TOWARD ONTOLOGIES AND SERVICES FOR ASSISTING INDUSTRIAL ROBOT SETUP AND INSTRUCTION

Mathias Haage, Anders Nilsson and Pierre Nugues
Department of Computer Science, Lund University, Box 118, 22100 Lund, Sweden
{*mathias.haage, anders.nilsson, pierre.nugues*}*@cs.lth.se*

Keywords: Industrial robot, ontology, web services, multimodal dialogue.

Abstract: Achieving rapid, intuitive, and error-free robot setup and instruction is a challenge. We present our work towards an assistive infrastructure for robot setup and instruction that attempts to address it. In this paper, we describe the ongoing development of a system that automatically generates multimodal dialogue interaction from product and process ontologies. The prototype currently generates two modalities, digital paper and spoken dialogue.

1 INTRODUCTION

Traditional robotics supports long-batch production and requires skilled personnel to handle setup and instruction. On the contrary, new robot markets often involve shorter series and small and medium enterprises. This means that the shift of products is faster and the change-overs often need to be carried out by non-experts. This sets new challenges for the robot user interfaces to be more intuitive and user friendly in order to reduce number of errors and cost/time (Haegele, 2007). Such challenges outline the need for assistive systems within the robot cell to make the operator less dependent on expert knowledge and turn complex tasks feasible. Examples that could benefit from assistance include calibration of tools, fixtures, and workpieces; usage of CAD/CAPP/CAM software such as task planners; configuration of process-specific software packages, such as the ABB palletizing PowerPac.

In this paper, we present an assistive infrastructure for robot setup and instruction that attempts to address these challenges. We introduce the ongoing development of a system based on semantic web technologies that automatically generates multimodal dialogue interaction. We also describe a prototype that currently generates two modalities, digital paper and spoken dialogue.

The purpose of an assistive system is to enhance the usability and usefulness of the robot and its connected resources (sensors, CAD/CAPP/CAM systems) through:

- The use of semantic standards in information exchange, such as RDF/S, OWL, SPARQL, and SWRL;
- Production documents such as product and process data including a semantic layer;
- The definition of compatible semantic layers so that they can be used across the relevant tasks, such as aiding cell calibration and robot instruction.
- Increased automation of tasks using the semantic layer, such as finding calibration sequences to make sure nothing is forgotten.

The roadmap we follow to implement the assistive infrastructure is based on the use of an ontological network to encapsulate knowledge about the product data and manufacturing processes. It requires the derivation of ontology concepts that will serve as the main data source to generate — or refer to — the complete specifications and the operating instructions used to automate information management necessary for task planning and execution.

2 MULTIMODAL FORM-BASED DIALOGUE

From the user viewpoint, the operation starts with an initial selection command from the operator to tell the machine which work piece to produce and possibly from positions and equipment data sensed by devices on the floor.

After the initial selection, the system extracts data from the ontology that enables the operator to configure the task and the product, and to prepare the task execution. The configuration step uses a multimodal interface that lets the operator fill in the different options. It ends with the monitoring and execution of the configured task.

The process flow uses conversion tools such as transformation rules, inference rules, and the JastAdd compiler (JastAdd, 2007) to select and convert portions of the ontology. This results in process and product data divided into configurable and nonconfigurable parts (Figure 1). The extracted data are first formatted as an XML document corresponding to a production sketch that we call the XML appconf.

From this sketch, the system automatically generates user interfaces with multiple input modalities for all the parameters. A first demonstration prototype will be available in Spring 2008.

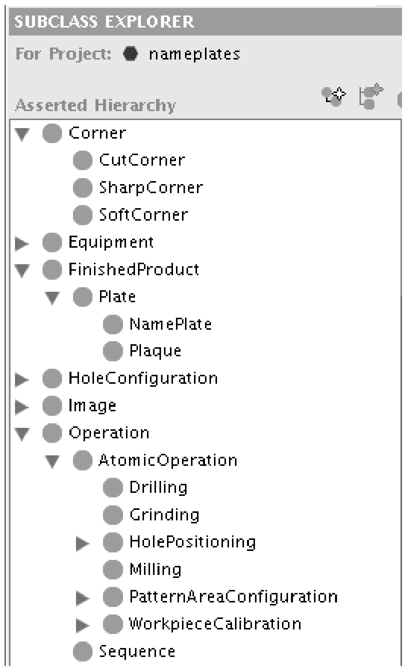
3 ONTOLOGY MODELING

In computer science, ontologies correspond to hierarchical models to represent concepts, objects, and their relationships. They enable systems to (Buitelaar, 2007):

- Encode and interpret data using a rich hierarchical and relational structure.
- Extract data and integrate them into applications.
- Share data with a common format.

As ontology modeling language, we have chosen the web ontology language (McGuinness and van Harmelen, 2007) and the Protégé toolkit (Protégé, 2007) as a data entry and validation tool. Both are well-established standards in their domain with a large developer's community. We are currently using them to build the ontology of a specific domain shown in Figure 2 that serves in the demonstration prototype. This ontology acts as an advanced data repository for the product configuration and the production operations. In the future, we will populate ontologies from manual modeling, specification databases, and 3D models. In addition to what we develop within SMErobot, the data model we will use could also possibly benefit from work being carried out at LTH for the SIARAS project (SIARAS, 2007) on production ontologies.

The conversion pipeline shown in Figure 1 uses W3C recommendations associated with the semantic web such as XSLT or SWRL. The choice of these tools needs some clarification. We first summarize



```
▼ Corner
    CutCorner
    SharpCorner
    SoftCorner
▶ Equipment
▼ FinishedProduct
  ▼ Plate
      NamePlate
      Plaque
▶ HoleConfiguration
▶ Image
▼ Operation
  ▼ AtomicOperation
      Drilling
      Grinding
    ▶ HolePositioning
      Milling
    ▶ PatternAreaConfiguration
    ▶ WorkpieceCalibration
    Sequence
```

Figure 2: An excerpt of the ontology detailing the operation hierarchy.

the concepts that are around OWL and then explain the conversion principles.

3.1 Resource Description Framework - RDF

OWL is based on the resource description framework, RDF (RDF, 2007). RDF models statements as triples in the form of a subject, a predicate (a verb), and an object. As an example, the statement *the milling process starts with a calibration* can be split into a subject, *the milling process*, a predicate, *starts with*, and an object, *a calibration*. Such triples are also named, respectively, the resource, the property, and the value. RDF is restricted to binary predicates.

This framework can use two encodings. The first one, called Notation 3, consists of sequences of textual triples and the second one adopts an XML syntax. The subject – the resource – must be a URI. The predicate or property, which is also a resource, is a URI too. The object or value can be a resource or a literal. To represent the example above, we use the lrc namespace – Lund Robotics Core – and URI http://cs.lth.se/lrc/ontologies/1.0/. This URI is still nonexistent when this article is being written. Using Notation 3, we can represent the example above as:

```
@prefix lrc: <http://cs.lth.se/lrc/ontologies/1.0/>.
<#milling_process> lrc:starts_with "calibration";
```

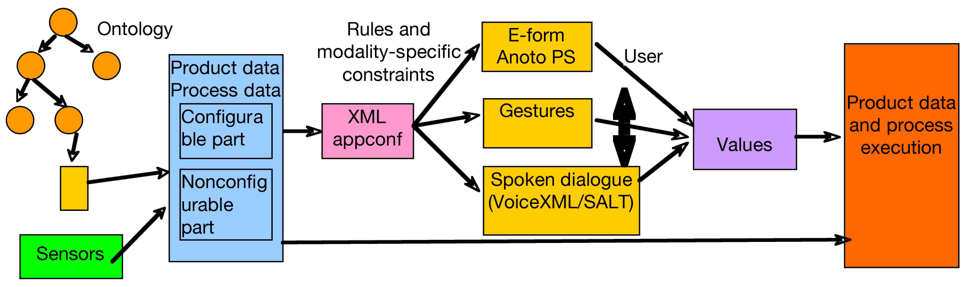


Figure 1: The process workflow.

And in XML syntax as:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:lrc="http://cs.lth.se/lrc/ontologies/1.0/">
  <rdf:Description rdf:about="#milling_process">
    <lrc:starts_with>calibration</lrc:starts_with>
  </rdf:Description>
</rdf:RDF>
```

More generally, triples (subject, predicate, object) can be encoded in Prolog or Datalog as *predicate(subject, object)* that is, with the sentence above as:

```
starts_with(milling_process, calibration)
```

3.2 RDF Schema – RDFS

RDF schema, RDFS, is built on RDF and defines two predicates that enable the programmer to build an ontology: *rdfs:Class* and *rdfs:subClassOf* (RDFS, 2007). The *rdfs:Class* element allows to declare a RDF resource as a class and the *rdfs:subClassOf* element allows to declare subclasses of a class and build a hierarchy. When a resource has been declared as a class, we can use *rdf:type* to create individuals member of this class. In addition, RDFS comprises similar predicates to build a property hierarchy.

In addition, RDFS has constructs to type the subject and the object of the triples. It corresponds to the domain and range of a function, and in the case of RDFS applies to properties. RFDS uses the constructs *rdfs:domain* for the subjects and *rdfs:range* for the objects to restrict the values of the two arguments of a property.

3.3 OWL

Although the combination of RDF and RDFS forms an ontology language, it lacks some features to build large, realistic ontologies. They include cardinality restrictions, Boolean operations on classes, etc. In addition, RDF and RDFS are not well coupled to logic and reasoning tools.

The web ontology language, OWL, is an extension of RDF and RDFS that attempts to complement them with better logic foundations and a support for practical reasoning (McGuinness and van Harmelen, 2007). It comes with three flavors of increasing expressivity – light, description logic, and full – that are upward compatible. Only the two first ones are guaranteed to be tractable in practice.

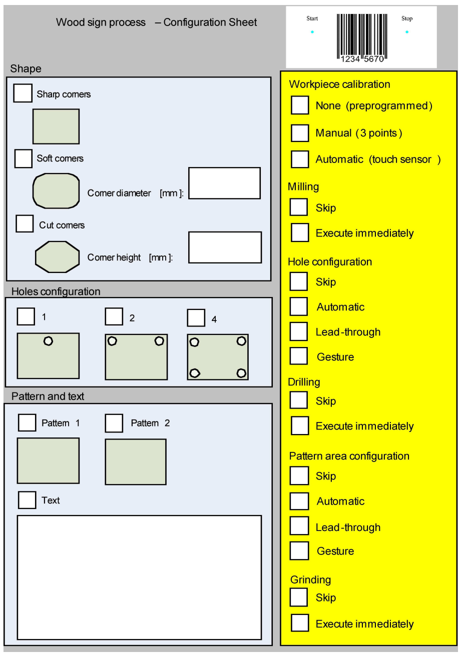
Important constructs of OWL include the *owl:Class*, which is derived from the *rdfs:Class*, two properties, *owl:ObjectProperty* and *owl:DatatypeProperty*, that relate objects to respectively another object or a data type value like a string, an integer, a float, etc., *owl:Restriction* that enables the programmer to use existential and universal quantifiers and cardinality.

4 PROTOTYPE SETUP

4.1 Nameplate Manufacturing

As described in (SMErobot, 2007b), the ontology programming approach uses automatically generated forms to select and configure both the task and the product. The prototype selected to demonstrate the concepts associated with our approach corresponds to the manufacturing of wood nameplates.

Figure 3 shows a configuration form where the left column configures the shape and looks of the plate. The right column configures the process for manufacturing the plate. In this example it is possible to skip steps, execute in a stepwise manner, and choose data acquisition methods for steps involved. The left column can be filled out at an earlier date while the right column is filled out close to task execution time. The upper right barcode identifies the process and is possibly unique to individualize the sheet.



Wood sign process – Configuration Sheet

Start 1234 5670 Stop

Shape

☐ Sharp corners

☐ Soft corners

Corner diameter [mm]:

☐ Cut corners

Corner height [mm]:

Holes configuration

☐ 1 ☐ 2 ☐ 4

Pattern and text

☐ Pattern 1 ☐ Pattern 2

☐ Text

Workpiece calibration

☐ None (preprogrammed)

☐ Manual (3 points)

☐ Automatic (touch sensor)

Milling

☐ Skip

☐ Execute immediately

Hole configuration

☐ Skip

☐ Automatic

☐ Lead-through

☐ Gesture

Drilling

☐ Skip

☐ Execute immediately

Pattern area configuration

☐ Skip

☐ Automatic

☐ Lead-through

☐ Gesture

Grinding

☐ Skip

☐ Execute immediately

Figure 3: Prototype form for manufacturing wood nameplates.

4.2 Nameplate Manufacturing Ontology

We have built a prototype ontology to encode the process templates and we are developing well-defined conceptual interfaces toward work cells (equipment, capabilities, communication) and process data, assisting construction of process templates, and assisting (manual/automatic) work cell reconfiguration.

The ontology is restricted to the prototype domain and Figure 2 shows an excerpt of it. It describes the finished products its components, together with possible features as well as the operations involved in the manufacturing process of the product.

4.3 Intermediate Appconf Representation

Given a product to manufacture, the conversion process extracts an intermediate, flat representation from the ontology. This representation is designed to be modality-independent, which makes it easier to build user views (forms, speech, gestures). We call it the application configuration – appconf.

As an example, in the application prototype we are building, the sheet requires the operator to supply data, such as the corner shape, the hole configuration, and the pattern and text (person name for instance). All these items are shown as input areas on the sheet in Figure 3. The intermediate appconf representation has corresponding elements representing all these configurable items, for instance the corner shape.

We give an idea of how to represent the corner shape options in the XML code snippet below. This code replicates the possible options, sharp, soft, or cut corners, with the images to display in the e-form using the *img* element and the messages to utter using the *snd* element in the case of a spoken dialogue.

```
<shape>
    <one-of>
        <option>
            <name>sharp corners</name>
            <command code="sharp.cd"/>
            <img src="sharp.jpg"/>
            <snd src="sharp.wav"/>
        </option>
        <option>
            <name>soft corners</name>
            <command code="soft.cd"/>
            <img src="soft"/>
            <param name="diameter" unit="mm"/>
        </option>
        <option>
            <name>cut corners</name>
            <command code=".cd"/>
            <img src="cut.jpg"/>
            <param name="height" unit="mm"/>
        </option>
    </one-of>
</shape>
```

Using this configuration sketch, presentation rules, and modality specific constraints, the conversion process produces displayable forms or spoken dialogues so that the operator can supply the missing parameters. Once the operator has filled in the data, the corresponding XML fragment is:

```
<shape>
    <name>sharp corners</name>
    <command code="sharp.cd"/>
    <img src="sharp.jpg"/>
    <snd src="sharp.wav"/>
</shape>
```

4.4 Methods and Languages to Extract Information from Ontologies

The conversion pipeline extracts and infers information from the ontology and generates the user input modalities. The appconf configuration sketch is an XML intermediate document between the ontology and the user interfaces. Unlike the sketch, the ontology is a structured and hierarchical representation, where features are shared and inherited across a variety of pieces and processes. This means that extraction is not trivial because the representation languages involve three complex and intricate layers: RDF, RDFS, and OWL.

Such a setting requires specific query languages and techniques. In addition to the ontology management, we need to process other XML documents in the processing chain such as the appconf sheet to convert them into forms or dialogue programs. We review here techniques and their application in the management of information along the conversion chain. They include accessing XML nodes, querying RDF triples, and reasoning about the ontology knowledge. Most difficulties come from the apparent masses of "solutions". Wikipedia lists not less than 11 different RDF query languages and ten OWL reasoners! We focus here on what have become the (likely) standards in their respective ecosystems.

4.4.1 XSLT

The simplest way to access and transform XML documents is to use the combination of XPath and the extensible stylesheet language transformations, XSLT (XSLT, 2007). XPath enables programmers to express a path and access nodes in an XML tree, while XSLT defines conversion rules to apply to the nodes. A typical application of XSLT is the transformation of XML documents into XHTML files destined to be read by web browsers.

Provided that the amount of paraphrasing (syntactic variation) is limited, XSLT XPath is fairly usable to run the conversions. From studies we have done, this is the case for the conversion of the appconf sketch to the user modalities. We are completing the implementation and integrating it in the prototype.

However, this is not the case for ontologies. They are built on OWL, which is built with RDF triples, which allows reformulating similar structures using different constructs. Querying ontologies require either query languages or reasoning rules. For a justification, see (Antoniou and van Harlmelen, 2004), pp. 100-102.

4.4.2 SPARQL

SPARQL (SPARQL, 2007) is a RDF query language. It enables the programmer to extract RDF triples using the SELECT keyword where the variables are denoted with a question mark prefix using a set of conditions defined by the WHERE keyword. It is also possible to build a new graph using the CONSTRUCT keyword. SPARQL's syntax is similar to that of the SQL language. The query below extracts all the pairs where *?subject* is a subclass of *?object*:

```
SELECT ?subject ?object
WHERE {
    ?subject rdfs:subClassOf ?object. }
```

However, as SPARQL makes the join operation implicit, it bears some resemblance with Prolog as in this query:

```
SELECT ?subject ?config
WHERE {
    ?subject rdfs:subClassOf <#FinishedProduct> .
    ?prop rdf:type owl:ObjectProperty .
    ?prop rdfs:range ?config . }
```

SPARQL is becoming a de facto standard for RDF. It is a stable language with a quality implementations from various sources. Competitors include XQuery, a generic XML query language, which has not gained acceptance in the RDF community.

4.4.3 SWRL

While SPARQL enables a programmer to extract information from an ontology, it is only designed for RDF. In addition, it cannot easily derive logical consequences from its results. To exploit fully the ontology knowledge, one needs a reasoning or inferencing mechanism. This is the purpose of a language like the semantic web rule language, SWRL (SWRL, 2007). SWRL rules have a Prolog-like structure. They consist of an antecedent corresponding to a conjunction of conditions (predicates) and a consequent. When the conditions are true, the consequent is also true and can be asserted. In addition to being a inference language, SWRL features an extension that lets it act like a query language, SQWRL.

SWRL is supported from the 3.4 version of Protégé in the form of a development environment with editing tools. This means that we can write, modify, and to a certain extent validate rules. However, version 3.4 is still in the beta stage at the time we are witing this paper. In addition, Protégé does not include a full-fledged inference engine. This means that it cannot by itself execute the rules. It just supplies a bridge that connects to an external module. So far, Protégé supports only one inference engine, Jess (Friedman-Hill, 2007).

4.4.4 JastAdd

JastAdd is not a query language in itself, but a general compiler construction tool with some very useful features; aspect oriented programming and attribute grammars. Using results from earlier work (Malec et al., 2007) we can automatically create a parser for an OWL ontology. Utilizing the aspect-oriented feature of JastAdd, we can then implement queries in the form of aspect modules that will be weaved in with the generated parser at compile time.

While it does not possess the expressive power of SWRL, it will enable the user to extract almost any information from the ontology with just a few 10s of lines of Java code.

4.4.5 Prolog

Prolog – or Datalog – is a last example of reasoning tool that could be used to extract information from the ontology. Some Prolog implementations have a RDF interface like SWI Prolog that has been used with success in semantic web applications (Wielemaker et al., 2007). It is then possible to query an ontology from a logic program and to run inference rules on the result.

As Prolog predicates and rules have much in common with SWRL, logic programs written in Prolog and SWRL would be very similar and with equivalent performances. Difference would come from the location of the bridge between the ontology and the inference engine, at the RDF level for Prolog, at the OWL level for SWRL.

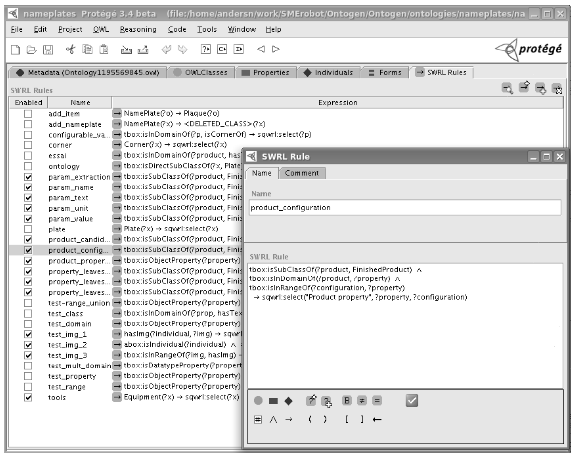
However, although Prolog is more expressive than other languages and has a proven record of industrial applications, Protégé does not support it. It is not standardized within the context of the semantic web either. This makes its choice, at the moment, riskier than the other options.

4.5 From an Ontology to the Appconf Sketch

The first step of the conversion pipeline generates the XML Appconf sketch from the ontology. As the SWRL formalism is more flexible and powerful, as well as adopted by the semantic web community, we are using it for this step in the demonstration prototype. As inference engine, we are using the built-in bridge that is for now only coupled to Jess.

However, SWRL is a new feature of Protégé 3.4 and athough it already supports many *abox* and *tbox* built-in predicates, it is still under active development. Many of the predicates are not yet implemented. The beta version status of SWRL pose timetable problems and we are also using SPARQL to query the ontology and write the rules.

We wrote rules using both formalisms to extract information from the ontology. We show below an example of SWRL rule that finds all the properties of the subclasses of FinishedProduct and Figure 4 shows a screenshot of the editing window in Protégé. We also show its counterpart in SPARQL. We embedded the rules in the Java prototype using the Protégé API that resembles SQL drivers.


nameplates Protégé 3.4 beta (file:/home/andersn/work/SMErobot/Ontogen/Ontogen/ontologies/nameplates/na
File Edit Project OWL Reasoning Code Tools Window Help
protégé
Metadata (Ontology1195569845.owl)
OWLClasses
Properties
Individuals
Forms
SWRL Rules
SWRL Rule
Name
Comment

SWRL Rules

Enabled	Name	Expression
☐	add_item	NamePlate(?o) → Plaque(?o)
☐	add_nameplate	NamePlate(?x) → <DELETED_CLASS>(?x)
☐	configurable_va...	tbox:isInDomainOf(?p, isCornerOf) → sqwrl:select(?p)
☐	corner	Corner(?x) → sqwrl:select(?x)
☐	essai	tbox:isInDomainOf(?product, has
☐	ontology	tbox:isDirectSubClassOf(?x, Plate
✓	param_extraction	tbox:isSubClassOf(?product, Finis
✓	param_name	tbox:isSubClassOf(?product, Finis
✓	param_text	tbox:isSubClassOf(?product, Finis
✓	param_unit	tbox:isSubClassOf(?product, Finis
✓	param_value	tbox:isSubClassOf(?product, Finis
☐	plate	Plate(?x) → sqwrl:select(?x)
✓	product_candid...	tbox:isSubClassOf(?product, Finis
✓	product_config...	tbox:isSubClassOf(?product, Finis
✓	product_proper...	tbox:isObjectProperty(?property)
✓	property_leaves...	tbox:isSubClassOf(?product, Finis
✓	property_leaves...	tbox:isSubClassOf(?product, Finis
✓	property_leaves...	tbox:isSubClassOf(?product, Finis
☐	test-range_union	tbox:isObjectProperty(?property)
☐	test_class	tbox:isInDomainOf(?prop, hasTex
☐	test_domain	tbox:isObjectProperty(?property)
✓	test_img_1	hasImg(?individual, ?img) → sqwrl
✓	test_img_2	abox:isIndividual(?individual) ∧
✓	test_img_3	tbox:isInRangeOf(?img, hasImg) →
☐	test_mult_domain	tbox:isDatatypeProperty(?propert
☐	test_property	tbox:isObjectProperty(?property)
☐	test_range	tbox:isObjectProperty(?property)
✓	tools	Equipment(?x) → sqwrl:select(?x)

Name: product_configuration

SWRL Rule

```
tbox:isSubClassOf(?product, FinishedProduct) ∧
tbox:isInDomainOf(?product, ?property) ∧
tbox:isInRangeOf(?configuration, ?property)
 → sqwrl:select("Product property", ?property, ?configuration)
```

Figure 4: A screenshot of the SWRL interface in Protégé and an example of a rule.

```
PREFIX list: <http://jena.hpl.hp.com/ARQ/list#>
SELECT ?product ?configuration
WHERE {
?product rdfs:subClassOf <#FinishedProduct> .
?property rdf:type owl:ObjectProperty .
{{?property rdfs:domain ?product} UNION
{?property rdfs:domain ?union .
?union owl:unionOf ?list .
?list list:member ?product}} .
?property rdfs:range ?configuration}
```

4.6 From the Appconf Sketch to Input Modalities

The second step of the conversion pipeline generates user input interfaces from the XML Appconf sketch. As final products frequently need to be customized according to the order, the manufacturing operator will be able to enter a part of the specifications at production time. In the demonstration prototype, we will investigate three configuration modalities that are core to the SMErobot project (SMErobot, 2007a), namely E-forms, gestures, and spoken dialogue.

We are developing tools to generate automatically the work piece production forms, the gesture tracking and interpretation module, or the dialogue specifications from the XML Appconf. Before the piece is manufactured, the operator fills in the remaining data corresponding to the final piece using the modality of her/his choice. To ease the interaction, we are investigating a framework to combine simultaneously the different modalities so that the operator can use speech and gestures at the same time for instance.

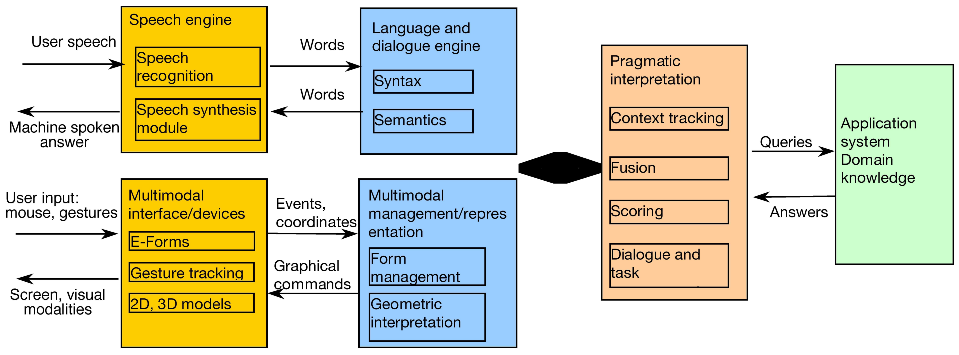


Figure 5: Multimodal dialogue architecture.

4.6.1 Transformation Language

As transformation language to produce the forms and the dialogue specifications, we are using XSLT. XSLT enables to apply transformations to Appconf nodes accessed via the XPath language. It can produce XML documents in formats like XHTML for the forms or VoiceXML for the dialogues. In addition, we are investigating the XSL-FO page-formatting standard where the description of a document content uses objects such as blocks, tables, footnotes, etc. It is richer than HTML-like descriptions and can be converted to PDF. The conversion of an XSL-FO document to a PDF uses a sequence of transformations that builds the XML tree, produces graphical objects, renders the objects as text areas with their pixel positioning, and finally generates PDF.

4.6.2 VoiceXML

The demonstration prototype includes a voice modality that enables the operator to configure the product through a dialogue and hence have his hands free while s/he fills in the manufacturing options. The dialogue uses a system-initiative scheme, which means that the dialogue structure resemble a form filling procedure where the user answers questions posed by the system.

We have chosen the Voice Extensible Markup Language, VoiceXML, to generate the dialogues. (VoiceXML, 2007) is markup language that enables a programmer to build form-based, goal-oriented dialogues (system-initiative mostly). The user fills fields in forms using speech, where the field input can be constrained with a grammar. It is designed to be integrated in a speech server and supports IP telephony. As VoiceXML is a standard, the programs should be portable to any platform that supports this language.

4.7 Perspectives: Multimodal Management

We will examine possible designs for the multimodal management architecture such as the one shown in Figure 5. It is mainly derived from a previous work of a member of the LTH team (Bersot et al., 1998; Godéreaux et al., 1998) and a recent system developed by the Bell labs (Ammicht et al., 2007; Potamianos et al., 2007). One input channel corresponds to the speech recognition module, which transcribes the user's speech into a word stream. The language engine then processes the character flow dealing with syntax, which is constrained by the VoiceXML structure, and semantics. The semantic module converts words into a semantic representation that is common to speech and other types of interaction. The other channel corresponds to form filling, which are processed by the multimodal manager. The pragmatic module merges data from both modalities and keeps track of the context and the application goals. The resulting answers are either converted into speech to the user by a speech synthesizer or presented visually by a visualizing module.

The multimodal architecture will use a client-server architecture and instantiate some of the modules shown in Figure 5. We are currently defining them. In the future, it could evolve into an integration platform enabling partners to plug their applications. As a use case we consider interactions where the user fills in the data using one modality, the corresponding client sends the data to the server, and the server updates the context of all the modalities. There are then continuous visual or audio updates of the current context. For instance, an audio message is synthesized each time the user has selected an option with the form, to confirm or remind the next actions. Modality switching could be carried out manually by the user or automatically.

5 CONCLUSIONS

We have described the design and implemention of an assistive infrastructure based on the use of an ontological network to encapsulate knowledge on the product data and manufacturing processes. We have implemented a prototype ontology that serves as the main data source to automatically generate digital forms and voice dialogues to configure a wood nameplate manufacturing process. As a perspective, we intend to synchronize modalities for a more flexible, efficient user input. A first prototype will be available for demonstration in Spring 2008.

ACKNOWLEDGEMENTS

The work presented in this paper is being financed mainly by the EU FP6 projects SMERobot and SIARAS. Partner companies within SMErobot with which we have close cooperation in this work include; ABB AB, Rinas ApS, and Visual Components Oy.

The respective developer teams behind the software tools we are using also deserve to be acknowledged: Protégé, Jess, and JastAdd.

REFERENCES

Ammicht, E., Fosler-Lussier, E., and Potamianos, A. (2007). Information seeking spoken dialogue systems part I: Semantics and pragmatics. *IEEE Transactions on Multimedia*, 9(3):532–549.

Antoniou, G. and van Harlmelen, F. (2004). *A semantic web primer*. The MIT Press.

Bersot, O., Guedj, P.-O. E., Godéreaux, C., and Nugues, P. (1998). A conversational agent to help navigation and collaboration in virtual worlds. *Virtual Reality*, 3(1):71–82.

Buitelaar, P. (2007). On the role of natural language processing in a data-driven approach to the ontology life-cycle. Keynote talk at TALN, Toulouse, France (http://olp.dfki.de/ontoselect/).

Friedman-Hill, E. (2007). Jess, the rule engine for the Java platform. http://herzberg.ca.sandia.gov/, site accessed December 2007.

Godéreaux, C., Guedj, P.-O. E., Revolta, F., and Nugues, P. (1998). *Virtual Worlds on the Internet*, chapter Ulysse: An interactive, spoken dialogue interface to navigate in virtual worlds. Lexical, syntactic, and semantic issues, pages 53–70, 308– 312. IEEE Computer Society Press, Los Alamitos, California.

Haegele, M. (2007). White paper on trends and challenges in industrial automation.

JastAdd (2007). The JastAdd Compiler-Compiler System. http://jastadd.cs.lth.se, site accessed December 2007.

Malec, J., Nilsson, A., Nilsson, K., and Nowaczyk, S. (2007). Knowledge-Based Reconfiguration of Automation Systems. In *Proceedings of IEEE CASE*, pages 170–175. IEEE.

McGuinness, D. L. and van Harmelen, F. (2007). OWL web ontology language. http://www.w3.org/TR/owl-features/, site accessed December 2007.

Potamianos, A., Fosler-Lussier, E., Ammicht, E., and Perakakis, M. (2007). Information seeking spoken dialogue systems part II: Multimodal dialogue. *IEEE Transactions on Multimedia*, 9(3):550–566.

Protégé (2007). The Protégé ontology editor and knowledge acquisition system. http://protege.stanford.edu, site accessed December 2007.

RDF (2007). Resource description framework. http://www.w3.org/RDF/, site accessed December 2007.

RDFS (2007). RDF schemas. http://www.w3.org/TR/rdf-schema/, site accessed December 2007.

SIARAS (2007). Skill-based inspection and assembly for reconfigurable automation systems.

SMErobot (2007a). The European robot initiative for strengthening the competitiveness of SMEs in manufacturing.

SMErobot (2007b). Preliminary description of concepts for high-level programming methods. Technical Report Deliverable DR2.6. Deliv.

SPARQL (2007). SPARQL protocol and RDF query language. http://www.w3.org/TR/rdf-sparql-query/, site accessed December 2007.

SWRL (2007). SWRL: A semantic web rule language combining owl and ruleml. http://www.w3.org/Submission/SWRL/, site accessed December 2007.

VoiceXML (2007). Voice extensible markup language (VoiceXML) 2.1. http://www.w3.org/TR/voicexml21/, site accessed December 2007.

Wielemaker, J., Hildebrand, M., and van Ossenbruggen, J. (2007). Using Prolog as the fundament for applications on the semantic web. In *Proceedings of the ICLP'07 Workshop on Applications of Logic Programming to the Web (ALPSWS-2007)*, Porto, Portugal.

XSLT (2007). XSL transformations (XSLT) version 2.0. http://www.w3.org/TR/xslt20/, site accessed December 2007.