A Survey on Domain-Specific Languages in Robotics

Arne Nordmann[1,2], Nico Hochgeschwender[3], and Sebastian Wrede[1,2]

[1] Cognitive Interaction Technology Excellence Cluster (CITEC), Bielefeld University, Germany
[2] Institute for Robotics and Cognition (CoR-Lab), Bielefeld University, Germany
[3] Department of Computer Science, Bonn-Rhein-Sieg University, Germany

Abstract. The design, simulation and programming of robotics systems is challenging as expertise from multiple domains needs to be integrated conceptually and technically. Domain-specific modeling promises an efficient and flexible concept for developing robotics applications that copes with this challenge. It allows to raise the level of abstraction through the use of specific concepts that are closer to the respective domain concerns and easier to understand and validate. Furthermore, it focuses on increasing the level of automation, e.g. through code generation, to bridge the gap between the modeling and the implementation levels and to improve the efficiency and quality of the software development process. Within this contribution, we survey the literature available on domain-specific (modeling) languages in robotics required to realize a state-of-the-art real-world example from the RoboCup@Work competition. We classify 41 publications in the field as reference for potential DSL users. Furthermore, we analyze these contributions from a DSL-engineering viewpoint and discuss quantitative and qualitative aspects such as the methods and tools used for DSL implementation as well as their documentation status and platform integration. Finally, we conclude with some recommendations for discussion in the robotics programming and simulation community based on the insights gained with this survey.

1 Introduction

Model-driven and domain specific development methods are recognized to cope with the challenges of building complex heterogeneous systems in domains such as aerospace, telecommunication and automotive [1] which face similarly complex integration and modeling challenges as advanced robotics. In the last years, this approach was actively adapted to the robotics domain to handle the complexity of robotics systems and help with the separation of concerns regarding the *functional architecture* and *software architecture*. One goal is to support the development and ease design space exploration. This requires to support the entire experimental toolchain ranging from purely functional modeling to software architectural and technical aspects such as software deployment.

The purpose of this survey is to report on the state of the art in domain-specific languages in robotics, and provide an overview on sub-domains relevant for programming and simulation of robotics applications that are already supported through domain-specific modeling methods. Similar surveys, yet for a wider scope, have been conducted by Biggs and MacDonald [2] as well as Van Deursen et al. [1]. One targeted audience of this survey is the potential DSL *users*, the domain experts looking for method and tool

support in their domain. This survey provides means to assess availability and usability of the DSLs to formulate their experimental or system hypotheses and generate reproducible experiments. We also target DSL *developers* and system integrators in robotics, to provide an overview on the state of the art, common solutions and best practices, and foster scientific exchange and community building inside the domain.

The paper starts with a short introduction to the core concepts of domain-specific modeling in Section 2 and defines a minimal set of methodological requirements on DSL approaches to be included in the systematic review. Subsequently, Section 3 analyses the targeted domain and exemplifies the use of domain-specific modeling techniques along a reference example from the RoboCup@Work competition. Section 4 explains how the literature survey was conducted along a defined protocol, while Section 5 classifies and quantitatively assesses the found literature, also providing an overview of several non-functional aspects. On that basis, Section 6 discusses some of these aspects along key publications and identifies best practices both for DSL engineering and DSL-related publications and documentation. Section 7 summarizes the survey and propose recommendations for further community development (exchange among researchers) and discusses requirements for re-use of knowledge provided with domain-specific models in the area of simulation and programming of robotic applications.

2 Domain-specific Languages

In order to perform a systematic review on domain-specific modeling for simulation and programming of robotics applications, a necessary prerequisite is to define what we consider a domain-specific (modeling) language and what we don't. According to van Deursen et al. [1], a DSL is defined as a "*programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain*". The abstractions and notations must be "*natural/suitable for the stakeholders who specify that particular concern*" [3]. These definitions already highlight two fundamental characteristics of well-designed DSLs: their expressive power targeted a specific domain and the definition of formal notations intuitively understandable for domain experts while being machine processable, eventually yielding executable models of robotics applications.

Model-driven software development with DSLs aims to extract agreed-upon syntax and semantics from the problem domain, e.g., by reviewing existing code examples and APIs, through the analysis of formal descriptions found in the literature or the application of further analysis patterns [4]. Based on the results of these domain analysis steps, the identified abstractions and desired notations can be realized as a DSL. Instead of hiding the domain concepts in a compilation unit implemented with traditional programming techniques, the DSL approach provides the specific abstractions at the model level. In contrast to *General Purpose Languages* (GPL) such as C++, Java, or Python, DSLs usually contain only a restricted set of notations and abstractions. Compared to *external* DSLs that define their own syntax and semantics, so-called *internal* DSLs are embedded in extensible general purpose languages such as Lua, Racket or Ruby. They extend the syntax and potentially the semantics of the host language with domain-specific notations and abstractions. This adds the expressive power of the DSL

to the GPL. While internal DSLs typically rely on (and are bound to) the execution semantics of their host language, external DSLs are transformed to a format that directly allows execution on a target platform or interpretation, e.g., through a virtual machine.

Similarly, *Domain-specific Modelling Languages* (DSML) that use graphical notations must be differentiated from general purpose modeling languages such as UML or SysML. While it is still possible to add domain-specific abstractions to these languages, e.g. using UML Profiles (cf. MARTE [5] to describe and analyze real-time systems), adding domain-specific notation to graphical modeling languages is much harder.

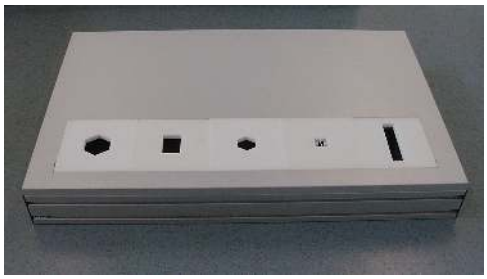
In order to efficiently implement and apply a DSL approach for the development of robotics systems and to fully exploit its benefits, DS(M)Ls are typically realized in toolchains tailored to model-driven development such as the Eclipse Modelling Project [6]. These so-called *language workbenches* such as MPS [7] offer extensive support for the development of the DSLs themselves and for the actual system modeling tasks performed by a language user. DSLs developed in these environments facilitate the users modeling tasks typically with textual and/or graphical editors with rich code completion and dynamic constraint checking. Furthermore, these environments provide extensions points to plug-in required model-to-model (M2M) and model-to-text (M2T) transformations in order to generate code from system models that integrates with the overall environment used for the development of a robotics application.

The above mentioned aspects comprise fundamental characteristics that need to be addressed in a DSL approach. Hence, the DSL approaches considered in this survey i) must provide a language definition or meta-model, e.g. Ecore or (E)BNF, ii) must be textual (internal or external) or graphical languages, iii) must provide an example of their concrete syntax (notation), iv) should[4] explain how a mapping to a target technology is achieved. While these criteria are formulated from a software engineering perspective, the most important criteria to decide whether a paper or article is included in the systematic review is whether or not it targets a relevant concern in the robotics domain. In order to allow for a more fine-grained mapping of DSL-related publications that conform to the criteria introduced above to the robotics domain the following section identifies a set of relevant sub-domains along a reference example that we consider particularly relevant and mature in the context of simulation and programming of robotics systems.

3 Domain Analysis

To exemplify the domain of this survey we use the Precision Placement Test (PPT) from the RoboCup@Work competition. It is a new competition in RoboCup that targets the use of robots in industrial scenarios where robots cooperate with human workers and machines for complex tasks ranging from manufacturing, assembly, automation, and parts-handling up to general logistics. The PPT exemplifies the complexity and huge variability of competences and capabilities required to develop today's robot applications. We consider this example to represent the current state of the art involving mature robotics disciplines so that we expect to find consolidated knowledge in the

[4] This relaxation allows to include purely analytical approaches in the review, which we also consider relevant contributions.



(a) PPT platform used in RoboCup@Work.

```
Robot Fancy {
RobotBase FancyBase {
    inertia_params{...}
    children { link1 via jA}
}
link link1 {
    id = 1
    inertia_params {
        mass = 1.0
        CoM = (0.5, .0, .0)
        Ix=0.0025  Iy=0.084 Iz=0.084
          Ixy=0.0  Ixz=0.0 Iyz=0.0
```

(b) Kinematics DSL example [10]

form of DSLs. In the following we will explain the PPT and also synthesize a core set of subdomains[5] which are relevant in solving the task and later used for classification of the surveyed publications. We are aware that this list is non-conclusive, but focus on these for the sake of brevity.

The main objective of the PPT is to assess the robot's ability to grasp and place objects into object-specific cavities (see Fig. 1a). The objects are taken from a set of (a priori known) standardized industrial objects such as screws, nuts, bolts and profiles. For the test a single robot is placed in front of a service area which stores the objects to be manipulated. The objective is to pick each object and place it in the corresponding cavity. Once the objects are picked up and placed or the time is over the task ends. To simulate and solve the problem one usually first needs to know the Robot Structure (1) of the target robot platform. This comprises representation of the actual physical realization of robot platforms (e.g., mobile base and manipulator) in terms of their mechanical structure and kinematic as well as dynamic properties. This subdomain roughly corresponds to Part B of Springer Handbook of Robotics (*Robot Structures*). Furthermore, Coordinate Representations and Transformations (2) between parts of the robot and its environment are required to enable computation of position, force, and velocity of the robot joints. This subdomain roughly corresponds to Part A in the Handbook of Robotics (*Robotics Foundations*). Exemplary DSL representatives for this subdomain are URDF [8] and the work of Frigerio et al. [9], shown in Fig. 1b. In general, the PPT demands advanced Perception (3) and Reasoning and Planning (4) abilities, namely to recognize and match objects and the correct cavities. Further, precise Manipulation and Grasping (5) abilities are required, namely to grasp and place the object in such a manner that it fits into the cavity. These subdomains roughly correspond to Part C (*Sensing and Perception*), Part A Chapter 9 (*AI Reasoning Methods for Robotics*), and Part D Chapter 28 (*Grasping*) in the Handbook of Robotics.

For each sub-task (object/cavity *detection/recognition* and object *manipulation*) there are several options to approach the problem which all require Coordination (6) primitives such as finite-state machines. For instance, the placement of objects in the cavities can be achieved through first perceiving and computing the position of the cavities and then generating a plan yielding a pose where the object can be dropped in the cavity. A more control-based approach is to compute an approximately position of the cavity and then placing the object on the arena and sliding the object into the cavity by

[5] Subdomains will be marked in the format Name (#) where # is a continuing number.

means of force-feedback. This approach demands advanced Motion Control (7) abilities in order to cope with uncertainties in the environment and fulfill constraints such as force-limits which corresponds to roughly to Part A Chapter 7 (*Force Control*) in the Handbook for Robotics. Example DSLs for this task are TFF [11] and iTaSC [12]. The presented capabilities all need to be integrated in an overall Architecture (8) with Components (9) as the basic building blocks which are preferable re-usable also for other applications. This domain corresponds roughly to Part A Chapter 8 (*Robotic Systems Architectures and Programming*) in the Handbook of Robotics.

4 Process

The selection of the publications for this survey focused on publications that developed domain-specific languages (DSL) to conceptualize aspects of the introduced domain or support certain research or engineering aspects. To find these, we scanned relevant robotics and software conference proceedings for a set of keywords. For the actual selection process we defined two inclusion criteria (IC) and two exclusion criteria (EC):

IC1	In proceedings of the International Conference on Intelligent Robots and Systems (IROS), International Conference on Robotics and Automation (ICRA), International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), Robotics: Science and Systems Conference (RSS), workshop on Software Development and Integration in Robotics (SDIR) or the workshop on Domain-Specific Languages in Robotics (DSLRob) **and** search matching one of the keywords *"domain-specific language"*, *"domain-specific modeling language"*, *"generative programming"*, *"specification language"*, *"description language"*, or *"code generation"*.
IC2	in proceedings of the Code Generation Conference (CG) and the International Conference on Generative Programming: Concepts & Experiences (GPCE) **and** search matching one of the keywords *"robot"* and *"robotics"*.
EC1	DSL does not model or support aspects of the introduced domain.
EC2	Either is no DSL or publication not complying with our definition from section 2, e.g. notation not documented via grammar or example.

Table 1: Inclusion criteria (IC) and exclusion criteria for publications in this survey.

IC1 included 208 publications after removing duplicates, IC2 included additional 2 publications, adding up to a total of 210 publications. We consider all publications from IC1 to pass EC1, as they passed through a review process of robotics conferences. EC2 filtered 169 publications, leading to a total of 41 publications that will be analyzed and discussed in the remainder of this survey.

5 Analysis

This section assesses technical aspects across the publications in this survey. The publications are analyzed along their subdomains according to Section 3 and regarding their temporal distribution as well as their utilized tool or method.

Subdomains A first analysis we did was grouping DSLs and their publications by common semantics, abstractions and use-cases according to the domain example introduced in Section 3. This is intended to serve as a map for potential DSL users as well as foster discussion and reuse of languages and the underlying models for DSL developers. The categorization is given in Table 2 and references the subdomains introduced in Sec-

	Subdomain(s)								
	① Struct.	② Transf.	③ Perc.	④ Plan.	⑤ Manip.	⑥ Coord.	⑦ Ctrl.	⑧ Arch.	⑨ Comp.
Muehe2010						□	■		
Akim2010						■			□
Reckhaus2010						■			
Frigerio2011	■	□					■		
Trojanek2011						■		■	
Anderson2011	■								
Romero2011									■
Ingles2010						■			
Angerer2012	□					■			
Klotzbuecher2012						■			
Laet2012		■							
Nordmann2012							■	■	□
Buchmann2013	■		□				■		
Hochgeschw2013			■						
Blumenthal2013	■								
Dantam2012						■			
Kilgo2012									■
Dhouib2012						■		■	■
Brugali2012						■		■	■
Vanthienen2013						■			
Klotzbuecher2011						■	■		
Loetzsch2006				■		■			
Steck2011						■			
Anderson2012	□								□
Haas2012						■			
Dai2002						■			
Manikonda1995						■	■		
Kunze2011	■			□					
Kanayama2000		■					■		
Rosa2007									□
Graves1999	■								
Tousignant2012						■			
Murray1992	■						■		
RuggGunn1994				■					
KressGazit2010	□			□		□			
Ljungkrantz2007	■					■			
Thomas2013						■	■		
Ferstenberg1986							■		
Bordignon2010	■								

Table 2: Overview of the surveyed DSLs and their subdomains: ■ = in focus, □ = partially.

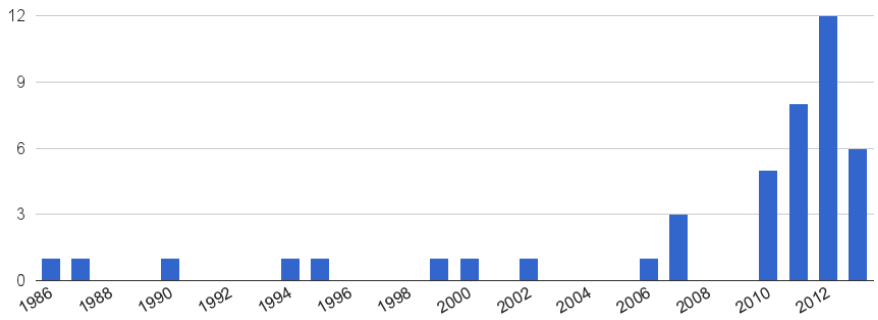


Fig. 1: Temporal distribution of the publications in this survey ranging from 1986 to 2013.

tion 3. The table is an initial version, though, that we intend to update continuously and maintain online[6], enriched with the aspects discussed in the remainder of this survey. The left-most column of Table 2 references this online table, as space constraints unfortunately don't allow citation of all surveyed publications.

The initial grouping by subdomains seems reasonable, as the assignment of most of the DSLs and publications to subdomains was quite straight-forward. However, the number of publications per subdomain varies significantly. Whereas we found no DSLs in the subdomain of Manipulation and Grasping (5), Coordination (6) for example seems to be quite well-covered as over 20 publications entirely or partially belong to this subdomain. The Robot Structure (1) and Motion Control (7) subdomains are also well-covered with roughly 10 publications each. These numbers may indicate how well-explored or even stable a discipline or subdomain is.

Temporal distribution Model-driven and domain-specific approaches are on the rise in robotics, we plotted the temporal distribution of the publications in this survey, as shown in Fig. 1. The distribution clearly supports a positive trend of DSLs in robotics respectively their publications, especially since around the year 2010 with several publications per year. This is equivalent to the start of the DSLRob workshop, the numbers, however, clearly exceed the number of DSLRob publications per year, proving that this is also a trend on general robotics conferences.

Methods / Tools This section analyzes the methods and tools that were used for development of the surveyed DSLs, as far as this is assessable via the publications or referenced documentation. This comprises tool support for developing external DSLs, as well as development of internal DSLs, as shown in Fig. 2.

The majority of DSLs assessed with this survey is realized as an external DSL. Although different tools and methods are being used, Fig. 2 shows that the *Eclipse Modeling Project* [6] (EMP) seems to be quite widely used. It therefore seems to be a good integration point and opportunity for DSL compatibility in this domain. Some

[6] http://cor-lab.org/robotics-dsl-zoo

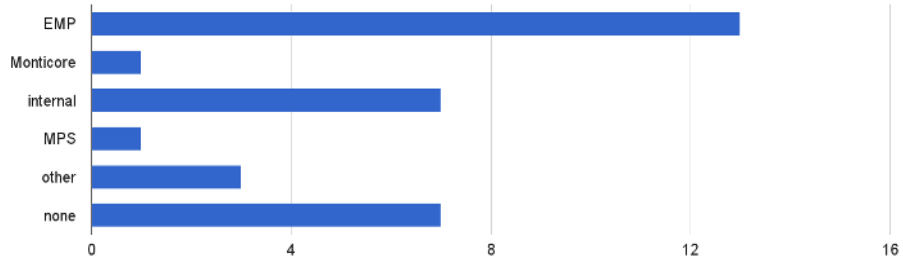


Fig. 2: Tools and methods used for the DSLs in this survey.

publications mentioned the extensive EMP tool-support explicitly as a big advantage of developing DSLs inside the Eclipse framework [13, 14], or the possibility of language re-use [15], as approaches within EMP share the same representation (Ecore). 7 publications in this survey developed their DSLs as internal DSL, for example in Lisp [16], and Lua [12]. 7 of the assessed publications developed DSLs and their tool-chain manually and without the aforementioned tool support, e.g. with custom parsers and tools.

6 Discussion

This section highlights different aspects of the surveyed DSLs as well as their publications that we think are important for i) language developers to enable language re-use, interoperability and discussing the core concepts, as well as ii) language users to allow assessing the availability and usability of the DSLs. We show different approaches to extract best practices in terms of documentation, accessibility and evaluation of robotics DSLs to make suggestions to the community. The need for this became clear during analysis of the publications for this survey, as lots of the aspects discussed here were largely undocumented and/or hard to access.

Accessibility and Documentation An important factor for re-use of DSLs, scientific exchange and community building around DSLs in robotics is their accessibility and documentation. This comprises several factors like technical accessibility (e.g. download of the language or models), licensing, and documentation of the DSL, its usage and execution context. Only a subset of the DSLs in this survey is documented in a way that would allow interfacing with it, e.g. with a documented meta-model [14, 17, 18]. While some publications give hints on the meta-model or show parts of it [19, 13], several publications document their meta-model mostly through exemplary models. Some DSLs are available for download as open-source software [18, 8, 20].

A good way to promote re-use of s DSL is to provide tutorials and examples of its usage, as download together with software frameworks and dependencies (if necessary), as done for example by [21, 8]. Laet et al. propose their semantics for standardization in the context of the robotics engineering task force [22].

Artifacts and Use-Case To assess the intended use of the DSLs, we looked at the artifacts generated (if any) and the context they are used in. While the DSL can be used

to generate visualizations of systems, e.g. the system architecture [23] or platforms [8], the main use-case for DSLs is to generate executable code to perform experiments or provide supporting routines. DSLs within the identified subdomains often cover similar use-cases. The Robot Structure (1) subdomain for example primarily targets controllers and platform as well as simulation support. Frigerio et al. [18] and Laet et al. [16] target kinematics and dynamics controllers that can be embedded in motion control systems. Bordignon et al. [24] exemplify the usage by generating code to simulate the specified robot platform in a particular simulation framework.

Artifact generation from DSLs becomes especially powerful and suited for re-use if the toolchain supports different M2M and M2T transformations. Either to generate different artifacts like visualization, computational routines and glue code [23], or executable code for different programming languages or software platforms [10, 11, 25].

Evaluation Evaluation of a DSL-based approach in their intended use-case is not only interesting from a developer's perspective, but also serves as a foundation for a decision from a user's perspective. A number of the surveyed publications evaluated the semantics or the generated artifacts. A surprising yet positive outcome of the analysis was, that quite a number of the DSLs in this domain are evaluated not only in simulation, but on real hardware [26, 14, 18, 13], and even on different platforms [19, 11].

We can roughly differentiate two different kinds of evaluation approaches: qualitative and quantitative evaluation. Qualitative evaluation is often done by conceptual discussions based on examples, e.g. portability of the semantics to different platforms [14, 19, 11]. Laet et al. [21] for example model some typical use-cases and show how common errors can be avoided by using its proposed semantics.

Özgür [27] lists four different quantitative benefits and corresponding metrics, that can be used to evaluate a model-based approach and can serve as a best practice:

1. **Efficiency** can be evaluated in terms of performance and memory utilization. Frigerio et al. [9] for example benchmarked the generated C++ code in its intended use-case, being forward and inverse kinematics and dynamics on different numbers of degrees-of-freedom.
2. **Scalability** in terms of compilation time and system size.
3. **Productivity** in terms of size, effort or number of change requests. Examples are Ringert et. al [28] and Romero-Garces et. al [29]. Both evaluate the usage of a DSL from the developers perspective against classical approaches by means of empirical software engineering. Non-functional aspects they covered comprise time spent for learning of the technologies, effort for fixing bugs, component re-use and complexity of understanding re-used software artifacts. [11] conducted hardware experiments on a PR2 and a KUKA LWR and analyzed the necessary number of lines of code for platform-independent and robot/framework specific code.
4. **Reliability**, e.g. in terms of defects introduced in a period of time.

Platform An important aspect of the generated artifacts and the model transformations is how tightly they are coupled to a certain platform. "Platform" in this context means the technical execution context, so the software framework, and all additional tools or libraries necessary to use the DSL or the generated artifacts.

First of all we have to differentiate between the DSL being used in a interpretation vs. a generation manner. Interpretation of a DSL is always being tied to a (DSL-specific) interpreter (e.g. [30]). For DSLs that are used in a generation manner, we differentiate between three classes of platform-dependency:

1. Proprietary solutions like *KRL* [31] and *RAPID* [32] that are targeted to a single platform and don't target openness or platform independence at all.
2. Generation of artifacts that are tied to or dependent on a library stack, software framework or runtime environment [13, 17, 33]. Some of the DSLs in this survey target a certain framework or environment, but come with exchangeable generators to explicitly allow re-use of the DSL and its concepts in different frameworks or environments as discussed above. Klotzbücher et al. [11] make the platform explicit, by distinguishing between platform-independent and platform-specific models.
3. Transformation of the DSL code directly to a general purpose language (e.g. Ada [14] or C++ [10]) being the most platform-independent option by reducing platform dependencies to a minimum, which provides clear advantages. It is easier portable, even to embedded systems [14], easier to re-use and eases scientific exchange. It also reduces assumptions about the platform from within the DSL.

DSL Development Process Mernik et al. [4] discuss that the identification and formalization of domain-specific abstractions is an important decision pattern for DSL development. However, to reuse, refine or to define new abstractions one needs to perform activities known in the area of *knowledge representation* such as domain and problem assessment and expert consultation. Unfortunately, in the assessed papers very little is written about the process *how* the abstractions have been identified, e.g. based on an ontology [25], a formalism [16] or a domain analysis [23]. One reason may be that the DSL developers are very often simultaneously also the DSL users and domain experts. Hence, assessing the domain is performed in an ad-hoc and implicit manner. To bring forward the DSL development in robotics we argue that robotic DSL papers should report also about the process of *how* and on which basis one developed certain domain-specific abstractions.

7 Synopsis

We surveyed the available literature on domain-specific (modeling) languages used for design, simulation and programming of robotics systems. The quantitative analysis supports that DS(M)Ls are a current active research field for simulation and programming of robots, however, compatibility and re-use of different DSLs and approaches is still an issue. Yet the *Eclipse Modeling Project* may serve as an integration platform for DSLs in robotics as it is already widely used. We further discussed, how different approaches to documentation, evaluation and platform-dependency affect the availability and usability of a DSL. We intend this survey to serve the robotics DSL community to foster exchange between DSL developers as well as providing an orientation for potential DSL users. Following the idea of the *EMF Concrete Syntax Zoo*[7] we intend to continu-

[7] http://www.emftext.org/index.php/EMFText_Concrete_Syntax_Zoo

ously maintain the survey as an online *Robotics DSL Zoo*[8] and invite the community to provide feedback and contribute. Future iterations of this survey will comprise further conference proceedings and include journal publications.

Acknowledgement This work was supported by a grant of the Cluster of Excellence Cognitive Interaction Technology (CITEC) at Bielefeld University. Nico Hochgeschwender received a PhD scholarship from the Graduate Institute of the Bonn-Rhein-Sieg University.

Bibliography

[1] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM Sigplan Notices*, 2000.

[2] G. Biggs and B. MacDonald. A Survey of Robot Programming Systems. *Australasian Conference on Robotics and Automation*, 2003.

[3] Markus Völter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering Designing, Implementing and Using Domain-Specific Languages*. 2013.

[4] M. Mernik, J. Heering, and A.M. Sloane. When and how to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[5] Sébastien Gérard and Bran Selic. The UML – MARTE Standardized Profile. In *The International Federation of Automatic Control*, pages 6909–6913, Seoul, Korea, 2008.

[6] Richard C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[7] JetBrains. Meta Programming System.

[8] Ioan Sucan. Unified Robot Description Format (URDF).

[9] Marco Frigerio, Jonas Buchli, and Darwin G. Caldwell. Code Generation of Algebraic Quantities for Robot Controllers. *International Conference on Intelligent Robots and Systems*, pages 2346–2351, October 2012.

[10] M. Frigerio, J. Buchli, and D.G. Caldwell. A Domain Specific Language for Kinematic Models and Fast Implementations of Robot Dynamics Algorithms. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2011.

[11] Markus Klotzbücher, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Reusable Hybrid Force-Velocity controlled Motion Specifications with executable Domain Specific Languages. In *International Conference on Intelligent Robots and Systems*, pages 4684–4689, 2011.

[12] Dominick Vanthienen, Markus Klotzbücher, Joris De Schutter, Tinne De Laet, and Herman Bruyninckx. Rapid application development of constrained-based task modelling and execution using Domain Specific Languages . In *International Conference on Intelligent Robots and Systems*, 2013.

[13] Andreas Angerer, Remi Smirra, Alwin Hoffmann, Andreas Schierl, Michael Vistein, and Wolfgang Reif. A Graphical Language for Real-Time Critical Robot Commands. In *Workshop on Domain-Specific Languages and models for Robotic systems*, Tsukuba, 2012.

[14] Piotr Trojanek. Model-Driven Engineering Approach to Design and Implementation of Robot Control System. *Workshop on Domain-Specific Languages and models for Robotic systems*, 2011.

[15] Sebastian Blumenthal and Herman Bruyninckx. Towards a Domain Specific Language for a Scene Graph based Robotic World Model. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2013.

[8] http://cor-lab.org/robotics-dsl-zoo

[16] Tinne De Laet, Wouter Schaekers, Jonas de Greef, and Herman Bruyninckx. Domain Specific Language for Geometric Relations between Rigid Bodies targeted to Robotic Applications. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2012.

[17] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann. A New Skill Based Robot Programming Language Using UML/P Statecharts. In *International Conference on Robotics and Automation*, 2013.

[18] M. Frigerio, J. Buchli, and D.G. Caldwell. Model based code generation for kinematics and dynamics computations in robot controllers. In *Workshop on Software Development and Integration in Robotics*, St. Paul, Minnesota, USA, 2012.

[19] M. Reckhaus and N. Hochgeschwender. A Platform-Independent Programming Environment for Robot Control. *Workshop on Domain-Specific Languages and models for Robotic systems*, 2010.

[20] M Lötzsch, Max Risler, and M Jungel. XABSL – A Pragmatic Approach to Behavior Engineering. *International Conference on Intelligent Robots and Systems*, pages 5124–5129, 2006.

[21] Tinne De Laet, Steven Bellens, Herman Bruyninckx, and Joris De Schutter. Geometric Relations between Rigid Bodies (Part 2): From Semantics to Software. *IEEE Robotics and Automation Magazine*, (September), 2012.

[22] Tinne De Laet, Steven Bellens, Ruben Smits, Erwin Aertbelien, Herman Bruyninckx, and Joris De Schutter. Geometric Relations between Rigid Bodies (Part 1): Semantics for Standardization. *IEEE Robotics and Automation Magazine*, (June), 2012.

[23] Arne Nordmann and Sebastian Wrede. A Domain-Specific Language for Rich Motor Skill Architectures. In *Workshop on Domain-Specific Languages and models for Robotic systems*, Tsukuba, 2012.

[24] Mirko Bordignon, Ulrik Pagh Schultz, and Kasper Stoy. Model-Based Kinematics Generation for Modular Mechatronic Toolkits. *International Conference on Generative Programming and Component Engineering*, page 157, 2010.

[25] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. In *Simulation, Modeling, and Programming for Autonomous Robots*, 2012.

[26] Ulrike Thomas, Bernd Finkemeyer, Torsten Kröger, and Friedrich M. Wahl. Error-Tolerant Execution of Complex Robot Tasks based on Skill Primitives. In *International Conference on Automation and Robotics*, Taipei, Taiwan, 2003.

[27] Turhan Özgür. *Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling in the Context of the Model-Driven Development*. Master, Blekinge Institute of Technology, 2007.

[28] J.O. Ringert, Bernhard Rumpe, and Andreas Wortmann. A Case Study on Model-Based Development of Robotic Systems using MontiArc with Embedded Automata. In *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme IX*, 2013.

[29] A. Romero-Garcés, L.J. Manso, M.A. Gutierrez, R. Cintas, and P. Bustos. Improving the Lifecycle of Robotics Components using Domain-Specific Languages. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2013.

[30] Henrik Mühe, Andreas Angerer, Alwin Hoffmann, and Wolfgang Reif. On reverse-engineering the KUKA Robot Language. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2010.

[31] KUKA System Software 5.5 - Operating and Programming Instructions for System Integrators. Technical report, KUKA Roboter GmbH, 2009.

[32] RAPID Overview. Technical report, ABB Robotics Products.

[33] A. Steck and C. Schlegel. SMART TCL: An Execution Language for Conditional Reactive Task Execution in a Three Layer Architecture for Service Robots. In *Int. Workshop on DYnamic languages for RObotic and Sensors systems (DYROS)*, pages 274–277, 2010.