Declarative-knowledge-based reconfiguration of automation systems*

Mathias Haage[1], *Jacek Malec*[1], *Anders Nilsson*[1], *Klas Nilsson*[1], *Sławomir Nowaczyk*[2]

Abstract

This article describes results of the work on knowledge representation techniques chosen for use in the European project SIARAS, *Skill-Based Inspection and Assembly for Reconfigurable Automation Systems*. Its goal was to create intelligent support system for reconfiguration and adaptation of robot-based manufacturing cells. Declarative knowledge is represented first of all in an ontology expressed in OWL, for a generic reasoning in Description Logic, and in a number of special-purpose reasoning modules, specific for the application domain.

1. INTRODUCTION

SIARAS is an acronym of an EU-funded (FP6 - 017146) STREP-project entitled *Skill-Based Inspection and Assembly for Reconfigurable Automation Systems*. Its main goal was to build fundamentals of an intelligent system, named *the skill server*, capable of supporting automatic and semi-automatic reconfiguration of a manufacturing processes. The ultimate purpose of this work is to provide support to robot users in small and medium-size enterprises (SMEs) in Europe, where usually there is lack of sufficient expertise regarding systems engineering and robot programming so that the companies are not able to solve complex reconfiguration tasks. Therefore a lot of SMEs refrain from buying advanced robot systems fearing the costs of external expertise. The skill server is expected to lower the cost associated with reprogramming a robot.

In SIARAS, we identified several types of basic concepts and related (non-procedural) knowledge: skills, devices, tasks, workpieces, and environment, that we used in the skill server (SkS), the major software system resulting from the project. From another perspective, we may speak about three kinds of knowledge: generic (no domain information required), general domain knowledge (time-related, geometry-related, etc.), and domain-specific (e.g., welding, assembling, machining, etc.). In accordance, the reasoning algorithms, their location, and interface have to be carefully determined.

*This work has been supported by the EU-project SIARAS (FP6 - 017146).

[1]Department of Computer Science, LTH, Lund University, Box 118, S–221 00 Lund, Sweden, {mathias,jacek,andersn,klas}@cs.lth.se.

[2]Laboratorium Informatyki, Katedra Automatyki, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza, 30-059 Kraków, Slawomir.Nowaczyk@agh.edu.pl.

In this paper we describe the solutions adopted in SIARAS for building the Skill Server and the results obtained so far. More details can be found in SIARAS deliverables [3, 16, 1] while a partial design has been described in our earlier paper [12]. This paper completes the introduction presented at the 9th KKR conference [2].

The paper is divided as follows: first we review the related work. Then we describe the ontology adopted as the basis for declarative knowledge representation. In the next section the system architecture is presented. Then follow details of the declarative representation of devices and skills. Afterwards comes a discussion of mechanisms implementing domain-specific knowledge. Finally, we briefly describe the system in action.

2. RELATED WORK

The research on knowledge representation has been extensively documented. One of the recent textbooks, offering a very good overview of the field, is [5].

The organization that made discussions on semantic web and, in particular, on ontologies popularized, has an extensive library of published documents at the W3Consortium's Semantic Web site [23]. In particular, the specifications of the most popular KR formalisms, like OWL [24] or DAML-OIL [7], together with available tools for using them, are provided there.

Production planning is usually considered (within the field of AI) to be a part of the automatic planning domain. However, besides the classical manufacturability analysis, reported recently in [8], there is in principle no thoroughly documented research on using ontologies in automated production planning. However, there is an extensive research aimed at supporting the engineering activities in production design by providing modeling languages and tools allowing formal, automatic analysis of the discussed process. One of such early works is [22] where an automatic planner for a robotized cell is proposed. Quite naturally, most of those formalisms and tools are heavily domain-dependent, with a small number of exceptions explicitly stating the goal of being general-purpose tools. We may name here the *Sensor Modeling Language* which offers a rich sensor ontology (see `http://www.sensorml.org` for an extensive documentation). A dual enterprise is the *unified robot modeling language*, (URML), from the University of Karlsruhe. Unfortunately, URML does not provide representation facilities for the dynamic aspects of robot performance.

Finally, an important attempt to formalize the language for speaking about production processes has been done at NIST, which created the Process Specification Language [20]. The language, and some of the associated tools, served as a reference point for the ontology developed within SIARAS.

From the application perspective, there has been some work done on configuration, although most work focused so far on configuring computer systems or services. For an overview, see [19]. A presentation origination from SIARAS research, but focusing on workpiece modelling, is given in [4]. Finally, the ideas developed in SIARAS become starting point for the concept of Knowledge Integration Framework being currently developed by the Rosetta project (`www.fp7rosetta.org`, see e.g. [17, 14]).

3. ONTOLOGY

We have decided to center knowledge representation around the concepts of devices (physical objects provided by their manufacturers) and skills, while task descriptions exist only during problem solving sessions, as dynamic structures. Tasks can be seen as (arguably, quite complex) combinations of skills and therefore there is no need to have them explicit in the vocabulary.

This static part of the knowledge is represented in an *ontology*: a data structure storing all the necessary relations between the terms used. Quite often ontologies are used for classification purposes. In the skill server case the classification is done when objects (devices) and their capabilities (skills) are introduced in the structure, therefore we can as well refer to it as plain taxonomy. The ontology forms a distributed system containing a quite complicated skill structure and libraries of devices (leafs of the taxonomy).

We have chosen the open source tool Protégé [13] for ontology creation and manipulation due to its openness and modifiability.

In our approach, the ontology is used for reasoning about skills matching particular tasks (after some initial re-parametrisation) and about devices offering those skills (under certain conditions). A pure ontology, which is nothing more than a set of logical statements, expressed in our case in OWL language [24] being a slightly weaker language than first-order logic, may be used for retrieval, pattern matching and simple classification (not really useful in our case), while other forms of reasoning, like planning, optimizations, consistency checks, etc., need to be done by more powerful reasoners, either general-purpose ones or those specialized to a particular application domain. The generic tools that have been used by us so far are Racer [9], Fact++[10], Algernon and Pellet. They differ in their reasoning power and efficiency, being able to handle either a restricted Description Logic language [21] (like OWL-DL offered by Protégé) in an efficient (polynomial complexity) manner [6], or a (more expressive) OWL-Full [24] representation, but using search algorithms of exponential complexity. We also have the possibility of choosing a different reasoner depending on the question asked, thus achieving flexibility and adaptability of the system.

4. SKILL SERVER DESIGN

Figure 1 shows the architecture of the skill server. Three components fit in the center of the picture: the main loop of the skill server, an *ontology*, and a *database*. The ontology module holds all the generic knowledge of the system. It corresponds to skills, tasks they can perform, sensors and actuators involved, operations of instantiated skills (i.e. with a fixed device associated with it), workpieces involved, etc.

The stated goal of the Skill Server is *reconfiguration* of an existing, correctly parametrised process. Thus, first the current task[†] has to be defined by a user (be

[†]By *task* we understand a description, possibly hierarchical, of the intended operation to be performed, leading to some intended state, e.g. "Mark the plate with Braille inscription". A *skill* is a concrete ability of some device, e.g. a robot equipped with a drilling tool can drill holes. Finally, a *process* is a concrete realisation of a task by a set of skills provided by the devices used in the workcell.

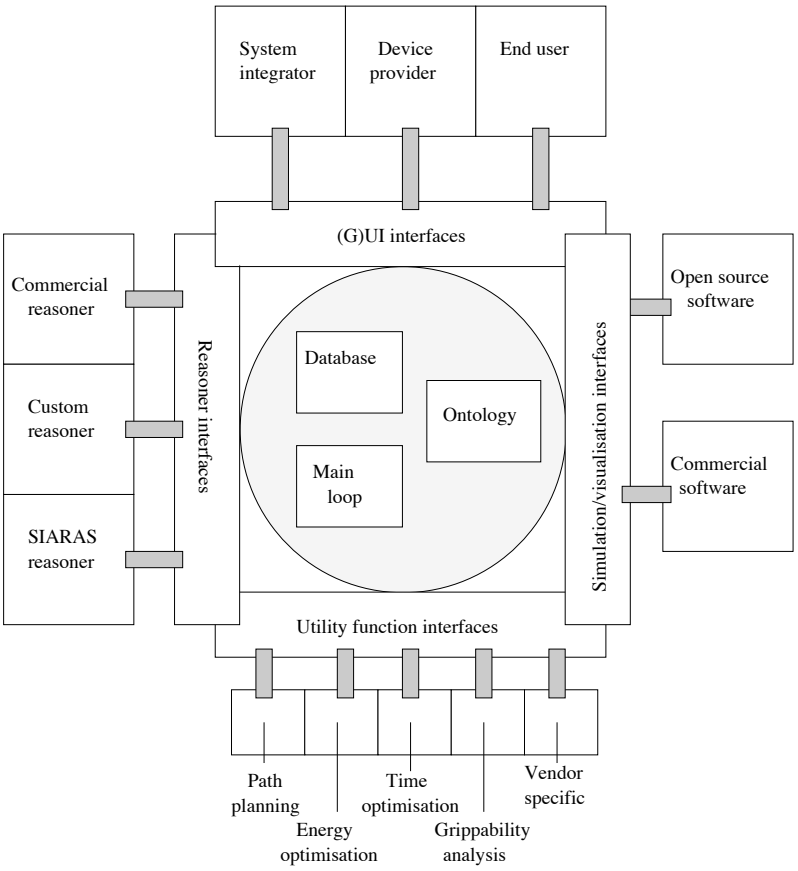

Fig. 1. The skill server architecture

it the system engineer or an end-system-user), using an appropriate user interface facilitating this duty. In order to constrain the task descriptions to ones understood by the skill server, the user interface has to consult the ontology in an appropriate way. As a result, the *actual task description* is created. It may be thought of as a data/knowledge structure, subsequently manipulated by the system. We use for this purpose a modified Sequential Function Chart (SFC), as defined by the IEC standard 61131-3 [18].

Next comes the main loop of the skill server. It begins with the user asking for a particular reparametrization or reconfiguration of the current task. Possibly it may consist, at least partly, of some manipulation over the actual task description ("What if?"). The server then analyses whether the current set of operations is still a valid realization of the task and, if not, it suggests changes. It employs both generic reasoning, available via external reasoners attached to the ontology, as well as using domain-dependent reasoning modules, represented here as *plug-ins*, attached to the core server using well-defined protocol and interface.

Finally, an important part of the original knowledge is this part of the ontology

that has been named *Device Library*. Formally, the device descriptions are elements (leafs) in the ontology. However, the device library forms *virtual parts* of the ontology, plugged-in as needed and as available. The library (or libraries) could reside and be maintained by device manufacturers (e.g. Abb, Kuka or Motoman), who would put in there everything that is necessary for a device to fit the (common) manufacturing ontology and to be meaningfully used, and reparametrized, by the skill server. This structure is named "database" in the Figure above.

5. CONTENTS OF THE SIARAS ONTOLOGY

The SIARAS ontology contains knowledge about (abstract) skills and about devices and their properties. This is an area for which ontology is well-suited, so it is both easy and efficient to specify that, for example, a concept of *vacuum gripper* is a subconcept of *gripper*, which in turn is a subconcept of *device*. Similarly, a *vacuum-pickup* skill is a subconcept of *pickup* skill. An ontology allows one to specify properties of each concept, with natural inheritance rules. Thus we can assign *mass* property with the concept of *device* and *number of fingers* with "finger gripper".

The first concept that has been introduced in the ontology is a *device*. We expect the device vendors to add their devices as leaves in the ontology, which ensures that they fit the classification properly and that all the required attributes are filled in. However, it is clear that not all constraints put on devices can be verified by an ontology reasoner, mostly due to lack of an appropriate domain knowledge, so we need to use more powerful tools for that.

Next, there is a need to represent connections between skills and devices. In particular, each device is able to perform one or more skills, and each skill can be performed by one or more devices. The hierarchical structure of the ontology is very useful here, as it allows us to specify that *vacuum gripper* has a *vacuum-grasp* skill, from which skill server will automatically deduce that it is a *gripper* and has, therefore, a (more generic) *grasp* skill as well.

Finally, when talking about robots, it might be necessary to specify that a *robot* uses a *gripper* to perform some *skill*. Doing it properly is outside the representational power of an ontology, at least in the form it has right now. We use the concept of *compound devices* for this purpose.

Tasks can be thought of as *generalizations* of skills along (at least) two axes: first, they are time-ordered sequences (or partially parallelized) of skills (or rather skill applications): for example, a *relocate* task can be seen as a sequence of *pickup*, *move* and *putdown* skills. In addition, it could make sense to have hierarchy of tasks, with *windshield fitting* task decomposing into *find windshield*, *position windshield* and *attach windshield* subtasks.

Tasks constitute means of achieving a particular goal, and a reasoner needs to be presented with at least some description of this goal. It needs to be able to reason about rationale for each task and about reasons why a particular device and particular parameters were chosen. At the very least it needs a list of criteria which distinguish acceptable execution of this task from unacceptable ones, leading to errors.

On the top level, the ontology is split into several categories:

ObjectBase Every physical object can be modeled as a simple *Part*, or as an *Assembly* consisting of parts or other assemblies. It is mainly intended to hold information about geometrical dependencies of objects (like e.g.,"The final product consists of a printed circuit board and a camera mounted on it."), as logical and functional dependencies, in particular regarding devices, are handled by the mechanism described below.

Operation The vocabulary needed for talking about operations[‡] that are performed by a device. An operation can be *Atomic*, i.e., an invocation of a skill, *Parallel*, forming a complex of two (or more) operations performed in parallel, or a *Sequence* denoting a complex of two (or more) operations performed one after another[§].

(Physical)Object A work cell consists of *PhysicalObject*s. Some objects, *Devices*, are active and have skills, while other, *Workpieces*, are passive and are being manipulated by the devices while executing tasks. The device hierarchy is quite deepin order to mirror the diversity of actual devices used in automation systems.

Property The *Property* hierarchy enumerates those properties of devices and skills which are interesting for the system to reason about. In particular, the subtree containing the physical properties of devices is very numerous, illustrating the focus taken during the prototyping. However, it is expected that the other branches of this hierarchy may grow as necessity arises.

Skill A *Skill* represents an action that might be performed (by a device) in the context of a production process. The skill hierarchy is divided into six subcategories, each for different types of skills identified by the SIARAS consortium. The major subcategory is *MainFunction* where the manipulation and manufacturing skills are defined.

There is also one important category, *CompoundSkills*, which is used for modeling complex skills which are aggregated from the simple skills found in the ontology. As an example we can name the DRILL skill, consisting of the following steps (corresponding to primitive skills): APPROACH (the start position), STARTROTATION (of the drill), MOVELINEAR (down), MOVELINEAR (up) and STOPROTATION. DRILL has its own purpose, different from purposes of its steps (namely to create a hole in the workpiece), and has to be treated as a unit in some contexts, while in others reasoners may need to analyze in detail each single step of this compound skill.

6. COMPOUND DEVICES

Most devices are not useful in isolation in a manufacturing cell, but must be combined with other devices to make a meaningful *compound device*. For example, consider

[‡] An *operation* is defined as an instantiated skill. It is the basic element of task representation.

[§] We have not made use of the *operation* concept in the deployed version of SkS.

```
controller_1: ABB_IRC5
ioboard_1: dig328
robot_1: ABB_IRB-140
clamp_1: AngleGripper
drillmachine_1: Bosch_GBM_10_RE
drillbit_1: DrillBit_HSS_8mm

SubDevice: controller_1,ioboard_1
SubDevice: controller_1,robot_1
SubDevice: drillmachine_1,drillbit_1
SubDevice: robot_1,drillmachine_1
```

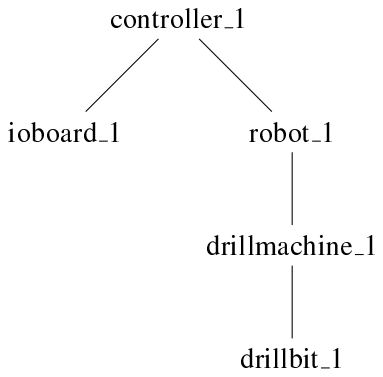

Fig. 2. Device specification from a task description on the left. Corresponding device tree on the right

a possible set of devices needed for an industrial robot to perform drilling in workpieces: robot controller, I/O board, robot arm, drilling machine, drill bit, fixture. None of these devices is by itself capable of drilling a hole at a specified location; a drilling machine can not position itself at the correct position, nor can it drill a hole without a drill bit attached to it. Only by connecting[¶] together all the devices mentioned above may the resulting *compound device* perform a drilling operation.

Since the skill server generates configurations for a robot cell, it must also be able to reason about how, and when, devices are connected to each other. We have therefore introduced a device relationship in the ontology, $hasSubDevice \leftrightarrow isSubDeviceOf$, in order to model compound devices. A difference compared to other relations in the ontology is that it is dynamic instead of static. Device instances in a device library will not typically be statically connected to any other device instance. Instead, a task description where a specific device instance is used, must also specify how it is connected to other devices listed in the task description. An example on specifying device relations is shown in Figure 2.

Yet another aspect of combining devices into compound ones is computing their properties out of the properties of their elements. In some cases this operation is obvious: e.g., a gripper can hold an object, thus a robot equipped with a gripper can also hold an object (simple inheritance). However, the allowed payload for such a compound device will not be inherited, but rather computed in a particular way. e.g. $\min(\text{payload}(robot) - \text{weight}(gripper), \text{payload}(gripper))$ There seems to be no obvious way to devise a generic inheritance mechanism for compounds; we currently assume that this is domain-dependent and normally specified by the expert user, although other possibilities may also be investigated.

[¶] *Connect* should here not be taken literally but in a logical sense: controls/is controlled by.

7. DEVICE LIBRARY

The properties identified for a device library to be usable and maintainable are:

- *distributedness*: the information may exist in several repositories, but needs to be dynamically merged in one place in order to be used by SkS;
- *heterogeneity*: the parts of ontology may be stored in different formats, e.g., core as an OWL XML file, device library as databases;
- *multiple access protocols*: we assume that parts of the device library (or, more generally, of the ontology) may be stored in several places, e.g.,
  - locally on the SkS server;
  - on a database server within the company using SkS;
  - on database servers provided by device manufacturers;
  - on web-sites of the device manufacturers;
  - in the devices themselves (with a specialized access protocol).

  Therefore there needs to be multi-protocol interaction between the skill server and the ontology providers, resulting in the actual ontology becoming available for computations when needed.

A preliminary study of such a distributed solution has been designed and implemented recently as a Master thesis project in Lund University [11].

One important aspect of the device library is that it must contain not only static information about the devices (like name, properties expressed using discrete parameters, e.g., weight, power consumption or accuracy) but also geometric information, which may be parametrized in some way (for this purpose we assume a COLLADA specification pointed to by an appropriate URI) and some description of the dynamic behavior of the device. The behavior may be described in many ways and it will be the visualization/simulation interface that determine what kind of descriptions will be necessary or useful. In SIARAS we have used Python programs runnable in the 3DCreate environment, both for visualization and for simulation purposes. In ROSETTA, the choice is Rapid language interpreted by the ABB Robot Studio environment. However, there may be more than one behavioral model, extending the utility of information contained in the device library.

8. WORLD MODEL

There have been two aspects of skill modeling in the skill server: static, contained mainly in the ontology, and dynamic, created for and in a particular task. The former one has been covered by the earlier sections. The second aspect is partially captured in the primary data structure of the skill server, named the *world model*.

The main structure of the world model (later WM) is a transition system (a sequential function chart in case of SIARAS, build using a dedicated user interface based on JGrafChart tool (`http://www.control.lth.se/grafchart/`)) and annotated with the necessary pieces of knowledge as e.g., purpose (which needs not to be obvious — in such a case it would have been retrieved from the ontology), optimization criteria to apply, etc. All skills are instantiated with parameters, including devices, and therefore may also be referred to as *operations*. Each operation has a set

of pre- and postconditions specified, and a geometrical model of the state before and after the operation is given. The pre- and postconditions are particularly important for synchronization of the operations, including workpiece manipulations and dependencies. Besides the geometry there is also necessity to include other aspects of the work cell model, i.e., communication between devices, timing constraints, used interfaces and signals.

In order to allow the reasoners to perform their duty, accessor methods should necessarily present the contents of the world model in different views. E.g., given that a compound device may consist of a robot arm, a drilling machine and a drill-bit, the apparent views are here robot-centric (joint-wise or end-plate-based), tool-centric (important for the performed operations, e.g. drilling) or process-centric, focusing on the parameters of the operation itself.

9. UTILITY FUNCTIONS

The design of the skill server assumed that knowledge is incrementally added to the server in form of utility functions (below shortly UFs), plug-in modules possessing specific knowledge about some aspect of manufacturing, and contributing to the skill servers effort by their expertise. The concept may be compared to the *blackboard architecture* [15], where modules called *knowledge sources* (KS) discover in a common data structure called blackboard how they can contribute to the problem-solving process by performing some computations or reasoning they are capable of. They do what they are defined for and store their results on the common blackboard for other KSs to use. In case of skill server, the knowledge sources are named *Utility Functions*. The assumptions underlying this computational model may be specified as follows:

- UFs have (read) access to the complete world model. There seems to be no point in hiding any information, except maybe intermediate results that shouldn't be put in the WM anyway;
- UFs return answers that SkS can understand, plus always an explanation string, understandable for humans (which may be displayed in the interaction area of a GUI, see below);
- UFs are always active and determine by themselves whether they are relevant, have enough data and what kind of result is appropriate at this moment. E.g., the latter may depend on whether SkS considers now a simple skill or a compound one, what is the context of the computation, etc.
- Skill server performs its functions by walking through the operations of the given task and analyzing their parametrization according to user's requests. At every moment SkS needs to know (as do all UFs it exploits) what is the current skill and what are all the compound skills containing it;
- UFs are also informed what is the expected outcome in the current state of reasoning, which may be one of the following alternatives: *show* (current parametrization), *evaluate* (feasibility), *find* (some parametrization, or acceptable parametrization, or even best parametrization), *tune* (the current parametrization further), and *make* (create configuration files);
- UFs are written by people not having contact with each other, not knowing

the details of other UFs, and therefore the interface needs to be sufficient to ensure interoperability between completely unknown pieces of code.

It should be obvious that in such case the UFs cannot modify the WM by themselves without creating havoc: they have to pass their results to the skill server, which then treats them appropriately, depending on what kind of function it is and what kind of result it provided. How to do it properly is a matter of policy, above everything else.

Utility functions provide the core knowledge about details of manufacturing, in particular about skills of devices and their concretizations in form of operations included in the current task description. Even though the core skill server has some reasoning capability for generic questions, the majority of work is normally done by the UFs.

10. DEMONSTRATOR

The results of the SIARAS project, in particular the Skill Server, have been presented in a demonstrator cell at the AUTOMATICA 2008 fair in Munich. The server, asked to reconfigure a particular manufacturing process (e.g., by changing workpiece material from wood to metal, or by changing the sensor used in quality control) could detect necessary changes to be done, verify the process consistency after changes, or even employ an external simulator to analyse the reachability conditions. More details regarding the demonstrator are available on the project home page `www.siaras.org`.

Of course, this demonstrator should be seen only as a proof of concept, i.e. that the algorithms and technology used to create the system have the expected potential and are principally correct. The next step is to fill the Skill Server with sufficiently much knowledge so that it becomes useful in an typical SME. This requires much work from the side of device manufacturers. The actual utility as well as scalability will not be possible to judge before the knowledge base is sufficiently large.

11. CONCLUSIONS

In SIARAS the declarative approach to generic automation-related knowledge representation in an ontology is combined with procedural (or sometimes rule-based) representation of domain-specific knowledge contained in the utility functions. It has been used in Skill Server which has been demonstrated in practice as a possible way of knowledge-based reconfiguration of a production cell consisting of two robots with exchangeable tooling and a vision-based quality assurance system.

We are convinced that this approach provides a powerful combination allowing to solve non-trivial reconfiguration problems, in particular given opennes of the architecture and modularity of the approach. However, the non-declarative part is a major obstacle for further scalability of the solution. Our next research step is finding a smart solution to this problem. Another future project is making the system open, accessible using open-source tools, like e.g. Blender, for visualisation and simulation. However, the ultimate verification will be done by end users eventually accepting this tool.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for insightful comments leading to improvement of this paper.

REFERENCES

[1] Ola Angelsmark et al. D2.4: Framework for task representation. Technical report, Lund University, 2006. SIARAS deliverable.

[2] Ola Angelsmark et al. Knowledge representation for reconfigurable automation systems. In: *Postępy Robotyki: Systemy i współdziałanie robotów* Red. K. Tchoń, pp. 129–138. Warszawa, Wydawnictwo Komunikacji i Łączności 2006.

[3] Ola Angelsmark, Jacek Malec, Sławomir Nowaczyk. D1.4: Methodology for skill encoding. Technical report, Lund University, 2006. SIARAS deliverable.

[4] Matthias Bengel. Model-based configuration — a workpiece-centred approach. In: Proceedings of the 2009 ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots. *Proceedings* Red. Jian S. Dai, London, UK, June, 2009, pp. 689–695.

[5] Ronald J. Brachman, Hector J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers 2004.

[6] Martin Buchheit, Francesco M. Donini, Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. Journal of Artificial Intelligence Research, 1993, Vol. 1, pp. 109–138.

[7] Dieter Fensel et al. OIL: An ontology infrastructure for the semantic web. IEEE Intelligent Systems, 2001, Vol. 16, No. 2, pp. 38–45.

[8] Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning, Theory and Practice*. Morgan-Kaufman 2004.

[9] Volker Haarslev, Ralf Möller. Racer: A core inference engine for the semantic web. Available at `http://www.franz.com/products/racer/`, 2002.

[10] Ian Horrocks. FaCT and iFaCT. In: Proc. Int. Workshop on Description Logics DL'99. *Proceedings* Red. P. Lambrix, et al, 1999, pp. 133–135.

[11] Petter Lidén. Distributed knowledge sources in SIARAS. Master's thesis, Department of Computer Science, Lund University, 2009. Available from `http://ai.cs.lth.se/education.shtml`.

[12] Jacek Malec et al. Knowledge-based reconfiguration of automation systems. In: *Proc. of the IEEE Conference on Automation Science and Engineering*, LNCS, pp. 170–175. Springer 2007.

[13] Marc Musen, et al. The Protégé ontology editor and knowledge acquisition system. `http://protege.stanford.edu`, 2006.

[14] Martin Naumann, Matthias Bengel, Alexander Verl. Automatic generation of robot applications using a knowledge integration framework. In: Proc. International Symposium on Robotics ISR 2010. *Proceedings*, Munich, Germany, June, 2010.

[15] H. Penny Nii. Blackboard systems: An overview. AI Magazine, 1986, Vol. 6.

[16] Anders Nilsson, Jacek Malec. D1.6: Implementation of software framework for skill representation. Technical report, Lund University, 2007. SIARAS deliverable.

[17] Jacob Persson et al. A knowledge integration framework for robotics. In: Proc. International Symposium on Robotics ISR 2010. *Proceedings*, Munich, Germany, June, 2010.

[18] PLCopen. IEC 61131-3: Programmable controllers – part 3: Programming languages. Technical report, International Electrotechnical Commission, 2003.

[19] Matthieu Quéva, Christian Probst, Per Vikkelsøe. Industrial requirements for interactive product configurators. In: Proceedings of the IJCAI-09 Workshop on Configuration (ConfWS-09). *Proceedings* Red. Markus Stumptner, Patrick Albert, Pasadena, CA, USA, July, 2009, pp. 39–46.

[20] C. Schlenoff et al. The Process Specification Language (PSL): Overview and v1.0 specification. Technical Report NISTIR 6459, NIST, Gaithersburg, MD, 2000.

[21] *The Description Logic Handbook*. Red. Franz Baader et al. Cambridge University Press 2003.

[22] U. Thomas, F. M. Wahl. A system for automatic planning, evaluation and execution of assembly sequences for industrial robots. In: Proceedings IROS 2001. *Proceedings*, Maui, Hawaii, USA, IEEE, November, 2001, pp. 1458–1464.

[23] W3C. Semantic web. `http://www.w3.org/2001/sw/`, 2001.

[24] W3C. Web ontology language (OWL). `http://www.w3.org/2004/OWL/`, 2004.