past [8, 12].RoboCup@Home [15, 16, 14] is a robot competition for domestic service robots under the roof of the RoboCup Federation. The robots have to fulfill tasks ranging from rather simple functions such as guiding a human through an apartment over serving drinks to complex missions like cooking meals or acting as a host at a party. Robots in @home are fully autonomous and they must be accessible and usable by non-expert users; this is most often realized via natural language-based control.

2 Qualitative Notions and Self-Maintenance

When operating in human-populated environments in collaboration with humans, it becomes obvious that numeric representation of the robot cannot quite be understood by a human. Humans rather tend to use qualitative representations of space than numerical ones. While humans are very good at interpreting qualitative distances such as "far" or "close" or orientations such as "the cup is left on the table behind the coffee pot" robots are not. One possibility (for robots) to deal with such qualitative notions is to use fuzzy predicates. A fuzzy predicate associates a number of quantitative values from a given domain (e.g. distances $1, 2, \ldots$) to a linguistic term (e.g. "far" or "close"). The association of quantitative and qualitative values is specified with a so-called membership function. Each quantitative value has a membership value, which defines, to what degree a value falls into a certain fuzzy category. An example of a membership function for qualitative unit distance is depicted in Figure 1. We extended the language Readylog with qualitative notions to be able to instruct the robot with command such as "get me the left cup on kitchen table". To this end, we developed fuzzy representation in the Situation Calculus [3, 11, 10] and integrated a control strategies into Readylog [9, 4].

Another challenge for robots working in human environments is that they need to operate for extended periods of time. Therefore, they need to be robust against internal shortcomings and they should continue to be functional even in the face of failures. More precisely, robots should be able to handle and deal with as many errors by themselves as possible. For that purpose, we developed a method to realize a limited form of self-maintenance [13]. It uses explicitly formulated constraints that associate actions with desired internal states of the robot. Before executing an action the robot checks whether the constraints are met and it schedules appropriate counter measures if this is not the case. For example, one could specify that the robot should only drive around if the collision avoidance module is running and in working condition. If then, the robot's next action is to move to a certain position but the collision avoidance component is non-functional, the system would restart it before attempting to execute the move action.

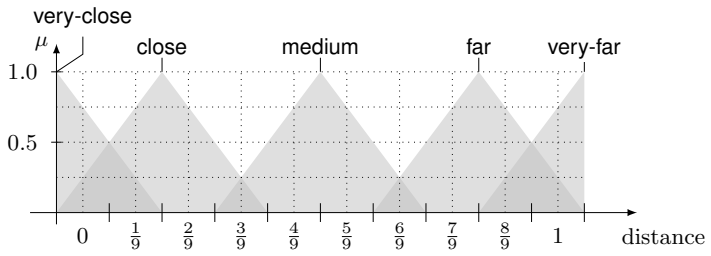

Fig. 1. Membership function for qualitative (unit) distance

3 Fuzzy Fluents for Abstracting Away Low-level Details

The basic motivation behind using fuzzy fluents in the high-level controller of our domestic service robots was to express qualitative facts about the world and reason with them. For translating qualitative notions "left of" or "close" into a quantitative representation that the low-level robot control system can use to perform its actions, we use fuzzy predicates for dividing continuous spaces (e.g. distance and orientation) into a number of equivalence classes. An additional nice feature of fuzzy predicates is that certain quantitative values can belong to more that one class at a time and that complex queries can be evaluated (see [3, 11, 10] for technical details). The linguistic classes and the membership functions can also be used to model different contexts. For instance, "fast" may mean 100 km/h in a self-driving car, while for a humanoid robot it means "100 cm/s". With modelling the membership function differently in different contexts, these details can he hidden away from the high-level reasoning system. This allows for using the same high-level controller in different contexts and on different platforms by simply providing an appropriate membership function of the qualitative terms used in the program. Extending this concept of "abstracting away" quantitative details of the low-level system can also be used to model self-properties of the robot or to allow a form of hardware abstraction of the low-level hardware system.

As an example application, consider a delivery robot scenario. Two different robot platforms might be equipped with different drive concepts, different payload capabilities, and different energy components. As a result, these two robots will have different maximum speed and they will be able to travel different total distances before they need to recharge. With our qualitative abstractions in place, both robots could use the same control program to conduct their delivery service. The only difference would be the individual membership functions for relating numerical values to their qualitative counterparts that are being used in the high-level program. Also, different payloads would automatically be accounted for when the control programs and constraints mentioned above would be using notions such as "heavy" by appropriate membership functions for different hardware platforms.

4 Discussion

In this paper we illustrated the use of qualitative notions for abstracting away low-level details in service robotic applications. Qualitative notions are realized by fuzzy fluents using membership functions that specify how much a numerical value belongs to a qualitative category. By using only qualitative terms in high-level control programs and also for formulating constraints that exist between actions and internal states of the robot, differences for varying platforms can be accounted for by just using different membership functions. This facilitates creating high-level control that is easily transferable from one robot platform to another. The proposed method could also be used with learning individual qualitative abstractions for different persons [7] or platforms.

References

1. Ferrein, A.: Robot controllers for highly dynamic environments with real-time constraints. Künstliche Intelligenz 24(2), 175–178 (2010)
2. Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains. Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics 56(11), 980–991 (2008)
3. Ferrein, A., Schiffer, S., Lakemeyer, G.: A fuzzy set semantics for qualitative fluents in the situation calculus. In: Proceedings of the International Conference on Intelligent Robotics and Applications (ICIRA'08), LNCS, vol. 5314, pp. 498–509. Springer (October 15-17 2008)
4. Ferrein, A., Schiffer, S., Lakemeyer, G.: Embedding fuzzy controllers into golog. In: Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-09). pp. 498–509. IEEE Press (August 20-24 2009)
5. McCarthy, J.: Situations, actions and causal laws. TR, Stanford University (1963)
6. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
7. Robinson, V.B.: Individual and multipersonal fuzzy spatial relations acquired using human-machine interaction. Fuzzy Sets and Systems 113(1), 133–145 (2000)
8. Schiffer, S., Ferrein, A., Lakemeyer, G.: Football is coming home. In: Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots (PCAR'06). pp. 39–50. ACM, New York, NY, USA (November 27-28 2006)
9. Schiffer, S., Ferrein, A., Lakemeyer, G.: Fuzzy representations and control for domestic service robots in golog. In: Jeschke, S., Liu, H., Schilberg, D. (eds.) Proceedings of the 4th International Conference on Intelligent Robotics and Applications (ICIRA 2011). LNCS, vol. 7102, pp. 241–250. Springer (2011)
10. Schiffer, S., Ferrein, A., Lakemeyer, G.: Reasoning with qualitative positional information for domestic domains in the situation calculus. Journal of Intelligent and Robotic Systems 66(1-2), 273–300 (2012)
11. Schiffer, S., Ferrein, A., Lakemeyer, G.: Caesar: An Intelligent domestic Service Robot. Intelligent Service Robotics 5(4), 259–273 (2012)
12. Schiffer, S., Niemüller, T., Doostdar, M., Lakemeyer, G.: Allemaniacs@home 2009 team description. In: Proceedings CD RoboCup 2009. Graz, Austria (2009)
13. Schiffer, S., Wortmann, A., Lakemeyer, G.: Self-Maintenance for Autonomous Robots controlled by ReadyLog. In: Ingrand, F., Guiochet, J. (eds.) Proceedings of the 7th IARP Workshop on Technical Challenges for Dependable Robots in Human Environments. pp. 101–107. Toulouse, France (June 16-17 2010)
14. Wisspeintner, T., van der Zant, T., Iocchi, L., Schiffer, S.: Robocup@home: Scientific Competition and Benchmarking for Domestic Service Robots. Interaction Studies. Special Issue on Robots in the Wild 10(3), 392–426 (2009)
15. van der Zant, T., Wisspeintner, T.: RoboCup X: A Proposal for a New League Where RoboCup Goes Real World. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005: Robot Soccer World Cup IX. LNCS, vol. 4020, pp. 166–172. Springer (2006)
16. van der Zant, T., Wisspeintner, T.: Robotic soccer. In: Lima, P. (ed.) Robotic Soccer, chap. RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications, pp. 521–528. I-Tech Education and Publishing (2007)

Architecture for Efficient Reuse in Industrial Settings

Rasmus Hasle Andersen[1], Anders Billesø Beck[1], Lars Dalgaard[1], and John Hallam[2]

[1] Danish Technological Institute,
Robot Technology,
Forskerparken 10, 5230 Odense M, Denmark,
Corresponding Author: raha@dti.dk
[2] University of Southern Denmark,
Mærsk Mc-Kinney Møller Instituttet,
Campusvej 55, 5230 Odense M, Denmark

Abstract. In this work we introduce the architecture used in the *DTI Robot CoWorker*, which is a flexible robotics software platform supporting easy extension of functionality, flexible component integration and intuitive human robot interaction – all with a strong focus on reusability. We employ a knowledge-based approach through a hierarchical *Hardware-Independent Capability Representation*, with which we are able to create reusable and hardware-independent process descriptions.

Keywords: Hardware-Independent Capability Representation, Reuse in Robotics, Robotic Architecture

1 Introduction

To strengthen the competitiveness of small and medium enterprises (SMEs), the creation of agile robots, which can easily be reconfigured for new tasks and operated by existing personnel, is needed and therefore system architectures supporting this are needed, so that flexible and assistive robotic installations will become viable in SMEs.

We here present our approach for realising such an architecture, namely the architecture of the *DTI Robot CoWorker*, which is a flexible robotics integration platform supporting flexible component integration and intuitive human robot interaction, and enables reuse of (sub-)process descriptions between hardware to solve different tasks. We use ROS [3] as middleware in the current implementation but from a design perspective, the architecture is middleware-independent.

We employ a knowledge-centred approach, by grounding system functionality following a hardware-independent representation. We call these grounded capabilities *Actions*. We have previously introduced this Action Framework in [1]. Actions serve to ground capabilities and abstract low-level complexities so that end-users and system operators need only worry about what is essential for them: THE PROCESS. Additionally we use a centralized Knowledge Base, storing and processing all semantically annotated information within the system.

2 Architecture Description

We propose a 4-layered architecture which has been designed with attention to the principle of "separation of concerns" introduced by Radestock and Eisenbach [4] such that independent system concerns are addressed explicitly. Communication, Computation, Configuration and Coordination were originally identified as the essential concerns and later Composition was added by Prassler et al. [2] as a fifth concern, which is likewise addressed at design time. We consider all five concerns in our architecture.

An overview of the architecture is given in Fig. 1, from which it is evident that the four layers of the architecture are `Computation`, `Configuration`, `Coordination` and `User Interaction`. A top-down hierarchical dependency exists between the layers, which ensures a proper division of responsibilities, and enables easy updates and extensions of individual layers.

In addition to the four separated layers, a `Knowledge Base` is present which is directly accessible from the Configuration and Coordination layers. User Interaction and Computation have access through these, to sustain separation. The Knowledge Base (KB) is the central information storage and interpretation module. All persistent information generated within the system is stored in the central KB, which handles the translation between raw-data and semantic data, ensuring that all modules have a common interpretation of raw data.

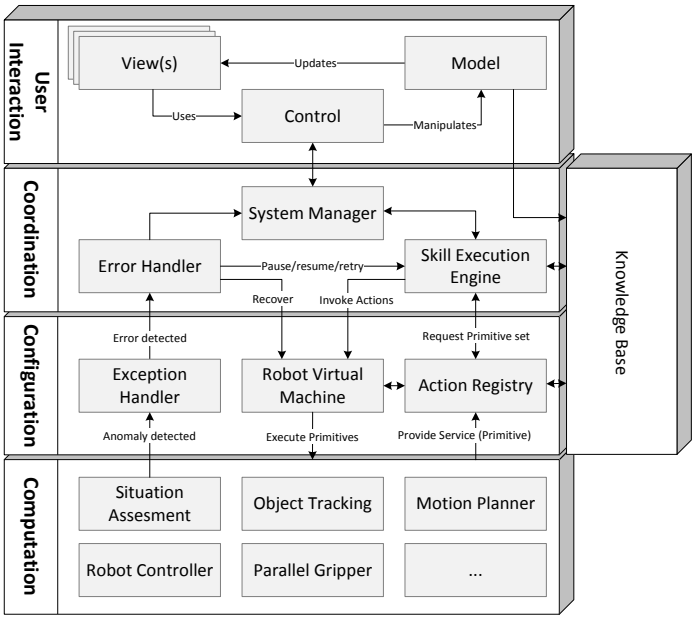

Fig. 1. The layered architecture of the DTI Robot CoWorker platform. The architecture consists of four layers and a shared Knowledge Base.

Computation is the layer in which all functionality performing actual computations is located, including hardware drivers, software interfaces, algorithms and so forth. Functionality is made available to the system through Computational Units by use of a service-based capability registration process provided by the Configuration Layer.

Coordinated execution and monitoring is addressed in the Coordination layer as well as error recovery in unexpected situations. The hardware-independent process descriptions, expressed as a graph of Actions, are executable through an online symbolic interpreter, which we call the *Robot Virtual Machine* (RVM). The RVM translates the symbolic Actions to specific capabilities provided by the connected units. At the top most level, we have the User Interaction Layer, which is the place where process descriptions are created through various interaction interfaces. The interfaces in the Interaction Layer are developed following the Model-View-Control pattern.

3 Discussion

We have successfully deployed the proposed architecture to more than seven different hardware platforms, in both commercial and research related activities, working with industrial manipulators and mobile platforms. We have tested more than 20 different tasks, and while doing so we have integrated a large amount of sensors and actuators, all supporting the dynamic registration process.

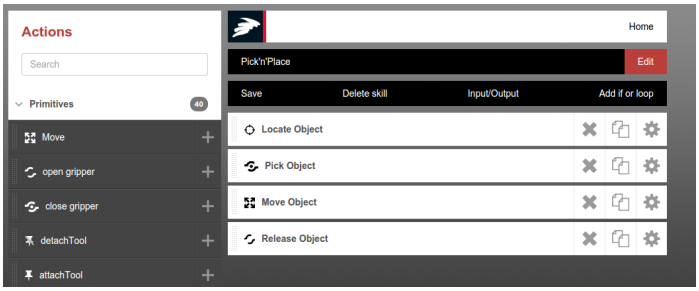

Fig. 2. An example of an end-user based instruction interface.

Reusability has been a main concern, and is two-fold within the architecture. One aspect focusses on reusing the actual functionality between platforms/installations, while the other focusses on reusing hardware independent Actions between tasks. Functionality reuse has been addressed by utilizing the grounded system capabilities provided by the Action Framework, such that external modules can easily be interchanged between physical platform configurations. This greatly reduces the integration efforts. Action reuse is realized by creating a hierarchy of the hardware independent actions, thus decoupling Actions from hardware, which can therefore be reused between tasks when parts

of the process are the same (even though the hardware is not). This reduces the required instruction time as the available library of Actions increases over time.

We enable non-technical end-users to intuitively specify new processes using the grounded capabilities without having to consider all complexities as these have been abstracted away, by using an intuitive end-user oriented GUI (an example of an end-user interface is given in Fig. 2. By simply arranging the available *Actions* in the desired order, a complete process description can be created without concerning one-selves with the underlying technical challenges of the connected hardware. At the same time expert users can create, via a different interface, very complicated applications beyond the scope of the end-user interface, thereby the system empowers experts as well as non-technical end-users.

The architecture facilitates end-user driven application-development, supports fast reconfiguration and enables adaptation to changes in production. Thereby it addresses many of the needs in modern robotic applications. By the realization of such intuitive and flexible robotic system, it becomes viable to automate high-mix low-volume production, which traditionally has not been automated.

Acknowledgments

The research leading to these results has been funded in part by the Danish Ministry of Science, Innovation and Higher Education under grant agreement #11-117525 and from the European Union's seventh framework program (FP7/2007-2013) under grant agreements #608604 (LIAA: Lean Intelligent Assembly Automation), and #287787 (SMErobotics: The European Robotics Initiative for Strengthening the Competitiveness of SMEs in Manufacturing by integrating aspects of cognitive systems).

References

1. Andersen, R.H., Sølund, T., Hallam, J.: Definition and Initial Case-Based Evaluation of Hardware-Independent Robot Skills for Industrial Robotic Co-Workers. In: International Symposium on Robotics. pp. 101–107 (2014)
2. Prassler, E., Bruyninckx, H., Nilsson, K., Shakhimardanov, A.: The use of reuse for designing and manufacturing robots. Tech. rep. (2009)
3. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T.B., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009) (2009)
4. Radestock, M., Eisenbach, S.: Coordination in evolving systems. In: Trends in Distributed Systems CORBA and Beyond, Lecture Notes in Computer Science, vol. 1161, pp. 162–176 (1996)