R5-COP

Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems

Tool Design

Tadeusz Dobrowiecki, István Engedy, Péter Eredics, András Förhécz, István Majzik (BME)

Project	R5-COP	**Grant agreement no.**	621447
Deliverable	D35.20	**Date**	01/31/2016
Contact Person	Tadeusz Dobrowiecki	**Organisation**	BME
E-Mail	dobrowiecki@mit.bme.hu	**Diss. Level**	PU

Document History			
Ver.	**Date**	**Changes**	**Author**
0.0	7.01.2016	Lay-out of the deliverable	Tadeusz Dobrowiecki (BME)
0.1	17.01.2016	Architecture, requirements	Tadeusz Dobrowiecki, István Engedy, András Förhécz (BME)
0.2	20.01.2016	Numerical reasoning	András Förhécz (BME)
0.3	25.01.2016	Architecture, requirements, methods	Tadeusz Dobrowiecki (BME)
0.4	27.01.2016	Comments, modifications	István Majzik (BME)
0.5	27.01.2016	Modifications, prepared for internal review	Tadeusz Dobrowiecki (BME)
0.6	30.01.2016	Modifications, after internal review	Tadeusz Dobrowiecki (BME), Pavel Smrž (reviewer) (BUT)

Note: Filename should be

"R5-COP_D##_#.doc", e.g. „R5-COP_D91.1_v0.1_TUBS.doc"

Fields are defined as follow

1. Deliverable number ***.***

2. Revision number:

draft version **v**

approved **a**

version sequence (two digits) ***.***

3. Company identification (Partner acronym) *****

Content

List of Acronyms

DB	Data Base
DL	Description Logic
FOL	First Order Logic
JSON	Java Script Object Notation
KB	Knowledge Base
KDB	Knowledge Database
KM	Knowledge Management
KR	Knowledge Representation
ORA	Ontologies for Robotics and Automation
OWL	Web Ontology Language
ROS	Robot Operating System
SMKB	Skill Model Knowledge Base
SQWRL	Semantic Query-Enhanced Web Rule Language
SWRL	Semantic Web Rule Language
SysML	System Modelling Language
UML	Unified Modelling Language
XABSL	Extensible Agent Behavior Specification Language
XMI	XML Meta Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
XSD	XML Schema Definition Language

1 Introduction

1.1 Summary

The deliverable presents the design of the high-level architecture for the Configurer Tool and the Skill Composer tool-chain. First the review of the user roles, interactions, and the subsequent requirements is presented. The requirements are mapped into a modular architecture organized around an ontology based knowledge base, interfaced functionally to all kinds of foreseen users. Architecture modules are described shortly, then the algorithmic methodology providing the main functions of the system is shown in detail and summarized with respect to the components, the data formats, and the availability.

1.2 Purpose of document

The aim of the document is to present high-level design of the decision support tools conceived in the Technical Annex (WP35), with general requirements summarized in the D35.10 "Requirements". The architecture builds upon the design of the Skill Model Knowledge Base, reported in the D13.11 "Skill Model Knowledge Base (tentative)", and now, in the D13.12. "Skill Model Knowledge Base (final)", as its central component, with the accompanying functional components providing means to solve configuration related problems, drafted in the D13.20 "Configuration Model".

1.3 Partners involved

Partners and Contribution	
Short Name	**Contribution**
BME	Architecture, analysis, review of methods and tools, conclusions
	....

2 Decision support system - the aims

In the WP35 the development of two skill related decision support tools was planned, namely the Configuration Tool and the Skill Composer Tool (R5-COP D35.10). The Configuration Tool was meant to provide the application user with the skill level configuration solution to the robotic system able to solve the application. The Skill Composer decision support system was intended to be the tool of robotic system designer, who would be able to experiment with various (software and hardware) implementation variants to obtain a realizable system. Both decision support systems were expected to be interactive, to provide explanations to the decisions, and to provide tools to refine, re-configure the proposals when the conditions of the posed problems (application specification, software/ hardware component supply) change.

Configuration Tool

The task of matching abstract application requirements to the available systemic software and hardware resources is decomposed into two mutually interdependent steps. First the skill configuration is developed, realizing abstractly the application requirements. This step is supported by the Configuration Tool. An available skill configuration means that the application can be "solved" into a functioning robotic system (at least in theory), furthermore the obtained skill configuration serves as a backbone to develop feasible component configurations. It also makes it possible to evaluate robotic configurations equivalent at the skill level, but drawing on from different component sets. The required knowledge is encapsulated in an ontology-based core knowledge base, the Skill Model Knowledge Base (SMKB), equipped with services supporting ontology reviewing, transformation, and reasoning. Skill modeling, in general, and the requirements for the Skill Model Knowledge Base were reviewed and summarized in the D13.11. Formal model of the Skill Model Knowledge Base and its architecture is reported in the D13.12.

Skill Composer

Skill Composer is a decision support tool to find out suitable software and hardware components realizing a given skill. The problem is involved due to a variety of available and mutually replaceable components and the natural hierarchy of skill notion. In addition the numerical parameters appearing naturally in the component specification (for example, execution time, power related requirements etc.) should be taken into account, aggregated to evaluate skills for efficiency and resource spending, elevating the usefulness of the decision support.

From the point of view of designing a full robotic system the services of the Configuration Tool and the Skill Composer functionally complement each other and to obtain a full view of how an application could be implemented as a robotic system (configuration), the services of both of them are needed.

With a sufficiently elaborated knowledge base and algorithmic background Skill Composer can provide a software/hardware instantiation of the skill configurations proposed by Configuration Tool. To achieve it the skill configurations (outputs of the Configuration Tool) must be passed over (by the user, or in some other way automatically) to the Skill Composer tool (see Fig. 1).

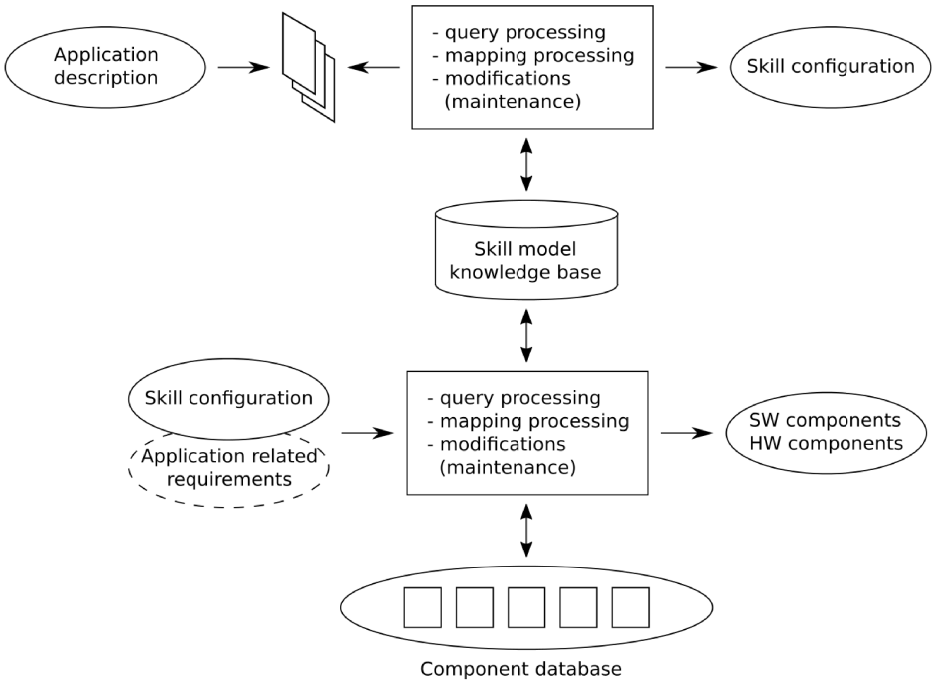

Fig. 1. Functional coupling of the Configuration Tool and the Skill Composer.

Although in the beginning of the R5-COP project both decision support tools were conceived to be entirely separated and independent system designs, with separate knowledge base components, see Fig. 2., it was recognized that their services are so strongly coupled, that a wise decision will be to integrate them and to design them as two independently usable components of the same integrated tool-chain, sharing the knowledge structures and their maintenance. Such integrated system design calls for a more differentiated and structured Skill Model Knowledge Base, than originally conceived, able to answer queries both about skill configuring and skill composing (these requirements were already taken into account in the knowledge base design reported in the D13.12).

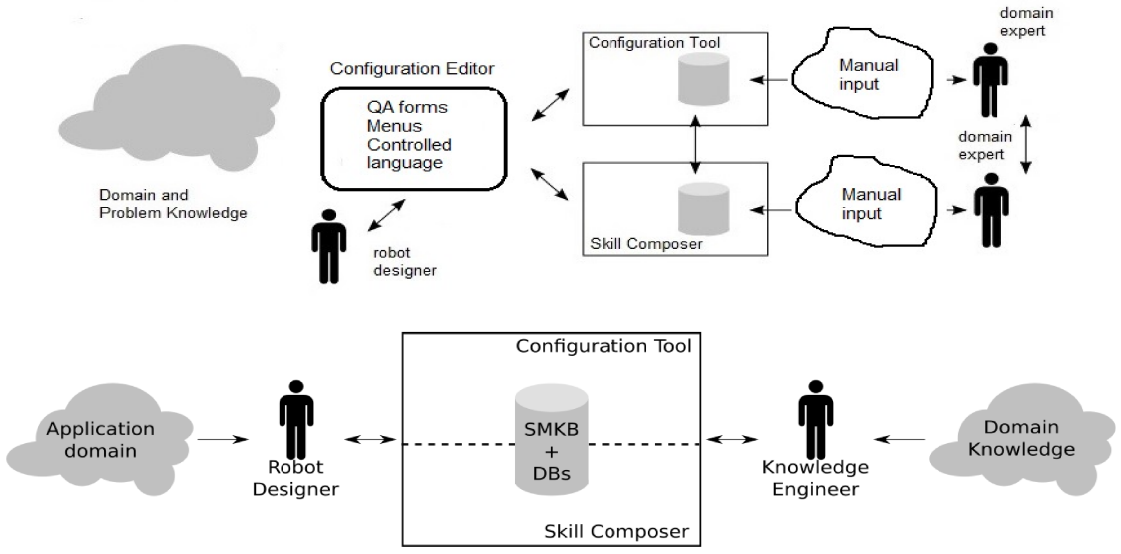

Fig. 2. Original conception of independent decision support tools and the adopted integrated system approach.

3 Requirements and design decisions

In an abstract way the working of a decision support system can be conceived as:

$$\begin{aligned} &\text{problem in user format} \Rightarrow \text{problem in tool format} \\ &\text{query in user format} \Rightarrow \text{query in tool format} \\ &\qquad\qquad\qquad\qquad \text{tool query processing} \\ &\text{solution in user format} \Leftarrow \text{solution in tool format} \end{aligned}$$

So the basic requirements should address the kinds of users, the interfaces, the transformations, and the processing.

Every knowledge intensive system has at least two kinds of users: system administrator responsible for keeping the knowledge (and other system services) ready to be used, and the true user bringing to the system problems to be solved (Sect 4).

Both users interact with the system in different way and require specific and different system interfaces (Sect 6). The interfaces interplay in the process of transforming the description of the problem on the user terms (as the user understands and handles it) to (and back) the description of the problem assumed by the built-in processing facilities (Sect 7).

Technically the typical processing services of an ontology based knowledge base (the Skill Model Knowledge Base is built from integrated ontology modules) is tracing the inheritance chains, establishing the existence of particular relations, and identifying concepts and entities based on queried concepts and relations (Sect 7.1).

For the intended application field it is not enough. From the application point of view the Skill Model Knowledge Base (SMKB) should facilitate the configuration and the re-configuration of the robotic systems where typical inferences could be to evaluate the realizability of skills in particular circumstances (variants in system components), or to identify analogous skills feasible in the modified circumstances. To this aim the typical ontology based reasoning should be complemented and integrated with numerical evaluation capabilities (Sect 7.2), considering that skill or HW/SW components can possess numerical attributes to be matched with the similarly numerical attributes of the application.

Lastly in a number of less structured application we can bring the user problem format to the system format so close that the format transformation is almost trivial and does not constitute a design difficulty. Not so here. In the robotic field the user operates with various structured semi-graphic models of different semantic content, and the ontology knowledge base also means structured models (with no matching semantic content). So the question of the user format-to-system format transformation (and back) is far from trivial (Sect 7.4).

Functionally the system should realize services for the system administrator for:

- the maintenance of the SMKB itself: inserting, deleting, expanding, verifying skills and other descriptions and knowledge chunks,
- the maintenance of the database components: inserting, deleting, expanding, verifying component descriptions and other documents,
- the maintenance of skill descriptions at the software/hardware component level:

inserting, deleting, expanding, verifying skill implementations with software/hardware components.

For the application user the realized services should be:

- provide easy to understand (native in the robotic design field) tools and interfaces to introduce the description of the problem to the system (the decision support tools),
- developing skill configuration for an application,
- summarizing/computing features of the software/hardware component coverage for a given skill,
- summarizing missing sub-skills and components,
- computing/qualifying some feasibility measures of implementing an application with the proposed skill configuration and components,
- searching SMKB for skills by activity, resources, conditions, etc.
- checking a given robot skill capabilities (i.e. given a software/hardware configuration, which skills can be implemented, "skill discovery").

The listed issues were already partly taken into account in the design of the Skill Model Knowledge Base where the following decisions were adopted (R5-COP D13.12):

- to integrate the skill model with the component model,
- to put the RA (Robotic and Automation) Ontology as the high level ontology over the skill model (Schlenoff 2012), (IEEE1872, 2015),
- to include a non-trivial option (i.e. going beyond the options defined in the standard OWL) for the numerical parameter modeling over components and skills.

4 Users

We can foresee three categories of users interacting with the system. System administrator, who is a knowledge engineer, but not necessarily knowledgeable in robotics, maintain the usability of the knowledge-base of the system. An ontology knowledge base is technologically complicated and even when we will use already predefined public modules, like Protege plug-ins and embedded description logic reasoners, maintaining ontologies is not a task for the robot designers.

Knowledge Engineer (KE) system administrator oversees the up-dating of the ontology and checking it for consistency. He also technically oversees updating and verification of other knowledge/data storage. He is responsibly for periodical system tests and for identifying causes of the erroneous functioning.

Taking into account that the Upper level ontology is the IEEE Standard Robotic and Automation ontology (ORA), which is under continuous development (R5-COP D13.12), upon any new release of the standard the KE system administrator has to up-load it to the SMKB. The most involved task next is to "stitch" it with the lower level ontology components.

Consider e.g. that in the current release of the ORA an individual robotic system "Care-O-bot-3 " is connected to the concept of "semiAutonomousRobot", but the future release of the ORA will already contain the notion of "householdRobot", or "serviceRobot", with some specialized features. It is in the interest of more efficient reasoning to reconnect the Care-O-bot-3 to the serviceRobot node, however it is retrograde in the inheritance and cannot be done automatically. Only and expert in the ontology and an expert in the robotics can do this task adequately. The KE system administrator can be both, but it is safer to assume that there is another **Robot Designer (RD) system administrator**, who knows insides of the knowledge base to the point of efficiently cooperating with the Knowledge Engineer, but also provides domain knowledge. Besides helping refining the connection between the Upper Level and the Middle Level ontologies (for details see the D13.12) RD system administrator is responsible for maintaining the actuality of the component knowledge and database (introduction of the new classes of software and hardware components, introducing advanced versions of the existing components, eliminating obsolete choices, refining skill and component descriptions). The simple addition of the new knowledge (individuals) to the knowledge base is relatively straightforward. It is the refinement which is more involved, can lead to inconsistencies, and demands the cooperation of the Knowledge Engineer and the Robot Designer system administrators.

A part of the Robot Designer system administrator task can be outsourced to the robotic community. Newly developed applications can introduce demands for skills not present in the system knowledge base, neither familiar to the RD system administrator himself. These new needs can be described in a controlled way and introduced into a remote (internet) database, from where they will be occasionally read, verified, and if sound, introduced in time into the system knowledge base.

The third category of users is the application user, i.e. a **Robot Designer user**, who asks for the decision support. Robot Designer user in his own interest can maintain four kinds of interactions with system. Robot Designer can learn about the task and the prospects of using the system by studying various documents provided by the system, manuals, data sheets, component descriptions, etc. Perhaps the most important of them will be a concise dictionary

or thesaurus of the lexical notions used by the system at the user interface. It will help the user to re-adjust to the system vocabulary the description of his problem, lessening that way the odds for errors.

Understanding well what information can be introduced into the system next the user builds models of his problem which will be then transformed into some internal format useful for the machine processing. At this moment the decision support system is in principle ready to work and to give advices. The system can assist the user in computing, comparing, modifying (skill) structures for given tasks, or can help cover the skill structures with system components, yielding executable skill options. Those can be compared and further modified.

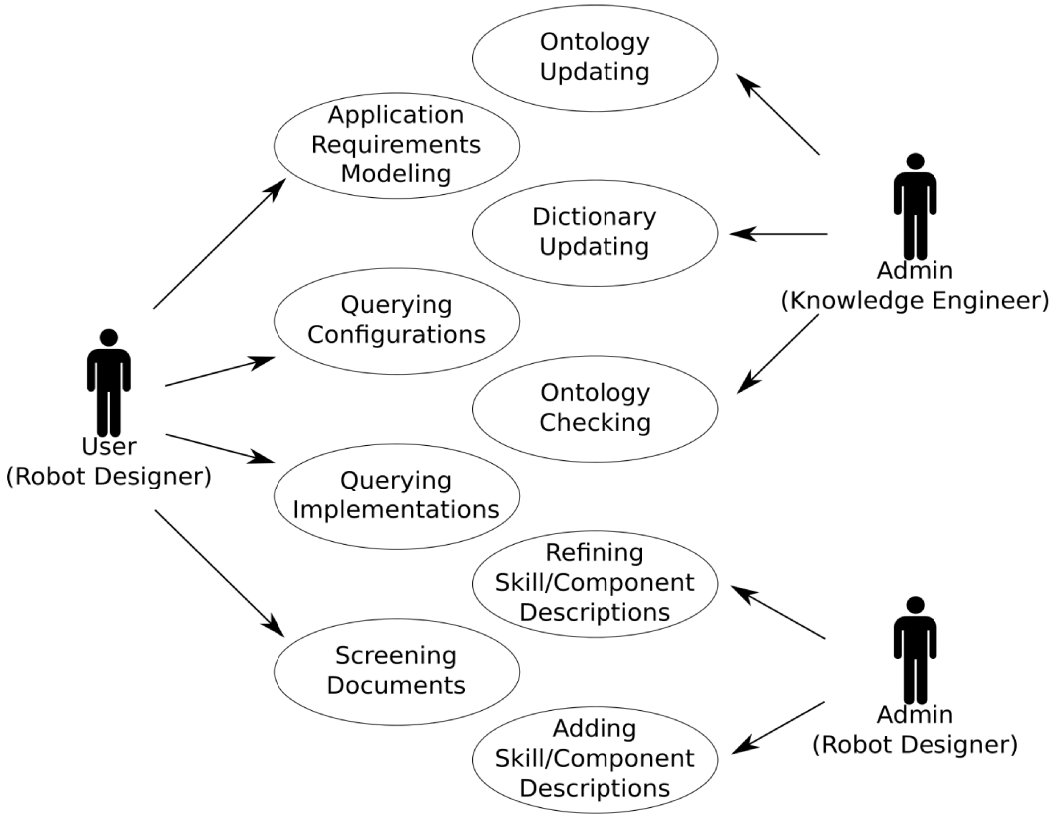

Fig. 3. System users with the respective fields of responsibility.

5 Interactions

Working session of a Robot Designer user is organized around three kinds of queries: queries revealing to the user system services and the GUI know-how, application model building queries and the substantive queries about configurations and implementations.

5.1 Querying know-how

It would be advisable to provide the user the following informative queries, essentially based on the browsing of the system knowledge and data bases:

- browsing definitions of the RA ontology concepts,
- reviewing the skills, the component classes, etc. names used/accepted by the system,
- reviewing particular skill definitions based on their relations to other ontology concepts (presenting ontology chunks in a readable natural language),
- reviewing the collections and descriptions of software and hardware components known to the system,
- reading FAQs, manuals, various repositories of suggested interactions,
- posing notes and questions to the system administrator.

The aim behind these services is to make the user familiar with the vocabulary accepted by the system and to help him to formulate legal models (i.e. unambiguously interpreted by the system).

5.2 Querying knowledge base for applications

After the user grasped the decision support functions offered by the system, the application designing session can start. First the user must introduce to the system the model of his problem, building it from scratch, or recalling a stored model for extensions and modifications.

Once the model of the user application is stored in the system and represented in the form native to the system reasoning, this model can be matched to the system knowledge structures and particular questions can be answered on this basis.

The system will probably keep an annotated trace of the user's session with the storage of the intermediate results, to be able to suspend it temporarily and to pick it up later, or to step back in time to make it possible for the user to introduce better models, knowledge, or to reformulate queries.

In the following RD stands for the Robot Designer user, and DS stands for the Decision Support system.

Presenting the problem

RD introduces, through the designated interfaces, models of the application (see Sect 7.4), as semi-graphic SysML models supplemented by natural language texts interpretable by the system (Sect 7.3). It is information about the:

- environment,
- tasks,
- mission,
- constraints,
- performance demands, etc.

DS evaluates the data and asks (if needed) for details.

RD supplies further details.

(DS transforms the models into the internal (OWL compatible) format, see (OWL))

Result: an internal representation of the problem stored for further analysis.

Developing skill configuration

RD identifies a stored problem, then asks for its skill-based evaluation.

DS performs analysis (reasoning) and presents a list of feasible skill configurations.

RD asks for ordering the list acc. to the supplied criteria, or RD asks for the best option acc. to the supplied filter (reasoning).

RD asks for the resource and performance summary (reasoning).

Result: qualified feasible skill configurations for the problem (stored).

Implementing skill

RD identifies a skill, then asks for its component implementation.

(RD can indicate some obligatory components to be taken into account in the evaluation).

DS performs analysis (reasoning) and presents a list of feasible skill implementations.

RD asks for ordering the list acc. to the supplied criteria, or RD asks for the best option acc. to the supplied filter (reasoning).

RD asks for the resource and performance summary (reasoning).

Result: qualified feasible skill implementation (stored).

Presenting robot system with variability options

RD introduces through the designated interfaces (as semi-graphic SysML models supplemented by natural language texts interpretable by the system), robot specification with all alternative sensor/ actuator/ etc. component configurations.

DS asks (if needed) for details.

RD supplies details.

Result: internal representation of the robot system, implicitly together with all its system configurations, stored for further analysis.

Verifications

RAD asks to recall a particular problem and robot description.

RD asks whether this robot can solve the problem.

DS recovers skill configurations found for the problem.

DS compares (reasoning) possible skill configurations with the available robot configurations and presents a qualified list (acc. to performance demands).

RD selects options and asks for resource and performance summary.

RD asks for the summary, how the selected robot configuration relates to the actual settings (case of reconfiguration).

DS: If no match possible, DS presents the summary of the most essential discrepancies between the problem requirements and the robot capabilities.

Result: internal representation of the robot configuration(s) suited to solve the problem, or optionally indications why the solution cannot be reached.

Remodelling the problem

RD asks to recover a particular problem.

DS presents its input forms.

RD introduces changes and asks for storage.

DS verifies that the original skill configuration solution is valid/not valid for the changes and modifies it if needed (reasoning).

If no configuration is valid, DS initiates (Developing skill configuration).

Result: internal representation of the modified problem stored for further analysis, and optionally qualified feasible skill configurations for the modified problem..

Remodelling the robot

RD asks to recover a particular robot specification.

DS presents input forms with content.

RD introduces modifications and ask for storage.

DS evaluates the modification and remakes the robot internal representation (less/more configurations).

Result: internal representation of the modified robot together with all its system configurations.

5.3 Querying system maintenance

Similarly a number of interactions must be provided for the system administrators.

KE system administrator, possessing deep knowledge about the system, does not require formatted queries. He must be able to reach the internal levels of the representation

(ontology) and processing (rules) directly. He can be helped by the purpose oriented editors, but such tools are already known and can be adopted here (Protégé).

RD system administrator is less knowledgeable and perhaps it would not be wise to give him rights to fumble with the deep components of the system. As his duty is primarily to advice the KE system administrator and to introduce new component and skill information, he may be provided by properly crafted input forms with the option that if their information is not acceptable to the system, it is put aside to be consulted with the KE administrator.

6 System architecture

6.1 Components

In the following we give a high-level design of the proposed system architecture. An open question not addressed in the design is to what extend make this system distributed. There are many options from the client-server kind of architecture to the downloadable stand-alone applications. The client side can be a simple web form based, or can be a downloadable component performing non-trival preprocessing and formatting. The following main components of the system were identified (see their relations in Figure 4):

Knowledge Base

The primary knowledge repository of the system comprising ontologies, ontological axioms, and reasoning rules. The principal component of this knowledge base is the Skill Model Knowledge Base described in the deliverables D13.11 and D13.12. (R5-COP D13.11), (R5-COP D13.12)

Document Database

The collection of forms for providing descriptions and queries. Storage of user sessions, results, and configurations. Storage for the information on individuals (software, hardware components), component data sheets. Storage of various documents, manuals, dictionaries, etc.

OWL Reasoner

Description Logic (DL) based reasoning component - an embedded fully functional Pellet reasoner (Baader 2005), (Horrock 1999), (Pellet).

Rule Reasoner

Emerald rule reasoner (Emerald) working with the SWRL rules (about the SWRL rules and their role in the OWL based reasoning see D13.12, and Sect 7.1, 7.2), but also with flexible numerical extensions and able to perform numerical computations over the DL reasoning level and to integrate them with logic based reasoning.

Skill Engine

Component transforming skill related user queries into the internal representation conform with that used in the knowledge base (i.e. ontology chunks in the OWL format, with numerical attributes).

Vocabulary and Grammar Dictionary

Collection of the acceptable vocabularies, dictionary definitions, and grammar chunks to support the controlled natural language entries. (acceptable vocabularies bridge the officially

sanctioned notions from the ORA ontology standard, the notions proposed in various robotic glossaries (for more detail see the D13.12) and the concept names used in the ontology definition in the knowledge base).

Natural Language Support

Controlled natural language interpreter and generator. Partially supported by the predictive text principle.

Skill Composer/ Configuration Query

The primary user GUI to conduct sessions. It yields the possibility to formulate queries addressing system knowledge base, previously introduced application and robot models, the relations of them, earlier results of the session, particular parameter data.

Application Model Builder

Component responsible for the administration of the (UML/SysML based) models transformed into the internal OWL format of the knowledge base.

UML - OWL Transformation

Model transformer between robot related SysML (and other) models and the OWL format used in the knowledge base. OWL-to-UML transformation is easy and is built-in on the standard basis into the ontology management tools. Not so the UML-to-OWL transformation (see Sect xxx). A problem to solve, as a standard solution to this problem does not yet exist.

Model Importer

User GUI making it possible to import SysML (and other) application and robot models.

Query GUI

User GUI to formulate queries. Structured as web forms with controlled natural language inserts.

Skill Descriptor Editor

The GUI for independent external experts to provide the system their ideas about skill and the information related to them.

Skill Description Forms

An interface to a remote server collecting skill related data from the prospective system users. The aim is to enroll independent robotic experts in the expansion of the system knowledge.

Document Browser

Document browser is a tool for the KE administrator to screen the content of the document

database. Some of the content of the document database is uploaded automatically (e.g. skill description forms), their usability must be verified and their content transferred into the system knowledge.

Knowledge Base Editor

The most essential, but standard component. The primary tool of the KE system administrator. Must be equipped with procedural support to screen, verify, and modify knowledge structures. For the ontology it will be a Protege based component, for the reasoning rules it will be a component from the Emerald system (Emerald).

Data Importer

Component responsible to distribute into internal storage the external data verified and accepted by the KE administrator.

External Database

Here belong external ontology repositories (e.g. the ORA), ROS package repositories, manufacturer data sheets, company web pages, maintenance manuals, etc. Some of their content can be directly down loadable to the system storage (e.g. sources in the OWL format), some must be interactively verified by the KE administrator, or even manually digested.

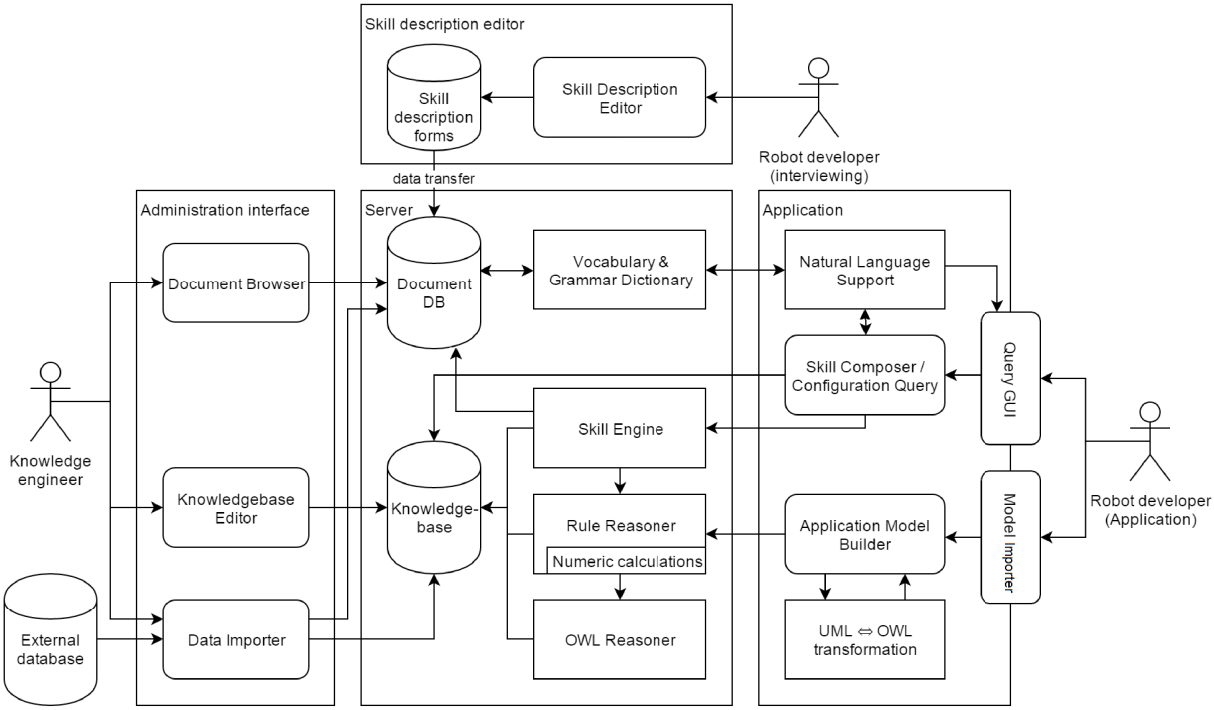

Fig. 4. System architecture.

6.2 Languages for data representation

In the following a summary of the data and knowledge description languages is presented.

The OWL ontology language will be used to describe skill knowledge, the knowledge about components types and hierarchy, furthermore the knowledge about the application task environment. OWL will appear in textual or graphical form on the ontology editing interfaces, and it will also serve as an internal representation in the reasoning rules. The OWL ontology language comes with different syntax: OWL2 Functional Syntax, OWL2 XML Syntax, (user friendly) Manchester Syntax, RDF/XML Syntax, and RDF/Turtle Syntax (the internal representation used in rules will draw form the user-friendly Manchester syntax).

SWRL is a logical rule language, which makes it possible to define Horn-clause (i.e. multiple positive conjunctive premisses and a single positive consequence) rules for the OWL ontology. Rules are formulated in an abstract syntax abstracting from the abstract OWL syntax, but can be also presented in human readable form. Rules will be used amplify the efficiency of the logical reasoning in incorporating numerical data, needed to express quantitative aspects of the application requirements and the parameters of the software and hardware components.

SysML is a visual modeling extension (dialect) of the UML, semantically extending the set of the UML diagrams to a general purpose (physical) system modeling language, more appropriate for engineering applications. Both OWL (any syntax) and UML (SysML included) can be transformed into XML based descriptions (XMI XML Meta Interchange) to facilitate the knowledge transfer. SysML (and derived UML profiles) will be used by the user to describe application task requirements and the expectations on various robot components.

JSON is a handy text file format structured in the user friendly way, with standard, or easily adaptable processing tools. I will be used to establish a well structured database on the concrete software and hardware components.

6.3 Summary of the components and technology

Component	Input/Output Data Format	Technology/ Algorithm	OTS/ Internally Developed
Knowledge Base	OWL (XML)	Logical (DL) reasoning	Internally developed
Document Database	XML, PDF, TXT, JSON, graphic formats	File directory. Respective file R/W	OTS
OWL Reasoner	OWL, internal formats	Logical reasoning	OTS (Pellet)
Rule Reasoner	SWRL, OWL (XML)	Rule interpreter	OTS (Emerald)

Skill Engine	OWL (XML)	XML processing	Internally developed
Vocabulary and Grammar Dictionary	XML, TXT	XML processing	partial OTS/ Internally developed
Natural Language Support	XML, TXT HTML	XML processing Web form processing	partial OTS/ Internally developed
Skill Composer/ Configuration Query	OWL (XML)	XML processing	Internally developed
Application Model Builder	OWL (XML)	XML processing	Internally developed
UML - OWL Transformation	SysML/ OWL	XML processing	partial OTS/ Internally developed
Model Importer	SysML (XML) HTML	UML editing XML processing Web form processing	OTS
Query GUI	HTML, TXT	Web form processing	partial OTS/ Internally developed
Skill Descriptor Editor	HTML, TXT	Web form processing	partial OTS/ Internally developed
Skill Description Forms	XML HTML	XML processing Client-server operations	Internally developed
Document Browser	HTML	Web form processing	
Knowledge Base Editor	HTML OWL (XML), SWRL	Web form processing Logical reasoning	Internally developed OTS (Protégé, Pellet)
Data Importer	HTML, XML	Client-server operations, browsing	OTS/Internally developed
External Database	XML, PDF, JPG	Web technology	OTS

7 Services and Algorithms

In this section we present the summary of formal methods underlying various algorithms to be implemented in the tool and involved in the reasoning and building and interpreting queries. A part of this material parallels section on reasoning from the D13.12.

7.1 Logical reasoning with OWL-based ontologies

Logical reasoning is main tool to handle state-of-the-art formal ontologies. Logical reasoning can compute derive new logical facts from the logical knowledge base, or can decide that some proposed facts are consistent with the knowledge base. In the tool-chain logical reasoning is needed in the Knowledge Base and the OWL Reasoner to compute answers to the user problems, and in the Knowledge Base Editor, to compute answers related to the consistency of the proposed extensions to the knowledge base.

The OWL and OWL2 ontology languages are equipped with proper formal semantics and this makes it possible to use logical reasoning to answer different inference problems with regard to an ontology. The most common inference problems are as following:

- Ontology Consistency: O ontology is consistent (or satisfiable) if a model of O exists.
- Ontology Entailment: O ontology entails O1 ontology if every model of O is a model of O1.
- Ontology Equivalence: O and O1 are equivalent if O entails O1 and O1 entails O.
- Class Expression Satisfiability: CE class expression is satisfiable w.r.t. O ontology if CE is not empty.
- Class Expression Subsumption: CE1 is subsumed by a class expression CE2 w.r.t. O if the CE1 class expression (concept) is a subset of the CE2 class expression (concept).
- Instance Checking: if a particular instance is a member of a given concept.

The OWL ontology language is equivalent with a particular kind of so called Description Logic (DL) (SHOIN), see (Baader 2005), (Horrock 1999). In the DL knowledge base there are schema (terminological) and data axioms. In its simplest form, terminological axioms can be used to introduce names (abbreviations) for complex descriptions. A TBox is constituted by a finite set of terminological axioms which define subsumption and equivalence relations on classes and properties. The assertional formalism (ABox) can be used to state properties of individuals. Assertional axioms or Assertions introduce Individuals, i.e. instances of a class, into the knowledge base and relate individuals with each other and the introduced terminology. A knowledge base for the reasoning is just a TBox plus an Abox (see Fig. xx).

The atomic reasoning options for the TBox and the ABox make it possible to implement the reasoning schemes addressing the ontology and listed above.

Tbox contains terminology axioms: definition axioms, e.g.:

$$\text{ServiceRobot} \doteq \text{Robot} \sqcap \neg\, \text{IndustrialRobot},$$

or General Concept Inclusion axioms, like:

$$\text{ServiceRobot} \sqsubseteq \exists\ \text{CONTROLS.User}$$

Abox contains individual information, e.g.: ServiceRobot(care-o-bot-3) or

$$\text{CONTROLS.User}\ (\text{john},\ \text{care-o-bot-3})$$

In the Tbox we can check the satisfiability, the subsumption, the equivalence, the disjunctiveness. In the Abox we can check the consistency, the instance check, the instance retrieval, the individual realisation, and the satisfiability of a concept.

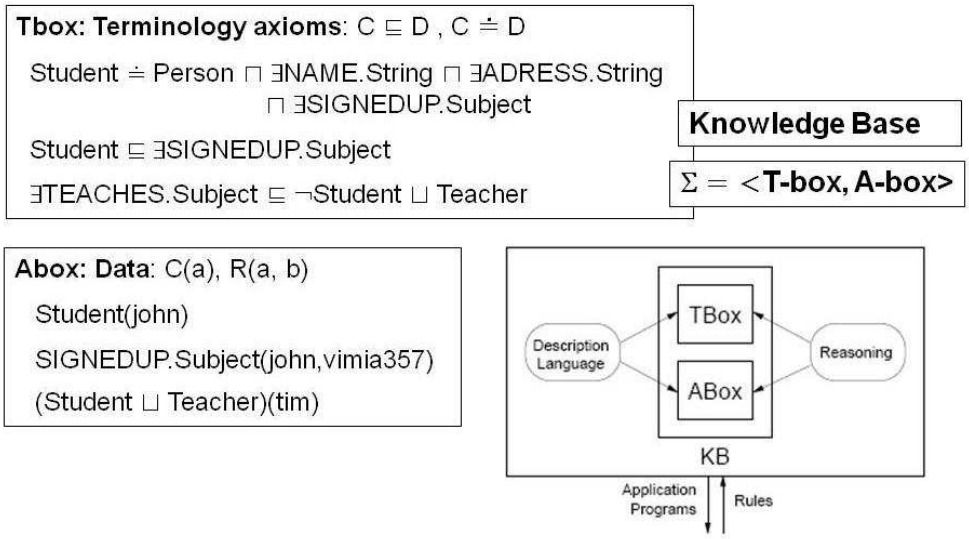

Fig. 5. Tbox and Abox knowledge base in the DL logic.

Reasoning schemes for the Tbox and the Abox can be model based (tableaux methods) or proof based (resolution method). The essence of the tableaux method is to construct the model of the negated query from the structure of the query itself. The construction branches out, but if every branch leads to contradiction, the negated query cannot be satisfied, which can be used to decide the subsumption of the examined concepts. The tableaux method is complete, sound, and terminating. As a result it builds a graph - a model of the query, which contains built-in individual information which can be used to answer the query.

Another option is to use the general proof schemes of the first order logic (resolution). The resolution is contradiction complete, i.e. can proof a contradiction whenever one exists. To use the resolution we must thus to negate the query, and when the contradiction is found, it means that the query is true. The individual information is usually found in the unification and substitution needed to perform the resolution step.

There are many OWL reasoners available. They may have different computational characteristics as they take a different approach in the inference problem. The most notables are (Pellet), (HermiT) and (Fact++), all supporting the OWL API, a Java programming

interface for OWL ontologies and reasoners.

Reasoning with rules in OWL implementation

Due to the nature of OWL, complex domain knowledge cannot be represented easily. To overcome limitations of OWL, the integration of description logics (DLs) and rule languages (typically Datalog) was investigated (Krotzsch 2011). There are several approaches which allow syntactically combining both OWL axioms and rules in ontologies and the combined formal semantics defines how the hybrid language is understood.

Semantic Web Rule Language (SWRL) is a proposal for extending OWL knowledge bases with the Unary/Binary Datalog RuleML sub languages of the Rule Markup Language (Rule-Markup). The proposal extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base. SWRL rules are an implication between an antecedent and consequent, both consisting of zero or more atoms. Atoms can be of the form C(x), P(x, y), sameAs(x, y) or differentFrom(x, y), where C is an arbitrary OWL class expression, P is an OWL property expression, and x, y are either variables, OWL individuals or OWL data values.

Extending OWL with SWRL rules results in a non-decidable logical formalism: there is no algorithm that can, in finite time, compute whether an axiom is entailed by a SWRL knowledge base. To overcome this limitation, SWRL has an alternative interpretation with limited expressive power: the DL-safe semantics. DL safety is a simple idea which is implicit in many rule systems and has been used in other contexts to regain decidability: variables in DL-safe rules bind only to explicitly named individuals in the ontology. Adding this restriction is sufficient to make SWRL rules decidable.

When extending the DL reasoner with rule-based reasoning capability one could create a hybrid reasoning engine, where the two knowledge bases are synchronized after each inference step. A popular implementation of a hybrid DL reasoner is the SWRLJessTab plug-in (Jess-Protégé) that supports the execution of SWRL rules using the Jess rule engine (Jess). A serious limitation of this approach besides performance considerations is that the rule engine may not capture all OWL axioms and as a result possibly inconsistent knowledge can be inferred by the hybrid system.

A cleaner approach is a rule engine specifically designed to work on OWL knowledge bases. In this case rule inferences are carried out by constantly querying the DL reasoner, hence all data is stored in a single location. The SWRL reasoner implemented inside Pellet is designed accordingly. In the tool-chain such rule interpreter is embedded into the Emerald rule interpreter (Emerald) (Sect 7.2), which will be used in the Rule Reasoner component.

7.2 Numeric types and numerical reasoning

At the reasoning level one of the main problems is the semantically sound integration of the conceptual logic reasoning with the parametric numerical reasoning. Consider an example from Fig. 6. The walking skill requires among others a working motor, which in turn requires energy (battery). So the skill in principle can be implemented, and the application level requirement for walking can be satisfied. But the devil is in detail. The application requirement specify 1h walking (or 10 km walking). Is the problem solved? Not really. The

battery capacity, coupled with the motor energy consumption numerically qualifies the skill, which only then is comparable to the application demands. So without certain ability to handle numerical knowledge, answering the user queries would be pointless.

The description logic behind OWL has very limited inference capabilities for numeric values, but has very good means in defining the value space for complex data models, also with numeric properties. More advanced calculations can be implemented with rule languages built on top of the OWL data model. (this Section parallels the respective section from the D13.12).

In the tool-chain the numerical evaluation (numerical reasoning) is built-in into the Emerald Rule Interpreter (Emerald), the primary reasoning engine of the system (used in the Rule Reasoner component). Emerald has its own rule language (based on SWRL) with a concrete syntax derived from OWL Manchester syntax and the semantic is an extension of SWRL DL-safe language semantics.

Data types in OWL

OWL DL uses the XML Schema Definition Language (XSD) (OWL2-Syntax Sec. 4, W3C-XSD) for data type definitions and semantics. The most frequently used primitive data types for numerical data are decimal, integer, float and double.

Decimal is the set of numbers that can be obtained by dividing an integer by a non-negative power of ten. The lexical space for decimal numbers can be defined by the regular expression: “(\+|-)?([0-9]+(\.[0-9]*)?|\.[0-9]+)”.

Integer is derived from decimal by having no fraction digits and disallowing the trailing decimal point, resulting in the standard mathematical concept of the integer numbers.

The value space of integer is the infinite set {...,-2,-1,0,1,2,...}.

The **float** data type is patterned after the IEEE single-precision 32-bit floating point data type, and similarly **double** is patterned after the IEEE double-precision 64-bit floating point data type, well known from many programming languages. Floating point numbers are often used to approximate arbitrary real numbers.

OWL2 defines two additional numeric data types, **owl:real** and **owl:rational**, with the value spaces “all real numbers” and “all rational numbers”, respectively. The owl:real data type does not directly provide a lexical representation, while owl:rational has a simple lexical form defined by the grammar

numerator '/' denominator

where both *numerator* and *denominator* are integers.

Data types similar to numeric values include booleans (xsd:boolean) and time instants (xsd:dateTime, xsd:dateTimeStamp).

Boolean represents the values of two-valued logic: {true, false}.

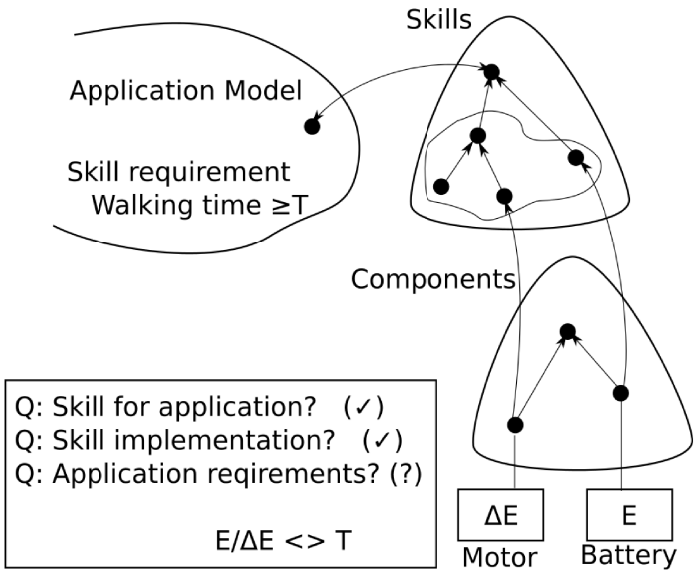

Fig. 6. Interaction of the numerical parameters within the ontology.

Custom data types to create complex values

The set of standard data types in OWL2 is somewhat limited and our application may require additional types to inference with. As an example vectors and matrices of real numbers are commonly used in robotics systems.

There is two distinct ways to store complex data in data types using the OWL2 language. One can store complex data using the binary types (xsd:hexBinary, xsd:base64Binary), encoding the data into a custom format. Then a custom reasoner can extract the information from these data chunks for specific properties. This means only the pre-defined set of data types are used, but does not use the language's capabilities in specifying the real data types of these properties.

A better way is to define custom data types. The language allows to define new data types with an arbitrary name and custom value spaces. These types are not part of the standard types, hence reasoners cannot infer about their values. As an example, in the following OWL ontology the data types "vector" and "matrix" are introduced, and two data-properties, "position" and "rotation" use these as their range, respectively. The ontology is presented in OWL Functional Syntax.

```
Prefix(r5:=<http://www.mit.bme.hu/r5cop#>)

Ontology(http://www.mit.bme.hu/r5cop/datatype-examples.owl

    Declaration(Datatype(r5:vector))
    Declaration(Datatype(r5:matrix))

    Declaration(DataProperty(r5:position))
    DataPropertyRange(r5:position r5:vector)

    Declaration(DataProperty(r5:rotation))
    DataPropertyRange(r5:rotation r5:matrix)
```

```
Declaration(NamedIndividual(r5:Object1))
DataPropertyAssertion(r5:position r5:Object1 "[10,0,20]"^^r5:vector)
DataPropertyAssertion(r5:rotation r5:Object1
      "[[0.5,0.866,0],[-0.866,0.5,0],[0,0,1]]"^^r5:matrix))
```

The value space of the data type can be restricted with a data type definition, but according to the OWL specification such data types have empty lexical spaces. The language does not allow such data types to occur in literals. Hence either one defines the allowed values of the data type in a data type restriction, e.g. using a regular expression:

```
DatatypeDefinition(r5:vector DatatypeRestriction(
      xsd:string xsd:pattern "\[[0-9]*(\.[0-9]+)?(,[0-9]*(\.[0-9]+)?)*\]"))
```

Or the datatype can be used in literals, e.g. to introduce data property values:

```
DataPropertyAssertion(r5:rotation r5:Object1
      "[[0.5,0.866,0],[-0.866,0.5,0],[0,0,1]]"^^r5:matrix)
```

Since we introduce the data type to store data, we should skip the restrictions from the ontology and enforce valid values with our custom reasoner.

Inferencing with numeric values in DL using facets

Facets are the only way to introduce constraints on the values on numeric data in OWL DL. The language has four constructs to bound the value space for a data type according to the data type's ordering:

- maxInclusive: inclusive upper bound, e.g. xsd:integer[<= 10]
- maxExclusive: exclusive upper bound, e.g. xsd:integer[< 10]
- minInclusive: inclusive lower bound, e.g. xsd:integer[>= 10]
- minExclusive: exclusive lower bound, e.g. xsd:integer[> 10]

The bounding values of facets are strictly constants. As an example, one can define an adult as follows:

```
Class: Adult
      EquivalentTo: Person and (age some xsd:integer[>= 18])
```

Two data-property values of the same instance cannot be compared in OWL DL.

Rule extensions and numeric inferences

SWRL allows for complex conditions combining data-property values, as variables may be bound to these values, and n-ary predicates can be used to evaluate functions. E.g. the area of a square can be defined in the following SWRL rule, using the built-in function "swrlb:multiply":

```
Rectangle(?r), hasHeight(?r, ?h), hasWidth(?r, ?w),
      swrlb:multiply(?h, ?w, ?a) -> hasArea(?r, ?a)
```

SWRL built-in functions

SWRL includes built-in functions that operate on the standard data types of OWL. These built-ins are based on the reuse of existing built-ins in XQuery and XPath.

Built-ins area divided into groups: comparisons, math, boolean, strings, etc. The previous example included a math built-in for multiplication. As an example, a comparison built-in can be used to define squares as rectangles with equal sizes, a definition beyond the expressive power of OWL:

```
Rectangle(?r), hasHeight(?r, ?h), hasWidth(?r, ?w), swrlb:equal(?h, ?w) -> Square(?r)
```

Math Built-Ins cover addition, subtraction, multiplication, division, integer division, modulo, power, unary addition, unary minus, absolute value, ceiling and floor limits, two kinds of rounding, sine, cosine, and tangent.

Some reasoning engines allow for the extension of SWRL built-ins. With this mechanism, one can extend the numeric capabilities of the language to custom data types. As an example, matrix multiplication can be defined as a custom built-in function in Pellet with the following Java class:

```
private static class MatrixMultiply implements GeneralFunction {
    public boolean apply(ABox abox, Literal[] args) {
        // first two arguments are the matrices to multiply
        Matrix A = new Matrix(args[0].getLexicalValue());
        Matrix B = new Matrix(args[1].getLexicalValue());
        Matrix AB = A.multiply(B);
        // the 3rd argument is the result of the multiplication
      args[2] = abox.addLiteral(ATermUtils.makeTypedLiteral(AB.toString(), MATRIX));
    }
    public boolean isApplicable(boolean[] boundPositions) {
        //the built-in is applicable for three arguments only
        return boundPositions.length == 3;
    }}
```

Emerald built-in functions

Emerald is an expert system built on top of OWL and SWRL, with a custom rule language with additional capabilities (Emerald). The system also allows for automated information collection from an end user by asking for relevant information wrt. a goal statement.

In Emerald functions can have object variables, and the whole ontology is accessible from within the function implementation. As a result functions are not limited by their fixed arity, but can query arbitrary-sized complex data structures.

7.3 Controlled natural language processing

Controlled natural language is an approach to bring closer the natural and contradictory requirements of the human user interacting at the system interface with the information

system. Human user has command of a flexible, universal and informal language tool, the computer system has command of equally universal formal language tool. The controlled natural language connect both worlds and can be used as an effective input and output tool. In the tool-chain in can be used at any user interface (Skill Description Editor, Query GUI, Model Interpreter, Document Browser, Knowledge Base Editor, Data Importer) where the user is given the possibility to introduce free textual descriptions.

Controlled natural language is such artificial language which in the interest of the successful computer processing limits the vocabulary, the grammar, and the semantics, yet keeping the natural language character of the developed language.

An important property of the controlled languages is that they retain the natural character. Sentences written in the controlled language are easy to understand, to interpret by humans. In case of controlled languages tailored for particular application the grammar can shrink even to some tens of rules. A vocabulary of a hundred (let it be a thousand) words is generally enough for every serious application (but the controlled languages are of course expandable).

In a number of the controlled natural languages the design of the grammar rules and the vocabulary is aimed not only at the easy computer processing, but at the unambiguous knowledge representation also. Sentences formulated in such **logic based controlled natural languages** can be transformed into first order logic sentences. The primary application field for such languages is the design of knowledge repositories (ontologies) and the facilitation of the knowledge base development for designers not versed in formal methods. Such languages frequently contain built-in knowledge query grammar rules and are equipped with reasoning engines.

Rabbit language can be used to edit ontologies in the OWL format (Rabbit). Similar tool is the Sydney OWL Syntax (SOS), which not only makes it possible to design OWL ontologies by people not versed in its details, but also makes it possible to translate the ontologies into plain English. Let us mention still the Controlled Language for Ontology Editing (CLOnE).

In the planned decision support tool the role of the controlled language is envisaged as follows. Minimally it can be used to generate comprehensible descriptions about various knowledge chunks and data stored in the system knowledge and data bases. The true gain will be if the controlled language would help at the input, facilitating for the user the unambiguous introduction of the textual information assisting the semi-graphical models, towards a better transformation into the system internal formats. The mentioned option that the logic based controlled languages are directly representable in logic which is also in the background of the ontologies could help much.

7.4 Model transformations

The greatest challenge in the design of the present decision support tool is perhaps the Robot Designer user interface (in the Model Importer component). The thinking of the user is structured and governed by various diagrammatic models. To divert him from them would mean a serious handicap and the possibility that the model of the application will be distorted and/or under-defined.

One could develop a well structured GUI, with structured input forms, where specific

information should be introduced being controlled by the form structures, captions, and controlled language. To design such GUI well, to obtain a meaningful user input, would mean however in practice that the GUI designer reproduces informally the user models.

User models are formulated in the UML and in the UML extension SysML. The decision support system works internally with the OWL. The transformation between the OWL models to the UML models is solved and is a built-in feature of the ontology editors. Not so the transformation from the UML models to the OWL models.

There is a number of reasons, why such transformation is not a trivial one (Kiko 2008). The main issue is that these languages were devised to fulfil different purposes. While OWL supports the representation of knowledge about a system, UML was developed primarily to support the construction of a (software) system. UML schemes are models, but the ontologies are not models in the same sense. As a result (Kiko 2008):

- Models focus on realization (ontologies do not)
- Ontologies are for run-time knowledge exploitation (models are not)
- Ontologies are for Representing Web Based Information (models are not)
- Ontologies are formal (models are not)
- Ontologies can support reasoning (models can not).

There are further differences also:

- Global scope and first-class status properties (In the OWL the general semantics of the properties is independent of a specific domain and range context. The UML does not sanction global scope of attributes and associations because of the closed-world interpretation placed on these constructs)
- Unique Name Assumption and Synonyms (OWL meets these requirements by allowing the definition of synonyms for classes, properties and individual descriptions, UML does not)
- Open-World vs. Closed-World Assumption (The UML is oriented towards data modeling and system construction so the represented knowledge is implicitly viewed as being complete. OWL, in contrast, interprets models as potentially representing partial knowledge).
- Profiles (The UML has the concept of a profile. The OWL does not)
- OWL does not use an extra packaging construct to separate terminological knowledge from extensional knowledge.
- Sufficient conditions and defined vs. primitive concepts (The OWL implements this feature by applying two terminological axioms, distinguishing the defining statement from the defined expression, and using anonymous class descriptions. The UML does not provide native assistance in the definition of sufficient conditions or defined classes, since the UML is intended to define constraints that have to hold in a concrete version of a represented system).
- Meta classes (The UML does not support meta-classes directly).
- Associations (The UML does possess some representation-oriented constructs that are not directly available in OWL: n-ary associations, qualified associations, bidirectional associations, and association classes. Two UML constructs – aggregation and composition – are not translatable to OWL).

The different assumptions used with OWL and UML often lead to different interpretations of common language constructs. As a result, it is impossible to translate OWL ontologies into

UML models and vice versa without the loss or corruption of information (Kiko 2008).

The situation is difficult but far from hopeless. There are multiple approaches and attempts to solve this problem, assuming some constraints minimizing the listed difficulties (Viademonte 2010), (Zedlitz 2012a,b). As both the UML and the OWL are organized around the hierarchy of abstractions (meta models, domain models, concrete models) there are attempts to build the transformation at the meta level (Zedlitz 2012a,b, 213), i.e. transform a concrete model to its meta level, via meta levels to the other side, then to the concrete model. Such approach bears advantage of being particular representation independent (like Functional-Style Syntax, Turtle Syntax, OWL/XML Syntax, Manchester Syntax, etc. (OWL)), however semantic finesses, which disappear at the meta level are difficult to be ported.

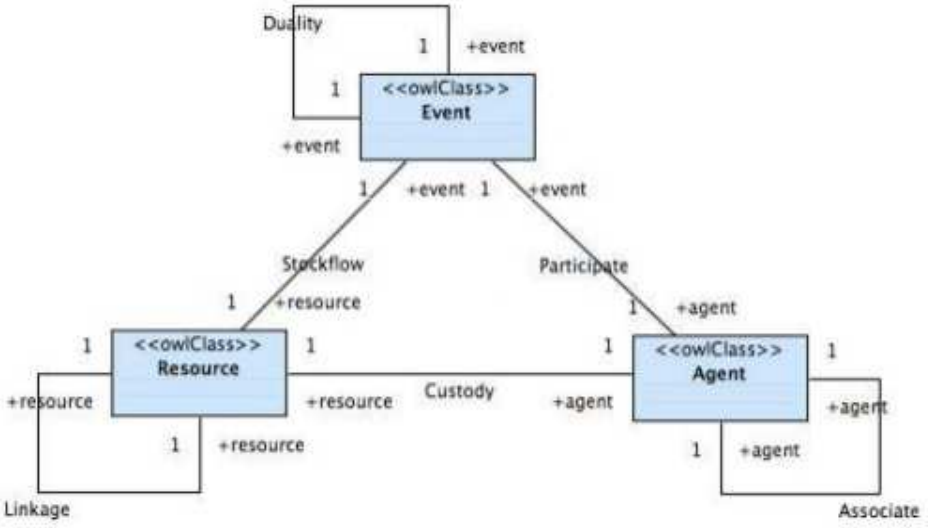

Fig. 7. The Ontology UML profile class-oriented stereotypes (from (Kuo 2011))

To be successful the authors made an interesting simplification. They restricted the UML class models. They were interested in the structural schema of an information system and did not have to take the behavioral schema into account.

The UML models can be focused on a particular domain by the mechanism of the profiles and stereotypes. In (Viademonte 2010) authors made a low level approach, designing UML-OWL transformation for a particular (enterprise model) profile. They built the correspondence between the needed UML and OWL concepts, but encountered difficulties with some of them. An additional reported difficulty was the dependence of the designed transformation on the specific UML modeling tools. It was necessary for them to hard code specific features, subject to specific software platform, XML encoding versions and namespaces as well. It was also necessary to take into account the discrepancies among distinct APIs for ontology processing and ontology editors.

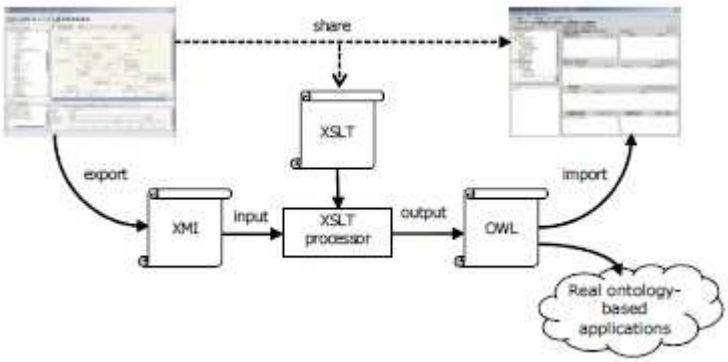

Fig. 8 XSLT principle: extensions of present UML tools for ontology development (from (Gasevic 2004), (Djuric 2005))

In (Gasevic 2004), (Djuric 2005)) (see also (Bridging-UML-and-OWL)) the authors built the Ontology Definition Metamodel (ODM) and the Ontology UML Profile (OUP) – a UML Profile that supports UML notation for ontology definition. They organized the transformation via the two-way mappings between OWL and ODM, ODM and OUP, and from OUP to other UML profiles. Their OUP used the standard UML extension and customization mechanisms, like stereotypes, tag definitions, tagged values, and constraints. Stereotypes enabled defining virtual subclasses of UML meta-classes, assigning them additional semantics. The authors expected that the designed OUP will:

- offer stereotypes and tags for all recurring ontology design elements, such as classes, individuals, properties, complements, unions, etc.;
- make specific ontology modeling and design elements easy to represent in UML diagrams;
- encapsulate ontological knowledge in an easy-to-read format;
- make possible evaluation of ontology UML diagrams e.g. for the possible inconsistencies;
- support ODM, hence be able to represent all ODM concepts.

The mapping from the OUP models to the OWL was realized via the XSLT with a set of rules (i.e. XSLT templates) matching XMI constructs and transforming them into equivalent OWL primitives. Some of the additional difficulties are reported in (Gasevic 2004).

An important add-on is the work of (Zedlitz 2013) on data formats, because the known approaches for UML-to-OWL (and reverse) transformations have neglected the problem of minute but important differences in data mapping.

Before we go further one must notice that nobody as yet attempted to transform from the ULM to the OWL models of the complexity adequate to describe the robotic systems.

What can we learn from the literature for the design of the user GUI? First of all that the transformation is possible, but it would be an errorr to be too general. The reported difficulties are substantial. The current approach (under detailed design) is of (Gasevic 2004) with the definition of a specialized OUP for the SysML schemes, and the careful choice of (Zedlitz 2012a,b) not to tackle all kinds of the SysML models (leave out the behavior), then to account for finesses and the difficulties in the properly tailored model transformation. The design tool will be a special purpose SysML(static)-to-OWL tool, not general, but well tailored to the needs of the configuration modeling.

8 Conclusions and open questions

The deliverable proposes a high-level architecture design for the decision support tool chain intended to help the robotic application designer in the configuration and verification of the robotic system solution. The scope of the application problems was drawn from the use case descriptions in the D11.10 and the abstraction level of the decision support is focused on the notion of robotic skills and system components realizing those skills. For this purpose Skill Model Knowledge Base was analyzed and developed in D13.11 and D13.12. The initial requirements regarding the tools were established in the D35.10 and further developed here. The scope of the configuration and verification problems is treated in the D13.20, and it is also where from the view of the user side modeling tools and user-system interactions came.

The system is to be implemented in the T35.4 and T35.5, and will be reported in the D35.40. The expected test environment for the tool-chain will be a robotic system configuration sub-problem dealing with various configurations and implementations of object handling (object detection, object recognition) capability. Such sub-problem lies at the heart of every robotic problem, where the robot must interact with the object rich environment, and can be realized with a variety of sensors and information processing algorithms. For the testing purposes the required knowledge base elements and component database items will be introduced manually into the tool-chain.

The tool-chain design brought into light also a number of difficult issues and open questions influencing the development and the implementation of the decision support tool-chain:

- The planned system is in many aspects experimental and pioneering. In the field of robotics there are no (known) similar developments on the decision support for an automated or semi-automated configuration and re-configuration.
- Many methodological aspects are new, with no experience for problems and solutions. Good trade-offs are unknown and perhaps one of the most important implicit aims of the present development is to gain enough experience.
- As mentioned before, ontologies evolve, and especially the ORA ontology evolves. Considering that the ORA ontology is in the core of the designed system, it is to be expected that the system will be (has to be) regularly up-dated, modified, and changed.
- Not everything (interaction) is equally well algorithmizable, not every conceivable algorithm is equally efficient. The logical reasoning is knowingly hard in time and memory (Horn clause rules fare better), and there is the exciting and key question of the model transformations.
- Even the partial solution to the SysML-to-OWL transformation problem will be one of the most important results obtained during the support tool design.
- It must be noted that for a moment free-hand in handling skills (names, definitions, relation to other skills, relation to the robotic system components) is actually possible because these notions are not yet regulated by any conceptual standard. However if such standard will appear, the system (knowledge base, reasoning, model transformations) must be corrected and adjusted, consequently the system must be re-designed.
- Lastly it must be also mentioned, that although the proposed architecture covers with its components the required functionality of the tool-chain, particular components can

be designed at varying level of complexity, leading to different versions of the tool-chain. For example there could be a version without the Skill Description Editor components, or without the option for the transfer of data from the external data bases. The spectrum of the file formats in the storage can also be limited, etc.

9 References

(Baader 2005) F. Baader, I. Horrocks and U. Sattler, **Description Logics as Ontology Languages for the Semantic Web**, Mechanizing Mathematical Reasoning, LNCS, pp. 228- 248, Springer Berlin / Heidelberg.

(CLOnE) A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Handschuh, **CLOnE: Controlled Language for Ontology Editing**, The Semantic Web, vol. 4825, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., Springer, 2007, pp. 142–155.

(Djuric 2005) D. O. Djuric, V. B. Devedzic, **Bridging MDA and OWL Ontologies**, J. of Web Engineering, Vol. 4, No.2 (2005) 118-143

(Bridging-UML-and-OWL) **ATL Use Case - ODM Implementation (Bridging UML and OWL)**, http://www.eclipse.org/atl/usecases/ODMImplementation/

(Emerald) **Emerald expert engine, regulation modelling and management,** http://web.multilogic.hu/emerald/, http://www.multilogic.hu/images/download/Emerald_Flyer_2_0_En.pdf

(FaCT++) **FaCT++**, http://owl.man.ac.uk/factplusplus/

(Gasevic 2004) Gasevic, D., Devedsic, V. and Damjanovic V., **Converting UML to OWL Ontologies**, Proc. of the 13th Int. World Wide Web Conf. on Alternate Track Papers & Posters, pp. 488-489, ACM, May 2004.

(HermiT) **Hermit OWL Reasoner**, http://hermit-reasoner.com/

(Horrocks 1999) I. Horrocks, U. Sattler, and S. Tobies, **Practical Reasoning for Expressive Description Logics,** In H. Ganzinger, D. McAllester, and A. Voronkov, eds, Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99), LNAI 170, pp. 161-180, Springer, 1999.

(Horrocks 2005) I. Horrocks, P. F. Patel-schneider, S. Bechhofer, and D. Tsarkov. **Owl rules: A proposal and prototype implementation**. J. of Web Semantics, 3:23–40, 2005.

(IEEE1872, 2015) **IEEE Std 1872™-2015, IEEE Standard for Ontologies for Robotics and Automation,** IEEE Robotics and Automation Society, Feb 2015, IEEE-SA Standards Board

(Jess-Protégé) **SWRLJessTab Protégé plug-in**, http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab

(Jess) **Jess, the Java Rule Engine**, http://herzberg.ca.sandia.gov/

(Kiko 2008) Kiko K., and C. Atkinson, **A Detailed Comparison of UML and OWL**. Technischer Bericht 4, Dep. for Mathematics and C.S., Univ. of Mannheim, 2008.

(Krotzsch 2011) M. Krötzsch, F. Maier, A. Krisnadhi, and P. Hitzler. **A better uncle for owl: nominal schemas for integrating rules and ontologies**. Proc. of the 20th Int. World Wide Web Conf. (WWW '11), pp. 645–654, New York, 2011.

(Kuo 2011) R. Kuo, **UML, OWL and REA-based Enterprise Business Model**, http://www.slideshare.net/rkuo/uml-owl-and-reaowl-andrea-basedenterprisebusinessmodel20110201a

(OWL) **OWL 2 Web Ontology Language Document Overview** (2nd Ed), W3C World Wide Web Consortium Recommendation 11 Dec 2012, (http://www.w3.org/TR/owl2-overview/)

(Pellet) **Pellet: The Open Source OWL Reasoner**, http://clarkparsia.com/pellet

(Protégé) **Protégé**, http://protegewiki.stanford.edu/wiki/Main_Page

(R5-COP D13.11) R5-COP D13.11 **Skill Model Knowledge Base** (tentative)

(R5-COP D13.12) R5-COP D13.12 **Skill Model Knowledge Base** (final)

(R5-COP D35.10) D35.10: **Requirements**

(Rabbit) G. Hart, M. Johnson, and C. Dolbear, **Rabbit: Developing a Control Natural Language for Authoring Ontologies**, The Semantic Web: Research and Applications, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Springer, 2008, pp. 348–360.

(Rule Markup) **Rule Markup Language**, http://wiki.ruleml.org/index.php/RuleML_Home

(Schlenoff 2012) C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer and E. Miguelanez, **An IEEE Standard Ontology for Robotics and Automation,** 2012 IEEE/RSJ Int. Conf. on Intell. Robots and Systems, Oct 7-12, 2012. Vilamoura, Algarve

(SOS) A. Cregan, R. Schwitter, and T. Meyer, **Sydney OWL syntax—towards a controlled natural language syntax for OWL 1.1**, Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, 6-7, June 2007, CEUR-WS, Vol. 258, 2007.

(SWRL) **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**, http://www.w3.org/Submission/SWRL/

(SysML) Systems Modeling Language, http://sysml.org/

(Viademonte 2010) S. Viademonte, Zhan Cui, **Deriving OWL Ontologies from UML Models: an Enterprise Modelling Approach**, British Telecom, GCTO, 2010.

(W3C-XSD) **W3C XML Schema Definition Language (XSD)** 1.1 Part 2: Datatypes. W3C Recommendation, 5 April 2012, https://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/

(Zedlitz 2012a) J. Zedlitz and N. Luttenberger, **Transforming Between UML Conceptual Models And OWL2 Ontologies**, Terra Cognita 2012 Workshop, Vol 6.

(Zedlitz 2012b) J. Zedlitz, J. Jörke, and N. Luttenberger, **From UML to OWL2**, Knowledge Technology, 2012, pp. 154-163

(Zedlitz 2013) J. Zedlitz, N. Luttenberger, **Data Types in UML and OWL-2**, SEMAPRO 2013: The 7th Int. Conf. on Advances in Semantic Processing