Experiences in Applying Artificial Intelligence within SIARAS Project

Sławomir Nowaczyk

Katedra Automatyki,
Akademia Górniczo-Hutnicza,
Slawomir.Nowaczyk@agh.edu.pl

Abstract. In this paper we present our experience in designing and building knowledge base for the SIARAS project. This project aimed to produce a tool, called Skill Server, supporting engineers during reconfiguration of manufacturing systems. We mainly focus on the process of developing knowledge representation used by Skill Server and discuss some lessons we have learned about applying Artificial Intelligence concepts in practical applications within high-tech industry. We discuss several types of knowledge that Skill Server needs to have access to, and how different KR solutions can be integrated together in a coherent system.

1 Introduction

In this paper we present our experience in designing and building knowledge base for the SIARAS project, focusing especially on how the approaches we used evolved as we gained more and more insight into the actual requirements of the system.

SIARAS is an acronym of an EU-funded (FP6 - 017146) STREP-project "Skill-Based Inspection and Assembly for Reconfigurable Automation Systems," running in the years 2006–2008. Its general aim was to support end users and engineers of manufacturing systems, including robotic ones, and to make production engineering easier (and thus cheaper) in several common circumstances.

The primary goal of the project was to build an intelligent system, provisionally called *Skill Server*, that would be capable of supporting automatic and semi-automatic reconfiguration of manufacturing processes in response to changing requirements. The main issue during the design phase was to merge two, somewhat opposed, views on the reconfiguration process: the top-down, AI-based approach and the bottom-up, engineering one.

The top-down approach describes the reconfiguration as a (re)planning problem. We are given a new task, usually expressed as a goal condition, possibly being a modification of an earlier, correct one. 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, we are to find such a sequence of operations that will ensure that the task is correctly executed, i.e. to find a plan that achieves the goal.

In the bottom-up approach, the Skill Server is used only for reconfiguration of an existing, correct, properly modelled production line. The system is not expected to propose novel solutions, nor to search for alternative ways of implementing the process. In particular, one should expect a detailed description of the task: what is produced and how (i.e. what are the steps of the process). Moreover, for each step, it should be clear how does it contribute to the goal. On the other side, available devices must be described in terms of operations they are able to perform (skills) and conditions under which they can operate. Skill Server needs to map task into skills and parametrise them appropriately.

It is rather obvious that the top-down AI approach is both computationally infeasible and impossible to model sufficiently well, while the bottom-up reparametrisation approach lacks generality and risks ending up as a database of *previously used* parameter settings for a number of devices in a number of scenarios. 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 previously considered cases and very similar ones only, thus precluding a more open-ended solution.

Taking this into account, we have settled for a layered approach, with reconfiguration level at the bottom and (re)planning level on top of it.

2 Knowledge Representation

Initially, we have identified several types of knowledge Skill Server will use: skills, devices, tasks, workpieces and environment. Most of them can be specified on, at least, two levels of abstraction: simplified, generic descriptions (like a universal "pickup skill") and instantiated ones (the operation of gripper G_1 picking the windshield W_1 in factory F, at time point t_n and position p_m). Throughout the project, however, we have been continuously investigating possibilities of introducing additional, intermediate levels of abstraction in between.

Nevertheless, in addition to the symbolic knowledge, there exist a number of domain-specific or device-specific procedures for calculating various aspects (e.g. trajectory planner, device calibration and reparametrisation procedures, etc.) which, in many contexts, should also be treated as knowledge. Despite several attempts, however, we have not managed to find an acceptable and useful way to put them into any kind of even semi-formal framework. This would be a very interesting research project in itself, but our experience shows that this knowledge is simply too diverse to formalise in the timeframe we had. Therefore, Skill Server considers such procedures as, essentially, black boxes.

The overall structure of the Skill Server is shown in Figures 1 and 2. They depict, from two different points of view, how the "core" Skill Server is integrated with various knowledge, reasoning and user interface modules and what kind of information sources it has at its disposal.

In effect, we have devoted most of our efforts to designing symbolic knowledge representation in a way that would be intuitive and useful in practice to our industrial partners (who, in general, have no background in AI) and, at the

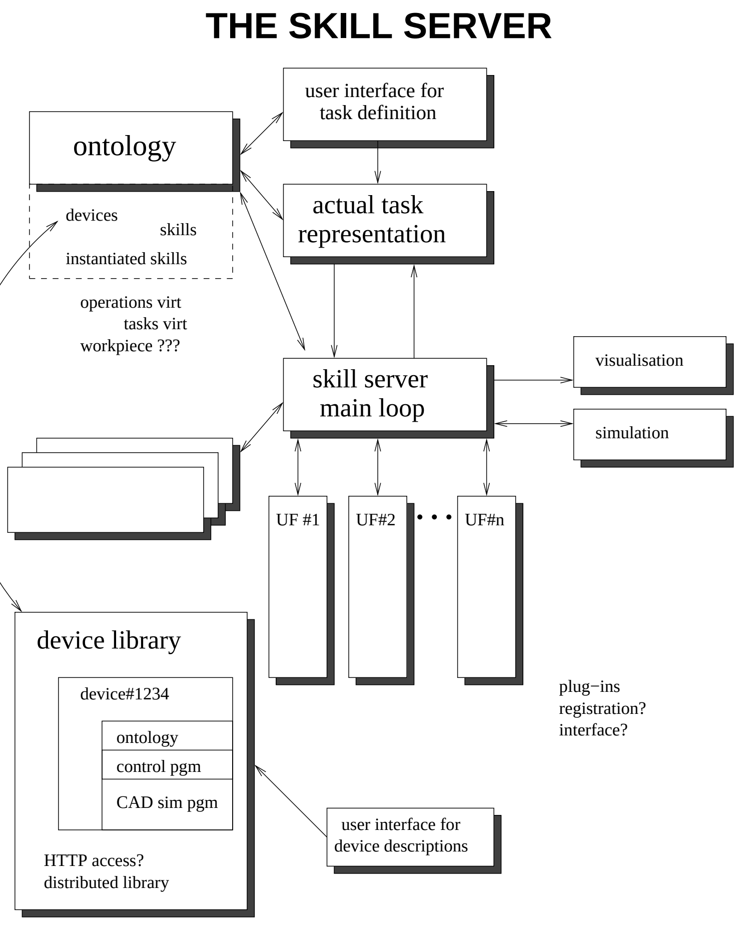

Fig. 1. Skill Server structure

same time, allow Skill Server to perform the reasoning tasks required. It turned out to be a surprisingly difficult task to agree upon a knowledge representation formalism within the consortium, however. In fact, while we (from the academia side) were already aware that there are costs associated with formal approaches to KR, we have not anticipated how difficult it will be to convince industrial partners that the benefits outweigh those costs.

In principle, the initial response was that any representation more advanced than *attribute-value* pairs was deemed as unnecessarily complex and still not expressive enough. Basically, the expectations seemed to be that we should use attribute-value pairs wherever appropriate, and full expressiveness of programming languages everywhere else. Finally, the consortium have eventually managed to agree upon using *Ontology* as a default knowledge representation formalism, provided we create a simple and intuitive one, not use existing.

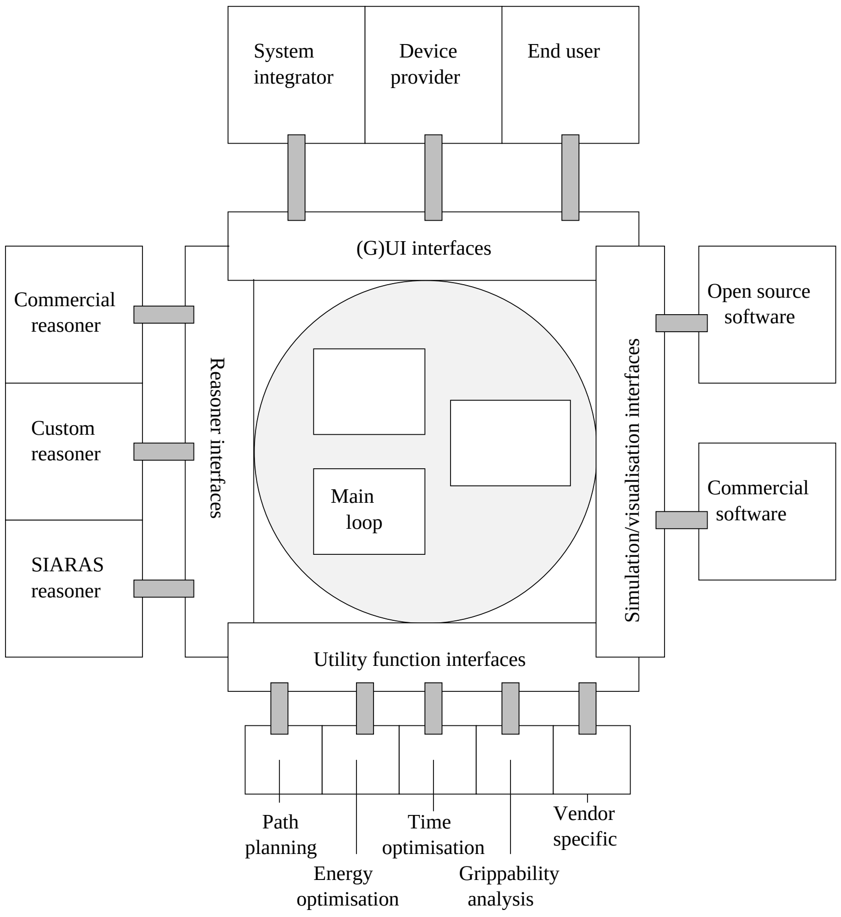

Fig. 2. Skill Server structure

One of the issues that caused the most discussions among the project partners was the contents and structure of the ontology. The latter question is mostly technical and amounts to fixing a representation which would make typical uses of the knowledge stored in the ontology as easy as possible. Still, during discussions, those two aspects seemed to be confused very often.

However, the former issue, i.e. what should the contents of the ontology be, was of paramount importance. There was little doubt that it should contain knowledge about skills and about devices providing them. What was not so obvious was the status of tasks: whether they should be present in the ontology as a separate entities.

There are strong arguments both for and against such solution. As tasks are one of the major concepts present in the basic idea of the Skill Server, they need to be included in the ontology (which serves as Skill Server's primary vocabulary) as well. Without their existence, no reasoning about them would be possible and thus the Skill Server would not be able to perform some of its basic functions.

However, on the most basic level, tasks necessarily have to correspond directly, on the one-to-one basis, with the skills. Otherwise there would be no possibility for the skill server to perform any kind of matching between them. Therefore, it is probably unwise to exactly duplicate the hierarchy of skills with a corresponding hierarchy of tasks. Moreover, the "lifetime" of tasks is also significantly different than the rest of information within the Ontology: skills and devices are almost eternal, in a sense that they can be input into the knowledge base once and are useful for everybody any time in the future. Tasks, on the other hand, are defined within a very specific context and usually only useful within a particular factory shop and for a limited time.

The ontology contains knowledge about (abstract) skills and about devices. 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". In a similar manner, a "vacuum-pickup" skill is a subconcept of "pickup" skill. Even though the expressive power of an ontology goes much further, we did in fact get most mileage out of it by treating it as little more than a glorified taxonomy. This part was considered sufficiently easy and sufficiently useful by all partners within a consortium to make it universally accepted. In fact, in some regards the ontology proved *too successful*, and there has been significant pressure to put virtually *all* knowledge there, even the kinds for which it is clearly inappropriate.

A useful feature of the ontology was that it allowed us to specify properties of each skill and device, with natural inheritance rules. Thus we were able to associate "mass" property with the concept of "device" and "number of fingers" with "finger gripper", to specify that "mass" can be expressed in "grams" or "pounds", and that "max image resolution" makes sense only for cameras, and not for robots. What we did not manage to do is to find a way to express the *purpose* of those properties in a way that would be accessible to the non-symbolic parts of knowledge, such as calibration procedure or specialised domain-dependent algorithms. However, this has more to do with how unstructured this procedural knowledge turned out to be and less with the ontology itself.

Another issue we have been debating many times within the consortium but never managed to reach a satisfactory conclusion was the concept of *compound devices*. For example, when talking about robots, it often appears useful to specify that a *robot* uses a *gripper* to perform some *skill*. It only seems natural to express that robot-with-a-gripper can perform more (or rather different) actions than robot alone. However, doing it properly turned out to be outside the representational power of an ontology, at least in the form we had it in the project.

3 Ontology Structure

We have decided to center knowledge representation around the concepts of devices (physical objects provided by their manufacturers) and skills (operations that can be performed). Task descriptions exist only during problem solving sessions, as dynamic structures, specific to a particular case. They can be seen as (arguably, quite complex) combinations of skills and parameters 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. While ontologies are often used for classification purposes, in our case the classification is done when objects (skills or devices) are introduced in the structure. The main use of the ontology is to allow reasoning about skills matching particular tasks and about devices being suitable for particular operations, as well as to standardise the nomenclature used and the relationships of different concepts.

For the prototype of Skill Server, we have chosen the open source tool Protégé for ontology creation and manipulation, together with reasoners such as Racer[1], Fact++[2], Algernon[3] and Pellet[4]. If Skill Server is ever to enter production stage, a specialised user interface will need to be developed, since general purpose tools we such as those are not appropriate for its intended end users. They worked fine within the SIARAS project, however.

The ontology contains three hierarchies: Devices, Skills and Properties. The Device hierarchy specifies what types of devices do we intend Skill Server to reason about. We have provided device manufacturers with a rudimentary interface for introducing their products: they specify where in the hierarchy should a particular device reside and, depending on that, they are asked to fill in values of appropriate properties. Those devices are, at least conceptually, leaves (instances) of the Device hierarchy. However, since we expect no centralised repository of all devices to be available and assume that data will be organised in a distributed manner (for example, downloadable from device manufacturers' web pages), we are actually storing all the devices in an SQL database.

The second hierarchy is the Skill hierarchy, which mainly aims at providing a common vocabulary between the device manufacturers (who specify which skills does a device offer) and the end users or systems integrators (who specify what effect do they want to achieve, or which task do they want performed). The Skill hierarchy is supposed to be a common ground where the capabilities offered by device manufacturers can be mapped into the requirements of the users. It performed satisfactorily during the project, including the demonstration stage, but it has by now became clear that we are missing a common understanding of an appropriate abstraction level for skills. For example, there was much discussion on whether there should be a "drill" skill, or if it should rather be modelled as a sequence of "start drill rotation", "move", "stop drill rotation" operations. In the end we have gone with the former approach, but we are not convinced this is the best solution and would rather have a setup where different abstraction levels could be used as appropriate.

There has been much debate about the need for complex (or composite) skills, but we do not think it is appropriate to represent them directly in an ontology. It would make more sense to provide a more expressive way of building them, using a language like Finite State Machines or Sequential Function Charts. The skills in the ontology should be simple enough that it is possible to reason about their constraints and effects of their application.

Finally, the Properties hierarchy forms a link between Devices and Skills. It is, in a sense, the first – simplest – way of specifying the dependencies and instantiating parameters (for example, that a particular gripper has a certain maximum weight limit).

Throughout the project we have kept our initial assumption that all the hierarchies will be defined by the SIARAS consortium and remain fixed, as the meaning of all the concepts used needs to be encoded into the reasoning mechanisms of the Skill Server. However, it has become apparent that this assumption is a very constraining one, so it would be good to investigate ways to lift it.

4 Knowledge Representation outside Ontology

The major part of knowledge representation we have decided to store outside the ontology are the tasks. Tasks can be thought of as *generalisations* of skills along (at least) two axes: first, they are time-ordered (or partially parallelised) sequences of skills (or rather of skill applications): for example, a "relocate" task can be seen as a sequence of "pickup", "move" and "putdown" skills. Second, it makes sense to have hierarchy of tasks, with "windshield fitting" task decomposing into "find windshield", "position windshield" and "fix windshield" subtasks.

In addition, tasks constitute means of achieving a particular goal, and Skill Server needs to have a representation of that goal, in particular to reason about rationale for each operation and about motivations why a particular device and particular parameters were chosen. At the very least it requires a set of criteria which distinguish acceptable execution of this task from unacceptable ones, and possibly ways to *compare* two solutions and determine when one of them is better than the other.

The bottom level of tasks hierarchy are *operations*, which are a kind of associations of skills, devices, workpieces, factory floor positions and time-line constraints. Where tasks are especially useful is on a higher level, when defining a concrete production process that will be the subject of reconfiguration. Therefore the ontology does not need them, or putting it differently, the only tasks that are available in the ontology are those that skills refer to.

We have chosen Sequential Function Charts[5] to represent tasks, since it allows us to specify temporal ordering of operations in an intuitive and yet flexible way. We have used the open source tool JGrafchart for our prototype.

Finally, we have implemented the core reasoning in Python programming language, "gluing" together the knowledge from ontology, SFCs and device-dependent procedures provided in the form of plugins.

In our approach the ontology is used for reasoning about skills matching particular tasks (after some initial re/parametrisation) and devices offering those skills under certain conditions. A pure ontology may be used for retrieval, matching and simple classification, while other forms of reasoning, like planning, optimisations, consistency checks, etc., need to be done by reasoners, either general-purpose ones, or specialised for the task. The generic tools that have been used by in the project (Racer, Fact++, Algernon and Pellet) differ in their reasoning power and efficiency, being able to handle either a restricted Description Logic language [6] (like OWL-DL offered by Protégé) in an efficient manner [7], or a full OWL [8] representation, but using exponential search algorithms. The user has the possibility of choosing a different reasoner depending on the question asked, thus achieving flexibility and adaptability of the reasoning part of the system.

We have also developed 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 optimisation 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 Work

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, and a very good overview of the field, is by Brachman and Levesque [9].

The work that originated discussions over semantic web and, in particular, on ontologies, has an extensive library of published documents available for example at the W3Consortium's Semantic Web site [10]. In particular, the specifications of the most popular KR formalisms, like OWL [8] or DAML-OIL [11], together with available tools for using those formalisms, can be found there.

Production planning is usually considered in AI to be a part of the automatic planning domain. However, besides the classical manufacturability analysis, reported for example in [12], 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 modelling 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 Modelling Language*, or Sensor ML for short, offering a rich sensor ontology (see `http://www.sensorml.org` for an extensive documentation). A dual enterprise, not exactly fitting our point of view either, is the *unified robot modelling language*, (URML), from the University of Karlsruhe, as URML does not provide representation facilities for the dynamic aspects of robot performance.

Finally, an important attempt to formalise the language for speaking about production processes has been done at NIST which created the Process Specification Language [13]. The language, and some of the associated tools, are serving as a reference point for the ontology developed within SIARAS.

6 Conclusions

In this paper we discuss the knowledge representation we use in the EU-funded project SIARAS, and present the most interesting points where real-life considerations clashed with our initial assumptions. We hope this overview will be helpful to other people designing knowledge bases for industry applications and will allow them to avoid some of the pitfalls we have encountered.

The main point of our discussion was the use of Ontology and what are the benefits and costs associated with representing knowledge in such structured way. We have discussed situations where ontology turned out to be a rather cumbersome tool, and also those where it proved to be very helpful. We wish we were able to offer some kind of evaluation of the approach we have chosen and of the representation we have ended up with, but unfortunately we do not have anything more concrete than our own reflections.

References

1. Haarslev, V., Möller, R.: Racer: A core inference engine for the semantic web (2002)
2. Horrocks, I.: FaCT and iFaCT. In: Proceedings of the Description Logics Workshop, DL'99. (1999)
3. Stoica, F., Pah, I.: Intelligent agents in ontology-based applications. In: 12th WSEAS International Conference on Computers. (2008) 274–279
4. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Description Logics. Volume 104 of CEUR Workshop Proceedings. (2004)
5. Fernández, J.L., Sanz, R., Domonte, E.P., Alonso, C.: Using hierarchical binary petri nets to build robust mobile robot applications: Robograph. In: International Conference on Robotics and Automation. (2008) 1372–1377
6. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook. Cambridge University Press (2003)
7. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. Journal of Artificial Intelligence Research **1**(1) (1993) 109–138
8. W3C: Semantic web (2003)
9. Brachman, R.J., Levesque, H.J.: Knowledge Representation and Reasoning. Morgan Kaufmann (2004)
10. W3C: Semantic web (2001)
11. Fensel, D., Horrocks, I., van Harmelen, F., McGuinness, D.L., Patel-Schneider, P.F.: Oil: An ontology infrastructure for the semantic web. IEEE Intelligent Systems **16**(2) (2001)
12. Ghallab, M., Nau, D., Traverso, P.: Automated Planning, Theory and Practice. Morgan Kaufmann (2004)
13. Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., Lee, J.: The process specification language (PSL): Overview and version 1.0 specification. Technical report, National Institute of Standards and Technology (NIST) (2000)