Proceedings of the 3rd Annual
IEEE Conference on Automation Science and Engineering
Scottsdale, AZ, USA, Sept 22-25, 2007

Knowledge-Based Reconfiguration of Automation Systems

Jacek Malec, Anders Nilsson, Klas Nilsson, Sławomir Nowaczyk

Abstract— **This article describes the work in progress on knowledge-based reconfiguration of a class of automation systems. The knowledge about manufacturing is represented in a number of formalisms and gathered around an ontology expressed in OWL, that allows generic reasoning in Description Logic. In the same time multiple representations facilitate efficient processing by a number of special-purpose reasoning modules, specific for the application domain. At the final stage of reconfiguration we exploit ontology-based rewriting, simplifying creation of the final configuration files.**

I. 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.

The *semantic web* project [1] 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 of a particular 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 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 industrial robot systems available today are only justifiable in enterprises with long or repeated product series and with highly trained staff to program and configure the robots. Small and medium enterprises (SMEs) typically manufacture many different products in short series and need to be able to reconfigure the manufacturing line within a day or so, which is virtually impossible with today robot systems. The project is expected to contribute to remedying this situation.

The approach taken assumes multiple knowledge representation formalisms coexisting in the system. Depending on what kind of reasoning needs to be performed the appropriate representation is chosen and a suitable reasoning engine is used to reach conclusions. Distributedness and multiple representations are the crucial factors allowing us to address non-trivial reconfiguration tasks. Efficient transformations between those representations are basic ingredient of the solution. In particular, the mechanism of *ontology-based compilation* is employed for creation of configurations, facilitating smooth transition from a generic process representation to device-specific code.

The paper is divided as follows. First we describe the project and discuss the choices to be made. Then the current system architecture is briefly presented, with focus on knowledge representation formalisms used. Next the ontology-based compilation is described in more detail. Finally we briefly present related works and make some preliminary conclusions.

This work has been supported by the EU-project SIARAS *Skill-Based Inspection and Assembly for Reconfigurable Automation Systems* (FP6 - 017146).

The authors are with Department of Computer Science, Lund University, Box 118, S–221 00 Lund. Sweden, [jacek, andersn, klas, slawek]@cs.lth.se.

II. 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 phase was to merge two, somewhat opposed, views on the reconfiguration process: the *top-down, AI-based* view and the *bottom-up, engineering* one.

A. 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 a plan such that the goal gets achieved).

In order to achieve sufficient precision one needs to define such terms as *task*, *skill*, *operation* and *plan*. In 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:

- **move(obj1,pos1,pos2)**
- 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 $f(x, y, z, t)$ (possibly holding something; note cartesian/polar coordinates)
- Move end effector of the robot from location $(\alpha_1, \ldots, \alpha_6)$ to another location $(\alpha'_1, \ldots, \alpha'_6)$ (note robot coordinates)

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 includes on the lower levels, among others, kinematic models, dynamic models, and collision-free path planning. One of the techniques particularly suitable to be used here is *Hierarchical Task Network Planning* [2].

We have defined a hierarchy 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 operations that might be requested from a production system). On the topmost level, the task of the skill server might be summarized as: 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 (of tasks and skills) 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 allows us to reason about parameters for concrete tasks, down to the level of actual control programs. The task gets the parameters instantiated by the values retrieved either from the skill description, at a symbolic level, or from an attached database of control programs, i.e., concrete realizations of the skills offered by the particular device performing it.

Summarizing, 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 and assembly operations), a database of actual devices available, and an efficient planning system. Its output is a specification of the production process, either fulfilling the criteria set by the user or explaining why they could not be met and suggesting changes that would remedy the problem.

In the general case the complexity of such a solution is, obviously, prohibitive. Therefore another approach needs to be considered for addressing the reconfiguration problem.

B. 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. A suitable hybrid (discrete and continuous) representation formalism needs to be used in this case.

Skills in this approach describe each device in terms of, among other attributes, (sub)tasks they can realize — a mapping to 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 current process description in some way. It might be a parameter change, an exchange of tasks, an exchange of boundary conditions, etc. Then the functionality of the skill server should be the following:

1) validity check of a process description (syntax, plausibility);
2) modification of the current implementation of the process, either reparametrisation or 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 updated 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.

C. Finding the Golden Middle

It seems that the top-down AI approach is computationally infeasible except in 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 extensibility to new domains or to new kinds of devices. On the other hand, such a database of specifics *will be* required anyway in any reasonable solution to the problem.

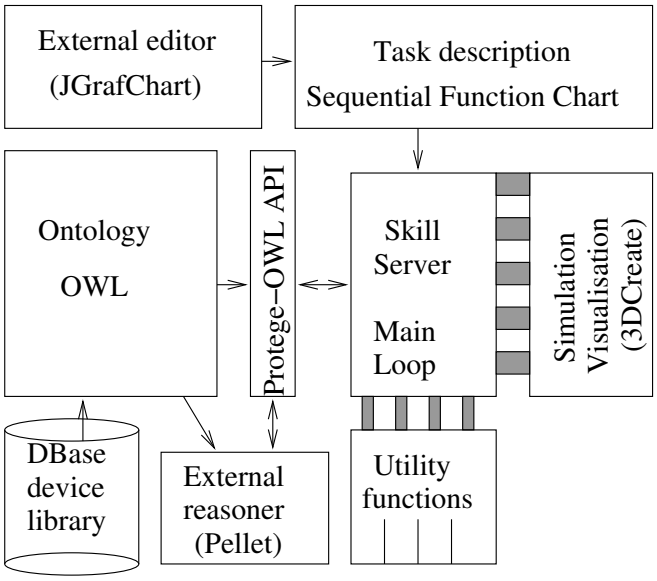

Fig. 1. Current architecture of the skill server.

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, the latter to be run only if necessary.

The main issue in such approach consists of deciding how to split a request into reconfiguration and replanning, i.e., what part of the query may be solved at the constraint satisfaction (reconfiguration) level and what requires replanning to be run. To some extent this is related to the complexity of reasoning required by a particular query: depending on how much knowledge we expect to be provided explicitly by the engineer (in the description of the current process) and how much would be necessary to be deduced.

III. Architecture

Figure 1 contains a sketch of the current status of the skill server. The design is based on the assumption, expressed in the following section, that the vocabulary elements used by the server are: tasks, skills, devices, workpieces and operations.

One can easily note that there are two strongly connected components in this picture: the main loop of the skill server, and a module named *ontology*. The ontology holds all the generic knowledge of the system, knowledge about skills (and tasks they are capable of performing, provided suitable devices), about sensors and actuators that are involved in performing skills (i.e., devices), the operations that may be performed by instantiated skills (i.e. with a fixed device associated with it), the workpieces involved in the production process, etc.

The flow illustrated from top to bottom of the figure corresponds to the intended mode of use of the skill server. First, the current task has to be defined by a user (be it system engineer or end-system-user), possibly using a suitable GUI. In order to constrain the task descriptions to ones understood by the skill server, the GUI 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 structure, subsequently manipulated by the skill server.

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. The server analyzes 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 via domain-dependent reasoning modules, illustrated here as *utility functions*, attached to the core server using well-defined protocol and interface. A prototype based on those ideas has been already implemented and is currently intensively tested. It has to be noted that even visualization and simulation are realized as plug-ins.

Finally, a separate object on the image is the database, used as that part of the ontology that contains the *Device Library*. Formally, the device descriptions are elements (leafs) in the ontology. It is expected that device library will form "virtual parts" of the ontology, plugged-in as needed and as available. The libraries could be distributed and maintained by device manufacturers, 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. Of course, appropriate maintenance tools are expected to be created in the future.

IV. Knowledge representation

We have identified several types of (non-procedural) knowledge the 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 generic "pickup skill") and instantiated ones (the operation of gripper G_1 picking the windshield W_1 in factory F). We do not exclude, however, a possibility of additional, intermediate levels of abstraction in between.

In addition to those, there is a number of domain-specific or device-specific procedures for calculating various aspects (trajectory planner, device reparametrization procedures, etc.) which, in some contexts, can also be treated as knowledge. In general, however, Skill Server treats them as black boxes.

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).

We have chosen the open source tool Protégé [3] for ontology creation and manipulation. In particular, Protégé allows one to use reasoners adapted to the complexity of the employed representation and offers a multitude of visualization modes.

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, 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 tested by us so far are Racer [4], Fact++[5], Algernon and Pellet. They differ in their reasoning power and efficiency, offering a possibility of choosing different reasoners depending on the questions asked, thus providing additional flexibility and adaptability.

We are also developing tools for storing and retrieving knowledge in appropriate data structures, so that the ontology can be easily extended by the system providers, while 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 as utility functions.

V. Ontology-based compilation

One of the final stages of skill server computations is the creation of appropriate configuration files for the devices used to perform the task. One of the problems to be faced when (re)configuring a production system is how to deal with the different peripherals that are used in the task. There are many different kinds of peripherals ranging from simple on/off sensors to advanced vision systems and all possible kinds of robot grippers. Different peripherals have different characteristics (size, weight, speed, payload) and are interfaced (mechanically, electrically, and communication) in different ways with the robots. Communication is often not standardized, and done on a quite low level. If the data describing a peripheral was available in a structured format, it would perhaps be possible to automatically generate the peripheral configuration for a specific robot model.

Then there is a problem which specific peripheral model to choose for a task. This depends on the constraints associated with the specific task; speed, accuracy, payload, and so on. Also here would a structured peripheral description be of help. A reasoning system could then try to find the best match from the given constraints.

The idea then is to use the ontology developed for skill taxonomy also for robot peripherals, such as grippers, using

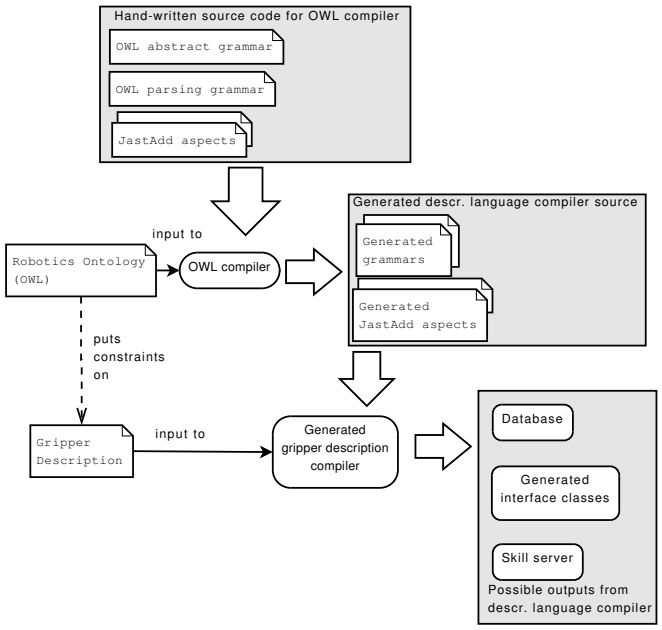

Fig. 2. Using OWL ontology to generate compiler for robotic grippers.

a standardized language. In the prototype we use OWL [6], but any ontology language could be used. This ontology then serves as specification language for peripherals, and is rather well-suited for automatic processing using either standard XML tools or tailor-made tools. However, a common problem with such specialized description languages is that developed tools are very tightly connected to a specific version of a specific description language. Tools made for gripper descriptions typically cannot easily be used for computer vision system descriptions. Changes in a description language will also imply changes in the tools.

As is not uncommon, such problems become easier to solve by moving up to the next abstraction level. By implementing a meta-compiler, a compiler for OWL that, as its output, generates a compiler for the description language specified in OWL, the abstraction level is raised. Instead of having to handle the dependencies between description language and tools manually, there is now one single specification for both the description language and the generation of tools. The fact that these description languages are XML-based helps in that the parsing syntax is given beforehand.

The general steps are depicted in Figure 2. A compiler for ontologies expressed in OWL is implemented using JastAdd [7] and a suitable parser generator. Given an OWL ontology as input, this compiler will then automatically generate the description of a compiler for the given description language. The automatically generated compiler can then be used to parse and analyze a description, and generate various forms of output depending on how the compiler was specified. Interface classes to sensors and actuators, or entries in skill server databases are just a few examples of what is possible to achieve.

The automatically generated code sometimes needs some additional hand-crafted code for a specific description lan-

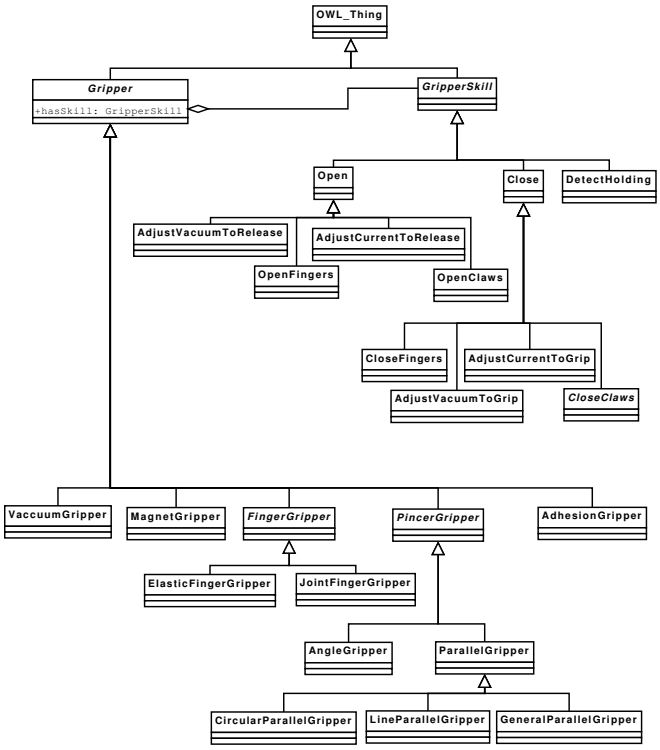

Fig. 3. Part of the robotic gripper model hierarchy.

guage. But it is still a far better situation than when the fundamental compiler description needs to be adapted after each change in the description language specification.

A. Prototype implementation

A prototype implementation of the OWL meta-compiler is currently being developed as a part of the SIARAS skill-server. As a realistic example on a non-trivial description language (the complete robot ontology being developed within the SIARAS project is far to large and complex to show here), consider a type hierarchy of robotic grippers, see Figure 3. Robotic grippers can be divided into four different categories based on gripping principle used; *vacuum grippers*, *magnetic grippers*, *finger grippers*, *pincer grippers*, and *adhesion grippers*. For finger- and pincer grippers, a more fine-grained division is needed because of functional differences with different mechanical solutions.

An OWL representation of the gripper model, as shown in Figure 3, serves as input to the OWL compiler. Even though it is rather hard to read for humans, as most XML syntax, it is quite easy to write a parser for OWL descriptions since a simple grammar only consists of the following two productions (in JastAdd syntax):

```
Element ::= Attribute* Element*;
Attribute ::= <STRING_LITERAL>;
```

Using the JastAdd tools, it is then easy to analyze the AST returned by the parser, and then generate a new abstract grammar from the AST.

```
Gripper : Thing ::= GripperSkill;
MagnetGripper : Gripper ::= ;
VacuumGripper : Gripper ::= ;
FingerGripper : Gripper ::= ;
JointFingerGripper : FingerGripper ::= ;
ElasticFingerGripper : FingerGripper ::= ;
PincerGripper : Gripper ::= ;
AngleGripper : PincerGripper ::= ;
ParallelGripper : PincerGripper ::= ;
AdhesionGripper : PincerGripper ::= ;
CircularParallelGripper : ParallelGripper ::=;
LineParallelGripper : ParallelGripper ::= ;
GeneralParallelGripper : ParallelGripper ::= ;

GripperSkill : Thing ::= ;
Open : GripperSkill ::= ;
OpenFingers : Open ::= ;
AdjustCurrentToGrip : Open ::= ;
OpenClaws : Open ::= ;
AdjustVacuumToRelease : Open ::= ;
Close : GripperSkill ::= ;
CloseFingers : Close ::= ;
AdjustCurrentToRelease : Close ::= ;
AdjustVacuumToGrip : Close ::= ;
CloseClaws : Close ::= ;
DetectHolding : GripperSkill ::= ;
```

B. Outlook

The current prototype, consisting of less than 400 lines of JastAdd code, can analyze a non-trivial OWL document and then generate a JastAdd abstract grammar for the description language as described by the OWL document. Regardless of which changes are done in the OWL-based specification, both the abstract and concrete grammars for the description language can be automatically generated.

In order to comprise a fully usable compiler for description languages for peripheral devices, some manually written code, here in the form of JastAdd aspects, is surely needed. If the language specification changes, there is a possibility that one has to make changes to this code also. However, there is a large difference between having to make minor changes in compact aspects, and changing the fundamental grammar descriptions in the compiler.

The described gripper-compiler is just one example of a tool to handle equipment interfaces. Many more end-effectors (for welding, glueing, grinding, and so on) exist, and there are many other types of external equipment (such as fixtures, feeders, and conveyors). Furthermore, interfacing in a robot work-cell involves more than the equipment, so in total there are at least the following items to cope with in terms of data interpretation:

- External equipment and their interfaces
- Control services
- Human operations
- Interaction devices
- Workpiece data and models
- Task descriptions and robot languages
- Model data for production processes
- Production or robot skill strategies

Up to now, there have been different standardization efforts in each of these areas. Most modern standards are XML based, which is sometimes referred to as being generic and

fully portable. However, having XML-based standards in each of the eight areas according to the items above does not really solve the problem; if these standards are based on different taxonomies, the data integration and coherence between different items remains a problem. That is, the different standards (despite being expressed in XML) are based on different terms, and the information integration still requires extensive engineering efforts.

By relating different standards to a common ontology, and using compilation in several stages as described above, we expect that the generation of configuration data in a robotized system can be much more generic. Clearly it will be a substantial effort to model all interfaces and devices, but if compiler technology can provide the means to interpret and relate terminologies from different areas, it should advance the state of the art not only in the area of reconfigurable systems but in the more generic *plug-and-produce robot systems* (`www.smerobot.org`) as well.

VI. Related works

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 [1]. In particular, the specifications of the most popular KR formalisms, like OWL [6] or DAML-OIL [8], together with available tools for using those formalisms, are available there. One of the recent textbooks on knowledge representation, offering a very good overview of the field, is [9].

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 [2], there is very little documented research on knowledge-based, automated production planning for non-trivial applications. An interesting recent result, documenting state of the art, is presented in [10].

Only very recently there is a growing interest in using ontologies and, more generally, "semantic web technologies", in factory automation [11]. 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. Most of those formalisms and tools are domain-dependent, with a small number of exceptions explicitly stating goal of being general-purpose, like e.g. the *Sensor Modeling Language* which offers a rich sensor ontology (`http://www.sensorml.org`). A dual enterprise is the *unified robot modeling language*, (URML), from the University of Karlsruhe. Unfortunately, it does not fit our work 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 [12]. The language and some of the associated tools are serving as a reference point for the ontology being developed within SIARAS.

Similar to ours, scalable, extensible architectures for intelligent automation have been proposed very recently, although in different contexts: either as agent-based distributed systems [13] or as object-oriented middleware systems [14], [15].

VII. Conclusions

In this paper we have presented the ongoing work on knowledge-based automatic reconfiguration system for robotized work-cells. Although offering 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 expected to coexist in the final system.

Knowledge representation in SIARAS is built around the concept of ontology, which, together with the pluggable, special-purpose reasoning modules, form the core of the *skill-server*. Late processing stages include ontology-based compilation for configuration purposes. The system is currently in its prototyping phase, with a demonstrator expected by the end of the year.

References

[1] W3C, "Semantic web," 2001, http://www.w3.org/2001/sw/.

[2] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning, Theory and Practice*. Morgan-Kaufman, 2004.

[3] M. Musen *et al.*, "The Protégé ontology editor and knowledge acquisition system," 2006, http://protege.stanford.edu.

[4] V. Haarslev and R. Möller, "Racer: A core inference engine for the semantic web," *Available at http://www.franz.com/products/racer/*, 2002.

[5] I. Horrocks, "FaCT and iFaCT," in *Proc. Int. Workshop on Description Logics DL'99*, P. Lambrix *et al.*, Eds., 1999, pp. 133–135.

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

[7] T. Ekman and G. Hedin, "Rewritable Reference Attributed Grammars," in *Proceedings of the 18th European Conference on Object-Oriented Computing (ECOOP) 2004*, ser. LNCS, no. 3086, 2004.

[8] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider, "OIL: An ontology infrastructure for the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 38–45, 2001.

[9] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[10] B. J. Clement, E. H. Durfee, and A. C. Barrett, "Abstract reasoning for planning and coordination," *Journal of Artificial Intelligence Research*, vol. 28, pp. 453–515, 2007.

[11] J. L. M. Lastra and I. M. Delamer, "Semantic web services in factory automation: Fundamental insights and research roadmap," *IEEE Trans. Ind. Informatics*, vol. 2, pp. 1–11, 2006.

[12] M. Grüninger and C. Menzel, "The Process Specification Language (PSL), theory and applications," *AI Magazine*, vol. 24, no. 3, pp. 63–74, 2003.

[13] W. Binder, I. Constantinescu, B. Faltings, K. Haller, and C. Türker, "A multiagent system for the reliable execution of automatically composed ad-hoc processes," *Autonomous Agents Multi-Agent Systems*, vol. 12, pp. 219–237, 2006.

[14] V. V. Vyatkin, J. H. Christensen, and J. L. M. Lastra, "OOONEIDA: An open, object-oriented knowledge economy for intelligent industrial automation," *IEEE Trans. Ind. Informatics*, vol. 1, pp. 4–16, 2005.

[15] I. M. Delamer and J. L. M. Lastra, "Service-oriented architecture for distributed publish/subscribe middleware in electronics production," *IEEE Trans. Ind. Informatics*, vol. 2, pp. 281–294, 2006.