Knowledge Representation for Reconfigurable Automation Systems*

Ola Angelsmark[1], Jacek Malec[1], Klas Nilsson[1], Sławomir Nowaczyk[1], Leonardo Prosperi[1]

Streszczenie

This article describes the work in progress on knowledge representation formalisms chosen for use in the European project SIARAS. *Skill-Based Inspection and Assembly for Reconfigurable Automation Systems* has a goal of creating intelligent support system for reconfiguration and adaptation of assembly systems. Knowledge is represented in an ontology expressed in OWL, for generic reasoning in Description Logic, and in a number of special-purpose reasoning modules, specific for the application domain.

1. Introduction

Knowledge representation is one of the major topics studied throughout the 50 years of artificial intelligence research. A multitude of methods has been developed, ranging from formal accounts, based on logical formalisms, through semi-formal approaches, to ad-hoc methods applicable in particular cases.

Knowledge representation, as many other subareas of AI, has been the subject to paradigm changes and, to some extent, has been continuously adapting to the fluctuating stream of research funding. Recent years have been mostly devoted to investigations of multi-agent systems and to the development of the so called *semantic web*. The semantic web project [10] aims to formalize large portions of knowledge in a form which enhances interoperability of usually distributed systems, and which introduces provisions for a common understanding of basic terms. The term *ontology* is normally used in this context to denote a logical formalization (using one of the available languages) of a particular (sub)domain of knowledge, stored in a commonly understood format and accessible via the world wide web or a similar mechanism.

Applicability area of this research is not limited to just the world wide web. Below we present our work in a project devoted to introducing limited elements of artificial intelligence into production design and production support (system integration). The general aim of the project is to support engineers and make the production engineering easier (and thus cheaper) in most circumstances. The project is funded by EU, in hope that obtained results and products will enhance the productivity in small and medium enterprises (SMEs), although this goal is outside the scope of this paper.

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

[1]Department of Computer Science, Lund University, Box 118, S–221 00 Lund. Sweden, `jacek@cs.lth.se`.

The paper describes the project, discusses the possibilities, introduces the KR method chosen, analyses consequences of this choice, presents briefly related works and makes some preliminary conclusions.

2. SIARAS

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 is to build an intelligent system, named provisionally *the skill server*, capable of supporting automatic and semi-automatic reconfiguration of a manufacturing processes.

The main objective of the design process was to merge two, somewhat opposed, views on the reconfiguration process: the *top-down*, *AI-based* view and the *bottom-up, engineering* one.

2.1. Top-down AI approach

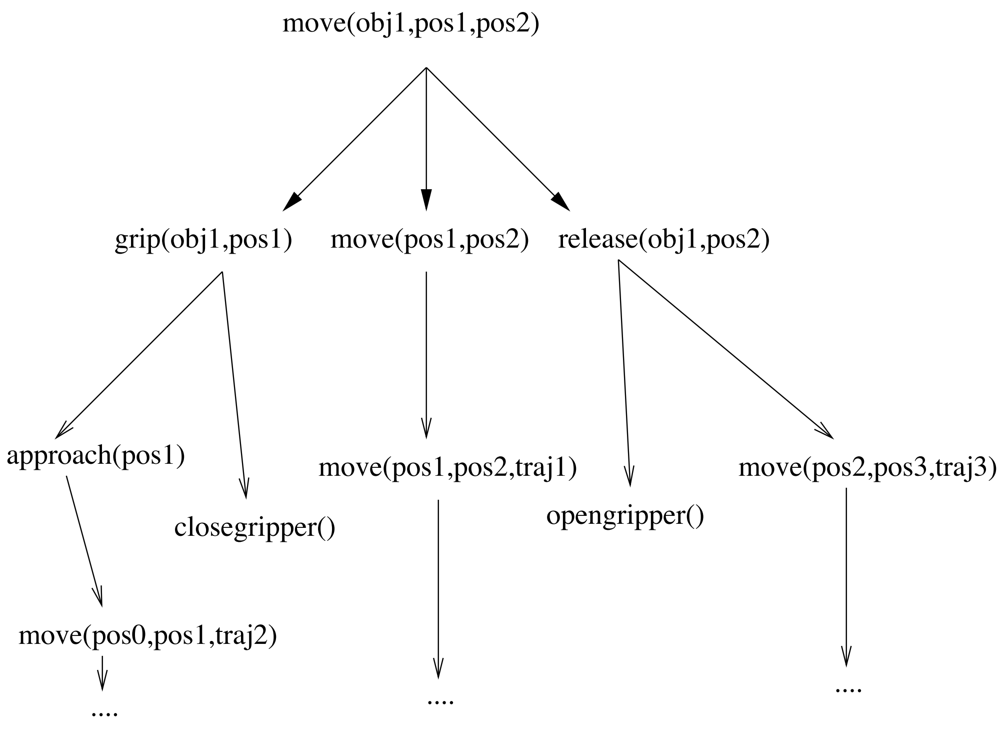
The top-down approach describes the problem of reconfiguration as a (re)planning problem: given a new task (usually expressed as a *goal* in AI), possibly being a modification of the previous one, and given a set of skills available in the system, understood as a description of the operations that might be performed by the devices available to the user, find such a sequence of operations that would ensure that the task is correctly executed (find such a plan that the goal gets achieved).

In order to achieve sufficient precision one needs to define such terms as *tasks*, *skills*, *operations* and *plans*. In the AI such representation is usually a symbolic one (e.g., an appropriate logic), formal and rather abstract: it does not specify the details of operations being discussed. For example, a robot would **move(object1, frompoint, topoint)** transforming a state in which **at(object1, frompoint)** holds into a state in which **at(object1, topoint)** holds. This level of abstraction is definitely not interesting for a production engineer, therefore one can foresee that a hierarchy of levels of abstraction will need to be employed here. This in turn corresponds to the notion of *hierarchical planning*, where one abstract operation is further specialized into more concrete ones, possibly passing a number of levels before the final, realizable plan is created. In our trivial example one could imagine at least the following levels of abstraction (see Figure 1):

- **move(obj1,pos1,pos2)**
- **move(pos1,pos2)** (possibly holding something)
- Move end effector of the robot from location $(x,y,z,\alpha,\beta,\gamma)$ to another location $(x',y',z',\alpha',\beta',\gamma')$ (note cartesian/polar coordinates)
- Move end effector of the robot from location $(x,y,z,\alpha,\beta,\gamma)$ to another location $(x',y',z',\alpha',\beta',\gamma')$ along a collision-free trajectory $f(x,y,z,t)$
- Move end effector of the robot from location $(x,y,z,\alpha,\beta,\gamma)$ to another location $(x',y',z',\alpha',\beta',\gamma')$ along a collision-free trajectory $f(x,y,z,t)$ minimising energy use (minimising operation time, etc.)

All those levels need to be represented and manipulated by the skill server. For that purpose the system needs a set of representation formalisms allowing reasoning and computations at each level of the hierarchy. This would include, on the lower levels, covering of topics such as kinematic models, dynamic models, and collision-free path planning.

One of the techniques particularly suitable to be used here, and offering appropriate computational tools, is *Hierarchical Task Network Planning* (cf. Chapter 11 in [5]). State-of-the-art planners have efficient implementations capable of successfully handling restricted domains, such as the one envisioned for the SIARAS project.



Rys. 1. A fragment of the hierarchy of abstraction levels.

Later in the paper we shall introduce two hierarchies representing knowledge about skills (i.e., capabilities of devices actually or potentially available for the end-user) and tasks (i.e., a generic representation of the operation that might be requested from a production system), respectively. On the topmost level, the task of the skill server might be to identify available skills in the production system at hand, identify the task to be executed (as stated by the user), and find a plan that would orchestrate skills so that this task is accomplished. That means that the two hierarchies should meet in some point, which corresponds to the concrete task (plan), where every single elementary subtask is realized by a concrete skill available in the system.

This meeting of hierarchies will allow us to reason about parameters for concrete tasks (actions in AI, operations elsewhere), down to the level of actual control programs, if necessary. The task will get the parameters instantiated by the values retrieved either from the skill description, at symbolic level, or from an attached database of control programs, i.e., concrete realizations of the skills offered by particular device performing it.

One particularly interesting aspect of this approach is the necessity of speaking about sensors as devices offering a skill of *information gathering*, i.e. ones producing input to the subsequent steps in the production chain. Planning system employed needs to, therefore, allow for representation of information-gathering actions — a non-trivial problem in itself. However, it will not be discussed in this paper.

In principle, every sensor may be described in many ways, depending on the considered application. E.g., a CCD camera may be used as a movement detector, color detector, presence detector, inspection device (classification: pass or no pass), classification device (into a number of classes), feedback-providing device (during welding or cutting), etc. At the bottom there is a physical description stating parameters like number of pixels, sensitivity, geometry of the lens, field of sight, and so on. At the top, there is direct correspondence to one of user-visible tasks. We could imagine a number of levels in between, details depending on how much do we want the skill server to reason about sensors and their skills.

Yet another interesting problem is how to describe sensor fusion for the purpose of reasoning, e.g., how to express the fact that a number of (say, linear) distance sensors may be used together for 3D triangulation. Should the compound skills be described as abstract operations, with preconditions, postconditions, etc? What would be the consequences of choosing this approach? The knowledge necessary for such reasoning is probably too complex to be included in a generic form. We assume that in the case they are necessary, such compound operations can be described as units – for the sake of simplicity.

To summarize, the skill server expects a large knowledge base with multiple representations for both skills and tasks, a description of the concrete task to be realized (a product description in some form, like CAD data or assembly operations — this is the actual input during from the end-users), a database of actual devices available, and an efficient planning system. Its output would be a specification of the production process, either fulfilling the criteria set by the user or explaining why could they not be met and suggesting changes that would remedy the problem.

In general case the complexity of such a solution would be, obviously, prohibitive. In our limited domain, however, there are chances of creating a solution that would be computationally feasible. At this point we consider knowledge engineering (extracting, codifying, implementing) as the major problem on our way towards the final product.

2.2. Bottom-up Reparametrisation Approach

The assumption in this approach is that the skill server is used only for reconfiguration of an existing, correct, properly modeled production process. The system is not expected to propose novel solutions, nor to search for alternative possibilities of implementing the process. Rather, it will be provided a complete description of a concrete, feasible and correct production process. In particular, one should expect a description of the task, i.e., what is produced, how (what are the steps or states of the process, i.e.

the subtasks) and how each step (subtask) of the process contributes to the goal. For a particular task this would probably form a tree-like structure. The knowledge base would contain a number of such trees (i.e., a forest), possibly represented as a directed acyclic graph. Specification language is left open until next section.

Skills in this approach describe each device in terms of, among other attributes, (sub)tasks they can realize — a mapping to the leaves in the task hierarchy should be, therefore, straightforward. Some of the interesting problems are the specification of boundary conditions between the elementary tasks (success/failure distinction, conditions of merging two tasks in series or in parallel, etc.) and, again, of a hierarchy of skill descriptions, from abstract to concrete.

The skill server is expected to maintain the matching between skills and tasks, temporal ordering of the skills, and details about execution of each skill (a parameterized description, possibly a control program as well).

The skill server may, thus, be queried in the following manner: an engineer (a sufficiently knowledgeable user) modifies the process description in some way. It might be a parameter change, an exchange of tasks, an exchange of boundary conditions, etc. An interesting open problem is the expected Human-Computer Interface as it will affect the set of possible questions.

Functionality of the skill server should be the following:

1. validity check of the process description (syntax, plausibility);
2. modification of the current implementation of the process, one of:
    - process reparametrisation;
    - process reconfiguration; or:
3. description of the problems encountered;
4. description of recipes (e.g., possible design patterns to employ);
5. suggesting solutions (depending on the vocabulary).

In step 2, after validating and accepting user input, the skill server should be able to analyze the new process regarding:

- feasibility (do the devices provide all the skills required by the new task);
- accuracy;
- reproducibility, robustness, reliability;
- timing, power consumption;
- collision avoidance;
- handling of error situations;
- cost.

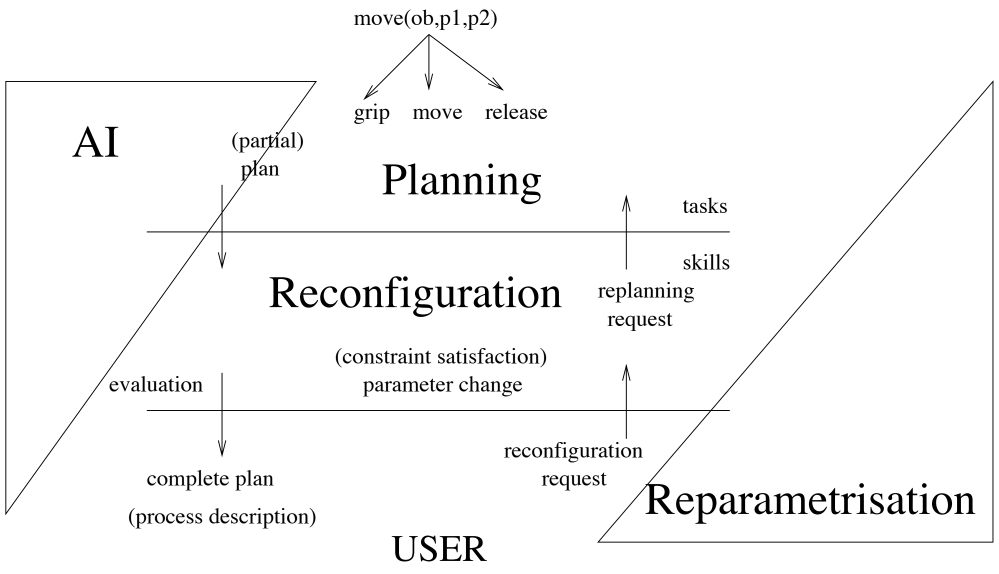
The steps 4 and 5 need to be repeated until the final acceptance by the user or until an apparent failure to deliver a solution. The main goal of the system is to **decrease the cost and time of reparametrisation or reconfiguration of the production process.** In particular, it is not to design a totally new process from scratch.

It should be stressed that we aim at a system that offers substantial ease of operation, as well as additional assistance to the users compared with other state of the art solutions.

2.3. Finding the Golden Middle

It seems that the top-down AI approach is computationally infeasible except in the very restricted cases, while the bottom-up reparametrisation lacks generality, as it would mainly consist of maintaining a database of possible parameter settings for a number of devices and retrieving appropriate values (possibly starting some simulations and running specific evaluation algorithms) from the database, given appropriate query. The main issue with this approach is guaranteeing scalability and extendibility to new domains or to new kinds of devices. There is a risk of limiting the approach to the considered cases and very similar ones only, thus precluding a more open-ended solution. On the other hand, such a database of specifics *will be* required in any reasonable solution to the problem.

In order to make sure that we do not loose the larger perspective while we aim at restricting ourselves to a feasible problem, we can imagine a layered approach, with reconfiguration level in the bottom and replanning level on top of it, to be run only if necessary (see Figure 2).



Rys. 2. Functional layers.

The issues involved with this approach would be deciding how to split a request into reconfiguration/replanning, i.e., what part of the query may be solved at the constraint satisfaction (reconfiguration) level and what requires replanning. To some extent this is related to the complexity of reasoning required by a particular query: depending on how much knowledge do we expect to be provided explicitly by the engineer (in the description of the current process) and how much would be necessary to deduce from it. This problem is closely related to the task of *plan recognition* (cf. [5]), where an automatic reasoner is expected to find out the purpose of a particular sequence of actions.

Yet another major challenge to the design lies in attaching all the lower-level algorithms specifically to particular skills (like, e.g., collision-free trajectory planning for robots, simulation programs, equational models, etc.) and making sure that their usage by the skill server will be orchestrated seamlessly.

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.

The 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) 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).

A carefully chosen set of representation primitives, together with a rather rich ontology and a set of efficient (i.e., polynomial) reasoning algorithms (available for as complex systems as description logics) allows us to keep the extension possibility open while focusing on a concrete demonstrator case for the beginning. As the project is done in cooperation with a German/Greek company INOS, providing system integration for automotive industry, we focus on a number of test cases proposed by the industrial partners.

We have chosen the open source tool Protégé [3] for ontology creation and manipulation due to its openness and modifiability. In particular, Protégé allows one to use reasoners adapted to the complexity of the employed representation, as discussed in the next section, and offers a multitude of visualization modes.

4. Formalisms

In the approach we have chosen 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 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 [6], Fact++[7], Algernon and Pellet. They differ in their reasoning power and efficiency, being able to handle either a restricted Description Logic language [9] (like OWL-DL offered by Protégé) in an efficient (polynomial complexity) manner [2], or a (more expressive) full OWL [11] 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.

We are also developing tools for storing and retrieving knowledge in appropriate data structures, so that on one hand the ontology can be easily extended by the system providers, while on the other hand it may benefit from distributedness, letting some parts be completed and stored at the device manufacturer's site. Yet another set of requirements is put on the reasoning process by the list of optimization tasks that may be requested by the user. Due to their computational complexity, and to their specificity to particular devices, they cannot be implemented in a general-purpose manner but rather require their specific reasoning blocks fitting the structure of the server.

5. Related works

The research on knowledge representation has been extensively documented, both in general textbooks on artificial intelligence and in numerous books devoted solely to this domain. One of the recent ones, offering a very good overview of the field, is [1].

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 [10]. In particular, the specifications of the most popular KR formalisms, like OWL [11] or DAML-OIL [4], together with available tools for using those formalisms, are available 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 [5], there is in principle no 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. Quite naturally, most of those formalisms and tools are heavily domain-dependent, with a small number of exceptions explicitly stating goal of being general-purpose tools. We may name here the *Sensor Modeling Language*, or Sensor ML for short, 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, it does not fit our point of view too well, since 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 [8]. The language, and some of the associated tools, are serving as a reference point for the ontology being developed within SIARAS.

6. Conclusions

In this paper we have presented the work on knowledge representation tools (formalisms and associated reasoning methods) that will be employed for creating and using the adaptive manufacturing ontology within the SIARAS project. Although offering the advantages of limited domain, adaptive manufacturing systems are still way too complex to be amenable to completely automatic analysis, therefore, a combination of automatic reasoning, domain-dependent non-formalisable computations, and user consultation are envisioned in the final system.

Knowledge representation in SIARAS is build around the concept of ontology, which, together with the pluggable special-purpose reasoning modules, form the core of the *skill-server*. The system is currently in its late design phase and a prototype implementation is expected during summer.

LITERATURA

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

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

[3] M. Musen et. al. The Protégé ontology editor and knowledge acquisition system. Primary web site, 2006.

[4] D. Fensel et al. OIL: An ontology infrastructure for the semantic web. IEEE Intelligent Systems, 2001, wolumen 16, numer 2, s. 38–45.

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

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

[7] I. Horrocks. FaCT and iFaCT. In: Proceedings of the International Workshop on Description Logics (DL'99). *Proceedings* Red. P. Lambrix et al, 1999, s. 133–135.

[8] C. Schlenoff et al. The Process Specification Language (PSL): Overview and version 1.0 specification. Raport instytutowy NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.

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

[10] W3C. Semantic web. Main Web site, 2001.

[11] W3C. Web ontology language (OWL). Main Web site, 2003.