Overview of an Ontology-based Approach for Kit-building Applications

Zeid Kootbally
Department of Aerospace and Mechanical Engineering
University of Southern California
Los Angeles, CA, USA
zeid.kootbally@nist.gov

Thomas R. Kramer
Department of Mechanical Engineering
Catholic University of America
Washington, DC, USA
thomas.kramer@nist.gov

Craig Schlenoff
Intelligent Systems Division
National Institute of Standards and Technology (NIST)
Gaithersburg, MD, USA
craig.schlenoff@nist.gov

Satyandra K. Gupta
Department of Aerospace and Mechanical Engineering
University of Southern California
Los Angeles, CA, USA
skgupta@usc.edu

***Abstract*—The Agility Performance of Robotic Systems (APRS) project at the National Institute of Standards and Technology (NIST) is using Web Ontology Language (OWL) ontologies for modeling in a robotic kitting workstation. The new technical idea for the APRS project is to develop the measurement science in the form of an integrated agility framework enabling manufacturers to assess and assure the agility performance of their robot systems. This framework includes robot agility performance metrics, information models, test methods, and protocols. This paper focuses on the information models and describes how they are used to introduce robot agility for the kitting domain. OWL class model files are generated automatically from XML schema model files. Files of OWL instances conforming to an OWL class model are generated automatically from XML instance files by automatically built translators.**

***Keywords*-Web Ontology Language; kit building; robotics; agility; knowledge representation; information model;**

I. Introduction

Robots will be a pervasive part of our lives in the coming decades. Whether it is behind the scenes in assembling products that you use in your everyday life, or helping you parallel park your car, robots are already playing a role in what we use and how we get around.

However robots are not good at everything. For the most part, robots perform best in highly structured environments, where objects are in well-known, predictable locations. Robots are also not known for "thinking on the fly" very well. They are best when they can be trained to perform a very specific activity, which requires a very specific set of motions, and that activity can be performed in the exact same way many hundreds or thousands times. Not surprisingly, robots have been adopted much more in high volume, repeatable operations such as car manufacturing than they have been in smaller job shop type operations where only a handful of similar products are being made at a given time.

Another way to describe this is that robots are not considered agile. But, in order for them to be useful to small manufacturers and to also allow larger manufacturers to offer more automated customization of high volume parts (think cars and cell phones), they need to be. The Agility Performance of Robotic Systems (APRS) project at the National Institute of Standards and Technology (NIST)[1] is addressing this challenge. The goal of this project is to enable agility in manufacturing robot systems and to develop the measurement science which will allow manufacturers to assess and assure the agility performance of their robot systems. Key areas of robot agility include: 1) the ability of a robot to be rapidly re-tasked without the need to shut down the robot for an extended period of time when a new operation needs to be performed, 2) the ability of a robot to recover from errors, so that when a part is dropped, for example, the robot can assess the situation and determine the best way to proceed to accomplish the goal, and 3) the ability to quickly swap in and out robots from different manufacturers so that a company is not tied to a single robot brand.

NIST is developing software, standards, and performance metrics to allow aspects of agility to happen. NIST is developing the Canonical Robot Command Language (CRCL) [1] which is a low-level messaging language for sending commands to, and receiving status from, a robot. CRCL is intended primarily to provide commands that are independent of the kinematics of the robot that executes the commands. This allows robots to be more easily swapped in and out since the robot commands are represented in a robot-agnostic format. NIST is also in the process of developing a set of performance metrics to measure robot agility, and is validating them at an upcoming Agile Robotics for Industrial

[1]https://www.nist.gov/programs-projects/agilityperformance-robotic-systems

Automation Competition (ARIAC)[2] to be held in 2017 in conjunction with the Institute of Electrical and Electronics Engineers (IEEE).

One of the key aspects needed to enable robot agility is the ability for the robot to represent knowledge about the environment (and its own capabilities) in such a way that it can reason over it and take action based on what it has learned. NIST has chaired an IEEE Working Group which developed the IEEE 1872 Standard (Core Ontology for Robotics and Automation (CORA)) [2]. This standard defines a core ontology that allows for the representation of, reasoning about, and communication of knowledge in the robotics and automation (R&A) domain. This ontology includes generic concepts as well as their definitions, attributes, constraints, and relationships. These terms can be specialized to capture the detailed semantics for concepts in robotics sub-domains. The standard has chosen to use the Standard Upper Ontology Knowledge Interchange Format (SUO-KIF) [3] to represent concepts and their associated axioms.

In this paper, the authors describe how the concepts of three ontologies in Web Ontology Language (OWL) [4] format, consistent with CORA, were specialized to enable agility in industrial robotics and the infrastructure that was built to convert the concepts into various representations that could directly be applied to the robot control system. The effort described in this paper deals with kitting or kit building. In kitting, parts are delivered to the assembly station in kits that contain the exact parts necessary for the completion of one assembly object.

This paper is structured as follows: Section II describes the information models used within this project. The authors also briefly describe the tool used to translate XML (eXtensible Markup Language) Schema Definition Language (XSDL) and XML files to OWL files in Section III. Section IV details the design methodology that relies on OWL files to perform kitting in the APRS project. Section V gives conclusions and future work.

II. Information Models

The APRS project makes use of three ontologies that can be applied to the kitting domain. The authors used a set of closely related C++ software tools for manipulating XSDL [5], [6], [7] files and XML instance files and translating them into OWL class files and OWL instance files [8]. The authors use the translators to translate XML instance files to OWL instance files. The OWL class files are used as input to a tool that generates a database schema automatically. The APRS work in OWL generation was reported in [8]. Modest improvements have been implemented since then.

[2]https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation

A. KittingWorkstation Model

The KittingWorkstation model describes the objects in the current kitting scenario. This file contains all of the basic information that was determined to be needed during the evaluation of the use cases and scenarios. The knowledge is represented in as compact of a form as possible with knowledge classes inheriting common attributes from parent classes. In Figure 1 and similar figures (which were generated by XMLSpy[3]), a dotted line around a box means the attribute is optional (may occur zero times), while a $..\infty$ underneath a box means it may occur more than once, with no upper limit on the number of occurrences. The main types (i.e., classes or datatypes) of the attributes of a kitting workstation are shown in Figure 1. More information on this model can be found in [9], [10].

B. Action Model

Planning for kitting relies on the Planning Domain Definition Language (PDDL) [11] domain file. PDDL is a community standard for the representation and exchange of planning domain models. In order to operate, the PDDL planners require a PDDL file-set that consists of two files that specify the domain and the problem. From these files, the planning system creates an additional static plan file. The authors explored the idea of automating the generation of PDDL domain and problem files by representing the components of a PDDL domain file [12] in an Action model. The Action model imports the KittingWorkstation model to include all the parameters defined in PDDL predicates and functions. Figure 2 depicts the information model for a PDDL domain file. A user only needs to create an XML file which consists of actions that a robot needs to perform to build a kit.

C. Robot Capability Model

The application of robotics in manufacturing assembly is hindered by their lack of agility, their large changeover times for new tasks and new products, and their limited reusability. One of the main causes for these hindrances is the lack of understanding of robot capabilities as they pertain to assembly tasks. In this context, a robot capability refers to the ability of a robot to perform a specific action on a specific object or set of objects. Measuring robot capabilities for a specific domain provide multiple advantages such as creating a process plan that assigns the best robot to each operation needed to accomplish the given task.

The Robot Capability model depicted in Figure 3 was produced based on the set of components identified from a thorough literature review [13]. The capability model

[3]Certain commercial/open source software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors or NIST, nor does it imply that the software tools identified are necessarily the best available for the purpose.

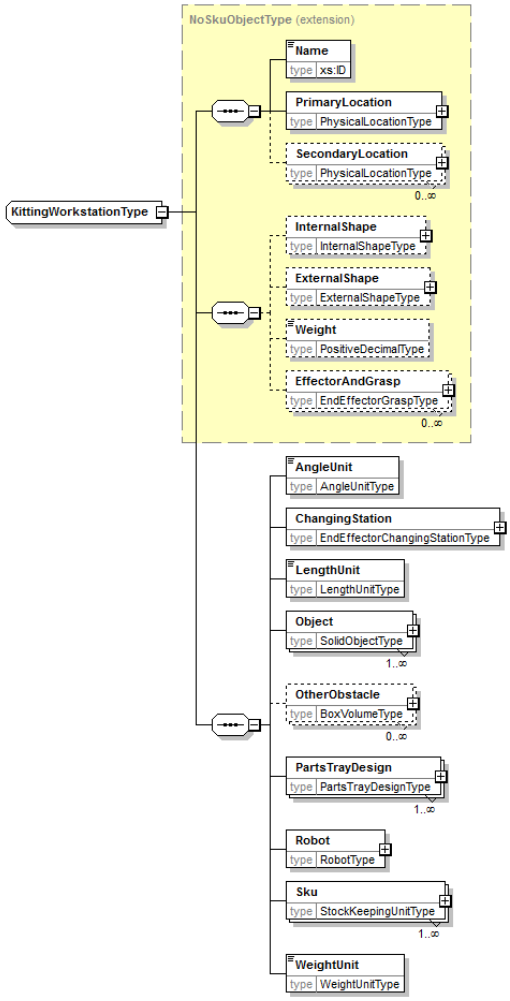

Figure 1. Information model for kitting.

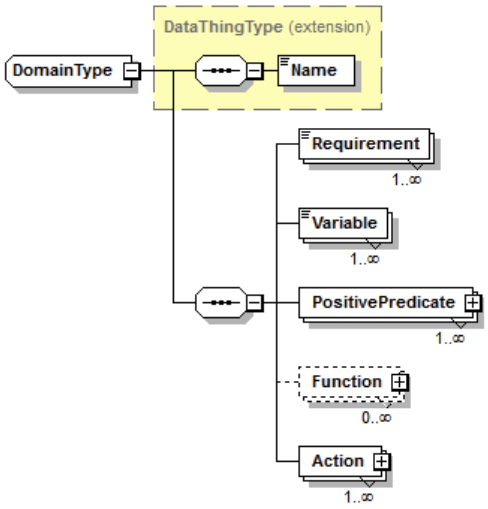

Figure 2. Information model for PDDL.

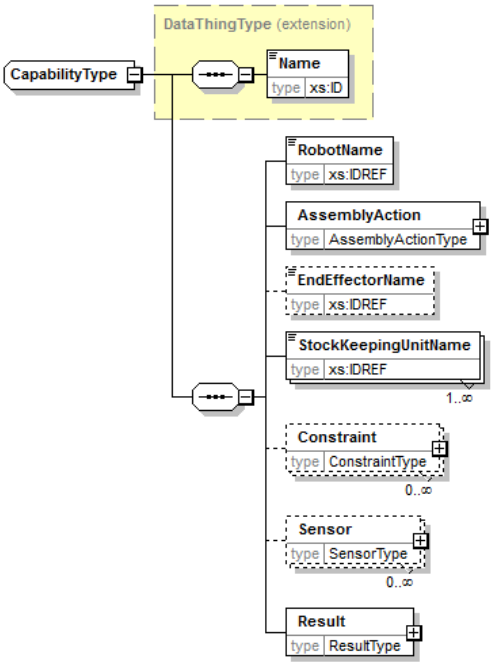

Figure 3. Information model for robot capability.

consists of pointers to elements defined in the Kitting-Workstation model as well as new elements. Mandatory elements include a pointer to the robot element for which the capability is defined (e.g., *robot-motoman*), the definition for the assembly action performed by the robot (e.g., *pick-up-part*), one or multiple references to the stock keeping unit for the object involved in the assembly action (e.g., *small-gear*), and the overall result for the current capability (e.g., can pick up this part 85 % of the time).

III. Translation Details

To enable using OWL the authors have developed a set of automatic software tools for generating OWL class files from XML schemas and for generating XML to OWL instance file translators automatically. The authors use the translators to translate XML instance files to OWL instance files. A number of systems for converting XML files to OWL files have been developed. A survey of nine of these systems was made by Hacherouf *et al.* [14]. That survey does not include the system developed by the authors, in this paper called the ISD OWL Generation System [8]. The ISD system is built in C++ , using YACC and Lex [15] for parsing. C++ is generated from YACC by bison [16] [12] and generated from Lex by flex [17].

A. Obviating OWL Problems

A number of features of OWL [18], [19] and Protégé [20], a tool available for building OWL ontologies, make it impractical to build OWL models and instance files directly.

The primary reason for this is that user errors in spelling the names of NamedIndividuals, properties, and classes are not recognized as errors.

OWL's open world assumption allows that anything might be true that is not explicitly ruled out (1) by OWL statements directly, or (2) by reasoning from statements that have been made. The open world assumption is appropriate in some contexts, however the kitting domain may be readily handled under a closed world assumption. Using an open world assumption introduces difficulty without providing any advantages.

Also, if the name of a class, property, or individual is used without being explicitly declared as such in the file (as happens when a name is misspelled), that class, property, or individual is implicitly declared. Protégé does provide some help with spelling by having an auto complete window to use when expressions are being constructed. A misspelled term will appear as one of the choices while the user types, if the first few letters are correct.

Another problem is that, while constructing an OWL file, it is easy to omit OWL statements one intends to make. Omitting any one statement or any set of statements after the header in OWL class and instance files will not be an OWL error and will not cause Protégé to flag any error or give any warning. The same would be true of many other OWL files.

Finally, Protégé does not check completely whether an OWL file conforms to the OWL specification. For example, if an OWL instance file imports an OWL class file and the prefix declared for both files is the empty prefix, no error will be signaled, even though the OWL specification says explicitly that this is not allowed.

Some of these issues can be detected, and research aimed at developing better OWL consistency checkers is ongoing [21]. One utility, Pellet, offers some support for advanced reasoning and debugging [22]. In our tests, the Pellet command line linter was able to detect spelling errors within OWL, but Pellet was unable to detect a missing statement. Further, Pellet seems to support OWL XML syntax, but was unable to parse functional style syntax. Limitations still remain.

The use of an undefined type in an XML schema file is an error, and readily available XML tools will detect and flag it. Similarly, a missing element in an instance file will be detected and flagged. If a definition of references to unique identifiers (IDREF) is made to a definition of unique identifiers (ID) that has not been used, that will be detected and flagged. If a portion of an XML schema file is omitted, in many cases, that will be detected when the file is read, and in most cases an error will be signaled if an instance file is read that conforms to the complete intended schema file. Thus, almost all spelling errors that will pass in OWL will fail in XML, and most errors of omission that will pass in OWL will fail in XML.

The translation tools do not make spelling errors or errors of omission. Hence, by using them on tested XML schemas and instance files, correct OWL files may be produced. In addition, it is easier to work with XML files since (1) they are structured while OWL files are not, and (2) XML files are about half as long as the equivalent OWL files.

IV. Design Methodology

The authors have implemented the design methodology shown in Figure 4 to perform kitting with an industrial robot.

A manufacturing activity begins when the operator receives a request to fulfill an order. The request is submitted to a Task Scheduler that starts the kitting process. Information from the three ontologies described earlier is converted into a graph database with the Database Generator, a Java tool. The use of a graph database ensures real-time access to information on the environment, planning domain, and robot capabilities. The Database Generator "sees" an ontology as a graph. It has a top node (owl:Thing) and classes extending it. There are individuals that belong to classes and object properties connecting the individuals. Individuals can have data properties and annotations that can be represented as node properties and relationship properties or as relationship types.

Using the Action and Robot Capability ontologies, the Problem Generator [12] dynamically creates a PDDL problem file for the current domain. The Problem Generator parses the PDDL domain file to retrieve all the predicates and numeric-valued fluents (including robot capability values) and then uses a mapping file to retrieve their values in the graph database.

The current state of the world is updated by the Sensor Processing since objects in the environment could have been inadvertently moved since the previous time the database was updated. Once the problem file is autogenerated, both the problem and the domain files are used with a temporal planner to generate a plan file.

Next, an Executor application translates each PDDL action from the plan file into a series of Canonical Robot Command Language (CRCL) commands, and sends each to each Robot Controller. CRCL provides message content and syntax for a low-level robot command-status protocol. The Executor application continually tracks the execution status of each CRCL command, ensuring that the conditions and effects of the PDDL commands are met as the statements are processed. If failures are detected, e.g., dropped part, the Executor aborts the PDDL plan and calls for replanning. The current state of the world is maintained in a world model (WM) graph database which is updated by a sensor processing system, which continually estimates the state of the environment, synthesizes higher-level information about the state of the world by fusing lower-level sensor information, and updates the WM, which could be implemented in a distributed fashion.

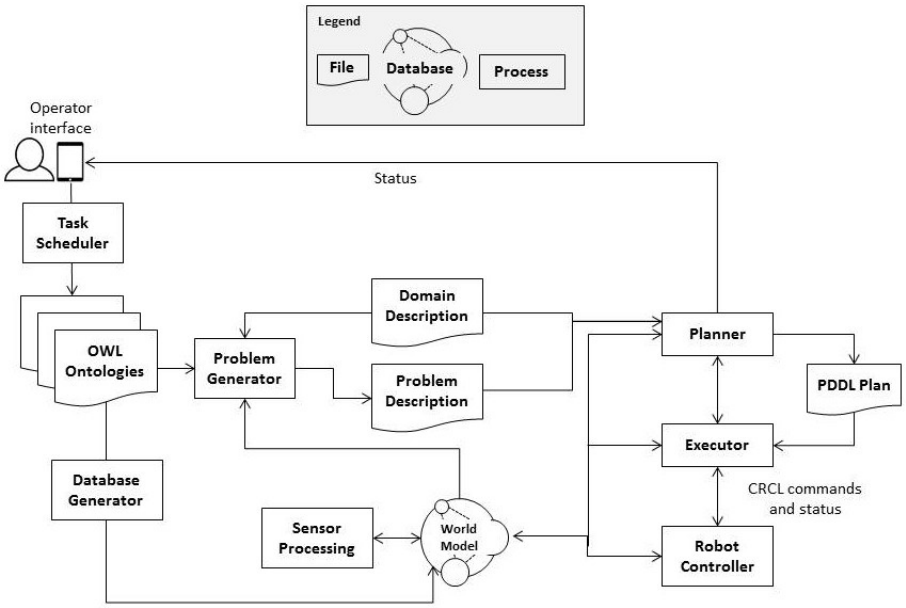

Figure 4. Design methodology for kitting.

The design methodology described above has been used at NIST to perform kitting with two different robots (Figures 5 and 6). Kitting was performed with both robots to pick and place components of a gear box (Figure 7). The KittingWorkstation model consisted of elements describing the workstation (robot, gears, etc) while the Action model described the different pick-up and place actions that the robots can use to reach a goal state. The design methodology was not altered in the two scenarios, only the information in the OWL files was modified to match the goal state (e.g., building a kit with two parts vs. a kit with four parts).

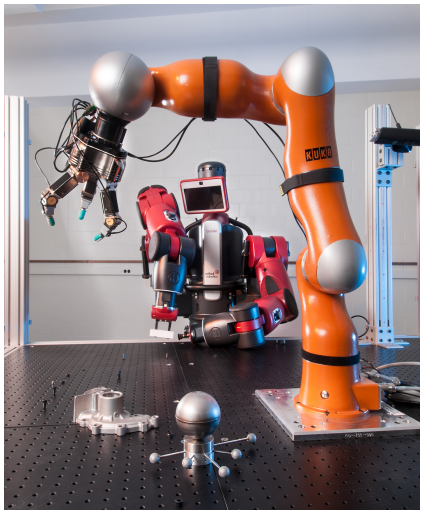

Figure 5. Kuka Lightweight Robot.

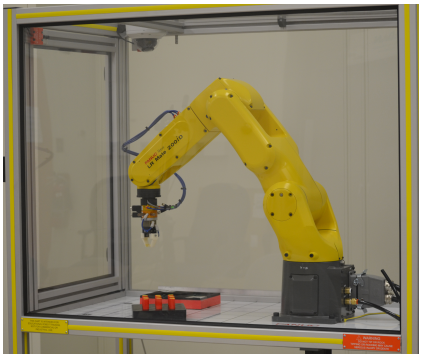

Figure 6. Fanuc LR Mate 200iD.

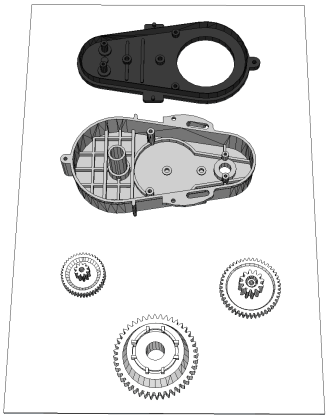

Figure 7. Parts for a gearbox kit.

V. Conclusions and Future Work

The Agility Performance of Robotic Systems (APRS) project aims at making industrial robots more agile to

tackle the challenges that small and medium manufacturers are facing. Implementing agility within the APRS project is performed with the kitting domain and starts with the definition of three information models which derive from the CORA model. This paper describes a KittingWorkstation model, an Action model, and a Robot Capability model. The KittingWorkstation model describes a kitting workstation including solid objects (e.g., parts) and data (e.g., poses). The Action model describes the structure components of a Planning Domain Definition Language (PDDL) domain file and is used to automate the generation of PDDL domain and problem files. The Robot Capability model expresses the capability of a robot at performing specific planning actions on a specific object or set of objects. Although significant efforts have been made to improve agility in manufacturing assembly, there is still a need to deal with action failures. It is the intention of the authors to develop a new model for failures. This model will encompass information on the different failures that a robot can encounter during kitting as well as the severity of the failures and a remediation plan for each type of failure.

ACKNOWLEDGMENT

Dr. Kramer acknowledges support for this work under grant 70NANB15H053 from the National Institute of Standards and Technology to the Catholic University of America.

Dr. Kootbally acknowledges support for this work under grant 70NANB15H249 from the National Institute of Standards and Technology to the University of Southern California.

REFERENCES

[1] F. Proctor, S. Balakirsky, Z. Kootbally, T. Kramer, C. Schlenoff, and W. Shackleford, "The Canonical Robot Command Language (CRCL)," *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 495–502, 2016.

[2] S. R. Fiorini, J. L. Carbonera, P. Gonçalves, V. A. Jorge, V. F. Rey, T. Haidegger, M. Abel, S. A. Redfield, S. Balakirsky, V. Ragavan, H. Li, C. Schlenoff, and E. Prestes, "Extensions to the Core Ontology for Robotics and Automation," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 3–11, Jun 2015, special Issue on Knowledge Driven Robotics and Manufacturing.

[3] A. Pease, "Standard Upper Ontology Knowledge Interchange Format," 2009. [Online]. Available: http://suo.ieee.org/suokif.html

[4] World Wide Web Consortium, "OWL 2 Web Ontology Language Structural Specification and Functional Syntax," 2009.

[5] W3C, "XML Schema Part 0: Primer Second Edition," in *http://www.w3.org/TR/xmlschema-0/*, 2004.

[6] ——, "XML Schema Part 1: Structures Second Edition," in *http://www.w3.org/TR/xmlschema-1/*, 2004.

[7] World Wide Web Consortium, "XML Schema Part 2: Datatypes, Second Edition," 2004. [Online]. Available: www.w3.org/TR/xmlschema-2

[8] T. R. Kramer, B. H. Marks, C. I. Schlenoff, S. B. Balakirsky, Z. Kootbally, and A. Pietromartire, "Software Tools for XML to OWL Translation," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST IR 8068, Jul 2015.

[9] S. Balakirsky, Z. Kootbally, C. Schlenoff, T. Kramer, and S. Gupta, "An Industrial Robotic Knowledge Representation for Kit Building Applications," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vilamoura, Portugal: IEEE, October 2012, pp. 1365–1370.

[10] S. Balakirsky, Z. Kootbally, T. Kramer, R. Madhavan, C. Schlenoff, and M. Shneier, "Functional Requirements of a Model for Kitting Plans," in *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*. ACM, 2012, pp. 29–36.

[11] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 431–433, 2003.

[12] Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, and S. Gupta, "Towards Robust Assembly with Knowledge Representation for the Planning Domain Definition Language (PDDL)," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 42–55, 2015, special Issue on Knowledge Driven Robotics and Manufacturing.

[13] Z. Kootbally, "Industrial Robot Capability Models for Agile Manufacturing," *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 481–494, 2016.

[14] M. Hacherouf, S. Bahloul, and C. Cruz, "Transforming XML Documents to OWL ontologies: A survey," *Journal of Information Science*, vol. 41, no. 2, pp. 242–259, 2015.

[15] D. Brown, J. Levine, and T. Mason, *Lex & Yacc*. OReilly Media, October 1992.

[16] C. Donnelly and R. Stallman, "Bison, The YACC-compatible Parser Generator," 2006. [Online]. Available: http://dinosaur.compilertools.net/bison

[17] V. Paxson, W. Estes, and J. Millaway, "Flex, Version 2.5.31 A Fast Scanner Generator," 2003. [Online]. Available: http://www.gnu.org/software/flex

[18] World Wide Web Consortium, "OWL 2 Web Ontology Language Primer (Second Edition) W3C Recommendation 11 December 2012," 2004. [Online]. Available: http://www.w3.org/TR/owl2-primer/

[19] ——, "OWL 2 Web Ontology Language Structural Specification and Functional Style Syntax (Second Edition) W3C Recommendation 11 December 2012," 2012. [Online]. Available: http://www.w3.org/TR/owl2-syntax

[20] M. Horridge, "A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, 1st ed," The University Of Manchester, Tech. Rep., March 2011.

[21] B. Motik, I. Horrocks, and U. Sattler, "Adding Integrity Constraints to OWL," *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, vol. 258, June 2007.

[22] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A Practical OWL-DL Reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.