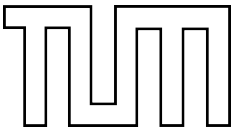


Université Paul Sabatier

Institut National des Sciences Appliquées

Technische Universität München

Lehrstuhl für Bildverstehen und wissensbasierte Systeme

Doppelpromotion / Joint Doctorate

Grounding the Interaction: Knowledge Management for Interactive Robots

Séverin Lemaignan

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Malik Ghallab
Prüfer der Dissertation:	1. Prof. Rachid Alami 2. Prof. Michael Beetz 3. Prof.-Dr. Joachim Hertzberg 4. Prof. Alessandro Saffiotti 5. Prof. Mohamed Chetouani 6. Prof. Dongheui Lee

Grounding the Interaction: Knowledge Management for Interactive Robots

Séverin LEMAIGNAN

September 3, 2012

There are more things in heaven and earth, Horatio,
Than are dreamt of in your philosophy.

Shakespeare, *Hamlet*, Act 1, scene 5, 159–167

Contents

One thesis, two places,
Three robots and four years,
Rachid and Michael sailing forward
On the merry-go-rounds of robotics.
Bears and bards, all the troupe of PhD students,
A wife and a daughter, a life and some science,
The rose city and all these friends
Who tile the world with colours,

To all of you, thank you!

Abstract

With the rise of the so-called *cognitive robotics*, the need of advanced tools to store, manipulate, reason about the knowledge acquired by the robot has been made clear. But storing and manipulating knowledge requires first to understand what the knowledge *itself* means to the robot and how to represent it in a machine-processable way.

This work strives first at providing a systematic study of the knowledge requirements of modern robotic applications in the context of service robotics and human-robot interaction. What are the expressiveness requirement for a robot? What are its needs in term of reasoning techniques? What are the requirement on the robot's knowledge processing structure induced by other cognitive functions like perception or decision making? We propose a novel typology of desirable features for *knowledge representation systems* supported by an extensive review of existing tools in our community.

In a second part, the thesis presents in depth a particular instantiation of a knowledge representation and manipulation system called *ORO*, that has been designed and implemented during the preparation of the thesis. We elaborate on the inner working of this system, as well as its integration into several complete robot control stacks. A particular focus is given to the modelling of agent-dependent symbolic perspectives and their relations to theories of mind.

The third part of the study is focused on the presentation of one important application of knowledge representation systems in the human-robot interaction context: situated dialogue. Our approach and associated algorithms leading to the interactive grounding of unconstrained verbal communication are presented, followed by several experiments that have taken place both at the *Laboratoire d'Analyse et d'Architecture des Systèmes* at CNRS, Toulouse and at the *Intelligent Autonomous System* group at Munich Technical University.

The thesis concludes on considerations regarding the viability and importance of an explicit management of the agent's knowledge, along with a reflection on the missing bricks in our research community on the way towards *"human level robots"*.

Keywords: Cognitive Robotics, Knowledge Representation and Reasoning, Human-Robot Interaction, Ontologies, Natural Language Processing

Résumé

Ancrer l'interaction: Gestion des connaissances pour la robotique interactive

Avec le développement de la *robotique cognitive*, le besoin d'outils avancés pour représenter, manipuler, raisonner sur les connaissances acquises par un robot a clairement été mis en avant. Mais stocker et manipuler des connaissances requiert tout d'abord d'éclaircir ce que l'on nomme *connaissance* pour un robot, et comment celle-ci peut-elle être représentée de manière intelligible pour une machine.

Ce travail s'efforce dans un premier temps d'identifier de manière systématique les besoins en terme de représentation de connaissance des applications robotiques modernes, dans le contexte spécifique de la robotique de service et des interactions homme-robot. Nous proposons une typologie originale des caractéristiques souhaitables des systèmes de représentation des connaissances, appuyée sur un état de l'art détaillé des outils existants dans notre communauté.

Dans un second temps, nous présentons en profondeur ORO, une instanciation particulière d'un système de représentation et manipulation des connaissances, conçu et implémenté durant la préparation de cette thèse. Nous détaillons le fonctionnement interne du système, ainsi que son intégration dans plusieurs architectures robotiques complètes. Un éclairage particulier est donné sur la modélisation de la prise de perspective dans le contexte de l'interaction, et de son interprétation en terme de théorie de l'esprit.

La troisième partie de l'étude porte sur une application importante des systèmes de représentation des connaissances dans ce contexte de l'interaction homme-robot : le traitement du dialogue situé. Notre approche et les algorithmes qui amènent à l'ancrage interactif de la communication verbale non contrainte sont présentés, suivis de plusieurs expériences menées au *Laboratoire d'Analyse et d'Architecture des Systèmes* au CNRS à Toulouse, et au groupe *Intelligent Autonomous System* de l'université technique de Munich.

Nous concluons cette thèse sur un certain nombre de considérations sur la viabilité et l'importance d'une gestion explicite des connaissances des agents, ainsi que par une réflexion sur les éléments encore manquant pour réaliser le programme d'une robotique *"de niveau humain"*.

Zusammenfassung

Verankerung der Interaktion: Wissensmanagement für interaktive Roboter

Mit dem Aufstieg der sogenannten kognitiven Robotik ist der Bedarf an mächtigeren Werkzeugen gestiegen, um das Wissen vom Roboter zu speichern und weiter zu verarbeiten. Diese Arbeit stellt zuerst eine Studie über die Anforderungen an solche Werkzeuge vor und schlägt eine neuartige Typologie von wünschenswerten Eigenschaften für Wissensrepräsentations-Systeme vor.

Wir führen dann ein solches System namens ORO ein. Wir zeigen seine innere Arbeitsweise sowie seine Integration in verschiedene Roboter-Architekturen. Ein besonderer Fokus liegt auf Agenten Perspektiven und ihre Beziehungen zur *Theory of Mind*.

Der dritte Teil der Studie stellt eine Komponente zur Verarbeitung von Dialogen vor, die die interaktive Verankerung der freien verbalen Kommunikation ermöglicht. Wir schließen mit mehreren Experiment-Berichten und einer Diskussion über die fehlenden Bausteine auf dem Weg zum *"human level"* Roboter.

Conventions and Notations

This thesis relies on several notations and specific writing conventions to describe symbolic knowledge and logical relations.

Ontologies and excerpts of ontologies presented in the work are mostly written in the W3C's OWL language. As a derivative of XML, it uses namespaces to declare the scopes of concepts. The main namespaces that are used in this work are `owl:`, `rdf:`, `rdfs:` (respective namespaces and schema namespace of the Web Ontology Language and the Resource Description Framework), `cyc:` (concepts defined in the OPENCYC upper ontology) and `oro:` (concepts defined in our *OpenRobots Common-Sense* ontology). For readability, the namespaces will be omitted when they are not required for the understanding.

The table below summarises the terminology that we use in this work to discuss knowledge representation questions. While these terms are generally not strictly synonyms, we will use them interchangeabily when no confusion may arise.

Entity, Element, Concept	Class (OWL), Concept (DL)	Relation, Property (OWL), Role (DL), (Binary) Predicate	Instance (OWL), Individual (DL)

Description Logic terminology (noted DL above) for classes (*i.e. concept*) and relations (*i.e. role*) will be used only in the specific context of Description Logic. In other cases, we use the term *concept* as a general term that encompasses *classes*, *properties* and *instances* in the OWL terminology.

Depending on the context, a logical *statement* is either a declarative sentence or the meaning of this sentence (in this case, it is a *fact* or a *belief*). Statements are generally represented as *triples* $\langle subject, predicate, object \rangle$. Statements that are explicitely added to a knowledge base are called *assertions*.

Again, we may use interchangeabily the terms *statement*, *assertion*, *fact*, *belief*, *triple* when no confusion arise.

Single concepts are typeset with this font: `concept`, while statements are typeset in this way (when represented as triples): ⟨`subject` *`predicate`* `object`⟩.

Relations between concepts and rules rely on logical connectors. The table below presents the most important ones that are found in this thesis.

$\models$	"models"
$\sqcap$	intersection or conjunction of classes
$\sqcup$	union or disjunction of classes
$\forall$	universal restriction
$\exists$	existential restriction
$\equiv$	class equivalence
$\emptyset$	empty set
$\wedge$	logical AND
$\vee$	logical OR
$\rightarrow$	implication

Finally, we punctually use the Manchester Syntax[1] to present in a readable way complex class expressions.

[1] http://www.w3.org/TR/owl2-manchester-syntax/

Chapter 1

Introduction: Robots, Interaction and Knowledge

Nao has been seen playing with autistic children, Justin is able to gently tap on the chocolate powder dispenser to prepare a hot chocolate, the PR2 robot(s) are taking orders and bring beers and popcorn around the labs, while Rosie pours pancake dough on a heater for the afternoon snack: if recent experiments conducted in the labs around the world are any indication, service robots are leaving the realm of Sci-Fi, dreams and phantasms to become a reality.

Nao, Justin, PR2, Rosie: robots play with children, prepare hot chocolates, serve fresh beers or make pancakes. Still in the labs, not yet with complex interactions with humans. What is missing for them to enter our homes?

From technological demos, these robots are now moving to real-world coworkers and companions, and they are undoubtedly to knock at our doors in the coming years.

The perceptual layers have moved up from traditional sensing modalities (camera images, laser scans) to synthetic pseudo-sensors like face recognition, SLAM-based localisation, or the Kinect-based human tracker.

Perceiving and understanding the environment is now a matter of rebuilding an

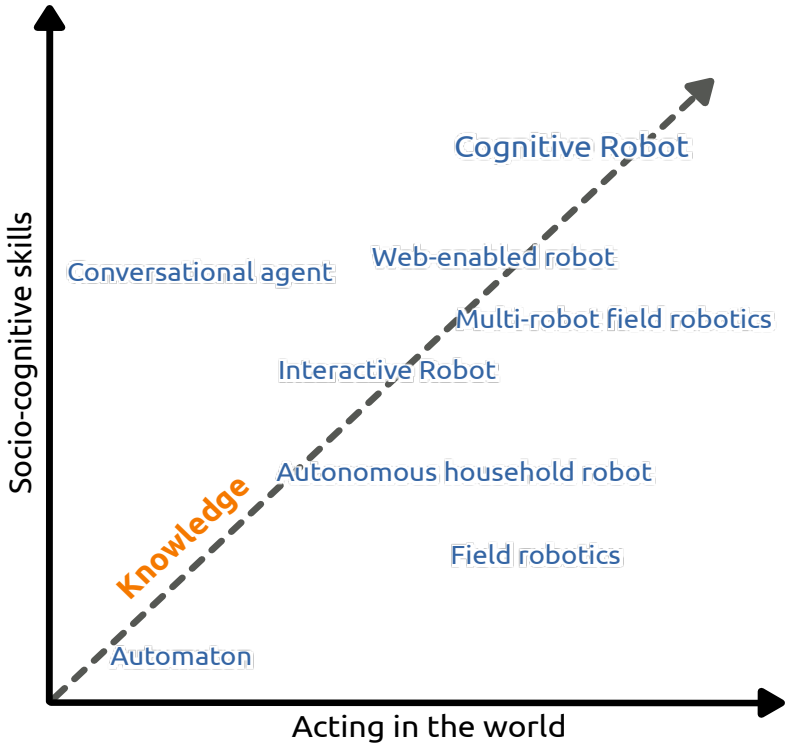


Figure 1.1: Towards the cognitive robot

internal, amodal, model of the environment, with two interleaved facets: a continuous, geometric world on one hand and a discrete, symbolic world on the other hand. This, by itself, is sufficient to build efficient, compliant, reactive manipulators.

But perceiving an inanimate environment is not enough for a companion robot: such a robot does not live in isolation in a world that would have been tailored to its capabilities. It lives in the real world, in interaction with other intelligent agents, and we want it to be endowed with social skills. It needs also to become aware of the human, as a physical entity of course, but also as a *mindful* entity. This implies that the robot is not only able to represent inanimate objects, not only able to represent its own mental state, but also able to guess and represent mental states of other agents, other intelligences. And interaction requires more, like communication skills and social capabilities: agency, perspective taking, theory of mind, not to mention the endless piles of information available from our connected world.

Figure 1.1 tries to relate socio-cognitive skills and acting skills of various domains of robotic research. Our focus in on the cognitive robot, and in this work, we postulate that further development of such robots relies on the acquisition of even better, richer, more diverse cognitive abilities, and that the key of this development lies in a better understanding of what *knowledge* means to a robot, and how to represent it. The What, and the How of knowledge for robots.

We propose to give life to these two questions by narrating a short story of two robots and a human...

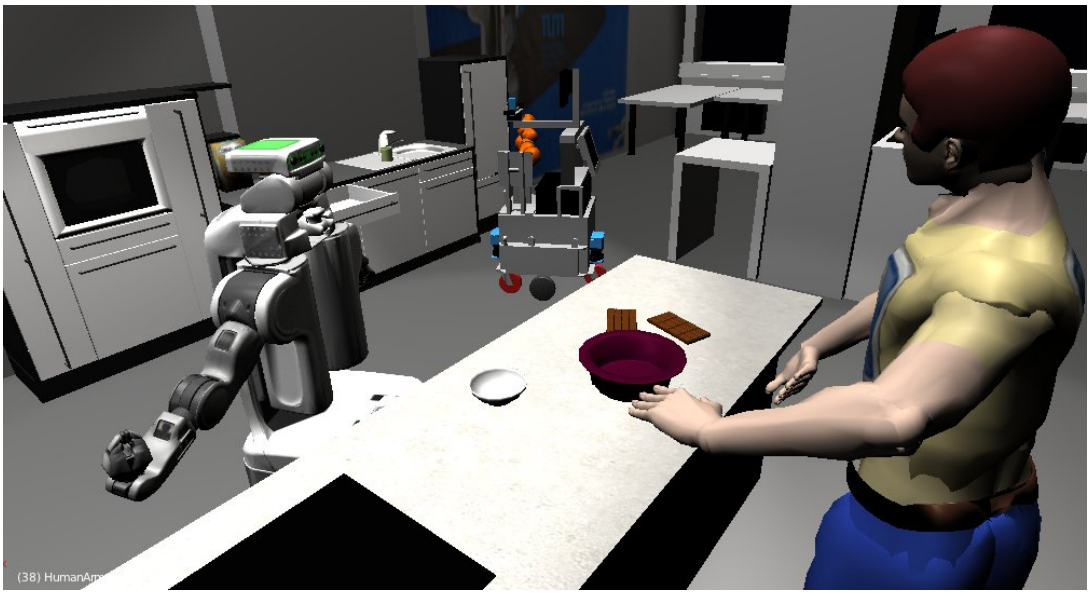


Figure 1.2: The *Brownie* scenario

1.1 A prototypical scenario

The aim of this imaginary scenario (that has not been implemented, neither in simulation nor on a robot) is to materialise early in this thesis the context and challenges of knowledge representation and manipulation for service, social and interactive robotics. It underlines the place, the role and the need of knowledge in a near-future, everyday situation where several robots and humans co-exist and cooperate. This scenario will also be a source of support examples for later sections of the thesis. Also, we have emphasised several keywords in this description: we will come back to them in chapter 2 to explain them formally and relate them to each other.

We entitle our scenario "the Brownie scenario" (Figure 1.2): Robi and Roba are two service robots, that can freely move and pick objects around (with possibly different hardware and software architectures, including different knowledge representation systems: typically, two robots built by two different companies). They cooperate with a human in a kitchen environment.

The main task of the scenario is the joint realization of a brownie, initiated by Tom, the human: "Let's make a brownie for tonight!".

The scenario is successful *1)* if the task is achieved (the brownie is baked), *2)* in a reasonable time (typically shorter than what it would have been required by the human alone), *3)* and the human is happy by the help it received (he/she did not feel useless, uncomfortable, unsafe).

We voluntarily do not detail the subtasks of the scenario, neither we define how they are shared amongst agents: our focus is on knowledge needs and flows.

A "first-order" analysis of this task leads to a rough partition of the required *representation* abilities:

1. Representation abilities related to the execution of a complex spatio-temporal task,

2. Representation abilities related to cooperation with other agents.

We can further refine these categories: to prepare and bake a brownie, the robot first needs to make sense of the term *brownie* itself: what is it? what is it used for? what is it made of? etc. We call this knowledge *common-sense knowledge* and the robot must be able not only to represent it, but also to have access to an initial source (for instance through a initial set of facts that are made available at startup, or via access to a Web-based knowledge base like Wikipedia)

Once bound to the action *make*, this should lead the robot to build and represent a *context*: we are in a scenario involving cooking. The context enables the robot to retrieve more common-sense knowledge, like that actions related to cooking often take place in the kitchen, cooking requires ingredients, utensils and a procedure that may be provided by a recipe.

These last assertions imply several other capabilities: "cooking often takes place in the kitchen" implies that representation of both uncertainty and likelihood is desirable. The fact that cooking is associated to a place further implies that the system models locations and is able to attach *thematic relations* to concepts (here, the likely location of the cooking action).

"cooking requires a procedure and ingredients" hints about another important feature closely tied on knowledge manipulation: *reasoning*. The robot can *infer* that cooking may require a recipe since a procedure and a list of ingredients are pre-requisites of the cooking action that may be provided by a recipe. If we omit the "may", this is a typical example of first-order logic reasoning. Many other reasoning techniques exist (including probabilistic ones – ones able to deal with the "may"), we shall illustrate some of them later in this scenario.

We mentioned that a recipe often provides a procedure (or a *plan*). The robot should be able to store this plan in a way that allows later execution. The plan is likely to contain *spatio-temporal constraints* (like "put the brownie in the oven for 20 min" or "let's cook *for tonight*") that must be as well appropriately handled.

To make decision, a robot may also want to *predict* the state of the world after some action ("if I leave the cake 2h in the oven, it will burn"). Such ability to project itself in future or, generally speaking, in other possible state of the world is related to several cognitive ability and reasoning techniques: *planning*, *projection*, *representation of possible worlds* and *non-monotonic reasoning*, in addition to common-sense knowledge and *physics-based* reasoning (that allows for instance to predict that an egg is likely to break if dropped).

Procedures are in addition often *underspecified*: we can expect the recipe to provide a cooking duration, but we usually do not expect the recipe to tell us to first open the oven door, and then put the cake into it, since it is self-evident that the door must first be opened to put the cake in the oven. Our cognitive robot should ideally be able to detect and possibly complete such underspecification.

Then, we want our three agents to cooperate. This, in turn, leads to another set of cognitive abilities.

Cooperation in our scenario can intervene at many places. For instance, an agent may want to inform another one about the number of eggs that are necessary for the brownie. This *helping* behaviour makes sense only if the first agent knows that the

recipient agent both needs the information but does not know it. This in turn requires the robot to be able to model the knowledge of the other agents: to think *from the perspective* of another agent (an idea that is related to the availability of a theory of mind, we will come back to it later on).

Ability to communicate is one important pre-requisite to collaboration. Communication in general requires the addresser and the addressee to share a common interpretative framework (a shared common-sense knowledge – or cultural background – and a shared context). In our scenario, the agents are working in a kitchen. This element of context does not however suffice if, for example,the human asks a robot to "give [him] the bowl". Behind the symbol "bowl", which physical entity are we actually talking about? If we want to talk and act on the world, this so-called *grounding* operation is essential. It is a bidirectional process: in covers the *top-down* operation (from the symbol to the percept) and the *bottom-up* converse (retrieval or creation of symbols from perception).

A related ability is called *pre-supposition accommodation*: if one of the agent moves behind another one, with the brownie dough in its arm, and says "be careful, I'm behind you!", we want the first agent to be able to represent both symbolically and geometrically (because, for instance, if the agent want to move, it must take into account the new obstacle) something that is not directly perceived.

Also central to cooperation are the notions of *joint intentions* and *joint goals*: to help the human during the cooking session, the robots need to track how far they are into the recipe, what is the next step the human is likely to go for, how tasks are currently split between agents, what action is currently blocking the procedure, etc. This knowledge should let the robot identify the intentions of other agents and create accordingly joint goals. Hence, a knowledge representation system aiming at dealing with cooperative behaviours is likely to have goal management structures taking explicitly into account other agents' actions and goals.

In order to effectively share tasks, the robot must also know what it is capable of: *capability introspection* (both in term of general capability and of immediate ability) is thus often desirable. It can be extended to general introspection (like the ability to tell "who I am" or "what do I think of") that may be required for the interaction.

Last but not least, our scenario assumes implicitly *natural interaction* between humans and robots (as shown by the casual style of the order "Let's make a brownie!"), and we want to ensure that the knowledge available to the robot provides efficient support to the natural language understanding (for instance by adopting models and vocabulary that are both well suited for machine processing and remain as close as possible to the humans own structures and vocabulary), and also to *non-verbal forms of communication*, like gestures.

Before explaining with more details in the next chapter the keywords we have emphasised in the scenario, we would like to briefly focus on the challenges specifically related to the human-robot interactions. Not only in term of knowledge representation, but more broadly in term of specific cognitive capabilities.

Figure 1.3: Interacting with the robot in an everyday situation: the human asks for help in vague terms, the robot takes into account the human's *a priori* knowledge and spatial perspective to refine its understanding of the question.

1.2 Robots for interaction

This work comes indeed from researches in the specific context of the human-robot interaction, or, to put it another way, in the context of interaction for *joint action* with humans, in a *situated* environment (figure 1.3).

"Let's bake a brownie for tonight!", *proposes Tom. The robots smoothly prepare all the ingredients, and they start to cook together a delicious cake...*

Natural interaction and cooperation are actually the current (dare we say, *short-term*) targets for the human-robot interaction community. The "Brownie scenario" we presented above belongs to the broad class of *interactive manipulation problems*: several agents agree on a (more or less implicit) joint goal that requires some sort of cooperation to be successfully achieved. This class of problems involves both dialogue and manipulation and they are often not completely defined at start-up: they require iterative, interactive resolution (step-by-step process, questions-answers,...).

What are the cognitive prerequisites for such a sentence –"Let's make a brownie for tonight"– to be understood by the robot, correctly interpreted in the spatial and temporal context of the interaction, and eventually transformed into a set of actions? We distinguished four main tasks in [74]:

1. building and maintenance of a consistent geometric model of the current situation, acquired through perception or deduction from previous perceptions,

2. building of an unambiguous and complete symbolic representation of concepts

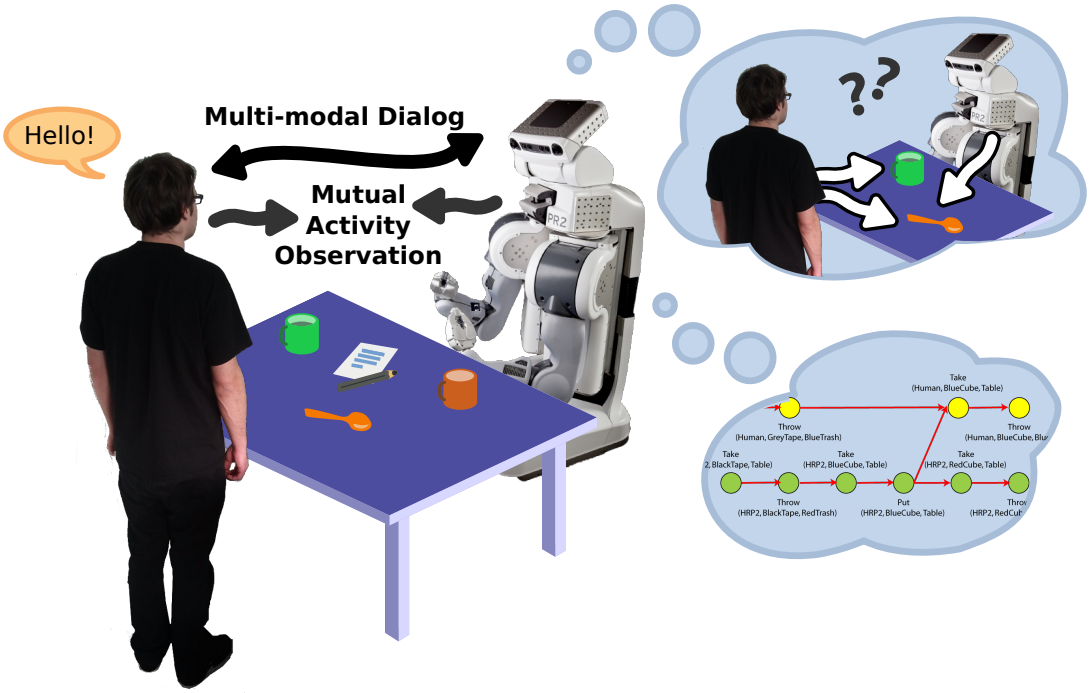


Figure 1.4: A robot reasoning about human-robot interaction and anticipation of human activities: sources of knowledge are multi-modal dialogue and observation of the environment and the human activities. The robot "knows" and reasons about the fact it is observed by the human.

(objects, agents, actions...) underlying the interaction, and practical for decision-making processes,

3. establishing the joint goal(s), building and maintenance of iteratively shared (human-robot) plans,

4. refinement and execution of the computed plans, and monitoring of those achieved by the human partner.

While each of these items is equally important to actually perform the interaction – and we will present (with illustration from experiments) how our knowledge representation system integrates and communicates with other processes to form a *knowledge-enabled* robotic architecture –, the thesis focuses on the second point: it presents techniques, developed and used on several real robots, for the symbolic representation of environment and mental models suitable for grounded situation interpretation, decision-making and control.

Models for the interaction Figure 1.4 summarises the main aspects of the interaction, that are to be translated into models. From the robot perspective, several cognitive skills are involved: dialogue processing through verbal and deictic modalities (what does the human say? What attitude – gaze orientation, postures, gestures... – does he express?), acquisition and maintenance of one or several models of the environment, not only from the robot point of view, but also from the other agents' points of view, anticipation (what are the intentions of the human? Can I predict and anticipate his/her actions?),

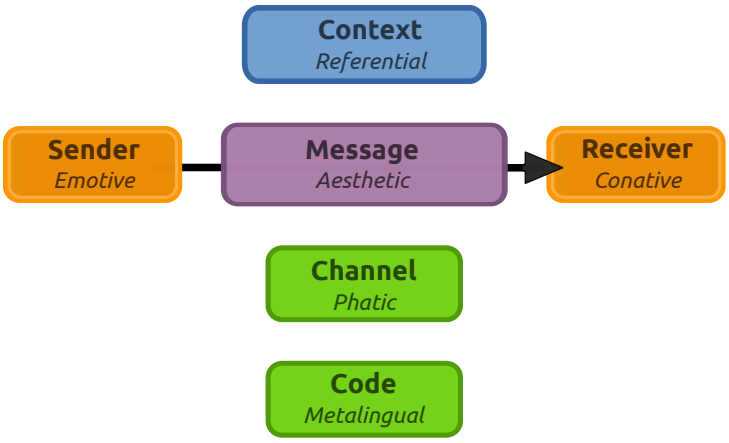


Figure 1.5: The *Communication Model*, as proposed by Jakobson [56]. In bold characters are the *communication dimensions*, in italics, the corresponding *communication functions*.

planning and control (how would I proceed further towards the goal?), monitoring of the other agents' activities (do we have an effective cooperation?) and the overall progress of the task.

As we shall see, all these cognitive capabilities also translate into requirements on the knowledge representation systems that we want to clarify.

The need of communication is probably the most salient one. The classical model of communication proposed by Jakobson in 1960 (figure 1.5) exposes in a bright way the main functions involved in a communication, be it verbal or non-verbal. While the *channel* and the *code* are the technical side of the communication, the *message* in relation with the *context* are directly concerned with the question of the *meaning*. And the meaning is itself tightly bound to the knowledge available to the agent.

The question of the communication between a robot and another agent (*agent* in a broad sense: another robot, a human, but also a remote knowledge base or the robot's developer) actually underlies many of the challenges of knowledge representation: how to represent the knowledge I want to exchange, and how to recognise, represent and share a context that ensures that both ends of the communication channel correctly interpret the message. Or, to put it another way: how to ensure that the *meaning* is correctly conveyed around while conducting social interactions?

1.3 The challenges

From the set of questions raised in the previous paragraphs, we can now articulate the challenges that are to be tackled in this field of knowledge representation for service or companion robotics.

The first challenge is to... clarify the challenges (!) of knowledge representation: we said "knowledge", we said "reasoning", we said "representation", but clear definitions are yet to be provided. Numerous requirements on what Newell calls the *knowledge level* of intelligent agents have emerged from our *Brownie* scenario, but how do they articulate? Are they comprehensive?

To allow further progresses in the field of cognitive robotics, we think it is mandatory

to lay down solid theoretical and practical foundations to the knowledge needs of service and interactive robots. This is our first challenge.

The second challenge is more technical: how to build such a "knowledge-enabled" robot? Since many years, researchers create and study so-called cognitive architectures, implemented as abstract, computer-based, virtual models of the human cognition. Robots, as embodied and interactive agents, raise specific issues. What are those? Which are the right technical approaches to tackle them? Can we build today at least an instance of such a cognitive system, and if we can not, why that? How the abstract idea of knowledge translates into practical, meaningful concepts?

Our third challenge relates to the specific question of the human-robot interaction. We claim that robots now belong to the realm of social individualities. What does that mean? Which consequences does that have on our initial knowledge challenge? How does it translate into practical issues, like natural language understanding?

All the contributions (summarised in the next section) of this thesis can be related to one of these three challenges, and we hope they contribute to the progress towards the understanding of these questions.

1.4 Contributions

We have presented our challenges: this section now summarises the main contributions of the thesis, both from a scientific point of view and from a technical point of view.

1.4.1 Scientific contributions

The starting point of our thesis is the feeling that a better understanding of the knowledge needs of robotic applications in human, *i.e.* complex, dynamic, semantically-rich, environments, would be beneficial to the research in cognitive robotic.

Building upon an extensive review of the literature and the formulation of several interaction scenarii (that themselves led to experiments on real robots), we have iteratively refined the "knowledge for interaction" problem. The formalization of this question is one of the main scientific outcomes of this work: we have listed and organised into a typology a set of desirable characteristics of knowledge representation systems for service robotics (chapter 2).

This typology aims at offering a comprehensive and consistent base to evaluate existing systems and to draw new research perspectives. It also enables to better assess the progresses of the Service Robot and Human Robot Interaction research communities towards the long term goal of *human-level artificial intelligence* for robots, as would say McCarthy.

Another scientific contribution of this thesis is its participation to narrow down the gap between research on embodied and disembodied artificial agents: we have tried to bridge experiences learned from years of research on disembodied cognitive architectures (both from the computing science and neuropsychology communities) with the constraints from real-world systems that weigh on robotic architectures. Notably, we have tried to identify theoretical reference contributions from the diverse

fields of cognitive sciences that are relevant to *knowledge-enabled* robotics. We have also proposed reference implementations on robots for some of them.

At the architectural level, our work also helps to better understand the knowledge flows in modern cognitive architectures for robots. By introducing *explicit* knowledge in our architectures, it allows the humans that design and program robots to *talk about* and question this knowledge: it singularises and materialises concepts that were beforehand often diffuse and ubiquitous. This leads us to define the idea and propose an implementation of a *knowledge-oriented* architecture (section 7.2).

This work has also several more focused scientific contributions. The centralised semantic architecture that we propose is original. While it exhibits shortcomings for some cognitive tasks, it also proposes novel efficient ways to represent and manipulate knowledge simultaneously for multiple agents (chapter 3). Along with the survey of current knowledge systems that we have conducted, it effectively completes the picture of available designs of knowledge representation systems.

Amongst the cognitive abilities that our developments have enabled, a particular scientific focus was led on the acquisition and modeling of multiple, agent-dependent symbolic worlds. This opened new perspectives related to *perspective-aware* reasoning or *theories of mind* for robots that are detailed in this work.

We also have a scientific contribution on the grounding of human-robot dialogue in natural language (chapter 5). We have algorithmically formalised a grounding process that takes advantage of multi-modal communication (verbal, deictic and immanent) and handles the semantics of several more complex language features like quantification. This system also has contributions related to the semantic validation of thematic roles and interactive disambiguation that takes into account human attentional focus.

1.4.2 Technical contributions

This thesis has four major technical contributions: the software development of the *ORO server* as a semantic blackboard dedicated to robotic applications, the design of the *ORO ontology* as a domain-specific common-sense ontology tailored for service robotic needs, the pervasive integration of a new semantic layer into several existing robot architecture, and finally, the software development of *Dialogs*, a module for natural language grounding.

The main software contribution of the thesis is the development of an open-source, versatile and light-weight knowledge base that stores in a formal framework based on first-order logics both the robot's own beliefs and the mental models (as perceived by the robot) of every other cognitive agent that the robot interacts with (chapters 3 and 4). This tool, called *ORO*, is implemented as a platform/middleware-agnostic server, and exposes to the robot's modules several advanced reasoning services (via the integration of external reasoners). This software project is now publicly available, used by other laboratories, and comes with extensive documentation and bindings for several mainstream languages (C++, Python...) and middleware (ROS, YARP).

In parallel of this development, and in collaboration with other developers, we have also drafted (and partially implemented) a proposal for a standard API for knowledge manipulation that supports the specific needs of robotic applications (section 2.4).

Coming along with the ORO server, we introduce in this thesis the *ORO common-sense ontology* which is a proposal of an upper ontology for service and interactive robotics (section 3.3). This ontology consists of about two hundred classes, relations and rules that are relevant for the modeling of the robot's beliefs and state, and the interactions with other agents (humans or robots). This ontology also tries to stay closely aligned with the standard OPENCYC upper-ontology to guarantee interoperability with semantic web resources and other robots.

A third technical contribution is the introduction of a new knowledge-oriented, event-driven communication model between high-level decisional modules (section 4.4): by introducing the notion of *semantic events*, the ORO server enables the development of new executive layers that combine reactive behaviour with high-level abstractions: for instance, triggering a behaviour when a human looks at the robot while sit, can be expressed in our architecture as a single proposition: `subscribe([* type Human, * looksAt myself, * isSitting true], behaviour_callback()).` This highly expressive event model opens a new range of development opportunities for decisional modules.

During the preparation of the thesis, we have also developed a new stand-alone natural language processor for English language (chapter 5). It takes advantage of the different symbolic models exposed by the ORO server to analyse, resolve the semantics and ground dialogues. It can process orders, questions and positive assertions and translates them into new symbolic facts. It includes a custom grammatical parser, a reverbalization module, several discrimination strategies, including interactive ones. The application is developed in Python (about 15K lines of code), can be used in real-time on the robot, and is accompanied by a speech recognition interface developed as an Android application.

A last notable software contribution is our involvement in the MORSE simulator for academic robotics. We have played a central role in the original design and development of the core functionalities of this open-source simulator which is now used by over twenty laboratories world-wide. While this project as a whole is not directly related to main topic of the thesis, we have led the effort towards effective simulation of human-robot interaction in MORSE. It is briefly presented at section 6.2.1.

1.5 A reader's guide

The thesis in one hour

Because of the contingencies of this world, we acknowledge that the complete reading of this thesis may not fit in one's tight schedule.

If you have only about one hour to dedicate to this work, we suggest to read the following sections in that order:

- What are the challenges? (section 1.3, page 8),
- Contributions (section 1.4, page 9),
- The ORO functional overview (section 3.1, page 57),

- The first interaction experiment (section 6.2.3, page 121),
- The evaluation of ORO and other knowledge representation systems (section 6.1, page 107),
- And finally, the discussion on perspectives (section 7.3, page 138),

Hopefully, this quick overview of the work can help you to select sections that you may want to visit more in depth.

For the patient reader

Roughly speaking, the thesis is organised in three parts: an analysis of knowledge representation systems for service and personal robotic, the presentation of ORO, our own implementation of such a knowledge representation system, and finally we report on practical uses of explicit knowledge manipulation on robots, first for natural language processing, then through several experiments.

The first part is covered in the chapter 2: after a discussion on what we call "knowledge" in our context, we explore its importance by listing, in a typology of characteristics, the requirements of our robots related to knowledge management. This chapter is completed by a survey of eight systems for knowledge management that have been already deployed on real robots.

At the end of the thesis, we give a second look at these systems to try to draw a picture of the overall landscape of knowledge representation approaches in the robotic research community, to identify new possible research directions.

The second part is covered by chapters 3 and 4. Chapter 3 presents the functional side of ORO server, some of the algorithms that are implemented, and discusses its knowledge model (the ORO *common-sense ontology*). The technical side is presented in chapter 4 where we emphasise the integration of ORO within a larger robotic architecture. The articulations with perception, planning and control are presented.

Chapters 5 and 6 form the third and last part of the thesis. Chapter 5 details *Dialogs*, a module for situated dialogue grounding that takes advantage of the symbolic knowledge exposed by ORO, and chapter 6 presents several evaluations of our work through various experiments conducted during the four years of the thesis preparation.

We conclude the thesis with a discussion of several issues related to knowledge management in service robots (importance of embodiment, relationships between the symbolic and continuous realms, etc.) and some remarks that could further improve knowledge representation and management in future robotic architectures.

Chapter 2

Symbolic Knowledge Representation

2.1 Knowledge and robotics

The idea of *Cognitive Robotics* was coined in the early 1990s by Reiter. In a chapter on that subject in *Foundations of Artificial Intelligence* [78], Levesque reminds about the manifesto they wrote together in 1998:

> Central to this effort is to develop an understanding of the relationship between the knowledge, the perception, and the action of [...] a robot. The sorts of questions we want to be able to answer are
>
> - to execute a program, what information does a robot need to have at the outset versus the information that it can acquire *en route* by perceptual means?
> - what does the robot need to know about its environment versus what need only be known by the designer?
> - when should a robot use perception to find out if something is true as opposed to reasoning about what it knows was true in the past?
> - when should the inner workings of an action be available to the robot for reasoning and when should the action be considered primitive or atomic?
>
> and so on. With respect to robotics, our goal (like that of many in AI) is *high-level robotic control*: develop a system that is capable of generating actions in the world that are appropriate as a function of some current set of beliefs and desires.

Indeed, pervasive knowledge could safely be considered as the prominent characteristic of cognitive robotics. This chapter is dedicated to an analysis of what is knowledge for a robot, and what are the important features of knowledge and knowledge representation that are relevant to cognitive robotics.

The next section attempts to give a practical definition of knowledge for robotics, with a few of its major characteristics. We then review some existing material from diverse fields of cognitive robotics to propose our own *typology* of the needs and

characteristics of knowledge representation for service and interactive robotics. About fifty such items are identified, defined, and organised in a large set.

We then put into practice this reference by surveying eight systems and architectures for robots. Their main strengths are underlined, in order to depict the state of the research in knowledge representation for robots.

Finally, we conclude this chapter on symbolic knowledge representation by briefly presenting a novel API for knowledge manipulation, jointly designed with several other researchers on knowledge representation.

What do we call "knowledge"? Since we will discuss at length the concept of knowledge in the context of robotics in the coming pages, it is useful to make our terminology explicit.

Be it in philosophy, cognitive sciences or computer sciences, reaching an agreement on a definition of "knowledge" seems difficult.

Allen Newell's famous *Knowledge Level* [98] can be a starting point: for Newell, *knowledge* is a medium between *agents* and *goals*, *actions*, *bodies*. Whereas the symbol level deals with representation, the knowledge level deals with language, semantics; whereas the symbol level deals with inference, the knowledge level deals with entailment. We will, at the conclusion of the thesis, give a second look to this distinction.

In our robotic context, we define knowledge as a narrower concept, while keeping Newell's link to actions: "knowledge" is for us *a set of interrelated logical facts that are meaningful to the robot executive controller*. By *meaningful* we mean that can possibly be interpreted to lead to a purposeful action. We will see that our main challenge while designing a cognitive architecture is furthermore to make this knowledge as *explicit* as possible.

The relation of *data* and *information* to knowledge is a debated epistemology question known as the "DIKW" hierarchy question. In this thesis, we will associate data to low-level material like raw sensor output, and information to uncontextualised symbolic facts.

To give a example, we can imagine a human reading a book while being tracked by a Kinect sensor: the pose of the human skeleton in the world would be the data, the fact `looksAt(human, book)` as interpreted by a geometric reasoning module would be the information, the fact `looksAt(john, war_and_peace)`, fully grounded and connected to the whole knowledge base of the robot would be proper knowledge.

This simple example also acknowledges the tight coupling between the symbolic and the geometric realms: while AI at its origins was mostly a matter of symbolic models, it has been since recognised that not only that the mind is not a purely abstract system, disconnected from the physical world, but even more, cognition fundamentally relies on its relation to the physical world (so-called *embodied cognition*). Varela [138] is one of the main discoverer of these mechanisms, and coined the concept of *enactivism* as the theoretical framework that study the links between cognition, embodiment and actions.

From the perspective of communication, knowledge is for us an information *interpreted in the cultural and social contexts* of the robot. This translates into three practical features: knowledge is made of statements that are *contextualized*, *grounded*, and *limited* to a domain of validity. These three features have important consequences for the way

a knowledge representation and storage system must be designed. Let us examine them:

Contextualizing is the ability for a cognitive system to connect a fact with a *cultural context*, an *interpretive frame* and the set of other facts previously acquired by the agent.

Since machines are limited to syntactic (in contrast to semantic) processing, we are mostly looking for a syntactic (*i.e.* , based on symbols) matching between concepts representations (in our case, sets of alphanumeric characters)..

The *cultural context* is a broad set of common, general facts that are considered widely accepted among the interactors (*e.g.* "bottles may contain water"). This knowledge is often referred as *common-sense knowledge*.

By *interpretive frame* we mean that a concept may have different interpretations depending on the agent, the current situation or the time frame the statement belongs to. Since a fact in one frame can be different (or even inconsistent) with a fact in another frame (for instance, one object can be visible for the robot and invisible for another agent), the underlying knowledge representation system must properly handle these interpretive frames.

Note that effectively representing a context is a rather different task than identifying it. This aspect will be further discussed at the end of this work.

Grounding corresponds to the identification or creation, and then, maintenance of a link between the symbol (the syntactic form of knowledge the computer will manipulate) and its semantics, *i.e.* its meaning, anchored in the world (the relations between the symbol, the referent of the symbol, and mediating minds is classically referred as the *semantic triangle*, and has been extensively studied in linguistics). The issue of grounding is well known in cognitive science and is summarised by Harnard [44] by this question: "how the semantic interpretation of a formal symbol system can be made intrinsic to the system?". This issue has a very practical importance in robotic: for a robot to be both endowed with a symbolic representational and reasoning system, and able to *act* in the physical world, it must ground its knowledge.

Domain of validity specifies the scope in which an information is (believed to be) true. It covers several aspects: temporal, situational and probabilistic. While related to the previous concept of *interpretive frames*, the domain of validity addresses the question whether a fact must be or not considered in a given context. This validity limitation is not usually carried by the fact itself. In the previous example, for instance, the robot observes a human sitting at a table. The fact "a human is sitting at the table" is true only for a limited period of time, until the human stands up. This period of time is not directly accessible (the robot does not know how long the human plans to stay), but the knowledge representation must be able to deal with this uncertainty and should explicitly label this fact as being limited in time.

To know if a fact is *permanent* or *transitional* (Pollock [105], page 51) is difficult (especially considering that a feature may be considered as permanent or not depending of the context: within the situation "a family meal", the fact "the human is sitting at the table" could be considered as permanent. Conversely, "ground is static" is generally

considered as a permanent fact, expect if we are talking of planetary mechanics for instance. The difficulty lies in the selection of the relevant situation in which reasoning must be carried out at a given time) and have currently to be defined in the cultural background of the robot.

These three aspects lead us to envisage the question of knowledge representation from two perspectives: elements that are *essential* to the knowledge (without those, informations could not become knowledge), and processes that are necessary to *produce* knowledge.

Knowledge is essentially dependent on the ability to represent:

- links, connections between atoms of knowledge,
- a general cultural background, in the form of common-sense knowledge,
- interpretive frames, contexts, restrictions on the domain of validity.

It must also rely the following active processes to:

- acquire and maintain knowledge perceived from the physical world or retrieved from other sources (interaction with other agents, web-based contents,...)
- add and connect new facts to existing ones,
- monitor contexts and accordingly manage the validity of the stored knowledge,
- ensure the logical consistency of the knowledge repository, and explicit inconsistencies when required[1].

We have already seen in the imaginary scenario introduced in chapter 1 that many other cognitive abilities related to knowledge representation and manipulation are required by service robots to actually operate, and the above items are more a high-level view of what knowledge *intrinsically* need to exist. The next section aims at providing a large, comprehensive set of cognitive abilities related to knowledge, that encompasses both essential and non-essential features of knowledge representation systems.

2.2 A typology of knowledge representation requirements for robotics

This section now focuses on formalizing the knowledge representation issue: we aim first at establishing a comprehensive typology and nomenclature (figure 2.1) of representational needs for robotics in the specific context of service robotics, before painting, at section 2.3, the current landscape of approaches to the knowledge representation problem in the research community. For each such "dimensions" of knowledge representation system, we provide a short definition accompanied by links to relevant literature.

[1]One may argue that the real world is however inherently inconsistent; we will discuss several aspects of inconsistencies representation and management later on.

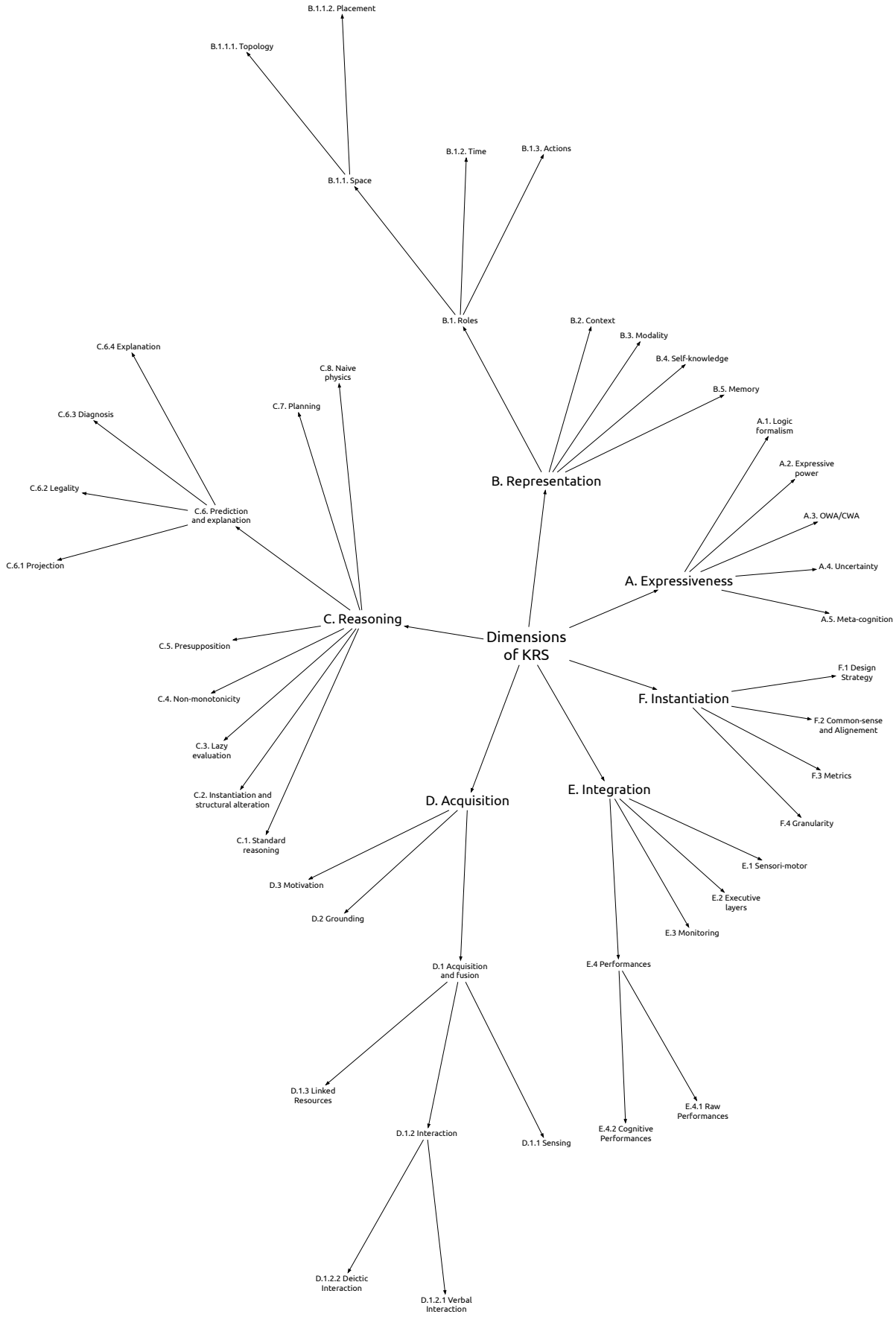


Figure 2.1: Taxonomy of the analysis dimensions of knowledge representation systems for service robotics.

The typology has been built from three main sources: a review of the existing literature on that topic that we present in the next section; the survey of eight knowledge representation systems already deployed; our own experience, acquired during the thesis preparation with the help of many discussions with researchers from both CNRS and TUM, that allowed to interweave two slightly different perspectives on knowledge in robotics.

We also wish to mention that the short presentation of each feature does not claim to be a comprehensive summary of the field: it is beyond our capabilities to cover in a few lines all what the areas of *time representation*, *planing* or *formal logic* have said during the last 30 years. We indeed focus on the aspects relevant to knowledge representation and certainly involuntarily omit many significant works in these fields.

2.2.1 Previous work

As said, the typology we propose is in part based on a comprehensive synthesis of classifications and analysis found in the literature. This synthesis is focused on cognitive abilities strictly related to knowledge manipulation in the context of service robotics.

Levesque and Lakemeyer [78] present in their chapter on Cognitive Robotics several characteristics of knowledge representation systems for robots, stressing the need of representing the *dynamics of the world*. Sensing is included in the knowledge representation via *fluents*; they introduce the idea of *possible worlds* to represent distinct parallel mental models; action representation and reasoning about tasks is discussed in the context of *situation calculus*; *open world* vs. *closed world* approaches are mentioned. They also discuss how robot programming and knowledge representation can be related. We integrate most of these items in our typology.

In a slightly broader context, Heintz et al. [50] define *knowledge processing middleware* as systems supporting "declarative specifications for flexible configuration and dynamic reconfiguration of context dependent processing at many different levels of abstraction". They identify six characteristics: the system must be able to *merge informations* from different, possibly distributed sources; it should support quantitative as well as qualitative processing of information, it should offer *bottom-up* and *top-down* processing, it should be able to deal with *uncertainty*, allow for "flexible configuration and reconfiguration" (which require what we call here *non-monotonicity*) and finally *meta-knowledge* and *introspective capacities* ("declarative specification of the processing functionalities").

Several surveys compare global cognitive architectures [72, 139, 26]. Langley, Laird and Rogers [72] distinguish nine capabilities: recognition and categorisation, decision making, perception and situation assessment, prediction and monitoring, planning, reasoning, execution control, interaction and learning/remembering/introspection. They also separately identify four *properties* of a cognitive architecture, that categorise how knowledge is handled by the architecture: representation of knowledge, organisation of knowledge, utilisation of knowledge and acquisition and refinement of knowledge. This categorisation had a notable influence on our typology, and many of these categories are also present in our proposal.

Vernon et al. [139] split these architectures into two broad categories: the *cognitivist* ones (where cognition is considered as an explicit computation problem, often based

on symbol manipulation), and the *emergent* ones (where cognition only exists as a result of the interaction of the system with its environment). The approaches presented in this chapter are, at a few exceptions, prototypical *cognitivist* approaches that aim at making knowledge explicit within the robot architecture. Vernon et al. propose twelve *characteristics of cognitive system* to compare architectures. Amongst them, they mention the *inter-agent epistemology* (how the structure of the world is captured in a representation and shared), the relation to *embodiment*, the ability to *anticipate* and to *adapt*, and the mechanisms of *motivation*. While presented at the level of the whole robotic architecture, these features also translate into knowledge representation strategies and are relevant to our study.

Chong et al. [26] also provide a recent review of the main cognitive architectures, with a focus on eight functions: perception, memory, goals management, problem solving capabilities, planning, reasoning, learning and links to neurobiology.

At an even broader scope, several authors from fields that are connected to robotics have previously listed desirable features of artificial systems aiming at rich cognitive abilities.

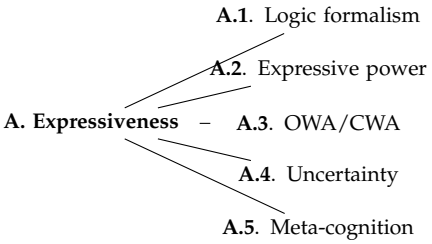
For instance McCarthy recently listed in [93] the challenges he identifies on the road to a *human-level AI*.

- the ability to *"operate successfully in the common sense informatic situation"*,
- the necessity of relying on mathematical logic, as the most fruitful formalism for machine intelligence,
- the ability to deal with *approximate concepts and approximate theories* (that would include representing them, and reasoning with them),
- non-monotonic reasoning,
- what McCarthy calls *Elaboration Tolerance*: the ability to extend *on demand* the closed domain of interpretation for a given assertion,
- the ability to formalise and reason about contexts,
- reasoning about events, and in particular, actions,
- the capacity of introspection,
- and finally, he points the issue of giving computer the right heuristics for decision making.

Coming from the perspective of natural language processing in situated context, Roy and Reiter summarise in [112] what they see as the main challenges to be tackled by knowledge representation systems: *cross-modal representation* systems, association of words with perceptual and action categories (*grounding*), modeling of *context*, definition of the right *granularity* of models, integration of *temporal modeling and planning*, ability to *match past (learned) experiences* with the current interaction and ability to take into account the *human perspective*.

Knowledge representation systems in robotics are directly affected by these points, and we indeed integrate them in our typology, in slightly reformulated ways.

2.2.2 What can be represented?



This first axis of analysis is its intrinsic expressive power. It answers the question: what can be possibly represented. When it explicitly exists, the *language* of representation plays here an obvious role.

Main logic formalisms

The main role of a knowledge representation system is to provide an adequate representation system to formally store facts and concepts that could be informally described in natural language.

Formal logic aims at providing such a representation system with the added value of providing a tractable support for inference and reasoning.

Most (but not all) of the systems we have surveyed rely on a particular logic formalism. The choice of the formalism has a strong impact, on one side, on the range of ideas that can be expressed conveniently (*practical expressiveness*) or at all (*theoretical expressiveness*), on the other side, on the ability to solve the fundamental inference problem (called *satisfiability*: is a given logical sentence true in my model?) in a tractable manner.

A large number of logic formalisms do exist and we briefly present below the most relevant ones for systems actually deployed in robotic architectures.

Predicate logic is the family of logic formalisms the most commonly found in knowledge representation. It distinguishes itself from the simpler *propositional logic* by the use of quantification to increase generality. *First-order logic* (FOL) is the subpart of *predicate logic* where the objects of *predicates* (or *formulae*) are simple *terms*, while in *higher-order logics*, predicates can be themselves objects of other predicates.

Horn clauses are an important subset of FOL because the satisfiability of a set of such clauses is a P-complete problem (*i.e.* practically tractable). A Horn clause is a disjunction of literals (a *clause*) with at most one positive literal: $\neg p \vee \neg q \vee \cdots \vee \neg t \vee u$, which can also be represented as $(p \wedge q \wedge \cdots \wedge t) \rightarrow u$. Important logic programming languages like Prolog are based on Horn clauses.

The family of *Description Logics* [9] also plays an important role. It is also a subset of the first-order logic, with some extensions in second-order logic. Description logics are notable because most of them are known to be decidable (but not always in a practically tractable manner). In description logic, axioms are build from *concepts*, *roles* (that are unary or binary predicates) and *individuals*. The W3C OWL-DL standard is a widely-used language to describe domains with the description logic.

Because description logics have been originally created from the perspective of a *knowledge representation language* and not a logic language, their terminology (*concept* or *class*, *role* or *property*, *individual*,...) is well-suited to knowledge description.

Modal logic, that allows for statement qualification like *possibility* or *necessity*, have been shown to be closely related to description logics [10]. Modal logic allows to represent conveniently parallel possible worlds and facts like "the robot knows *that the human knows* how to read a recipe".

Temporal logic are designed to represent and manipulate assertions whose truth value may vary in time. We introduce one of its key idea (the *fluents*) later in the typology.

One last class of logics that is of particular relevance for robotic applications is the *probabilistic logics* or *Bayesian logics*. These logics provide a formal framework to reason on propositions whose truth or falsity is uncertain. We elaborate below on the representation of uncertainty.

Note that most of these logic formalisms are still active research field on their own, and practical considerations (especially the availability of reasoners efficient enough for on-line use on a robot) often constrain the choice of a logical formalism and a level of expressive power.

Expressive power

Logical formalisms each bring a certain level of expressive power. For instance, the following classical syllogism can not be represented in propositional logic because of the use of *universal quantification*:

1. All men are mortal,
2. Socrates is a man,
3. Therefore, Socrates is mortal

However, the following weak version of the syllogism can be represented in propositional logic:

1. If Socrates is a man, then Socrates is mortal,
2. Socrates is a man,
3. Therefore, Socrates is mortal

Generally speaking, expressive power comes at the cost of more complex *satisfiability* and *consistency*[2] computations, possibly leading to untractable, if not undecidable (*i.e.* systems where it is proven that a proposition can not be decided to be true or false) problems.

Figure 2.2 shows that the expressive power of description logics and Horn clauses partially overlaps. In section 2.2.4 we mention extensions to description logics based on rule systems that bring closer the two approaches.

The relationships between expressive power and reasoning complexity that follow has been extensively studied for Description Logics. Zolin [149] maintains a "complexity navigator" that allows to conveniently explore these relationships and indexes most of the literature on that subject.

[2]We precise these concepts at section 2.2.4.

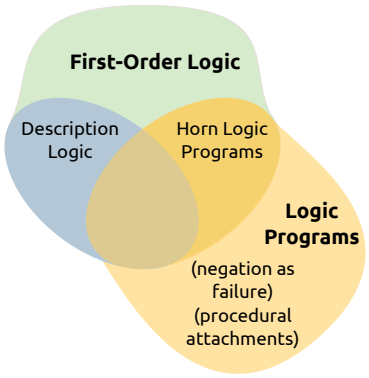


Figure 2.2: expressiveness overlap of Description Logics and logic programs based on Horn clauses, taken from [40]

It can be noted that the relation between expressiveness and reasoning complexity is fragile: for instance, adding the following axiom $\langle$`farFrom` *`disjointProperty`* `near`$\rangle$ (that states that two individuals can not be at the same time near and far from each other) changes the expressiveness power of the ORO Common-Sense ontology (presented at chapter 3) from $\mathcal{SHOIQ}(\mathcal{D})$ to $\mathcal{SROIQ}(\mathcal{D})$[3]: this seemingly innocuous assertion change the complexity class of the whole ontology, and the concept satisfiability reasoning problem switches from a *NExpTime-complete* problem to a *NExpTime-hard* problem (*i.e.* , *at least* as hard as the hardest problem in *NExpTime*).

This "instability" has practical consequences on run-time performances on the robot because a light alteration of the knowledge structure can lead to very noticeable performance drops.

Open world and close world assumptions

The *close world* (CWA) vs. *open world* (OWA) assumptions name a modelling choice on the *completeness* of a knowledge domain. In the close world assumption, a proposition that can not be proven true is assumed to be false (*negation by failure*), while in the open world assumption, a proposition may be considered either true, false or unknown.

This distinction is important in robotics where the robot may have to manipulate concepts with only partial knowledge on them. For instance, let us imagine a robot that sees a bottle on a table, whose bottom is hidden by another object. The robot can not prove that the bottle is indeed *on* the table. A knowledge representation system relying on the closed world assumption would then assume the bottle is *not* on the table ($\neg R^{CWA}_{isOn}(bottle, table)$) whereas with the open world assumption, the proposition $R^{OWA}_{isOn}(bottle, table)$ would be undecided. Example in table 2.1 provides another example of consequences of the CWA/OWA choice on reasoning.

The OWL language is specifically known to assume an open world. Domains constrained with the closed world assumption lead to more tractable inference problems, and allow for instance the use of logic languages like Prolog. Thus, several approaches exists to *locally close* a domain (*cf* Levesque [78], section 24.3.2 for a summary of those ones).

[3]See appendix A for a brief explanation of this notation

Action	Part involved
`PickSoftly`	hand
`PickAndPlace`	arm, hand
`MoveArm`	arm

Table 2.1: Assuming the question is: *select actions that do not require to move the arm*, a CWA reasoner would return `PickSoftly` whereas an OWA reasoner would not return anything if the `PickSoftly` action is not explicitly said not to involve the arm.

Representation of uncertainty and likelihood

Sources of uncertainty for a robot are two-fold: uncertainty *intrinsic* to facts (like *"It may rain tomorrow"*), uncertainty caused by imperfect perception of the world (*"Is the bottle really on the table?"*). Most logics do not account explicitly for uncertainty. It must be either relied on specific logics (like Bayesian logics) or on extensions of classical logics.

Meta-cognition: knowledge on the knowledge

As stated by Cox and Raja [28], meta-cognition is composed of both *"meta-level control of cognitive activities and the introspective monitoring of such activities to evaluate and to explain them"*.

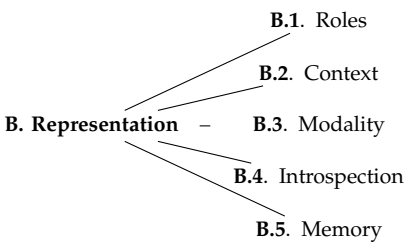
Sloman proposes in [125] a detailed analysis of meta-cognition and its different aspects in both natural (human) and artificial systems.

A knowledge representation system endowed with *meta-cognition* is not only able to manipulate knowledge but also to exhibit and manipulate the structure of its knowledge and the reasoning process. For instance, the ability to explain a logical inconsistency in a KRS is a meta-cognitive function, as is the ability to expose and alter the knowledge structure (these two reasoning techniques have their own entries in the taxonomy, at section 2.2.4.

At section 2.2.3 below, we discuss the idea of introspection. Meta-cognition can be viewed as the technical facet of the introspection in general.

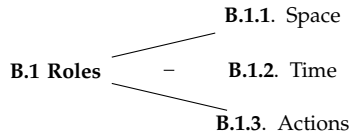
2.2.3 How things are represented?

We do not discuss in this section the general strategies to construct a knowledge model (they will be presented in section 2.2.7). We focus here on questions that involve representational challenges (time, space, context) or require specific cognitive capabilities (theory of mind, introspection, memory).



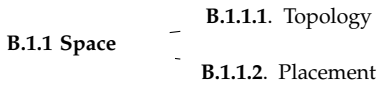
Role representations

This section discusses strategies and approaches to represent in a knowledge model three important *roles*: spatial relations, time and representation of actions.



Representation of space Symbolic representation of space is a widely studied topic. In particular, a large literature corpus is available on spatial ontologies.

Two main classes of spatial relations are usually represented at the symbolic level: the topology of environments and the placement of physical entities.



Topological maps are abstractions of an environment as graphs where nodes represent places and edges represent connections between places. Because they are symbolic representation, topological maps allow higher-level reasoning (such as containment, connectivity, regions) than metric maps.

One important contribution to the building of a coherent representational stack for space representation is the Spatial Semantic Hierarchy [66] introduced by Kuipers. It consists in multiple interacting representations of space, both qualitative and quantitative, that span from sensor-level representation to ontologies of places and regions.

Symbolic representation of entities placement can be absolute or relative. The relation `isOn`, for example, leads to absolute statements: the validity of the relation is independent of the nature of its subjects and objects, and is also independent of the observer.

On the contrary, the relation `nextTo` is relative, and depends on the relative size of the subject and the object. Two houses distant of 2 meters from each other can be considered as next to each other. Two ants separated by 2 meters are not next to each other.

The relation `leftOf` is another example of relative spatial relation, this time because it depends on the observer viewpoint.

Choices must be made in the knowledge representation system to adequately represent relative spatial relations. Options include the computation of such relations only on-demand, when the context is known (which viewpoints, etc.) or storage of these facts in different models, one per agent.

Representation of time As an agent acting at human-like time scale and dealing with temporal concepts (like actions), a robot needs to represent and reason about time. Time representation is split into two distinct abilities: representing time points (both in the past – which is roughly equivalent to assignment of timestamps to events the robot perceives – and in the future), and representing passing time (situations, durations, timespans) like in *"the eggs will be cooked in 10 min"*.

This is usually formalised as a disjunction in time representation between *events*, that is, any durationless temporal concept, and *situations*, that is, any temporal concept with a non-zero duration[4].

Numerous techniques to represent and reason about time have been devised. Amongst the most significant ones, we can mention Allen's interval algebra [5] and Ghallab's chronicles [39]. The concept of *fluent* also play an important role for time representation: fluents are properties (or *conditions*) that change over time, like $sees(agent1, apple, t)$.

We call a system that does not account for time (*i.e.* that mentally permanently lives in present) *atemporal*.

Actions Events and actions are two temporal concepts that are of particular importance to robotic systems, as systems that perceive, react and perform in their environment.

While representation of events boils down to label a timestamp, representation of actions are more complex since not only they deal with durations, but they also imply semantic interactions between concepts. From a taxonomy point of view, actions are a particular type of event that normally leads to a situation corresponding to the action realisation.

Thematic roles [41] (also found as *semantic roles* or *theta roles* in the literature) allow to semantically qualify the parameters of an action. The recipient of the action, the performer, the object acted upon, the destination are some example of common thematic roles. Table 2.2 presents a more comprehensive list of thematic roles proposed by [1] (see [43] for a comparison of other sets of thematic roles present in the literature).

In the context of robotic, *task* often represents an aggregate of (atomic) actions. We use the term here as the representation of the abstract model of an action. Tasks usually specify at least the conditions required to performed the action and the consequences of the realization of the action, and are central to the decisional layers of the robot, in particular for the planning, monitoring and execution control activities. Knowledge representation systems may thus have specific mechanisms to represent them (and possible, to reason about them, as presented in section 2.2.4).

An technical report we have written on task modeling in OWL ontologies is available in appendix B.

Plans representation is closely related to action and task representation. It is specifically discussed at section 2.2.4.

Context modeling

Knowledge is contextualized information: it is essential for the robot to associate the facts it represents to a *context*. The context carries the keys for the interpretation of the information and set a common ground for interaction (and, in particular, communication [56]). It implicitely defines the domain of validity of the facts and carries the *common-sense* knowledge required to fill the gaps of the knowledge explicitely shared between the agents.

[4]Other definitions of a situation do exist, notably in the context of *situation calculus* [79], Reiter and Levesque consider a situation to be an history of actions.

Role	Meaning
Agent	The *doer* or instigator of the action denoted by the predicate.
Patient	The *undergoer* of the action or event denoted by the predicate.
Theme	The entity that is moved by the action or event denoted by the predicate.
Experiencer	The living entity that experiences the action or event denoted by the predicate.
Goal	The location or entity in the direction of which something moves.
Benefactive	The entity that benefits from the action or event denoted by the predicate.
Source	The location or entity from which something moves.
Instrument	The medium by which the action or event denoted by the predicate is carried out.
Locative	The specification of the place where the action or event denoted by the predicate in situated.

Table 2.2: A list of thematic roles, as proposed by Aarts [1]. Depending on the action, only certain roles are meaningful.

Context is generally difficult to recognise, represent and reuse because it is multiform and largely implicit nature. It is also never unique: at a given moment, several contexts, at different temporal, spatial, social scales, overlap.

In the current literature in robotics and cognitive architectures, the term *context* usually simply refers to a set of beliefs that initiate a representation (and reasoning) frame: in [76], the robot creates a context of interaction with a specific human by storing in a separate model the beliefs of this human and using this knowledge when dialoging with the human, the reasoning network is reinitialised in the GLAIR architecture when the hypotheses that defined the current situation are not believed anymore [118].

This acception of *context* is simplistic, and omit the overlapping and multi-scale aspects of context modeling.

We see context representation as one of the main challenge of knowledge representatio in general, and we will further discuss the importance and issues brought by context modeling in the conclusion of the thesis.

Modality, contingency and theory of mind

Linked to the context representation, but seen from another angle, knowledge representation systems may support logical *modality*. A knowledge model is logically modal if is support the concept of *possible worlds*, *i.e.* , parallel beliefs models (or *interpretations*) that can be independently accessed.

A *contingent* proposition is defined as neither always true (a tautology) nor always false (a contradiction) in every possible world: its truth value depends on the context. In knowledge representation systems for robotics that support logical modality, interpretations are often initialised with a common set of initial beliefs (like common-sense knowledge). This initial common knowledge is hence true in every possible worlds for

the robot, and thus does not belong to its contingent knowledge.

On the contrary, alternative mental models with contingent knowledge may be used to represent different (possibly hypothetical or even imaginary) views on the world, from the robot own perspective or context, or from other perspectives computed by the robot.

The representation of the mental perspective of other agents has a particular importance in human-robot interaction. It relies first on the ability to literally *view* the world from a standpoint which is not *egocentric*. This cognitive ability is referred as *perspective taking*. Flavell [36] and Tversky [136] define the psychological grounds of perspective taking, that are themself originated in Piaget's work on cognitive development (recent studies on infants include Moll [95] for instance). Perspective taking begins to be studied on robots as well [133, 21, 111].

The idea of a *theory of mind* [106] emerges from the perspective taking ability. It can be defined as the ability for one to understand and acknowledge that other intelligent agents can have their own mental state (that includes beliefs, intents, desires, knowledge) that is possibly different from one's own. The *attention* plays a central to the development and recognition of a theory of mind [14, 77].

A notable consequence of having a theory of mind is the representation of *false beliefs*, *i.e.* , facts that are believed to be true for an agent, but false for other ones. The *Sally and Ann* experiment [77] (figure 2.3) is a classical example of a false belief situation. In 2.3(d), Sally thinks the ball is in the beige box because she did not see Ann moving it. An external observer asked "Where will Sally look for the ball?" would answer "in the blue box" without a theory of mind (*i.e.* , a model of the knowledge of Sally), whereas it would correctly answer "In the beige box" with a theory of mind.

Scassellati [116] is one of the first to have implemented a theory of mind on a humanoid robot.

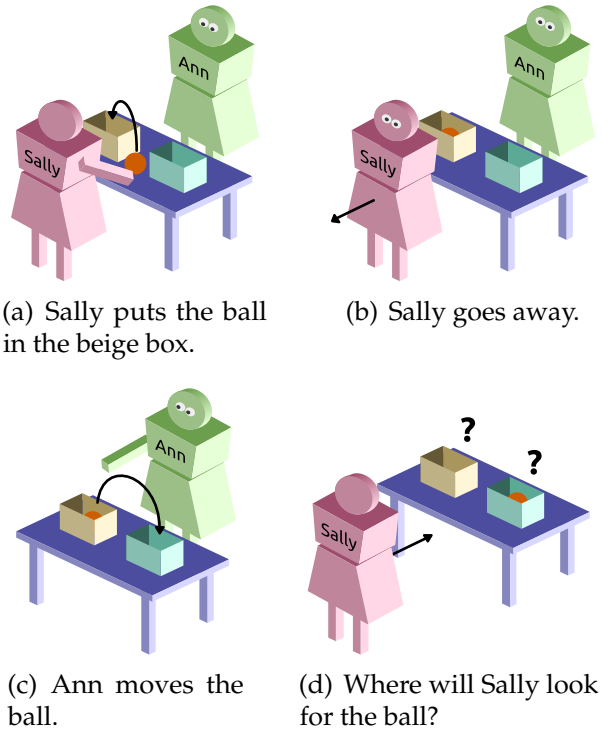
Self-knowledge: Who am I? What can I do?

Self-knowledge *Self-knowledge* is the term used in philosophy and psychology to describe the knowledge that an individual has, acquires or infers about itself through its experiences. It answers the question "What am I like?".

Introspection is the technical meaning to access to self-knowledge by the ability to self describe: what are my capabilities, what is my state (performing some action, idling, etc.), what are my beliefs, what are my intentions and my plans?

Introspection must be distinguished from meta-cognition: While introspection may require meta-cognition (for instance to be able to expose its internal knowledge), it is not always mandatory. The current state of the robot can be represented as a simple instantiation of a specific category (for instance, if the robot gives an object to the human, this state could be represented with the triples [`robot` *`performs`* `action1`, `action1` *`isA`* `Give`].

Modeling of the robot capabilities A particularly important aspect of self-knowledge for robots relates to the description of its own capabilities: which sensors/actuators/-computation services exist and are currently available ? While at a first level, these descriptions can be static (*e.g.* the robot has one laser scanner and two arms), at more



(a) Sally puts the ball in the beige box.

(b) Sally goes away.

(c) Ann moves the ball.

(d) Where will Sally look for the ball?

Figure 2.3: The standard "Sally and Ann" false-beliefs experiment, taken from [77].

advanced levels, the description is updated and reflect the current (and possibly past and future) state of the robot. Note that these descriptions may also involve geometric descriptions (a kinematic chain, the pose of a device, etc.) that may be deported outside of the main knowledge base. Efforts trying to formalize, maintain and expose the capabilities and state of a robot are not new (and ground themselves in work and techniques for self-descriptive remote procedure calls in computing science), but take a renewed importance with applications for high-level multi-robot cooperation.

Recent works by Kunze et al. [68] seeks at defining a formal language to represent the capabilities of a robot.

Memory

Memory has been studied at length in the cognitive psychology and neuropsychology communities: Atkinson and Shiffrin [7] introduce the idea of *short-term* and *long-term* memory, Anderson [6] splits memory into *declarative* (explicit) and *procedural* (implicit) memories, Tulving [134] organises the concepts of *procedural*, *semantic* and *episodic* memories into a hierarchy. Short-term memory is refined with the concept of *working memory* by Baddeley [11] (Figure 2.4).

It is worth emphasising that if memory is commonly associated to the process of forgetting facts after a variable amount of *time*, it actually covers more mechanisms that are relevant to robotics, like selective remembering triggered by a specific context or reinforcement learning.

Most knowledge representation systems offers some kind of memory as a pool of facts that are not forgotten by the robot until it is halted (this memory is often referred

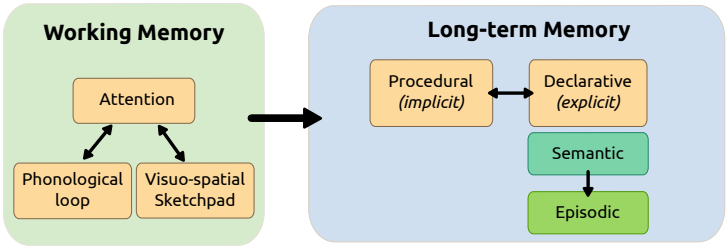
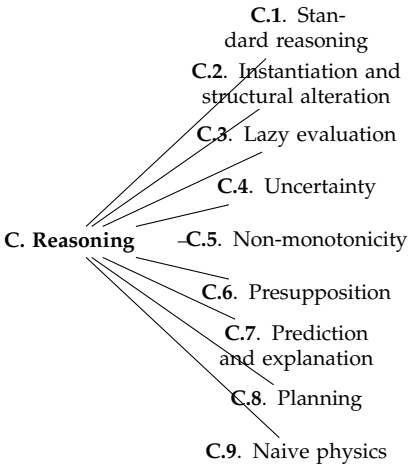


Figure 2.4: Overview of the main types of memories, based on [7, 6, 134, 11]

as a *working memory*, but with a meaning unrelated to Baddeley's definition). Some systems may propose persistent storages that allow the robot knowledge to grow over time, while others may offer a larger range of memory categories, like short term memory (that lasts for a couple of seconds) or episodic memory (that allows the robot to selectively remember facts associated to specific events).

In the larger field of cognitive architectures, the SOAR architecture [73] is one of those that tries to reproduce a human-like memory organisation. The GLAIR cognitive architecture also have a concept of long term/short term and episodic/semantic memories.

2.2.4 Reasoning techniques



Standard reasoning techniques

We call *standard reasoning techniques* techniques based on logical inference, using resolution algorithms like *forward chaining*, *backward chaining* or *semantic tableaux*.

Main reasoning problems include *concept satisfiability*, *consistency checking* and *instance checking*.

Concept satisfiability verifies if it is possible to find a non-empty *interpretation* of a concept (or an expression defining a concept) in the knowledge model. For instance, the formula $\texttt{Plant} \wedge \texttt{isRed}$, which defines the concept of red plants, is satisfiable in a model $\texttt{KB}$ iff $\exists a, \texttt{Plant}(a) \wedge \texttt{isRed}(a)$, *i.e.* if we can find at least one red plant a in our model.

Checking the consistency of a model is equivalent to checking the satisfiability of each of the concept defined in the knowledge model.

Instance checking consists in verifying that an individual a is an interpretation of a concept (or concept expression) C in the knowledge model. A typical example would be that we are provided with an instance `object1` and we want to know if this object is a kind of `Bottle` or `Glass`.

Inferences can also be drawn from other constructs, whose availability depends on the representation language. OWL, for instance, has constructs for:

- class subsumption (to represent inheritance relations)
- reasoning on roles properties, including:
  - entailments based on roles domain and range (for instance, if the domain of the role `thinksOf` is known to be `ThinkingAgent`, then `thinksOf`$(a, b) \rightarrow$`ThinkingAgent`(a)),
  - universal, existential and cardinality constraints,
  - several second-order predicates (inverse, symmetry, transitivity, etc.)
- *class restrictions* like:

  `Bottle` $\equiv$ `Artifact` **that** (`hasShape` **value** `cylinderShape`)
- set operations like:

  `Color` $\equiv \bigcup$ (`blue`, `green`, `orange`, `black`, ...)

Rule Languages As mentioned earlier, knowledge models based on description logics can be extended through rule languages (typically for OWL, the SWRL language).

An intersection of properties is an example of expression that can only be represented with rules. For instance:

`looksAt(?agt, ?obj)` $\wedge$ `pointsAt(?agt,?obj)` $\Rightarrow$ `focusesOn(?agt, ?obj)`

Dynamic instantiation and alteration of the knowledge structure

The content of a knowledge base is often conveniently divided into a *structural part* that defines the conceptualisation of a domain in term of vocabulary and relations between the concepts, and an *instantiation* of this structure into concrete entities.

The terms TBox and ABox are commonly found to describe these two different types of statements in ontologies: TBox statements describe a system structure (made, for example, of a set of classes and properties) whereas the ABox contains TBox-compliant statements that are asserted in this structure.

Knowledge representation systems allow to modify the ABox, which can be considered as the dynamic part of the knowledge base. Alteration of the ABox include the addition or retraction of relations between existing instances, and the addition or removal of instances. We call the later *dynamic instantiation*: the capability for a system to create new instances at run-time. The term dynamic instantiation applies primarily to concrete entities (typically, a new object is discovered by the robot, a symbolic instance

is created for it), but may also apply to abstract entities (like an instance of action, of a feeling, etc.).

The knowledge representation system may also allow to alter the TBox. This requires the underlying reasoning systems to be able to dynamically take into account structural changes in the knowledge base.

Example of TBox alteration include modification of the taxonomy (like addition or retraction of a `subClassOf` relation), changes to the asserted domain or range of a predicate, addition or retraction of rules.

Supporting TBox alteration has notable consequences on the learning capabilities of the system: teaching general facts (*i.e.* , facts at the level of whole categories) like "cars go on roads" to a robot requires an alteration of the TBox.

Lazy evaluation

Lazy evaluation describes the ability for a KRS to delay active knowledge acquisition or reasoning operations until the value is actually needed.

For instance, a system that computes the symbolic relative placement of two objects only when this fact is required to answer a query would be said to adopt a lazy evaluation strategy, whereas a system that computes *a priori* such relations (and by consequence, carries out such computation for possibly all known objects) would be said to use a *strict evaluation* policy.

Lazy evaluation has an immediate impact on efficiency and scalability of the system, and some problems may even be only tractable with a lazy evaluation strategy (the relative placements of object is an example of combinatory explosion in strict evaluation approaches, although this issue could be mitigated in real use-cases by heuristics that would select a subset of objects to evaluate).

One downside is that the knowledge base never contains explicitly the complete set of beliefs of the robot. This limits for instance the ability for the robot to react to logical conditions that involve facts that are lazily evaluated ("trigger a callback when object A is behind object B" would not be triggered with a purely lazy evaluation strategy for spatial relations).

(Non) monotonic reasoning

Monotonic reasoning means that addition of new assertions to a knowledge base can only extend the set of assertions that can be inferred, while a *non-monotonic* reasoning scheme may lead to retraction of facts. McCarthy coined a famous example to illustrate the need of non-monotonic reasoning:

> Consider putting an axiom in a common sense database asserting that birds can fly. Clearly the axiom must be qualified in some way since penguins, dead birds and birds whose feet are encased in concrete can't fly. A careful construction of the axiom might succeed in including the exceptions of penguins and dead birds, but clearly we can think up as many additional exceptions like birds with their feet encased in concrete as we like. Formalised non-monotonic reasoning provides a way of saying that a bird can fly unless there is an abnormal circumstance and reasoning that only the

abnormal circumstances whose existence follows from the facts being taken into account will be considered.

An important application of non-monotonic reasoning is the representation of change: for example, to make a brownie, one needs to crack eggs and mix them to the chocolate. The eggs disappear and are replaced by a dough:

$\texttt{Egg}(a) \wedge \texttt{Chocolate}\,(b) \wedge \texttt{MakeDough}(a,b,c) \rightarrow \neg\,\texttt{Egg}(a) \wedge \neg\,\texttt{Chocolate}(b) \wedge \texttt{Dough}(c)$

The insertion of the proposition $\texttt{MakeDough}(a,b,c)$ leads to retraction of other facts. This rule requires non-monotonic reasoning to be applied.

Default logic is one of the formal logics that account for representing general truth and exceptions to it (for instance, *tomatoes are red, in general*). However, due to computational complexity of these models (most of inferences in default logic are known to be NP-complete problem), classical logics and most of the existing reasoners do not allow non-monotonic reasoning. For instance, the SWRL rule language, usually associated to the OWL-DL ontology language, does not allow non-monotonic reasoning (only so-called DL-safe rules are allowed).

One important exception is the *negation as failure* inference rule, as implemented by PROLOG for instance, that allows for non-monotonicity within the closed world assumption.

Presupposition accommodation

Presupposition accommodation [140] is the ability for a system to automatically create a context allowing to make sense of a proposition.

Applied to robotics, we can imagine a human telling a robot "Please get me the bottle that is behind you". If the robot has not yet see what is behind it, it needs to assume (and represents in its knowledge model) that a undefined bottle can be found somewhere in the half of space behind it.

A knowledge representation system able to cope with presupposition accommodation would be able to take into account this (usually under-defined) information that is not grounded into perception for later inferences.

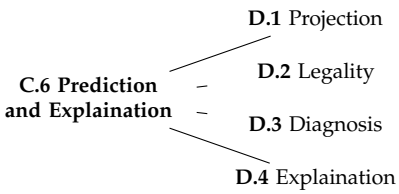
This ability to imagine a physically state of the world that is not actually perceived can be seen as the converse of the grounding ability.

Note also that presupposition accommodation implies a bidirectional link of the symbolic knowledge model with a geometric (or physical) model of the environment.

Prediction, projection, explanation

Levesque [78] distinguishes two main tasks related to reasoning on actions and consequences of actions, the *projection task* and the *legality task*.

We call *diagnosis* the converse of the projection task: the ability to track back the origin of a decision, and *explanation* the more general ability to explicit a reasoning or a decision.



Projection task : determining whether or not some condition while hold after a sequence of actions. The projection task is a typical non-monotonic reasoning task, since at each step, the system must add but also retract beliefs, as defined in the tasks post-conditions.

Legality task : determining whether a sequence of action can be performed starting in some initial state.

The projection and legality tasks are illustrated in appendix B where the tractability of task representation in DL ontologies is discussed.

Diagnosis : this corresponds to the ability to rewind on past events in case of failure to provide possible explanation. This can be seen as the temporal reverse of the projection task. Because of their modelling of situations as an history of actions, derivatives of the GOLOG logic programming languages are a good example of the diagnosis task integrated to the knowledge representation system [42].

Explanation Diagnosis is also linked to the *explanation* or *justification* capabilities that may be offered by the knowledge representation system. The explanation of an entailment is the sequence (or set of sequences if several are possible) of reasoning steps that allow to reach a conclusion. An explanation can also conversely explain why a statement leads to a contradiction.

The following example shows an explanation for an inconsistency in a particular knowledge base: by adding the statement ⟨`robot1` *`belongsTo`* `human1`⟩, we observe that an inconsistency is triggered. The reasoner provides the four following observations to explain the inconsistency:

1. `robot1` **Type** `belongsTo` **some** `Thing`
2. `belongsTo` **Domain** `Artifact`
3. `robot1` **Type** `Agent`
4. **DisjointClasses:** `Agent,Artifact`

This explanation directly portrays the underlying structure of the knowledge model: we can understand that a robot is modelled here as an agent, that agents and artifacts are disjoint classes and that only artifacts can belong to someone, hence the inconsistency. A KRS may provide mechanisms to expose this kind of analysis, either automatically when inconsistencies occur, or on demand.

Explanation of contradictions plays a particular role for robots: from a cognitive point of view, a logical inconsistency (*i.e.* , a contradiction) can be viewed as a *cognitive conflict* or *cognitive dissonance* (*i.e.* , two incompatible models of the world that must be dealt with). Being able to expose and explain such cognitive conflicts eases the control

of the behaviour of the robot in unexpected semantic situations, and forms a first step towards an adequate reaction.

Cognitive dissonance is also identified by developmental psychologists like Piaget as a motivation factor for a child to progress through the various stages of its cognitive development. This has been also studied in robotics [102].

The cognitive capability of *justification* is also intimately linked to the meta-cognition capability and participates to the overall *cognitive observability* of the system.

Task planning

Symbolic task planning is the ability for a robot to select a sequence of actions in order to reach a given final state. This capability is closely related to the previously mentioned projection and legality tasks: for a robot to plan, it must be able to build hypothetical states of the world that would follow from the successive application of actions (prediction task), and at each step, select possible, legal actions based on their pre-conditions (legality task). As already mentioned, these tasks are highly non-monotonic.

Symbolic task planning in general is a large research field [113]. The so-called Classical Planning Problem, first, is characterised by a unique known initial state, durationless deterministic actions which can be taken only one at a time, and a single agent. STRIPS and PDDL are amongst the commonly used languages for representing such planning problems.

Planning with nondeterministic durationless actions with probabilities can be represented as discrete-time *Markov decision processes* (MDP), and when full observability is replaced by partial observability, we deal with *partially observable Markov decision process* (POMDP).

Hierarchical Task Networks (HTN) are another common formalism for planning problems, where an initial set of tasks (*High Level Tasks*, HLA) is decomposed into either primitive actions or a new set of subtasks.

From the observation that the core reasoning techniques (back and forward chaining) are shared between planners and reasoners used in knowledge representation systems, task planning can be considered within the KRS.

Since McCarthy's *Situation Calculus* in 1963, numerous knowledge representation formalisms dedicated to representation and reasoning about actions and situation have emerged: besides situation calculus, *fluent calculus* and *event calculus* are the main ones. Thielscher [131] recently proposed a unification of these approach in a new *action calculus*.

The GOLOG [80] language and its derivatives (like READYLOG [35] and INDILOG [42]) propose implementations of the situation calculus focused on robotic applications.

It is however also common to rely on external components dedicated to symbolic task planning (in particular because of the non-monotonicity requirements) with a tight link to the KRS for domain retrieval and/or resulting plan storage.

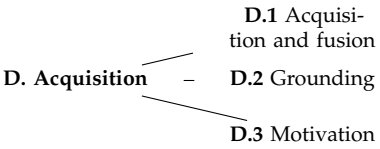
Finally, in the context of interaction between several agents, the management of *joint intentions* and *joint goals* [132, 32] are additional aspects that have to be represented and appropriately handled by the planning subsystem.

Physics-based reasoning

As embodied entities, robots have to interact with physical entities. *Naive physics reasoning* covers all the everyday reasoning the humans unconsciously perform, like taking into account gravity ("if I drop a ball, it falls down") or common physical properties of objects ("a glass may break if dropped", etc.). Many of the interactions with our everyday environments are ruled by such laws that are difficult to exhaustively encode.

Some systems [67] rely on external dedicated physics engine to compute symbolic facts from on-demand physics simulation. This requires a tight integration between the symbolic model and a geometric model that carries the geometries and physical properties of objects.

2.2.5 Acquiring knowledge



Knowledge acquisition and modalities fusion

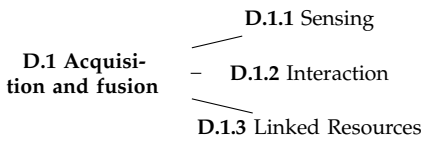
In our context, *acquiring knowledge* means to build new logical statements from data sources and to anchor them into the existing knowledge. We consider three possible sources of data: proprioceptive/exteroceptive sensing, interaction with other agents, humans or robots, and remote knowledge bases. The acquisition process has generally two steps: the information acquisition by itself, and the *transformation* of the information into knowledge, *aligned* with the robot existing model (following our terminology for information and knowledge, as discussed at the beginning of the chapter).

It must be observed that knowledge acquisition is generally not done directly in the knowledge representation system. External components (often aggregated into knowledge acquisition *pipelines*) are usually required to convert percepts into symbolic facts and to ground them.

While we do not review in this article all these systems, the whole process of knowledge acquisition is central in cognitive robotic architecture and the design of knowledge representation systems can influence or be influenced by the approach to knowledge acquisition.

In particular, complex robotic systems often require multi-modal perception capabilities (for instance, a robot can only interpret an utterance like "this is a plate" if it is able to understand gestures, understand natural language and merge them in a timely manner). Multi-modal interpretation can take place at various levels, but in many cases (especially if the modalities are of very different natures, like in the example above) merging will require symbolic-level reasoning. The KRS has a direct impact on the feasibility and ease of such operations.

Let review shortly the three sub-categories of knowledge acquisition.



Sensing From the point of view of knowledge representation, the sensing capability can be split into proprioceptive sensing (*i.e.* , sensing of the robot own internal state) and exteroceptive sensing (sensing of the robot environment). The (physical) introspection capabilities of the robot relies on the former.

Exteroceptive sensing is the most obvious and largely studied mean of knowledge acquisition. Traditional sensing devices (IR, cameras, laser-scanners), while still present and widely used (navigation based on 2D localisation and obstacle avoidance is today the standard), are being step-by-step replaced by *synthetic sensors*.

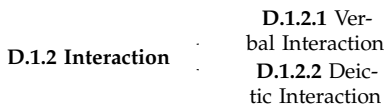
These synthetic sensors include post-processing to provide higher-level percepts that ease the grounding. The prototypical example of such a device is the Kinect sensor. At a first level, it replaces traditional stereo vision algorithms by providing a fast, robust depth map. At a second level, it provides accurate, real-time tracking recognition and tracking of whole body poses of human.

Other examples of such synthetic sensors exist: face recognition, off-the-shelf performant SLAM solutions, automatic cluster segmentation in point clouds (with the PCL library), etc.

While the progresses of these sensing technologies are remarkable, one field of perception remain a very difficult challenge: accurate and generic object recognition. Most of the current approaches to object recognition rely on a mix of point cloud segmentation and fitting with visual feature recognition (SIFT-like algorithms), but it remains a slow and fragile task.

Interaction Interaction with other intelligent agents (humans or robots) is another important source of knowledge acquisition. It relies obviously on some form of sensing (from speech recognition to gesture recognition) but we distinguish it from the previous section because *interaction* implies a form of communication. Communication is associated to specific functions (as shown on Jakobson's diagram, figure 1.5, page 8), and in particular, it implies a shared context (usually implicit) between interactors.

We distinguish between two main interaction channels: verbal and deictic.



The field of verbal interaction processing for robots spans from pattern-based, constrained sentences recognition to natural, bidirectional, unconstrained verbal communication. A large literature corpus exists on Natural Language Processing (NLP) which is presented at section 5.1.2, page 97.

NLP is an established research field by itself, and while the robotic community is still lagging behind on many theoretical aspects, it brings one important aspect: the embodiment. Because the interactors, both the robot and the human, are establishing a communication within a shared physical context, the verbal communication channel

is complemented by deictic channels, back channels and possibly shared physical experiences: a human can show something to a robot, saying "Give me this". This is not possible for a virtual agent.

Several of the knowledge representation systems we present have developed specific built-in mechanisms or extensions to parse, ground and possibly rebuild natural language.

Deictic (used in the literal meaning of "display, demonstration, reference") interaction is also an established field of research in human-robot interaction. Common deictic forms of communication [81] include attentional focus (via face and gaze tracking) and joint attention, pointing, emotional expressions (based on face expressions, postures, emotional gestures).

Like other knowledge acquisition modalities, the recognition and interpretation of deictic communication is rarely directly included in a KRS, but, as previously mentioned, the symbolic representation of such communication acts is relevant and important to achieve successful multi-modal interactions.

Linked Knowledge Resources Robots, and in particular service robots, have usually an access (with possibly security-related constraints) to the World Wide Web and remote knowledge stores.

The current shift towards the Semantic Web (*i.e.* structured, annotated data that are easily machine-processable) makes increasingly easy to have robots to reuse autonomously this knowledge. The DBPedia project, for instance, illustrates well the tendency: it provides an automatically generated RDF version of the Wikipedia encyclopedia.

In its current state, the relevance of the DBPedia project in our context is however limited: the triples that are extracted are mostly "mechanical" (like the categories of a term, or factual informations extracted from Wikipedia's InfoBox), and the vast majority of the knowledge actually contained in the encyclopedia pages remains out of reach of automated parsers. Efforts in that direction however exist [99, 34].

Another notable project that seeks at providing large amount of machine-friendly common-sense knowledge is the MIT's OpenMind project [119]. The project is designed to let the general public easily add common-sense statements in semi-controlled natural language, which is then processed to a publicly available ontology.

Another approach that is easier at short-term to let robots remotely access knowledge repositories consists is building robot-specific shared repositories, with declarative and/or procedural (*i.e.* , plan library) knowledge. The RoboEarth [143] project is an example of such an effort.

Grounding/anchoring strategies

Grounding (also called *anchoring* when specifically referring to the building of links between *percepts* and *physical objects* [27]) is the task consisting in building and maintaining a bi-directional link between sub-symbolic representations (sensors data, low-level actuation) and symbolic representations that can be manipulated and reasoned about [44].

Being embodied entities with interaction with other embodied entities as a fundamental requirement, robots and robotic is deeply concerned by the grounding issue.

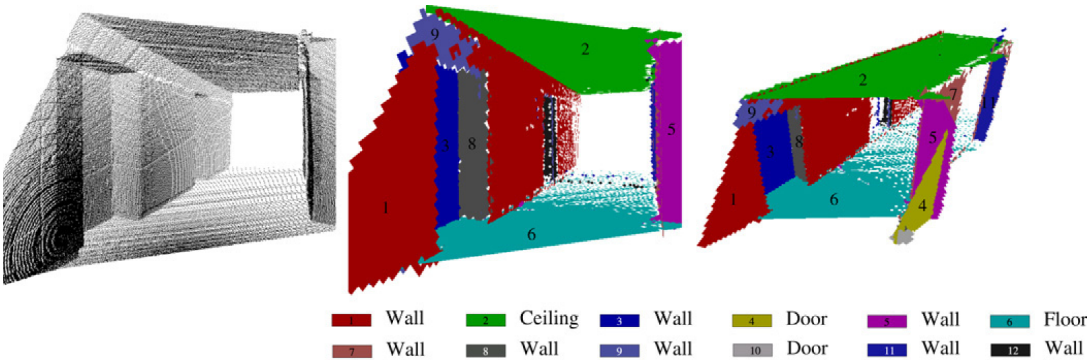


Figure 2.5: Example of a semantic map, taken from [100].

Being actually implemented on real service robots, all the symbolic knowledge representation systems that we review in this study have some kind of grounding process. Numerous approaches exist, like amodal *proxies* [54], grounded amodal representations [3, 91], semantic maps (Figure 2.5, [100, 37, 19]) or affordance-based planing and object classification [84, 137].

Intrinsic motivation and curiosity

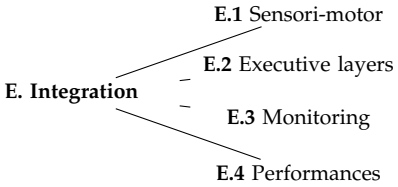
The reasons for a robot to acquire knowledge are diverse, and usually external to the robot itself. It is often driven by the requirements of tasks that the robot has to execute. In this case, motivations are managed by the execution controller and are mostly invisible to the knowledge representation system.

However, motivation can also be intrinsic, driven by the internal state of the robot's beliefs, without external pressure or reward. In this case, the

In the context of robotics, Oudeyer notes however that *the information that is compared* [to compute a level of motivation] *has to be understood in an information theoretic perspective, in which what is considered is the intrinsic mathematical structure of the values of stimuli, independently of their meaning.* [102].

Psychological grounds of motivation are summarized in [102] while the main approaches to computational motivation, divided into knowledge based models, competence based models and morphological models, are surveyed in [103].

2.2.6 Practical integration in robotic architectures



Knowledge representation systems do not mean anything to robots if they are considered in isolation. This section proposes categories of features related to the integration of the KRS into a larger software architecture that includes perception routines, decision-making processes and actuation control.

We also mention some practical aspects of a real-world system, like performances and monitoring tools that come along with the KRS.

Integration with sensori-motor layers

We have previously discussed (section 2.2.5) the principles of the grounding process that aims at establishing and maintaining a connection between percepts (and to a lesser extend, low-level actions) and symbols.

While every real-world cognitive robot need some kind of grounding, the actual implementations lead to very different information flows.

The systems can be roughly split into two classes: *passive* knowledge repositories that process symbolic facts produced by lower-level sensori-motor layers (*push* flow); *active* knowledge managers that directly query (possibly by polling or on-demand) low-level layers.

This macroscopic distinction is however mostly a matter of defining the frontiers of the KRS: some systems like KnowRob [130] encompass geometric reasoning layers that would be considered as external by other systems like ORO [75] that focus on the symbolic fact storage and rely on a ecosystem of independent modules to provide and consume symbolic knowledge.

Other systems do not fit either in such a partition between active and passive systems because they do not stand as independent modules but exist as diffuse, *ubiquitous* knowledge manipulation system (case of the CAST knowledge model [54]), for instance because they are primarily language [35, 114].

Integration with executive layers

Conversely, the knowledge management module need a tight integration with the decision-making processes. As for the integration with sensori-motor layers, the borders of the KRS can be fuzzy and vary from one architecture to another: many consider symbolic task planning as an integral role of the KRS, while other have dedicated extensions for planning, some integrate learning as an on-the-flight process that is part of the KRS, others as an independent deliberative process, etc.

The actual integration techniques vary also widely, from language extensions (like the integration of CRAM [15] with KnowRob) and client-server architectures, to event-driven models (SHARY and ORO [3]). Choices at this level have notable consequences on the whole design of the upper control architecture of the robot, in particular regarding its modularity and the ease of addition of new components.

Monitoring and debugging

It is common to have knowledge representation systems at the heart of a cognitive robotic architecture, and therefore KRS are easily "buried" in the system.

At the same time, the symbolic model often provides a valuable synthetic view on the whole state of the robot, furthermore easily understandable by the human developer (the fact ⟨`human1 isSitting true`⟩ is easier to interpret than the suite of relative coordinates of each joints of the human skeleton, as provided by the human tracker, for instance).

Tools to trace and visualise at run-time the evolution of the knowledge structure and contents may be available with the KRS, as well as post-processing tools that run on the trace to analyze *a posteriori* the cognitive behaviour of the robot.

Evaluation of performances

Benchmarks of symbolic systems for robots are hard to conduct for several reasons: identifying good metrics for robotic experiments in general is difficult because of the complex interactions between tenth of modules running in parallel, and isolating one specific component is difficult. Also, knowledge representation systems are often tightly coupled to the other modules. To quote Langley [72]:

> The conventional wisdom of software engineering is that one should develop independent modules that have minimal interaction. In contrast, a cognitive architecture offers a *unified* theory of cognition with tightly interleaved modules that support synergistic effects.

The lack of standard API for knowledge services makes it also hard to switch between KRS to compare them.

Finally, because service robots are designed to act in rich, dynamic environments, possibly with humans, building repeatable experiments is challenging, and quantitative measurements are often not the right metric [72].

We will however present here some quantitative metrics (related to scalability, for instance), followed by qualitative evaluation approaches, grouped under the term *Cognitive Performances*.

E.4 Performance Evaluation
- **E.4.1** Raw Performances
- **E.4.2** Cognitive Performances

Raw Performances The *raw performance* is evaluated on quantitative benchmarks. The main metric is the *scalability* $\mathcal{S}^{KRS}$ of the system with the size of the knowledge base $\sigma = |\Delta^{\mathcal{I}}|$ (in term of atoms or statements).

We call *relaxation time* $\mathcal{R}^{\mathcal{M}}$ the (averaged) time required by the system after a model modification of type $\mathcal{M}$ before being available for further interaction, and *query time* $\mathcal{Q}^{\mathcal{M}'}$ the (averaged) time to execute a query of complexity $\mathcal{C}$ on the KRS. The type $\mathcal{M}$ of model alteration is either an ABox modification (addition/removal of an instance) or a TBox alteration (addition/removal of a class, a class restriction or a rule).

Temporal scalability is defined in term of the nature of the function $f^{\mathcal{M},\mathcal{C}}(\sigma) = \mathcal{R}^{\mathcal{M}}(\sigma) + \mathcal{Q}^{\mathcal{C}}(\sigma)$ (*i.e.* the relation of the relaxation time and query time to the knowledge base size). *Space scalability* is the relation of memory consumption to the size σ. The scalability is tightly coupled to the expressiveness of the underlying knowledge model, which need to be known for the scalability measurement to be meaningful.

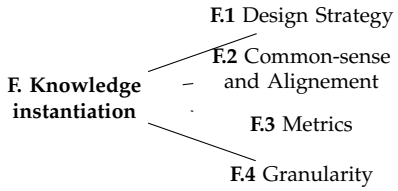
Because of coupling and repeatability issues we have mentioned, raw performances of KRS are often benchmarked with synthetic datasets (which leads to another issues: how to assess the meaningfulness of the performance of a reasoner on an artificial ontology? [13]) or "toy" experiments that do not always model the whole complexity of real-world application [26].

Cognitive Performances While evaluating the raw performances of knowledge representation systems in a relevant manner may be difficult, the cognitive performances of the robot as a whole can be also evaluated.

Langley et al. [72] propose five such dimensions of evaluation: the *generality* of the system (can it adapt easily to new tasks?), the *rationality* or relevant of the inference/reasoning/decisions the system take, the *reactivity* and *persistence* that evaluates if the behaviour of a cognitive system is appropriate under unpredicted changes, the *improvability* of the system as a function of the knowledge added to it, and finally, the resulting *autonomy* of the system.

Cognitive performance can also be evaluated with the support of tools developed in cognitive psychology. Several standard tests (like False-Belief experiments [77] or the Token test [31]) have been used to judge the cognitive abilities of robots [91, 21].

2.2.7 Knowledge instantiation



This last branch of our taxonomy looks at the actual *content* of the knowledge base: the knowledge instantiation. Here, *instantiation* does not only refer to the instantiation of the knowledge structure (what we have called the ABox), but also includes the knowledge structure itself (the TBox).

While we have previously mentioned features of knowledge representation systems that enable the robot to fill its knowledge base with content and alter the knowledge structure, most of the systems also come with a certain amount of initial knowledge that often includes *common-sense* knowledge (*i.e.* facts that widely known to humans, and hence often implicit: "to put a cake in the oven, one must first open the oven's door").

The design strategy, the choice to rely on common-sense knowledge or not, the reuse of standard ontologies, the quantity of *a priori* knowledge are many parameters that lead to different knowledge models.

Design strategy

The main challenge of knowledge representation can be summarised as *How to model the real-world state and interactions in a symbolic way, processable by the robot to make decisions*. We have already introduced the term *grounding* to describe the (bi-directional) process of binding percepts to symbols.

We have also seen that the instantiation of the knowledge (*i.e.* , the actual, practical knowledge available to the robot) comes either from some variant of perception plus grounding, or from knowledge that the developer considers as already meaningful for the robot: remote semantic databases or initial (common-sense or situation specific) knowledge.

This translates into two main strategies to drive knowledge instantiation: a *top-down* design or a *bottom-up* design.

A bottom-up approach to knowledge instantiation takes the output of sensors as the primary source of knowledge: instances of objects, agents, and their relations directly result from what is perceived. No abstract, "from the more generic to the more specific" process takes place.

Steels [127] considers this approach to be a solved issue, and according to Sloman [123] (and his stance against the *"Symbol Grounding meme"*), since bottom-up grounding boils down to grounding of *somatic* concepts (*i.e.* roughly, the sensori-motor relationships that the robot learns from its interaction with the world), it constrains in an unacceptable way the range of concepts accessible to the robot.

Knowledge instantiation can also be approached as a *top-down* activity: in natural language grounding, for instance, the robot needs to automatically bind an abstract representation (a group of words uttered by a human) to an unambiguous, context-dependent, internal concept. This concept may (or may not) be *a priori* available to the robot as a pre-loaded ontology (what we previously called the cultural background of the robot). In any case, the robot must conduct a cognitive process that leads from an abstract concept to a concrete entity.

The bottom-up and top-down strategies also reflect how the knowledge structure itself is constructed: either by successive classification and refinement of percepts, or from generic categories, typically extracted from standard upper-ontologies.

Common-sense and alignment with standard upper-ontologies

At section 2.2.5 we have presented how remote knowledge bases are a valuable source of knowledge, including common-sense knowledge, that robot can extract.

Building *a priori* knowledge with a top-down strategy leads to populate the upper part of the taxonomy with abstract concepts like *Thing*, *Time*, *Person*, etc. The organisation of the whole knowledge depends on the design of this abstract part of the common-sense knowledge.

Different knowledge structures at this level can lead to serious misunderstanding: for instance, if one robot considers the concept of a *person* as representing some intelligent, possibly disembodied, entity, and another robot represents a *person* as a subclass of mammals, their models of the world are likely to be conflicting.

To be able to successfully exchange knowledge with other systems (robots, databases, natural language parsers, etc.), the common-sense knowledge of robots is thus often aligned with a standard upper-ontology.

Many such upper-ontologies exist (table 2.3 lists some of them). While CYC and SUMO are current the main two, many efforts take place to make these ontologies compatible with each other (in particular by using the WORDNET thesaurus as intermediate unambiguous source of semantics).

Metrics and quality criteria

Qualitative and quantitative metrics give an insight on the size, complexity and effectiveness of ontologies used with the robots.

Project	**Terms**	**Assertions** (triples)
CYC	> 300 000	> 3 000 000
YAGO	> 10 000 000	> 120 000 000
SUMO	20 000	60 000
DBPEDIA (for English)	1 840 000	385 000 000
OPENMIND Common Sense (for English)		1 000 000

Table 2.3: Raw size of major upper-ontologies.

Table 2.3 gives such metrics for five major upper-ontologies commonly used in the semantic Web community (taken from [88] and the projects' respective websites). It is interesting to note the large variations between projects like DBPEDIA whose content is automatically generated from Wikipedia, CYC or SUMO, which are both hand written, but do not adopt the same strategy to select the knowledge to represent.

These metrics do not reflect adequately the *expressive complexity*, though: in most of these ontologies with a large amount of terms and assertions, taxonomic relations (`isA`) or technical predicates (URI, translations, etc.) account for a large part of the assertions, at the expense of real semantic relations.

Other metrics that include the type of predicates that are used, or the computed DL expressiveness, can be more significant (some are presented in table 3.2, page 66).

Qualitative evaluation of ontologies is also a well studied field. In Staab's *Handbook on Ontologies*, Vrandečić [141] provides a synthesis of qualitative criteria for ontology assessment. He lists eight of them: *accuracy, adaptability, clarity, completeness, computational efficiency, conciseness, consistency,* and *organizational fitness*. Details and relevant literature can be found in Vrandečić's chapter.

Granularity

Amongst the characteristics of ontologies, knowledge *granularity* qualifies the level of details or refinement of the knowledge stored. The level of granularity of a robot's ontology hints on the place of the symbolic layer in the whole robotic architecture: some systems (like OMRKF [128]) go as down as storing SIFT features (*i.e.* a large volume of numerical values) in the ontology. The storage of literal values is indeed a relevant case for robotics. Depending on the representation language, literal values can be naturally represented and processed (common case in logic programming language) or not (storing numerical value, let alone matrices, in OWL is cumbersome and not efficient due to the serialisation to XML).

The issue of the granularity of models can also be partially addressed by splitting the knowledge representation into a geometric level (where all numerical values are stored) and a purely symbolic level. The communication between the two layers is however a complex question.

It must finally be noted that complex robotic systems are likely to use different level of the knowledge depending on the task to achieve, and the granularity of the knowledge should probably be considered as a dynamic property.

2.3 Existing systems for knowledge representation in service robotics

Table 2.4 lists the knowledge representation systems that we have surveyed.

This section first clarify the inclusion criteria, and then briefly presents each of them. At chapter 6, we will consider again these systems, this time as a whole, to build a summary of the fields of knowledge representation that are adequately (or not) tackled by the existing systems.

Project	Category	Authors (Institution)	Project homepage	Programming language	Knowledge model/Logical Formalism	Main reference
ARMAR/Tapas	Formal	Holzapfel, Waibel (Karlsruhe TH)			TFS (Typed Feature Structures)	[51]
CAST Proxies	Ubiquitous	Wyatt, Hawes, Jacobsson, Kruijff (Brimingham Univ., DFKI Saarbrücken)			Amodal proxies	[54]
GSM	Structural	Mavridis, Roy (MIT MediaLab)				[91]
Ke Jia Project	Formal	Chen et al. (Univ. of Science and Technology of China)	www.wrighteagle.org/en	ASP (Answer Set Programming)	ASP	[24]
KNOWROB	Formal	Tenorth, Beetz (TU Munich)	ias.in.tum.de/kb/wiki	PROLOG	PROLOG + OWL-DL	[130]
NKRL	Language	Zarri et al. (Paris Est Créteil Univ.)		NKRL		[114]
OUR-K/OMRKF	Formal	Lim, Suh et al. (Hanyang Univ.)	incorl.hanyang.ac.kr/xe	?	DL + Horn Clauses	[82, 128]
PEIS KR&R	Formal	Daoutis, Coradeshi, Loutfi, Saffiotti (Örebro Univ.)	www.aass.oru.se/~peis	C, CYCL	CycL (1st and 2nd order logics, modal logics)	[29]

Table 2.4: List of surveyed systems. Categories are *Formal* for systems that have a formal underlying knowledge representation, *Ubiquitous* for systems where knowledge is fully distributed, *Language* for languages used as KRS on robots or *Structural* for KRS where knowledge is represented as special data structures.

Survey Inclusion Criteria

Every robotic system has, implicitly or not, some knowledge representation systems. It may range from a simple state vector to an explicit symbolic knowledge base. This survey focuses on the right end of this spectrum: symbolic systems, suited for abstract reasoning.

Besides, we have decided to restraint the set of systems to those actually implemented on robots, and used in semantic-rich environments (*i.e.* dynamic, partially unknown environments with a large range of different entities which may have interactions). The typical scenario that would involve such robots is the *Brownie Scenario* already presented at section 1.1: a service robot in a human-friendly environment like a kitchen.

We have limited ourselves to systems that 1. run on *service robot* (that is, robots that interact with objects in a semantic-rich environment primarily designed for humans), 2. ground the knowledge in the physical world (physically embedded systems able to assess their environment), 3. are able to merge different knowledge modalities, 4. are able of on-line, dynamic knowledge acquisition and reasoning (*i.e.* not simple static databases).

These criteria exclude platforms like DyKnow [49] which are focused on data fusion and knowledge grounding at lower levels.

We have also chosen not to include the GOLOG language and its derivatives [80, 35, 42] in this survey. While several implementations on robots, including service robots, do exist, the focus of this language is on representation and reasoning about actions and situations, and the link with symbolic, abstract knowledge is not explicit.

While classical cognitive architectures like Soar [73], GLAIR [118] or ACT-R have declarative knowledge modules [30] and have been recently used on service robots (see ACT-R/E [60] for instance), they are also absent from this survey because we did not find much references in the literature on knowledge manipulation and representation applied to real-world robotic scenarii for these architectures.

A comprehensive reference on (bio-inspired) cognitive architectures is also available from the BICA Society [126].

2.3.1 ARMAR/Tapas

Tapas is the name of the knowledge representation system and dialogue manager found on the ARMAR III robot [51] for the Karlsruhe Institute of Technology.

Knowledge in Tapas exists as procedural knowledge (plans) and declarative knowledge. The later is split into *lexical knowledge*, *semantic knowledge* and a database of identified objects (with their properties). The *lexical knowledge* contains lexical and grammatical informations about the objects. The *semantic knowledge* is organised into an ontology relying on *typed feature structures* (TFS [23], a formalism originating from the computational linguistics community, and a superset of first-order logic).

Tapas has a strong focus on natural language grounding. It proceeds by generating grammars from properties represented in the ontology to parse and understand dialogue.

Another focus is put on handling unknown words and objects. Tapas provides

routines to recognise unknown entities, and propose and interactive and iterative verbal process to categorise (including adding new categories) those new concepts.

Experiments TAPAS has been used experimentally in a kitchen environment where naive users had to ask the robot for an object and get information about another object.

2.3.2 CAST knowledge model

CAS (*CoSy Architecture Schema*) Toolkit [48] is a comprehensive toolkit aimed at building cognitive architectures for robots through a set of interconnected *SA* (*subarchitectures*). The CAS does not expose a central knowledge base as seen in other works. It represents instead knowledge as unrooted *proxies*. Those proxies are formally defined in [54] as $p = \langle F_p, u_p \rangle$ where F_p is a set of instantiated features (like ϕ_{red}^{Colour}) and u_p a *proxies union* that form an equivalence class corresponding to one entity.

A union of proxies forms a global amodal representation of an entity, that can be explicitly shared and manipulated. Being not centralised, the knowledge model can be qualified of *ubiquitous*. Furthermore, knowledge source in the CAS architecture is tightly bound to the on-line grounding process (be it grounded in perception or in dialogue). While nothing seems to prevent it, no *a priori* knowledge (including common-sense knowledge) is used.

Knowledge sharing is ensured by the event mechanism of CAST: modules can monitor proxies for alteration by other modules. Jacobsson et al. mention how this can apply to reinforcement learning: the vision module creates a proxy for an orange object. This proxy get monitored by a learning module. In parallel, the proxy is bound to an union by the natural language understanding module that add new a feature like *"this object is a fruit"*. The learning module is called back, and can add this new information to its model.

In the presented implementation, the CAST knowledge model does no allow for effectively representing actions or temporal information.

Knowledge Acquisition Several techniques for knowledge acquisition have been explored within the CAST framework. Cross-modal knowledge fusion [46] is well studied, and the interaction with natural language processing [65, 64] is a particular emphasise of the project.

In [45], Hawes et al. also explore *curiosity* mechanisms in the context of spatial representations with the robot *Dora*.

Experiments CAST has been used in several experiments, including table-top manipulation (with a focus on language understanding) and more recently on the Dora robot [45] for indoor exploration.

2.3.3 GSM

GSM (for *Grounded Situation Model*) [91] is a knowledge representation system primarily built to "facilitate cross-modal interoperability", especially in the context of verbal

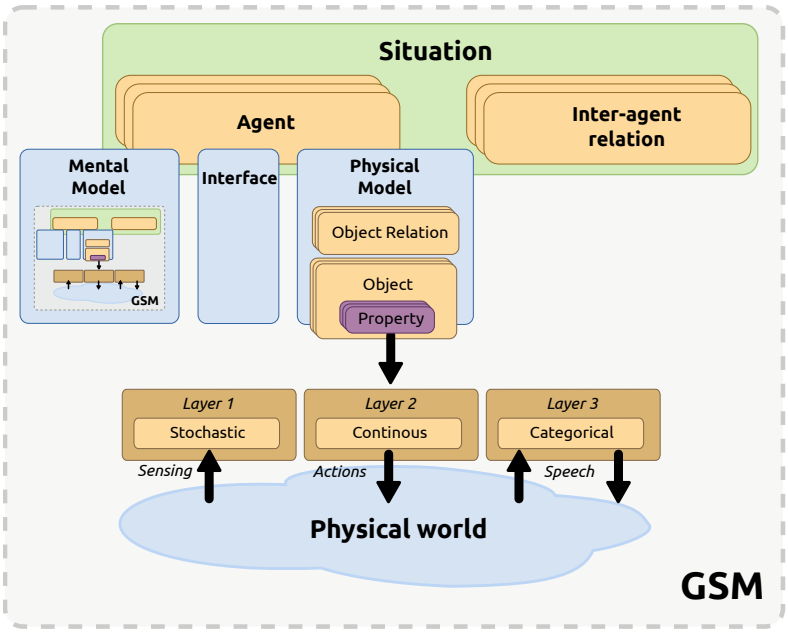


Figure 2.6: Simplified hierarchical structure of the Grounded Situation Model, based on [91].

interaction with a robot.

GSM does not rely on any formal language but rather on a layered data structure (figure 2.6) that organises the surrounding world into agents and relations between agents. Each agent (any animate or inanimate object) is attached to a physical model (made of *body parts* that have properties like their position, color, etc.) and a mental model (which is a recursively embedded GSM, thus allowing a sort of theory of mind).

Properties are represented in three layers: a stochastic representation, close to sensory percepts, a *continuous single-valued* encoding of the stochastic model, and a discrete, categorical model.

One notable feature of GSM is the *bidirectionality* of the grounding process: not only sensor percepts are abstracted into categories suitable for human conversation, but human utterance (like "There is a red ball in the center of the table") can also be turned into property descriptions. This basically enable the knowledge representation system of the robot to *imagine* entities.

GSM also features several strategies for managing time and events. *Moments* are created by storing timestamped snap-shots of GSM, and *event classifiers* allow to define and detect events.

Experiments GSM has mostly been tested on table-top manipulation and interaction tasks (a "conversational helping hand" as stated by the authors) implemented on a 7-DOF arm equipped with force feedback, cameras for blob tracking and speech recognition (Sphinx4). Mavridis and Roy provide in addition an in-depth analysis of the performance of GSM by the mean of a standard psycholinguistic test, the *Token test* [31].

2.3.4 Ke Jia Project

The Ke Jia project [24] integrates on a mobile platform a knowledge representation language with natural language processing, task planing and motion planing.

Knowledge representation relies on *Action Language C*, itself based on *Answer Set Programming* (ASP) [38]. These languages, that are syntactically close to Prolog, are based on *stable models* of logic programs, and support non-monotonic reasoning. Default and non-monotonic reasoning has been especially researched within the Ke Jia project for symbolic task planing [57] and underspecified natural language processing.

Amongst other features, the natural language processing capabilities of the system support acquisition of new logical rules at run-time.

Experiments The Ke Jia robot has been demonstrated in several tasks involving human-robot interaction with natural language. These tasks include a task with multiple *pick & carry* that are globally optimised, naive physics reasoning via taught rules or more complex scenarii with the robot delivering drinks, taking into account changing and mutually exclusive preferences of users.

2.3.5 KnowRob

KNOWROB [130] is an integrated knowledge management system developed at the Technical University of Munich. It is build as a set of modules (figure 2.7 organised around a core reasoning system written in Prolog. This core module interfaces through Java/Prolog or C/Prolog APIs with external modules.

Extension modules can plug into the system to provide specialised reasoning capabilities or interfaces to external data sources, *e.g.* to read object detections from the vision system. These modules operate on the level of instances (ABox).

Knowledge model KNOWROB can load OWL ontologies, and the *KnowRob-Base* ontology is provided as a common-sense ontology, with a focus on household and kitchen domains. KNOWROB also store and reason on introspective knowledge through the *Semantic Robot Description Language* [68] that allow to represent symbolically the capabilities of the robot, and is used for planning.

Reasoning Techniques Amongst the notable KNOWROB extensions, PROBCOG [55] is an effort to provide probabilistic reasoning based on bayesian networks, integration with naive physics reasoning has been studied [67], automatic parsing of Web resources in semi-natural language has been also experimented with [99].

Grounding KNOWROB offers a mechanism called *computables* that allow to evaluate certain predicates by calling external dedicated functions (for instance, the valuation of a proposition like ⟨`object1 isOn object2`⟩ is computed when required by calling a specific geometric reasoning module). In combination with Prolog's lazy evaluation strategy, this supports a good scalability.

Computables rely on various subsystem to evaluate. In particular, it relies on the CoP framework [62] and *semantic maps* [19] for the recognition of objects and environment.

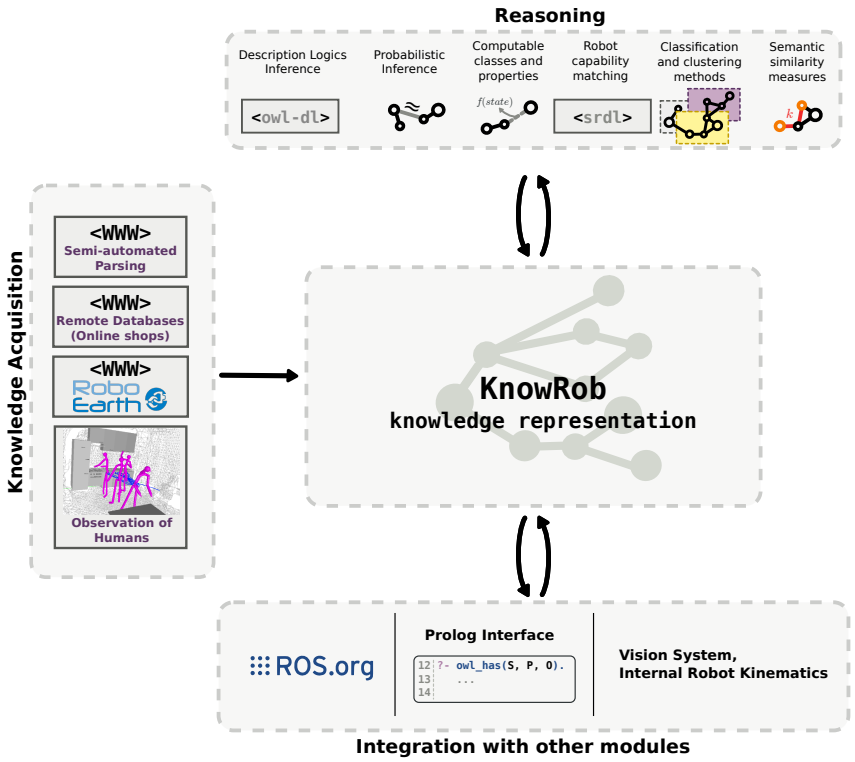


Figure 2.7: Overview of the KNOWROB framework, taken from [129].

Experiments KNOWROB has been deployed on several scenarii at the Technical University of Munich, on the PR2 robot and on a 2-arm custom mobile manipulator in the scope of the "assistive kitchen" [16] project. These experiments include retrieval and automated parsing of recipies from the Web, retrieval and manipulation of various kitchen tools, cooperation between two robots.

2.3.6 NKRL

NKRL stands for *Narrative Knowledge Representation Language*. While this language is developed since a long time by Zarri [147, 148], recent research direction include application to the robotic field [114]. NKRL is not *per-se* a knowledge representation system, as it is primarily a language. However, it is used as the representation and reasoning mechanism for robots by Sabri et al.

Knowledge representation The NKRL language semantics are stored in two ontologies: an ontology of concepts Ω and an ontology of events Ψ. The ontology of events is made of action or situation templates. Templates are a set of predicates (MOVE, PRODUCE, RECEIVE, EXPERIENCE, BEHAVE, OWN and EXIST) associated to thematic roles. Grounding and reasoning with NKRL is based on template matching.

Experiments The main scenario of development for NKRL-based robots is the Smart Home and monitoring of eldery people. Knowledge acquisition partially rely on

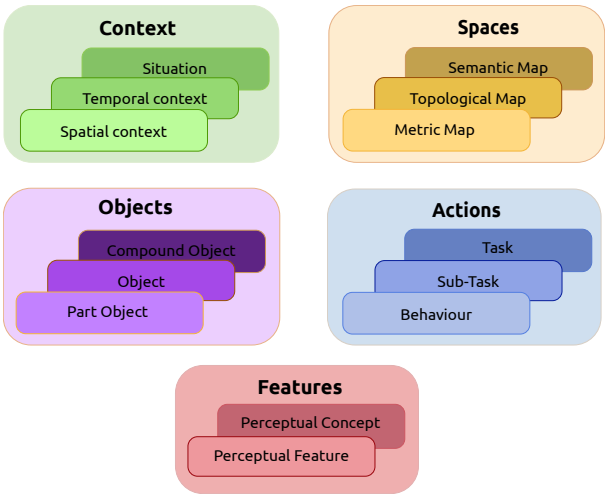


Figure 2.8: OUR-K organises knowledge into fives *classes*, each composed of *levels*. Figure based on [82].

ambient intelligence (RFID, pressure sensors in the chairs, etc.). The scenario is still being implementated.

2.3.7 OUR-K and OMRKF

The Ontology-based Unified Robot Knowledge [82] (OUR-K) framework, successor of the Ontology-based Multi-layered Robot Knowledge Framework [128] (OMRKF), is a knowledge representation system based on five inter-related *classes* of knowledge (figure 2.8). It proposes a layered approach to knowledge representation that allows to integrate the grounding process to the knowledge representation process. OUR-K knowledge model is implemented with a mix of Description Logics for the concept hierarchies and Horn clauses.

Each level of knowledge is build as three stages of ontological realization: a *meta-concept* (the level itself, like "temporal context", "behaviour" or "object feature"), a taxonomy of concepts inside this level (for instance $cup : Object \sqsubseteq tableware : Object$) and an instantiation of the taxonomy ($cup1 : cup$).

Representation The environment is represented in OUR-K in the $spaces : Model$ knowledge level as a classical three layers mapping (metric, topological and semantic maps). Objects (in $objects : Model$) are localised in $spaces : Model$ through Voronoi nodes.

The knowledge class $Context$ proposes an explicit statement of spatial context (mostly geometric relations between objects), temporal context and a more general *high-level* context, inferred from spatial and temporal contexts.

Finally, the $Activity$ knowledge class store compound actions in a HTN-like structure, exploited at run-time by a planner.

Experiments Experiments conducted with OUR-K and OMRKF include finding kitchen objects and reporting about their state to a human. This experiment also

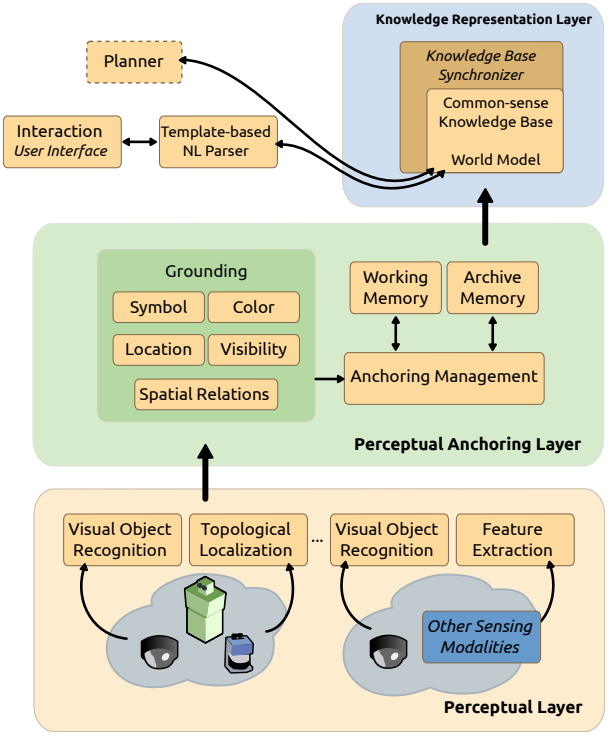


Figure 2.9: The PEIS knowledge representation system, taken from [29]

shows how OUR-K can deal with objects only partially matched by their descriptor by introducing a $candidate()$ function.

2.3.8 PEIS KR&R

PEIS ECOLOGY [115] is a software *ecosystem* that aim to binds autonomous robotics with ambient intelligence (network of sensors). *PEIS* stands for *Physically Embedded Intelligent System*: every robots or intelligent device in the environment is abstracted as a PEIS.

Each PEIS physical component is running a *PEIS Kernel* instance. Communication between instance relies on a custom P2P communication protocol.

The PEIS architecture allows for adding new abilities through software components sharing the common *tuple space*.

We consider here the semantic layer [29], referred as *PEIS KR&R*, that includes symbolic representation and reasoning.

Knowledge model The PEIS Knowledge representation system relies on the RESEARCHCYC and CYCL language to represent knowledge. The CYCL language allows to represent first order logic sentences and has extensions for modal logics and higher order logics.

As a system relying on CYCL, contexts can be expressed as *microtheories*: the truth or falsity of a set of statement depends of the *microtheory* in which these statements are evaluated.

The PEIS KR&R system is deeply integrated to the general PEIS Ecology *smart* environment. Figure 2.9 gives an overview of the interactions between PEIS knowledge processing layers.

Knowledge Acquisition The primary source for knowledge acquisition is perception. The PEIS ecosystem provides a SIFT-based object recogniser used in conjunction with ceiling cameras for object localisation. Other perceptual modalities are available (like human tracking, ambient environment monitoring).

A template-based natural language parsing system may also be used to add new assertions to the system.

The system can ask the human for help to disambiguate between concept names.

Anchoring Daoutis et al. formalise the issue of anchoring as finding a *predicate grounding relation* $g \subseteq \mathcal{P} \times \Phi \times D(\Phi)$, where $\mathcal{P}$ is a set of predicate symbols, Φ a set of percept's attributes, and $D(\Phi)$ the domain of these attributes.

In the current implementation, object category (returned by the SIFT classifier), color, location, spatial relations (both topological – *at*, *near* – and relative to the robot – *left*, *behind*, etc.) and visibility are the five classes of extracted attributes.

Integration in the robot architecture The PEIS framework offers through the *PEIS middleware* a practical way to insert a new component into the shared *tuple space*. Thus, the KR&R module can be seamlessly integrated into the PEIS ecosystem.

Experiments Experiments involving PEIS take place in a Smart Home environment (*PEIS Home*). The implemented case studies explore dialogue-based interaction with the robot about known objects.

2.4 An interface for knowledge manipulation in robotics

During the preparation of the thesis, discussions with several people involved in knowledge representation (namely Dominik Jain, Lars Kunze, Michael Beetz) have led to the draft of a generic API for knowledge access and exchange between robotics components.

This section presents this effort of standardisation that is (partially) implemented by the ORO server, presented in the next chapter.

2.4.1 Rationale and general considerations

The original idea comes from the acknowledgement that more and more software components for robotics want to store or use symbolic data. Since established international efforts at defining standard for inter-component communication like ROS have already proved their usefulness, one single API for different knowledge representation and management systems could be equally useful.

The API is designed for robotics (even if probably useful in other contexts): it aims to be simple and practical for clients by focusing on core knowledge operations (addition of knowledge, retraction, querying) with consistency constraints; it explicitly supports uncertain knowledge and multiple models (modality); it makes clear how knowledge is added or retracted with explicit policies.

We have attempted to design it in a way that do not restrict expressiveness (any logical sentence that can be expressed in the logic of predicates, with a probabilistic extension, can be manipulated by the API), and a simple extension mechanism should permit future evolutions in a backward compatible way.

Besides facilitating exchange of knowledge contents between systems by ensuring one standard formalism, another outcome of the adoption by several KRS of this API is that it allows easy switch between semantic engines (and thus benchmarking and sharing of unit-tests).

This API was developed with Prolog-based knowledge systems, Description Logics-based knowledge bases and Markov networks in mind, and should cover as well other systems related to predicate logics (with or without a probabilistic extension).

Besides standard operations on axioms and taxonomy, the API aims to cover:

- probabilities associated to statements
- management of several models
- explicit policies to add, retract or, more generally, alter knowledge (for instance, to guarantee consistency when adding knowledge)
- specific, implementation-dependent, extensions through the `special` method.

Implementations are not always expected to cover to whole API, but must have a predicable behaviour when a part of the API is not implemented. In particular, the API makes no assumptions on implementations regarding:

- the actual supported expressiveness (the API allows to express general first-order logics statements, but the underlying implementation may support only a subset, for instance, Description Logics)
- Closed-world assumption vs Open-world assumption
- Reasoning capabilities

2.4.2 The Knowledge API

The API is divided in five parts:

1. Methods related to service management,
2. Methods related to knowledge alteration,
3. Methods related to knowledge querying,
4. Methods related to models manipulation and finally

Key	Values	Meaning
`method`	`add` *(default)*	the statements are added to the knowledge base, without ensuring consistency.
	`safe_add`	the statements are added only if they (individually) do not lead to inconsistencies.
	`retract`	the statements are removed from the model. Associated probabilities are discarded.
	`update`	Updates objects of one or several statements in the specified model. If the predicate is not inferred to be *functional* (*i.e.* , it accept only one single value), behaves like `add`.
	`revision` or `safe_update`	Updates objects of one or several statements in the specified model if it does not (individually) lead to inconsistencies. If the predicate is not inferred to be *functional* (*i.e.* , it accepts only one single value), behaves like `safe_add`.
`model`	`all` *(default)*	all existing *models* (section C.4) are impacted by the change.
	a valid model id or a set of valid model id	only the specified model(s) are impacted

Table 2.5: Knowledge revision policies.

5. Methods related to taxonomy walking.

Parts 2 and 3 are the two main parts, involved with knowledge manipulation.

Knowledge Alteration Methods in part 2 are build around the generic `revision` method, that takes as parameter a set of logical propositions and a policy.

A policy is represented as a set of `(key, value)` pairs whose possible values are presented in table 2.5.

Knowledge Querying The main method that allow for knowledge retrieval is `find`. A `find` query is build as a set of partial statements (*i.e.* , statements with named or anonymous unbound terms) that form a pattern. It returns statements matching the pattern.

"Shortcut" methods are offered by the API for common operations (adding/retracting a statement, checking if a statement exists, etc.). Where relevant, probabilistic versions of the methods are also defined.

The complete API reference is provided in Appendix C.

Chapter recap

That concludes the chapter on Symbolic Knowledge Representation for robotics.

In that chapter, we have first discussed a definition of *knowledge* in our context of service robotics and human-robot interaction. We have presented several references from the literature regarding the identification and classification of prominent features of knowledge representation systems.

We have then introduce a comprehensive typology of such features, that comprises of about fifty concepts sorted into six main categories: features related to knowledge expressiveness, features related to representation techniques, features related to reasoning, features related to acquisition and grounding of knowledge, features related to the integration of a KRS into a larger robotic architecture, and finally, features that characterise the represented knowledge itself. Each of the fifty concepts has been briefly presented with references to the literature.

Finally, we have surveyed eight systems for knowledge representation in service robots and underlined their main strengths.

The next chapter introduces ORO, a tenth KRS that we have designed and implemented during the thesis preparation.

Chapter 3

The OpenRobots Ontology Framework

This chapter introduces the *OpenRobots Ontology* server and its common-sense knowledge base.

We present here the functional description of `oro-server`, and detail its knowledge model. Its actual implementation is discussed in the next chapter.

We also present the *OpenRobots Common-Sense Ontology* that contains most of the knowledge at hand when the robot starts.

3.1 Functional overview

We have adopted a centralised approach for knowledge management called ORO [75]. The platform is designed as a central knowledge storage service implemented as a server where the robot components can add or query statements at run-time. Figure 3.1 illustrates the main functional components of ORO.

At the core, ORO is build around the OpenJena[1] ontology management library, connected to the Pellet[2] reasoner.

A front-end accepts and manages connections to clients. The clients' requests are processed by a set of internal modules: basic operations on statements, but also higher cognitive and human-robot interaction related features are available. External plugins can also be added via a specific extension mechanism.

Besides acting as a facts database, the ORO platform exposes several functions: operations on knowledge statements relying on inference (through a continuous first-order logic classification process), management of *per-agent* symbolic models, categorisation of sets of concepts and profiles of memory (that enable the robot to "forget" about some facts).

ORO also provides an event mechanism that allows components to be triggered when specific events occur. A component can for instance subscribe to events of kind [**?agent** *isVisible true*, **?agent** type Human]. As soon as the perception layer detects a human in the robot's field of view and accordingly updates the knowledge base, the executive layer is triggered. The event framework also takes advantage of the inference

[1] `http://www.openjena.org`
[2] `http://clarkparsia.com/pellet`

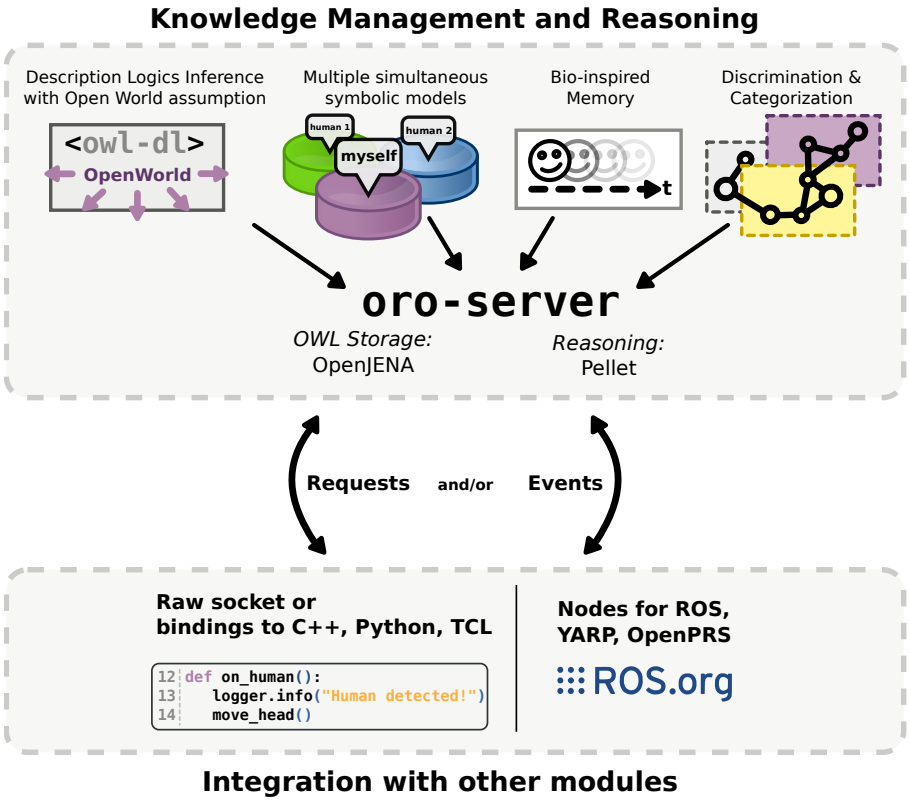


Figure 3.1: Overview of the ORO architecture.

capabilities of ORO. Thus an event can be indirectly triggered if its triggering conditions can be inferred to be true.

3.2 The ORO knowledge model

3.2.1 Expressiveness

Unlike systems relying on logic programming, ORO is purely based on Description Logics: the ORO knowledge model is based on RDF triples (*i.e.* exclusively binary predicates). Triples ⟨`subject` *`predicate`* `object`⟩ are the atoms of knowledge for ORO.

Knowledge in the ORO server is represented with OWL2. As already mentioned at section 2.2.4, the available constructs include:

- inheritance relations, *e.g.* :

  ⟨`Bottle` *`subClassOf`* `Container`⟩ $\models$ *all bottles are containers*,

- property axioms
  - specification of predicates' domain and range, *e.g.* :

    ⟨`thinksAbout` *`domain`* `IntelligentAgent`⟩ $\models$ *only intelligent agents can think*,
  - cardinality constraints (including `allValue`, `someValue`, `hasValue`),

  - property characteristics (symmetry, transitivity, reflexivity, antisymmetry, etc.)

- class restrictions like:

  `Bottle` ≡ `Artifact` **that** (`hasShape` **value** `cylinderShape`)

- set operations like:

  `Color` ≡ **unionOf**(`blue`, `green`, `orange`, `black`...)

DL-safe[3] SWRL (*Semantic Web Rule Language*) rules are also supported. For example: `looksAt(?agt, ?obj)` ∧ `pointsAt(?agt,?obj)` ⇒ `focusesOn(?agt, ?obj)`

The formal expressiveness of the current version of the ORO common-sense ontology (commit `19f1fcf27` in the public repository[4]) is $\mathcal{SROIQ}(\mathcal{D})$, which correspond to the OWL2 language full expressiveness (the appendix A presents the usual naming conventions of Description Logics expressiveness).

Table 3.2, page 66 gives quantitative details on the type of axioms used in the ORO common-sense ontology.

Reification Since RDF triples constrain to binary predicates, *reification* is often required to express n-ary relations. For instance, the relation *A gives object B to C* can not directly be represented in RDF. This relation is reified as {⟨`act1` *type* `Action`⟩, ⟨`act1` *performedBy* `A`⟩, ⟨`act1` *actsOn* `B`⟩, ⟨`act1` *receivedBy* `C`⟩}. As long as the instance `act1` exists in the knowledge base, the original relation *A gives object B to C* is considered to hold. This kind of reification is common in the ORO knowledge model.

Reification can also take place at a meta-level (this is the level usually intended by the term reification): a triple ⟨`subject` *predicate* `object`⟩ can be itself reified in {⟨`stmt1` *type* `Statement`⟩, ⟨`stmt1` *hasSubject* `subject`⟩, ⟨`stmt1` *hasPredicate* `predicate`⟩, ⟨`stmt1` *hasObject* `object`⟩}. This level of reification allows to characterise the knowledge atoms themselves, for instance to specify when the atom was added. The section on memory management in ORO server, below, gives examples of usage of this meta-cognition feature.

Note that in traditional logical programming like Prolog, reification is rarely strictly required since no constraints hold on the arity of predicates. To store the date of creation of a facts, one could simply add it as a supplementary argument of the predicate[5].

Open World Assumption Following the OWL language model and as mentioned at section 2.2.2, the ORO knowledge model makes the open world assumption.

This allows to easily represent that a fact is unknown (by simply not stating it in the knowledge base), but also requires to carefully explicit what the entities are and are not.

[3] In DL-safe rules, variables bind only to explicitly named individuals in the ontology.

[4] Clonable from `http://git.openrobots.org/git/robots/oro.git`

[5] In the case of time representation, however, reification — or, in the case of logic programming, second order logic — often takes place through the *fluents* mechanisms, see section 2.2.3

3.2.2 Special representation techniques

Representation of alternative knowledge models

As pictured in Figure 3.1, ORO stores independent cognitive models for each agent it interacts with. When the ORO server actually identifies a new agent (or infers that some instance is an agent), it automatically creates a new, separate, in-memory OWL model for that agent. Then, different robot components, like execution control or situation assessment, may store the agents' beliefs in separate, independent models. All knowledge processing functions in the robot's primary model are equally available in every agent's model, which allows us to store and reason on different (and possibly globally inconsistent) models of the world.

Each of these models is independent and logically consistent, enabling reasoning on different perspectives of the world that would otherwise be considered as globally inconsistent (for instance, an object can be visible for the robot but not for the human. This object can have at the same time the property `isVisible` *`true`* and `isVisible` *`false`* in two different models).

We present at section 4.4.1, page 82, a 3D real-time environment, SPARK, that allows to compute on-line several symbolic properties that are dependent on the perspectives.

Theory Of Mind and contexts By maintaining independent mental states for each agent it interacts with, we consider the robot to be endowed with a simple *theory of mind* [116]: the robot can explicitly model the beliefs of its interactors, it expose them to the control architecture, and the same set of cognitive abilities are available on these secondary model as on the main model: reasoning, inconsistencies detection, events, etc.

Proper *false beliefs* experiment, similar to the Sally and Ann experiment presented in the previous chapter, has been recently conducted with ORO by Mathieu Warnier, as reported in [145]: in this experiment, two humans observe a table with several objects, then one leaves while the other one moves around some objects. This leads to two different set of beliefs on the world, which the robot explicitly stores and updates when necessary (if the human comes back and check the table, for instance).

These multiple models can also be viewed as different *interpretive frames*, allowing the robot to interpret the same reality from different points of view. In this sense, each model carries a context of interaction. In chapter 5, we present how such agent-dependent contexts are used by a natural language processor to make sense of user sentences from his/her point of view.

Multi-lingual support

The RDF specification supports internationalisation by the way of *language tags*: plain literals may have an optional language tag (taken from the standard RFC-3066) that tells in which human language the literal is expressed.

ORO benefits from this mechanism, and can be configured to use a specific language as default. When an language is explicitly selected, the translated labels of concepts (when available in the underlying ontology) are used instead of the default English ones.

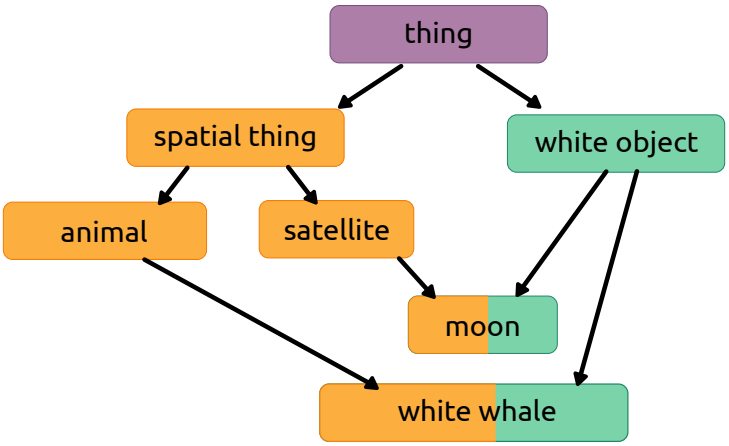


Figure 3.2: Sample taxonomy to illustrate *common ancestors* algorithms.

Since only the labels (*i.e.* the human-friendly name of the concepts) are subject to translation, changing the default language of the ORO server has no semantic impact: entities in the ontology always refer to same concepts. Same inferences are drawn, same connections to knowledge sources are made, etc. The strength of semantic approaches is here well illustrated.

3.2.3 Reasoning techniques

Standard inference services

As explained in the next chapter, we use the Pellet open-source reasoner to reason on the knowledge base. This enables to expose several standard inference services: consistency checking, concept satisfiability, classification and realisation (the most specific classes that an individual belongs to).

In case of inconsistency in one of the knowledge models stored by ORO, an explanation of the inconsistency is proposed, in a human-readable form, for debugging purposes. The automatic exploitation of the explanation by the robot executive controller is yet to be developed.

Grounding, classification and discrimination algorithms

ORO server implements several algorithms to identify similarities and differences between concepts (classes or instances) [110]. The main ones are presented in this section.

Common and first different ancestors The *Common Ancestors* algorithm (algorithm 3.2.1) returns the classes that are the "first" common super-classes of the two concepts.

Algorithm 3.2.1: COMMONANCESTORS($concept1, concept2$)

$\mathcal{I} \leftarrow$ SUPERCLASSES($concept1$) $\cap$ SUPERCLASSES($concept2$)
return $(c \in \mathcal{I} | \text{SUBCLASSES}(c) \cap \mathcal{I} = \emptyset)$

Taking the taxonomy in figure 3.2 as example, the common ancestors for the pair `{white whale, moon}` are `{spatial thing, white object}`, *i.e.* the set of classes that belong to the intersection of the super-classes of both the concepts and that have no sub-classes in this intersection.

The common ancestors are useful to determine the most precise class(es) that include a given set of individuals.

Algorithm 3.2.2: FIRSTDIFFERENTANCESTORS($concept1, concept2$)

$$\begin{cases} \mathcal{C} \leftarrow \text{COMMONANCESTORS}(concept1, concept2) \\ \mathcal{S} \leftarrow \text{SUPERCLASSES}(concept1) \cup \text{SUPERCLASSES}(concept2) \\ \textbf{return } (\forall c \in \mathcal{C}, \text{DIRECTSUBCLASSES}(c) \cap \mathcal{S}) \end{cases}$$

The *First Different Ancestors* algorithm (algorithm 3.2.2) returns the list of direct sub-classes of the common ancestors. They are intuitively the most generic types that *differentiate* the two concepts. In the taxonomy figure 3.2, two instances `a` and `b` of respectively `white whale` and `moon` have as first different ancestors the two sets `{animal, satellite}` (subclasses of ancestor `spatial thing`) and `{white whale, moon}` (subclasses of ancestor `white object`).

Clarification Algorithm During interactions with other agents, the robot is often required to figure out which individual correspond to a description like "red object", "a bottle", "a book larger than this other one", etc. This is a key part of the grounding capability.

Clarification and discrimination algorithms are based on what we call *descriptors*: descriptors can be properties of individuals, either acquired by the robot are statically asserted in a common-sense ontology. They are also the result of other reasoning algorithms like the *Common Ancestors* and *Different Ancestors* algorithms presented above. We shall see later how symbolic knowledge is first acquired from geometric reasoning or natural language processing, and we consider in this section that the *clarification* process is based on an established ontology, like the sample proposed in figure 3.3.

Based on this ontology and a given partial (or complete) description of an object (list of attribute-value pairs), the robot is able to identify the referred object the following way (Algorithm 3.2.3). First it obtains all objects that fulfil the initial description by querying the agent model in the knowledge base. Based on the result it either succeeds (obtains one single object), fails (no object with that description could be found) or obtains several objects. In this last case, a new descriptor is added (mark 1) to the initial description and the process starts over again until all possible descriptors have been added.

Failure occurs hence in two cases: when the description does not match any object from the robot's knowledge, either because the robot's knowledge is incomplete (the human refers to an unknown descriptor or descriptor value), or when a set of candidates could not get successfully discriminated with the available descriptors.

Two options are available to add new descriptors: directly asking the human for more information, or automatically searching a new attribute and ask the human for

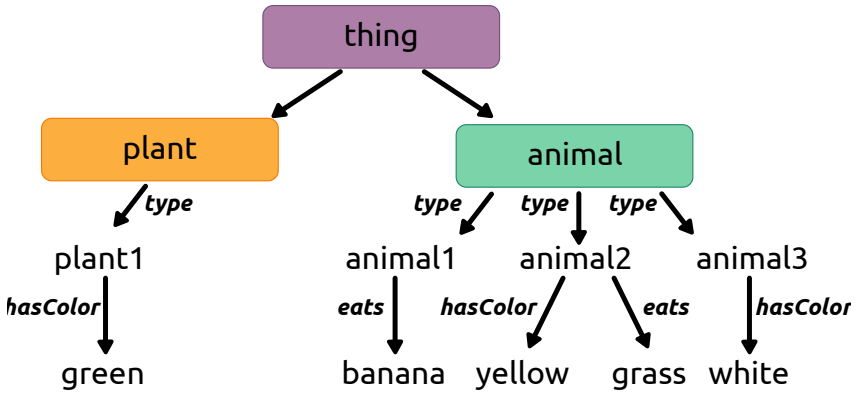


Figure 3.3: Sample ontology to illustrate the discrimination routines. `plant1` is an instance of `Plant` and `animal[1-3]` are instances of `Animal`.

its value. In the latter case, we need to automatically find the best discriminant for the current list of objects being evaluated (*candidates* in the algorithm).

Algorithm 3.2.3: DISCRIMINATION(*description*, *agent*)

$$
\begin{cases}
candidates \leftarrow \text{GETOBJECTFROMDESCRIPTION}(description, agent) \\
\textbf{if } |candidates| = 1 \\
\quad \textbf{then return } (candidates[0]) \\
\\
\quad \textbf{else if } |candidates| = 0 \\
\quad \textbf{then output } (\text{No object found!}) \\
\\
\quad \textbf{else} \begin{cases} description \leftarrow \text{ADDDESCRIPTOR}(description, agent) \\ \textbf{return } (\text{DISCRIMINATION}(description, agent)) \end{cases}
\end{cases}
\tag{1}
$$

Finding a discriminant We have implemented a set of semantic categorisation functions in ORO. One of them consists in looking for discriminants, *i.e.* descriptors that allow a maximum discrimination among a set of individuals.

We distinguish two types of discriminants. *Complete* discriminants are those attributes (or properties) that totally discriminate the set of individuals. In other words, properties whose values can uniquely identify those individuals. However, they are not always available. First, because two or more individuals may share the same value, and second, because not all individuals may share the same properties. Thus, *partial* discriminants are those that split at best the set of individuals in different subsets based

on some criteria.

Algorithm 3.2.4: GETDISCRIMINANT($individuals$)

```
P ← ONTOLOGY.GETPROPERTIES(individuals)
P̂ ← ∅
for each p ∈ P
  do { n_ind ← NBINDIVIDUALSWITHPROPERTY(p)        (1)
     { n_val ← NBDIFFERENTVALUES(p)                (2)
     { if n_val > 1
       then P̂ ← APPEND([p, n_ind, n_val])          (3)
RANK(P̂)                                            (4)
return (P̂[0][0])
```

The algorithm to determine the type of discriminant available (Algorithm 3.2.4) has the following steps (to better follow it, we base its description on the ontology example illustrated in figure 3.3). We search a discriminant for the following individuals: `plant1`, `animal1`, `animal2` and `animal3`. First we obtain the direct properties and classes for all the individuals, *i.e.* we do not consider all the hierarchy of properties and classes (in the example, `plant1` has two super-classes (`Plant` and `Thing`), but we only take the most direct one (the class `Plant`)). Next, we compute the number of individuals per property (mark 1) and the number of different values for that property (mark 2). For instance, for the property `type`: all the four instances have a type, $n_{ind} = 4$, and this property has two possible values (`Plant` and `Animal`), $n_{val} = 2$. If $n_{val} > 1$ (in other words, if not all individuals have the same value), then we consider that property as a potential discriminant (mark 3). Finally, we rank (mark 4) the list of potential properties following two criteria: the number of individual occurrences (*i.e.* we maximise the coverage of that property) and the values occurrences (*i.e.* the more distinct values, the better). The best discriminant corresponds to the first element of the sorted list. In other words, the class with higher number of occurrences and more variety in it. If several properties are equal, we return all of them.

In our example, the algorithm would return the type as best the partial discriminant. If we only consider the instances of the class `Animal`, it would return two properties equally discriminant: `hasColor` and `eats`.

We use this algorithm in particular for interactive concept grounding. We will detail this approach at chapter 5.

Memory

The ORO server offers a mechanism to mimic simple forms of biological memory. When new statements are inserted in the knowledge base, a *memory profile* is optionally attached to them.

Three such profiles are predefined: `short term`, `episodic` and `long term`. They each correspond to a different lifetime for the statements (respectively 10 seconds, 5 minutes and no time limit). After this duration, the statements are automatically removed from the knowledge base.

Name	Example
rdfs:subClassOf	⟨Human *subClassOf* Agent⟩
rdfs:subPropertyOf	⟨hasColor *subPropertyOf* hasFeature⟩
rdfs:domain	⟨thinks *domain* IntelligentAgent⟩
rdfs:range	⟨name *range* string⟩
owl:inverseOf	⟨sees *inverseOf* seenBy⟩
owl:FunctionalProperty	⟨age *type* FunctionalProperty⟩
owl:TransitiveProperty	⟨isAbove *type* TransitiveProperty⟩

Table 3.1: Some of the properties and classes defined in RDFS and OWL that allow to define the semantics and relations of terms within an ontology.

This approach is limited. In particular, *episodic* memory primarily refers to the semantics of the statements (that is expected to be related to an event) and not to a specific life duration. We discuss at the end of this work possible improvements.

Active Concepts We rely however on this short term memory for a particular use-case: *active concept*. Some modules, like our natural language processor (described at chapter 5), use the `short term` memory profile to mark for a few seconds important concepts that are currently manipulated by the robot. For example, if a human asks the robot: "Give me all red objects", the human, the `Give` action, and each red objects that are found are successively marked as *active concepts* by inserting statements like ⟨human *type* ActiveConcept⟩ in the short-term memory (which can be considered, in this case, to be a working memory, as defined at section 2.2.3). We use this feature to give a (visual) feedback to the users (section 4.3.2)

Knowledge structure alteration and learning

In the ORO server, the knowledge structure (*TBox*) is purely declarative and asserted as regular statements (like ⟨Location *subClassOf* SpatialThing⟩), following the RDF Schema (RDFS) and OWL language constructs.

In complement to the constructs already presented at section 3.2.1, table 3.1 lists the main properties and classes defined in RDFS and OWL that allow to describe the ontology structure. These constructs can all be inserted at run-time to alter the knowledge model of the server. They are immediately taken into account by all reasoning process taking place after this point.

While we do not claim to have addressed the issue of learning in general, TBox alteration coupled with language processing features, enables to implement specific learning mechanisms. For example, one can teach the robot that cats are animals by processing a sentence like "Cats are animals" into ⟨Cat *subClassOf* Animal⟩ and then adding it at run-time into the knowledge base. From that points, all entities asserted or inferred to be cats will be as well inferred to be animals.

The case study *Point & Learn* (section 6.2.2) presents some experimental results in this domain.

Expressiveness	$\mathcal{SROIQ}(\mathcal{D})$
Axioms	961 (annotations: 359)
Class count	108
Object property count	78
Data property count	13
Individual count	21
Functional properties	19
Inverse properties	9
Transitive properties	4
Symmetric properties	3

Table 3.2: Statistics on the ORO common-sense ontology.

Fast concept lookup

Because retrieving a concept from its label (for instance, a "location" is the human label for the concept `cyc:SpatialThing-Localized`) is a frequent operation, in particular for language processing application, the ORO server also provides a fast concept look-up mechanism to search for a concept identifier by its label. This also takes into account the chosen language (English, German, French...).

3.3 Knowledge instantiation: the OpenRobots Common-Sense Ontology

The ORO platform is made of the server that we have presented, and a common-sense ontology, the *ORO Common-sense Ontology*.

At start-up, the knowledge model of the ORO server is initialised with a configurable sets of ontologies that build together the initial pool of facts known to the robot: the common-sense ontology and optional, domain dependent ontologies.

Each time a new model is created (typically when a new agent is detected), it is also initialised with the same pool of facts. From the point of view of the robot, this ensure that all the different agents share the same background knowledge.

Usually, the ORO server is started with the common-sense ontology that we present in this section, and one scenario-specific ontology that usually contains the set of individuals with relevant properties needed by the experiment.

The table 3.2 gives a first overview of the extend and expressive level of the ORO common-sense ontology. While the raw numbers of axioms is in no way comparable to large upper ontologies (as presented at chapter 2), the modeling is also more expressive (with the use of second order predicates) than the ontologies that are partially or completely generated automatically.

The ORO common-sense ontology has been designed from two requirements: covering our experimental needs and conforming as much as possible to the OPENCYC upper ontology.

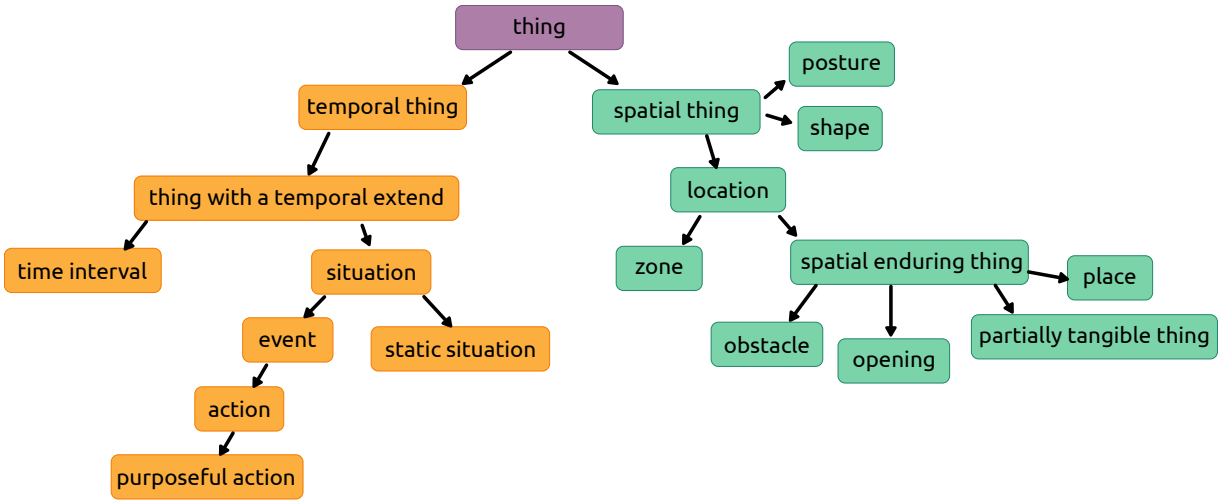


Figure 3.4: The upper part of the ORO common-sense TBox. All these concepts belong to the OPENCYC namespace.

This lead to a bidirectional design process: from *bottom-up* regarding the choices of concepts to model, *top-down* regarding the upper part of the taxonomy. This upper part of the ontology is pictured on figure 3.4. All the classes visible on this figure belong to the OPENCYC namespace (the `cyc:` prefix is omitted).

By *aligning* the upper part of the ontology on OPENCYC (as other KRS, like KNOWROB or PEIS K&R, did) has multiple advantages. First the design of this part of the ontology is generally difficult: it pertains to abstract concepts whose mutual relations comes to philosophical debates. The upper taxonomy of OPENCYC represents a relative consensus, at least within the semantic Web community. Then, because it is a well established project with numerous links to other on-line databases (like Wikipedia or WordNet), the reuse of important OPENCYC concepts ensures to a certain extend that the knowledge stored by the robot can be shared or extended with well-defined semantics. A good example is the concept of *Object*: In everyday conversation, an object is a relatively small physical thing, that can be typically manipulated. Normally, a human is not considered as an object. In CYC, an object has a more precise definition: it is something *partially tangible*. That includes obviously the humans, and actually many other entities that would not be commonly said to be objects (the Earth for instance). Thus the importance of relying on well-defined and standard semantics to exchange informations between artificial systems.

This figure 3.4 also illustrates the fundamental disjunction in the ORO model between *temporal* and *spatial* entities (formally, $(TemporalThing \sqcap SpatialThing)^{\mathcal{I}} = \emptyset$, with $\mathcal{I}$ the *interpretation* of our model).

The class `purposeful action` is the superset of all the actions that are voluntarily performed by the robot (or another agent). Subclasses (like `Give`, `LookAt`, etc.) are not asserted in the common-sense ontology, but are added by the execution controller (in link with the symbolic task planner) and the natural language processor based on what is actually performable and/or understandable by the robot at run-time.

The tree of figure 3.4 (this subset of the ontology is indeed a tree: this has however not to be the case in general, and, as a matter of fact, the TBox of the whole ORO

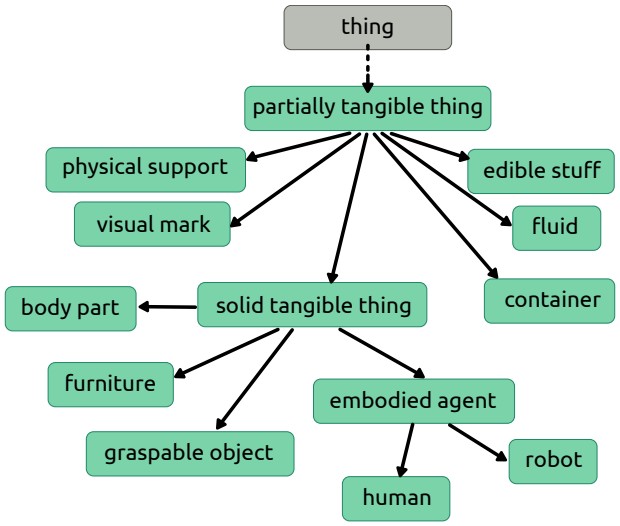


Figure 3.5: TBox of the specialisations of `PartiallyTangible`.

common-sense ontology does not form a tree) is not equally developed at lower levels. We have already briefly mentioned the developments of the actions. The other important part is the descendants of the `partially tangible thing` (what is commonly called an *object*). Figure 3.5 gives more details on this part of the ontology.

This excerpt from the ontology makes the bottom-up design process visible: only few types of *partially tangible* things appear, and only subclasses relevant to the context of service robotics in an human-like environment are present.

Lastly, the ORO common-sense ontology contains several rules and class expressions that encode non-trivial inferences.

The definition of the `Bottle` is a case in point. We already gave a simplified version, here the complete definition:

`Bottle` ≡ `Container` **and** `Tableware` **that** (`hasShape` **value** `cylinderShape` **and** `hasCharacteristicDimension` **only** *`int[>= 0.1, <= 0.3]`*)

If a human informs the robot that a given object is indeed a bottle, the robot can then infer much more on this object. And if the human affirms that a car is a bottle, the robot may question this assertion because of the inconsistent size.

Chapter recap

We conclude here this third chapter. This chapter was focused on the functional and algorithmic presentation of the ORO server, a *semantic blackboard* where robotic modules can write and querying pieces of knowledge.

We have mentioned how multiple mental models can be managed by the server, and we have also presented several active services, like the discrimination algorithms or the management of the memory.

Finally, we have presented the ORO *common-sense* ontology that provide the robot with an initial background knowledge, shared by all the agents.

The next chapter first gives some implementation and technical details about the ORO framework, and then present how ORO is integrated with other components on real robots. In particular, we detail the integration with the geometric reasoning module and the symbolic task planner.

Chapter 4

Implementation and Integration on Robots

4.1 A centralised server-based implementation

The ORO server is a multi-platform command-line application that starts a socket server to which clients can send requests to store or retrieve symbolic statements.

Figure 4.1 gives an overview of the ORO server architecture. The server is build on three layers: a front-end, in charge of the communication with external clients, a set of central modules that either handle incoming requests or provide background processing (like the event monitoring or the memory management), and a back-end, that stores the knowledge models in several parallel pools of RDF triples, one per agent. The back-end provides all the knowledge manipulation features required by the modules including reasoning.

The ORO server is written in Java 6. This choice is due to widespread use of the Java language in the semantic Web community that leads to the fact that most of the RDF/OWL API and reasoners are available as Java libraries. As mentioned previously, the ORO server relies on the Open Jena API for OWL manipulation and on the Pellet reasoner for all reasoning tasks. Java was thus the best candidate to glue them in a robotic-friendly knowledge base.

4.1.1 Front-end

Communication with external components is handled by the server front-end. It was originally build as a YARP node, but was eventually transformed into a simpler and more generic socket interface. The socket connector uses a simple ASCII protocol, presented in figure 4.2. Communication with dedicated robotic middlewares like ROS or YARP is provided by dedicated external clients. Section 4.2.2 briefly presents them.

When a request is received by the front-end, it is parsed, deserialised and dispatched to the module providing the requested service.

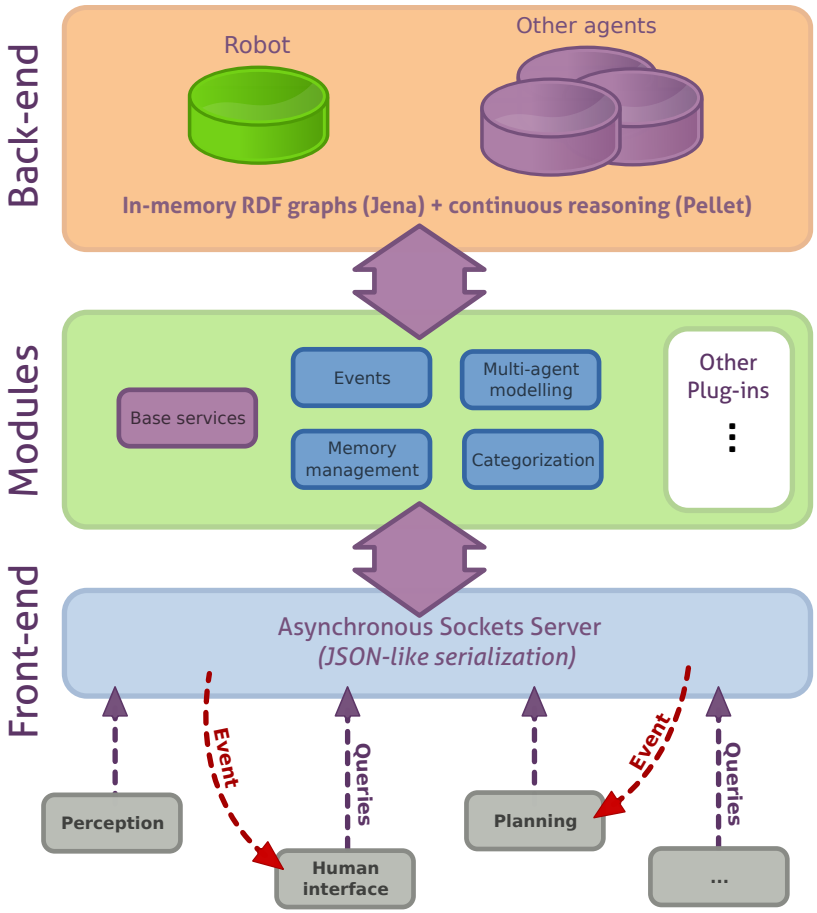


Figure 4.1: The software architecture of the ORO server.

```
method name
[parameter 1]
[parameter 2]
[parameter n]
#end#
```

(a) Client requests

```
ok
[return value]
#end#
```

```
error
[exception name]
[human-readable error]
#end#
```

```
event
event id
#end#
```

(b) Server answers

Figure 4.2: The ORO server protocol. Elements in square brackets are optional. Note that the `ok` and `error` message are synchronous server answers to client requests while the `event` message is produced asynchronously.

4.1.2 Modules

These *modules* are initialised and maintained by the server. They provide the actual features of the server as sets of *services* (like `add`, `getSubclasses`, etc.).

Some modules do not expose any services. They provide instead other form of knowledge management. For instance, the `MemoryModule` is in charge of the application of the memory policies. It discards old statements depending on the memory class they belong to (short term memory, long term memory, etc.).

Regular services (*i.e.* that are actual Java methods) are invoked by the front-end. They process the request and interact with the knowledge pool via the back-end interface.

Plugins The server can be easily extended by the mean of *plugins*. These are JAR files that are loaded at run-time and have access to the exact same internal APIs as regular modules like the event module or the categorisation module. ORO comes with a tool that ease the creation of new plugins by generating customisable templates. The process is documented in a tutorial available on-line[1].

4.1.3 Back-end

The back-end consists in a pair *{triples store, reasoner}*. For ORO, we make the choice to rely the open-source OPEN JENA library to store and access the RDF graph, in combination with the PELLET reasoner. However, due to clean separation, other APIs (like the MANCHESTER OWL-API) and reasoners (like FACT++ or HERMIT) could be used with little changes in the back-end API.

Open Jena

JENA [92] is a mature library for the semantic Web, originally developed by Hewlett-Packard labs, and now under the leadership of the Apache foundation.

As stated on its official website[2], the Jena Framework includes:

- an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats;
- an ontology API for handling OWL and RDFS ontologies;
- its own rule-based inference engine for reasoning with RDF and OWL data sources;
- stores to allow large numbers of RDF triples to be efficiently stored on disk;
- a query engine compliant with the SPARQL specification
- servers to allow RDF data to be published to other applications using a variety of protocols, including SPARQL.

[1] `http://www.openrobots.org/wiki/oro-server-plugins`
[2] JENA website: `http://incubator.apache.org/jena/`

Pellet

PELLET [120] is a reasoner for Description Logics developed by Clark&Parsia[3].

Pellet supports reasoning with the full expressiveness of OWL-DL ($\mathcal{SHOIN}(\mathcal{D})$) and OWL 2 ($\mathcal{SROIQ}(\mathcal{D})$).

Pellet provides all the standard inference services that are traditionally provided by DL reasoners: consistency checking, concept satisfiability, classification (computation of all the classes the instances belong to) and realisation (the most specific classes that a specific individual belongs to). It also supports DL-safe SWRL rule.

The use of a reasoner is completely transparent for the modules: the reasoner is automatically called when a model changes to classify it. Thus, queries to the knowledge models always access both the *asserted* and *inferred* sets of statements. As a consequence, the ORO server can be run with no reasoner without any visible API change for the modules. They will simply manipulate only asserted facts.

4.1.4 API

As of version 0.8, the ORO server API exposes about 50 methods (some of them are besides polymorphic) organised into seven categories: *Base*, *Agents*, *Administration*, *Concept comparison*, *Events*, *Queries* and *Taxonomy walking*.

Partial support for the *Knowledge API* (section 2.4) has also been added to ORO server 0.8: the newly introduced `revise` method that takes a *revision policy* as parameter is a versatile and generic mechanism that deprecates *de-facto* several methods currently exposed by the API.

The details of the current API is provided in appendix D.

4.1.5 Notes on the Java implementation

The ORO server has been developed with modularity in mind, thus most of its structure relies on clearly defined Java interfaces. Figure 4.3 presents the main ones.

The application entry point is the `OroServer` class. The class instantiate one front-end (`IConnector`), one main back-end (`IOntologyBackend`) and several modules (`IModule`), including plugins that are loaded at run-time.

Modules usually also implement the `IServiceProvider` interface to expose *services* (`IService`). These are the actual methods found in the server API. Java methods that belongs to the API simply need to be decorated with a `@RPCMethod` Java annotation to be automatically exposed. Extending the server API is thus simply a matter of annotating almost any Java method[4] with `@RPCMethod`. Listing 4.1 shows one example of a service annotated in such a way, the `add` method. This method convert a set of strings representing triples into `IStatements`, and add them to the triple store with the back-end method `oro.add`.

```
@RPCMethod(
        category="base",
```

[3] PELLET website: `http://clarkparsia.com/pellet`

[4] The only constraint being the input and output datatypes: currently, primitive types, simple collections and objects with explicit serialisation/deserialisation methods are supported.

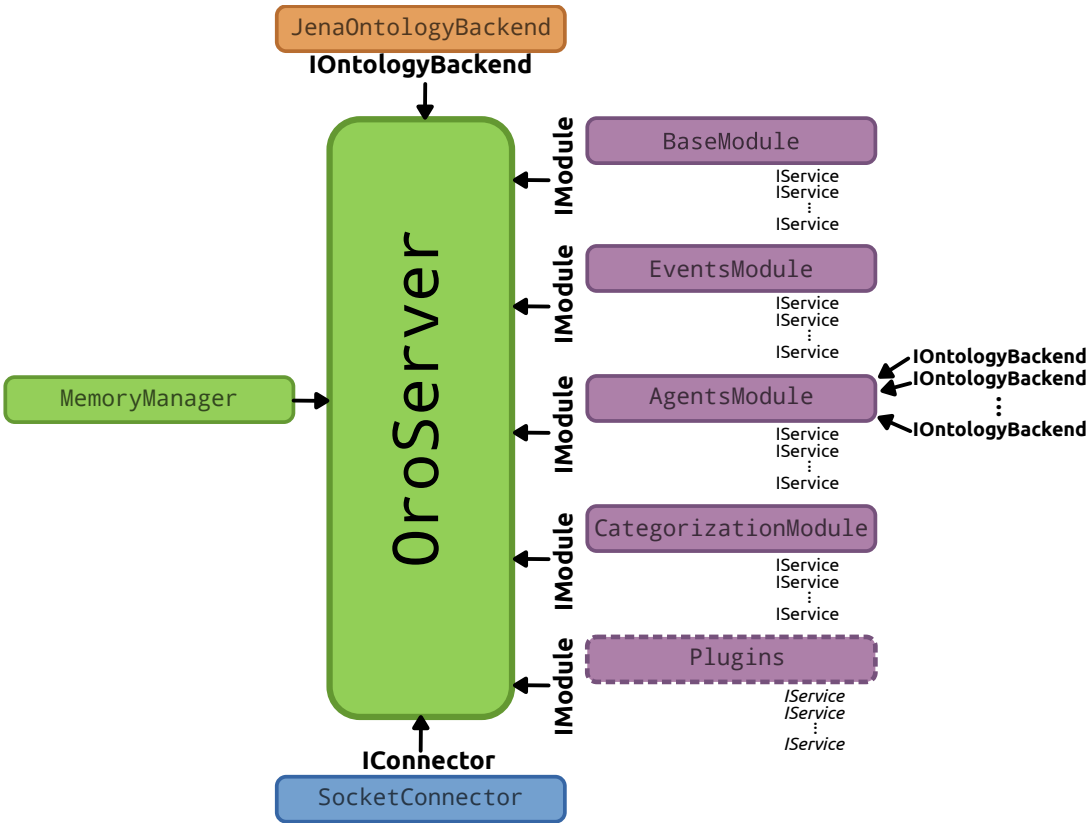


Figure 4.3: Main Java interfaces and classes of the ORO server.

```
        desc="adds one or several statements (triples S-P-O)."
)
public void add(Set<String> rawStmts, String memProfile)
{
    Set<IStatement> stmtsToAdd = new HashSet<IStatement>();

    for (String rawStmt : rawStmts) {
        if (rawStmt == null)
            throw new IllegalStatementException("Got a null stmt!");
        IStatement s = oro.createStatement(rawStmt);
        stmtsToAdd.add(s);
    }

    oro.add(stmtsToAdd, MemoryProfile.fromString(memProfile), false);
}
```

Listing 4.1: The `add` method from ORO `BaseModule`

As visible on the figure 4.3, the `AgentsModule` plays a particular role: it manages the alternative knowledge models of other agents, and store a `IOntologyBackend` for each of them. This module also re-export a large part of the services exposed by the `BaseModule` (methods for standard knowledge manipulation), but in their multi-agent versions.

This design issue (duplication of certain basic services) is due to the historical

development of the ORO server, and should be improved in future versions.

4.2 Bindings to other languages and components

To ensure the integration of the ORO server in existing robotic architectures, we have developed several idiomatic language-specific bindings, as well as bridges with two widespread robotic middlewares, ROS and YARP.

4.2.1 Language bindings

The main interaction gate with the ORO server is its socket interface (as presented above, at section 4.1.1). Sockets are light-weight, platform independent, and supported by every programming languages. Developing an interface with ORO server in a new language is hence relatively easy.

For our own needs, and because they are amongst the most widely used languages, bindings for Python and C++ are available by default with ORO server.

The complete Python and C++ API will not be presented here (their documentations are available on-line[5]), but we present two short example that demonstrate how the knowledge base can be integrated in code in a natural way.

Python

The Python script below (listing 4.2) demonstrates several of the interaction mechanisms with ORO server: model alteration, queries and events.

```
1  import time
2  import pyoro
3
4  def onEvent(evt):
5          print("God␣save␣the␣queen!␣" + evt + "␣killed␣Bond!")
6
7  try:
8          oro = pyoro.Oro()
9
10         oro += ["Spy␣rdfs:subClassOf␣Human",
11                 "bond␣rdf:type␣Spy",
12                 "bond␣rdfs:label␣\"Bond,␣James␣Bond\""]
13
14         if "bond␣rdf:type␣Human" in oro:
15             print("Alright,␣Bond␣is␣a␣human")
16
17         oro += "pr2␣rdf:type␣Robot"
18
19         for ag in oro["*␣rdf:type␣Agent"]:
20             print("Agent␣" + ag + "␣is␣here.")
21
```

[5] http://www.openrobots.org/wiki/oro-server-bindings

```
22          oro.subscribe(["?a kills bond"], onEvent)
23          oro += "pr2 kills bond"
24
25          time.sleep(1)
26          # the event should have been triggered
27
28  except pyoro.OroServerError as ose:
29          print('Oups! An error occured!')
30          print(ose)
31
32  finally:
33          oro.close()
```

Listing 4.2: Example of interaction with `oro-server` in Python

At line 8, we establish the connection to the server. We assume here that the server is launched on the default port and on the local machine.

At line 10, three facts are added to the knowledge base. The first one modifies the TBox of the ontology (alteration of the knowledge structure) while the two other ones modify the ABox (a new instance and a new label). Adding (or removing) triples from the ontology is done naturally with the += and −= operators.

At line 14, we check that a fact holds, either in the asserted model or in the inferred model (in this case, Bond is inferred to be a human because we added before that spies are types of humans). Here as well we use a natural idiomatic Python syntax that creates and executes a SPARQL query behind the scenes.

Line 19 shows another way to execute queries, with a dictionary-like accessor. Both humans and robots are asserted to be agents in the ORO common-sense ontology, thus this query returns a list `[bond, pr2]`.

Line 22 shows how events are created. We first subscribe to an event by passing a pattern (`?a kills bond`) and a callback (implemented at lines 4-5). Line 23 triggers the event, and the callback method is then invoked.

We have kept this example simple. The complete Python API allows to describe more complex events (when a fact can not be inferred, when a new instance of a given class appears, etc.), to manipulate different models, to walk through the ontology taxonomy, etc.

C++

The C++ bindings are provided as a library called `liboro`. The listing 4.3 presents some examples of its usage.

```
1  #include <iostream>
2  #include <iterator>
3  #include <set>
4
5  #include "liboro/oro.h"
6  #include "liboro/oro_library.h" // static library of ORO concepts
7  #include "liboro/socket_connector.h"
8
```

```
9  using namespace std;
10 using namespace oro;
11
12 class EventCallback : public OroEventObserver {
13     void operator()(const OroEvent& evt) {
14         cout << "Event triggered!" << endl;
15     }
16 };
17
18 int main(void) {
19   set<string> partial_stmts;
20   set<Concept> result;
21
22   SocketConnector connector("localhost", "6969");
23   Ontology *oro = Ontology::createWithConnector(connector);
24
25   //Creation of instances
26   Agent robot1 = Agent::create("Nice Robot", Classes::Robot);
27   Agent human = Agent::create("Young PhD", Classes::Human);
28
29   //First query
30   partial_stmts.insert("?mysterious rdf:type Agent");
31   oro->find("mysterious", partial_stmts, result);
32
33   copy(result.begin(),
34        result.end(),
35        ostream_iterator<Concept>(cout, "\n"));
36
37   partial_stmts.clear();
38   result.clear();
39
40   //More statements are added to the knowledge base
41   Object table = Object::create(Classes::Table);
42
43   Object unknown_object = Object::create();
44   unknown_object.assertThat(Properties::isOn, table);
45
46   oro->add(Statement("oro:isOn rdfs:subClassOf oro:isAt"));
47
48   Agent myself("myself");
49
50   myself.sees(unknown_object);
51   myself.sees(human);
52
53   //A query involving multiple partial statements
54   partial_stmts.insert("?mysterious isAt ?support");
55   partial_stmts.insert("?support rdf:type cyc:Table");
56   partial_stmts.insert("myself sees ?mysterious");
57
```

```
58    oro->find("mysterious", partial_stmts, result);
59
60    copy(result.begin(),
61         result.end(),
62         ostream_iterator<Concept>(cout, "\n"));
63
64    //Events
65    EventCallback ec;
66    Classes::Human.onNewInstance(ec);
67    Agent superman = Agent::create("Superman", Classes::Human);
68
69    sleep(1);
70
71    set<string> event_pattern;
72    Property flyingProp = Property("isFlying");
73
74    event_pattern.insert( superman.id() + " isFlying true");
75    oro->registerEvent(ec,
76                       FACT_CHECKING,
77                       ON_TRUE_ONE_SHOT,
78                       event_pattern, "");
79
80    superman.assertThat(flyingProp, "true");
81
82    sleep(1);
83
84    return 0;
85 }
```

Listing 4.3: Example of interaction with `oro-server` in C++

At lines 22 and 23, the `oro` object is built as a singleton. This actually connects the application to the ontology server.

At line 26 and 27, we create two new instances of agents labeled *Nice Robot* and *Young PhD* (the static types `Classes::Robot` and `Classes::Human` are generated from the ontology itself by a script).

A first simple query (line 30 and 31) return a `std::set` of concept IDs.

At line 41, a new object is created. No label is defined, but the class is set to be a table. On the contrary, at line 43, we create a generic object (only asserting it is an instance of `Object`).

At line 44, we assert a property (here also, the list of available properties is statically generated from the ORO ontology).

As shown at line 46, we can as well access the ontology at a lower level, directly adding (or removing) new triples. In this case, we modify the TBox of the knowledge base.

C++ objects can also be created by using directly the concept IDs, as shown line 48 for the special concept `myself` (an instance always representing the robot itself).

Some object properties frequently used in the ontology are available as methods, as seen lines 50 and 51.

Lines 54 to 58 show a more complex query, involving several partial statements. Named variables (like `?mysterious` or `support`) are used in the statements to reference the same entities.

Lastly, lines 65 to 80 show two ways of defining events with a callback functor (defined at lines 12-16).

To deal with real-world constraints, `liboro` also provides mechanisms to reconnect to the ontology server when the connection is lost, and has a built-in buffering system to increase bandwidth for components that produce a large amount of symbolic facts.

4.2.2 Interface with robotic middlewares

While convenient in certain cases, language-level interfaces do not usually offer the modularity and loose-coupling required by complex robotic architectures where tenth of modules, possibly spread over several computers, need to talk together. The acknowledgement of this issue has led to the development over the last ten years of numerous so-called *robotic middlewares* that abstract away inter-module communication (at the transports, protocols and programming interfaces levels).

We provide with ORO server wrappers for two of the main middlewares currently in use in the robotic community, ROS [108] and YARP [94].

These wrappers use the C++ or Python bindings previously presented to expose the features of ORO server as a stand-alone ROS/YARP nodes.

4.3 Monitoring and debugging

4.3.1 Logging and debugging tools

The ORO server offers several levels of logging, from almost silent to very verbose. In verbose mode (`debug` level), the server outputs exact incoming requests, along with millisecond accurate timestamps.

Those logs can then be replayed by a special tool written in Python that simulates the original communication with the server: timestamps are respected and if multiple clients where connected to the server, the log player forks one thread by client to simulate parallel access to the server.

Also available to the developer, an explanation of the inconsistencies when they occur is produced by the server, and helps to retrace the sequence of logical steps that led to the inconsistency.

4.3.2 Visualisation

Visualisation of ontologies is a difficult task in general because of their complex graph structure. In order to present anyway the content of the ontology to external observers, we have developed an OpenGL-based dynamic visualiser called `oro-view` (figure 4.4). This application connects to the ORO server and lets the user explore the taxonomy by simply selecting concepts on the screen. Nodes are then expanded, distributed in a force-directed layout, and further reveal the structure of the ontology. Because

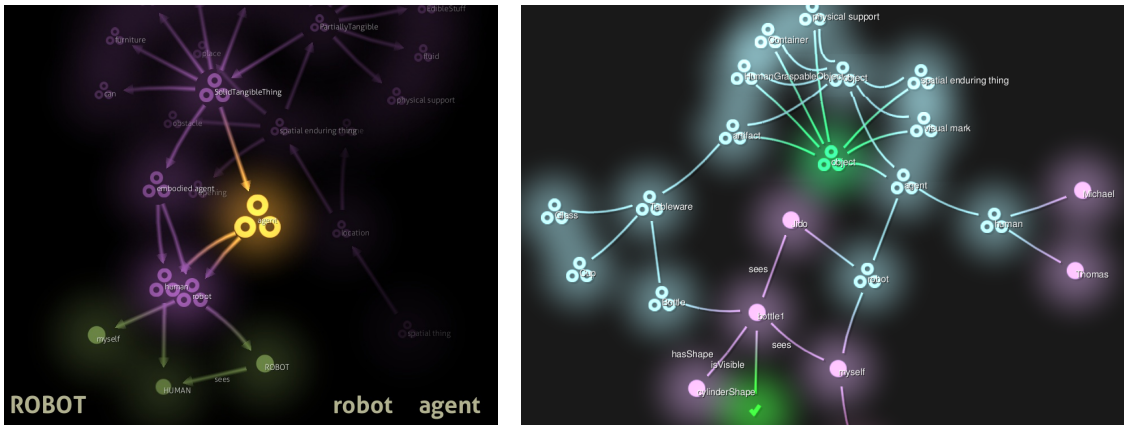


Figure 4.4: Two screenshots of the `oro-view` visualisation tool. On the left one, an `ActiveConcept` (in yellow) is visible.

`oro-view` takes on-demand its input from the server, changes in the ontology (new facts being added, etc.) are reflected in the viewer when the user refreshes nodes.

`oro-view` also subscribes at start-up to a special event for *active concepts* (it monitors new instances of the `ActiveConcept` class). Each time an individual is asserted to be an `ActiveConcept`, the server triggers back `oro-view` that creates a visual focus on the concept (the individual "pops up"). When displayed during experiments, this provides visual feedback to external observers. In particular, we made use of this feedback during the public performance of the *Roboscopie* theatre play (section 6.2.4).

`oro-view` also provides export to GraphViz `dot` format for latter reuse of the ontology graph in publications.

4.4 Integration in the robot architecture

Figure 4.5 presents the organisation of the upper software layer (the "decisional" or "cognitive" layer) of the service robots Jido and PR2 as currently in service at LAAS (this architecture is described in detail in [3]). The sensori-motor layer (bottom) is abstracted in SPARK, an intermediate amodal 3D model where geometric (and some temporal) reasoning take place.

The outcome of the geometric analysis, as well as the result of the dialogue processing module (DIALOGS), are stored in ORO, that plays the role of as a central knowledge hub. The symbolic knowledge base triggers events that are captured by a top-level execution controller.

In our architecture, the controller can rely on two specialised planners: MHP, a geometric motion and manipulation planner [122, 86, 104] and HATP, a symbolic task planner [4].

The dialogue processing module, as well as the symbolic task planner, also use the knowledge base to answer questions or initialise the planning domain.

During a typical interaction experiment (such an experiment is describe at chapter 6), the execution controller decides upon a goal to reach, requires a plan from the task planner, allocates the actions to the human and the robot, communicates the shared

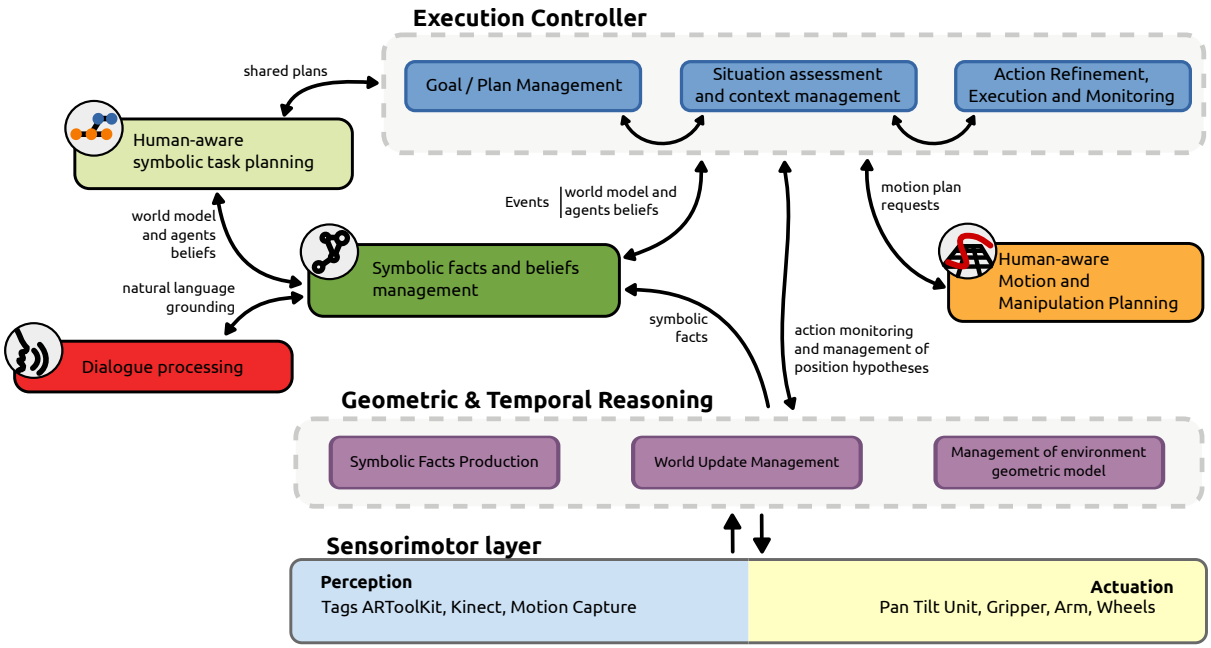


Figure 4.5: Software architecture of PR2 and Jido, two service robot interacting with humans at LAAS-CNRS.

plan to the human, and controls and monitors its execution. The operation continues until the goal is achieved, is declared unachievable or is abandoned by the human.

In this architecture, only ORO and Dialogs (the dialogue processing module that we present in the next chapter), as components, are the actual direct outcomes of this doctoral work. It is however important to present the other one all here as well since a knowledge base only make sense within a larger architecture, with knowledge providers and consumers. Furthermore, the approach to knowledge management introduced by this thesis had a strong influence on the design of the communication flows between all these components. Thus, this section introduces the software components that have been used in conjunction with ORO (mostly, but not only, on the LAAS robots), and details how these components produce, exchange and consume symbolic knowledge.

4.4.1 Acquiring and anchoring knowledge in the physical world: the SPARK module

Anchoring perceptions in a symbolic model requires perception abilities and their symbolic interpretation. In this section we present SPARK (*SPAtial Reasoning & Knowledge* [121]), a situation assessment reasoner that generates symbolic knowledge from the geometry of the environment with respect to relations between objects, robots and humans, also taking into account the different perspective that each agent has on the environment.

SPARK can be seen as an amodal geometric model of the environment that serves both as basis for the fusion of the perception modalities and as bridge with the symbolic layer.

Figure 4.6 shows a screenshot of the SPARK environment side-by-side with the real environment. In this example, objects are identified and localised through 2D barcodes. The human pose is tracked with a Kinect-like device (assisted by motion capture to accurately track the head motion, which is required to compute what the human is looking at).

The geometric model is continuously updated at run-time by the robot.

Building an agent-aware symbolic model of the environment

On Perspective Taking Visual perspective taking refers to the ability for visually perceiving the environment from other's point of view. This ability allows us to identify the objects in situations where the visual perception of one person differs from the other one. In developmental psychology, one typical example consists of two similar objects in a room (*e.g.* . two balls) where both are visible for the child, but only one is visible for the adult. Thus, when the adult asks the child to hand over "the ball", the child is able to correctly identify which ball the adult is referring to (*i.e.* the one visible from the adult point of view), without asking [95].

Besides, in order to compute a visual perspective, the actual visibility alone is not enough. We include not only what the other person sees in a given moment, but also what he *can* see with a minimal effort (moving the eyes or the head). To model the potential visibility of an object we compute a visibility ratio while turning the head of the agent model towards the object (figure 4.8, page 87).

Spatial perspective taking refers to the qualitative spatial location of objects (or agents) with respect to a frame (*e.g. the keys on my left*). Based on this frame of reference, the description of an object varies [87]. Humans mix perspectives frequently during interaction. This is more effective than maintaining a consistent one, either because the (cognitive) cost of switching is lower than remaining with the same perspective, or if the cost is about the same, because the spatial situation may be more easily described from one perspective rather than another [136]. Ambiguities arise when one speaker refers to an object within a reference system (or changes the reference system, *i.e.* switches perspective) without informing his/her partner about it [21, 111]. For example, the speaker could ask for the "keys on the left". Since no reference system has been given, the listener would not know where exactly to look. However, asking for "the keys on your left" gives enough information to the listener to understand where the speaker is referring to. On the contrary, when using an exact, unambiguous term of reference to describe a location (*e.g.* . "go north") no ambiguity arises.

In this work, we use two types of the frames of reference: egocentric (from the robot perspective) and addressee-centred (from the human perspective).

Symbolic locations Human commonly refer to the positions of objects with symbolic descriptors (like *on*, *next to*...) instead of precise, absolute positions. These type of descriptors have been studied in the context of language grounding [101, 89, 109, 59, 18]. In SPARK we focus agent-independent symbolic locations and agent-dependent, relative locations.

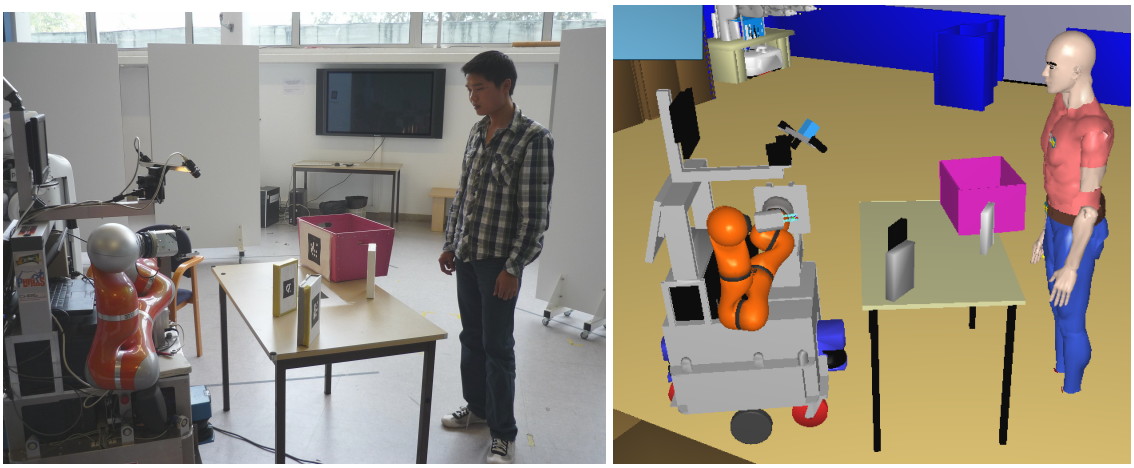


Figure 4.6: The robot represents at run-time its environment in a 3D model resulting of the sensors' inputs fusion (Kinect, motion capture, 2D barcodes tracking).

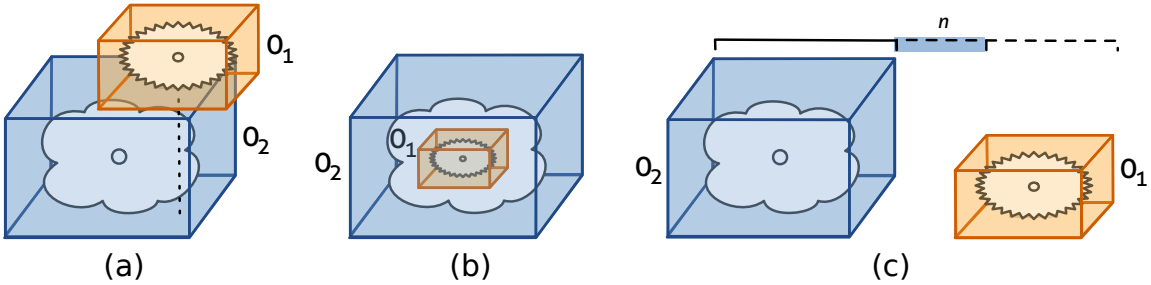


Figure 4.7: Spatial relations between two objects: (a) `isOn` relation, (b) `isIn` relation, and (c) `isNextTo` relation.

Agent-independent locations We can refer to object locations with respect to other objects in the environment, such as *above*, *next to*, *in*, etc. SPARK computes three main relations based on the bounding box and centre of mass of the objects (fig. 4.7):

- `isOn`: computes if an object O_1 is on another object O_2 by evaluating the center of mass of O_1 according to the bounding box of O_2.
- `isIn`: evaluates if an object O_1 is inside another object O_2 based on their bounding boxes BB_{O_1} and BB_{O_2}.
- `isNextTo`: indicates whether an object O_1 is next to another object O_2. We cannot use a simple distance threshold to determine if two objects are next to each other since the relation is highly dependent on the dimensions of the objects. For instance, the maximum distance between large objects (*e.g.* two houses) to consider them as being next to each other is much larger than the maximum distance we would consider for two small objects (*e.g.* two bottles). Thus, the relation between the dimensions and the distances of the objects are taken into account.

To ensure the different agent models are up-to-date, all these properties are always computed on-line, each time the current state of the world changes.

Table 4.1 lists all the symbolic placement relationships that are currently computed by the system.

Subject	Predicate	Object	*Notes*
Location	isAt ≡ cyc:objectFoundInLocation	Location	
	→ isOn ≡ cyc:above_Touching		
	→ isIn		
	→ isNextTo		
Location	isAbove ≡ cyc:above-Generally	Location	inverse of isBelow
			isOn ⇒ isAbove
Location	isBelow	Location	inverse of isAbove

Table 4.1: List of statements describing spatial relationships between objects. "→" indicates sub-properties. When existing, the equivalent predicate in the OPENCYC standard (prefix cyc:) has been added.

SPARK also compute symbolic facts related to agent independent world dynamics. The predicate isMoving states, for each tracked entity, whether it is currently moving or not.

Agent-dependent placements While in previous section we listed several *absolute* location predicate, many topological relations are directly dependent from the observation point.

The predicate hasRelativePosition represents spatial locations between agents and objects that are agent dependent. We compute these spatial locations by dividing the space around the referent (an agent) into n regions based on arbitrary angle values relative to the referent orientation. For example, for $n = 4$ we would have the space divided into *front*, *left*, *right* and *back*. Additionally, two proximity values, *near* and *far*, are also considered. The number of regions and proximity values can be chosen depending on the context where the interaction takes place.

Through perspective taking, SPARK computes for each agent a symbolic description of the relative positioning of objects in the environment (table 4.2).

Subject	Predicate	Object	*Notes*
Location	hasRelativePosition	Location	
	→ behind ≡ cyc:behind-Generally		inverse of inFrontOf
	→ inFrontOf ≡ cyc:inFrontOf-Generally		inverse of behind
	→ leftOf		inverse of rightOf
	→ rightOf		inverse of leftOf
Object	cyc:farFrom	Agent	
Object	cyc:near	Agent	

Table 4.2: List of statements describing relative spatial relationships between objects and agents.

Building a model of agents

Building a grounded symbolic model of the physical environment does not suffice in general to fully ground the human-robot interaction, and a model of the current capabilities of the agents surrounding the robot is also required.

There are a number of common properties for a robot and a human related to their capabilities in a given situation: they can both reach, grasp, look at, point at, etc.: we group them in the *Agent* category, defined as entities that can act in the environment and manipulate it.

SPARK computes the following capabilities from the perspectives of each agent:

- *Sees*: An important ability to know about an agent is to predict *What can it see?*, *i.e.* what is within its field of view (FOV). Being able to compute this information, the robot can reuse it for instance to infers which object a human is searching for (the one that is not currently visible, otherwise the user would not be searching for it). In figure 4.8*a* the field of view of a person is illustrated with a grey cone (broader one). While he is able to see the two small boxes on the table in front of him, the big box on his right is out of his FOV, and therefore, he is not able to see it.

- *Looks At*: this relation corresponds to what the agent is focused on, *i.e.* where its focus of attention is directed. This model is based on a narrower field of view, the field of attention (FOA). Figure 4.8*a* shows the field of attention of a person with a green cone (narrower one). In this example only the grey box satisfies the `looksAt` relation.

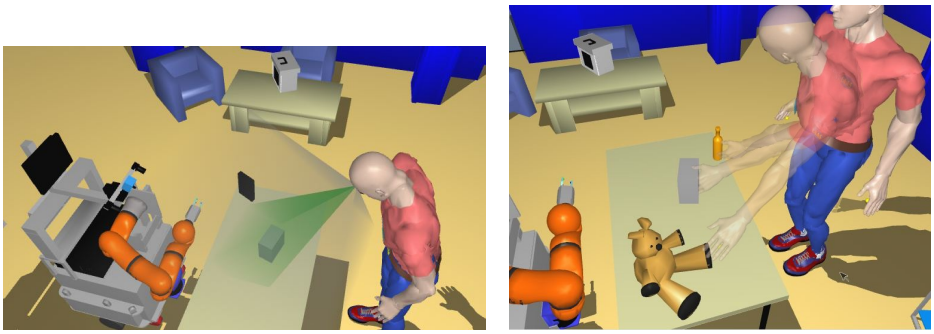
- *Points At*: verifies whether an object is pointed at by an agent. This relation is particularly useful during interaction when one of the agents is referring to an object saying “this" or “that" while pointing at it.

  If a larger object occludes a smaller one while an agent is pointing at them, the outcome of the evaluation will result only in one relation, *i.e.* ⟨`agent_01` *`pointsAt`* `object_01`⟩ since the small one is not visible to the agent. On the contrary, if the small object is in front of the big one, then both objects will satisfy the relation, which may generate an ambiguity (which object the agent refers to?) that is let to be solved by other discrimination algorithms.

  To make recognition more robust, these three first capabilities are filtered with an hysteresis function at the geometric level.

- *Reachable*: it allows the robot to estimate the agent’s capability to reach an object, which is fundamental for task planning. For example, if the user asks the robot to give him/her an object, the robot must compute a transfer point where the user is able to get the object afterwards. Figure 4.8*b* shows different reachability postures for each object on the table. In the example, the bottle and the box are both reachable for the human, but the teddy bear is too far. Instead, from the robot’s perspective, the teddy bear is reachable, while the bottle is not.

While the first three relations (`sees`, `looksAt` and `pointsAt`) are computed through a model based approach, the latter one is based on the Generalized Inverse Kinematics



(a) (b)

Figure 4.8: (a) Field of view (FOV) and the field of attention (FOA) of the human. (b) Different reaching postures for the human.

with pseudo inverse method [96, 12] to find a posture for the agent where its end-effector is at the centre of the object within a given tolerance.

Tables 4.3 summarises the predicates produced by SPARK during the agent capabilities analysis phase.

Subject	**Predicate**	**Object**	*Notes*
`Agent`	`looksAt`	`SpatialThing`	
`Agent`	`sees`	`SpatialThing`	
`SpatialThing`	`isInFieldOfView`	`xsd:boolean`	via inference: $\langle$`myself` *sees* `*`$\rangle \Leftrightarrow \langle$`*` *isInFieldOfView* `true`$\rangle$
`Agent`	`pointsAt` $\equiv$ `cyc:pointingToward`	`SpatialThing`	
`Agent`	`focusesOn`	`SpatialThing`	via inference: `looksAt` $\wedge$ `pointsAt` $\Rightarrow$ `focusesOn`
`Agent`	`seesWithHeadMovement`	`SpatialThing`	
`Agent`	`reaches`	`Object`	

Table 4.3: List of facts describing the attentional state and the abilities of an agent. `looksAt` is interpreted as an object *being in the field of attention* of an agent. An object is seen if it is visible for the agent without moving the head (*i.e.* , in *field of view*).

Table 4.4 lists the other symbolic facts that are produced and maintained by SPARK related to the general state of the agent.

4.4.2 Symbolic task planning

Complex human robot interaction also requires reasoning about the actions the agent can perform: How can they achieve a specific goal? What are the required actions to achieve this goal? Which actions can be performed by each agent? etc.

In the previous sections, we have seen how symbolic knowledge is produced and stored from the real physical world. In this section, we present one possible way to use these symbolic models of the environment and the interacting agents to produce a plan of actions for a complex goal.

Subject	Predicate	Object
`Agent`	`hasIn{Left|Right}Hand`	`GraspableObject`
`Agent`	`hasPosture`	`Posture`
`Agent`	`currentlyBodilyDoes`	`Action`

Table 4.4: List of statements describing the state of an agent in general. `Posture` can be either `standing` or `sitting`. The `currentlyBodilyDoes` predicate states the current action of the agent, be it intentional or not.

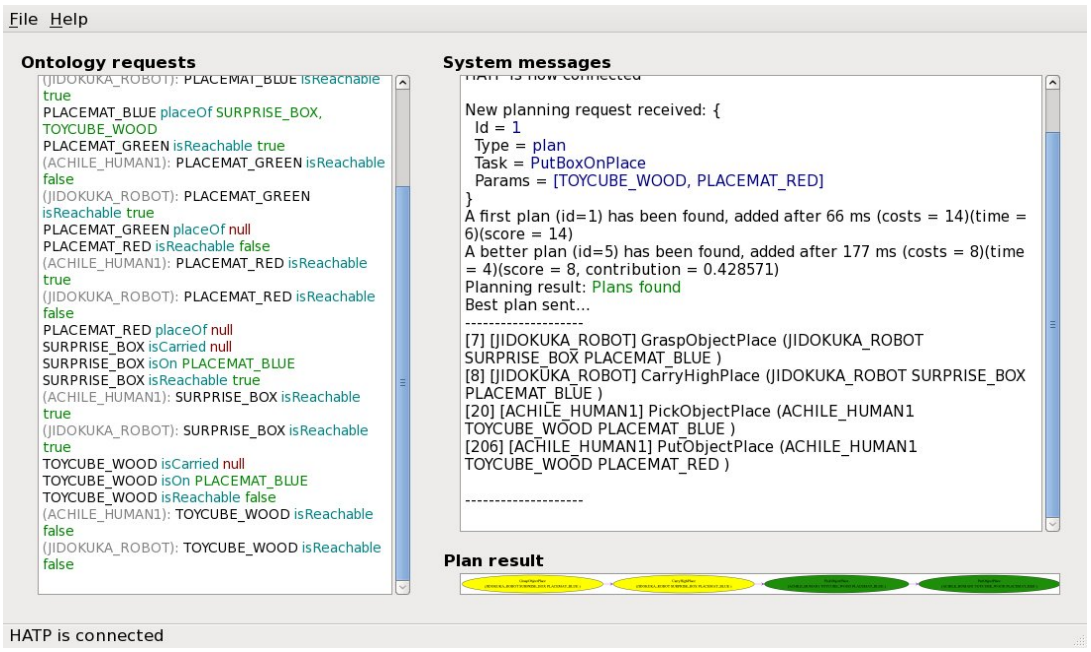


Figure 4.9: Screenshot of the HATP console. On the left panel, we see the results of the requests to ORO, on the bottom right the resulting plan.

In order to devise how a given goal can be accomplished, the robot has to elaborate a plan,*i.e.* a set of actions to be achieved by the robot and its human partners. We use in our architecture the HATP planner [4] (for Human Aware Task Planner, figure 4.9). HATP is based on a Hierarchical Task Network (HTN) refinement, which performs an iterative task decomposition into sub-tasks until reaching atomic actions [97]. The planning domain defines a set of methods describing how to decompose a task and can be seen as the procedural knowledge of the robot (note that this knowledge in stored in the own resources of the planner, not in ORO). HATP is able to produce plans for the robot's actions as well as for the other agents. It can be tuned by setting up different costs depending on the actions to apply and by taking into account a set of constraints called *social rules*. This tuning aims at adapting the robot's behaviour according to the desired level of cooperation of the robot.

Agents and action streams The robot plans not only for itself but also for the other agents. The planning domain of each agent is instantiated from the agent-specific model in the ORO server. The resulting plan, called "shared plan" is a set of actions that form a stream for each agent involved in the goal achievement. Depending on the

context, some "shared plans" contain causal relations between the agents. For example, the second agent needs to wait for the success of the first agent's action to be able to start its own action. When the plan is performed, causal links induce synchronisation between agents. Figure 4.10 illustrates a plan with two streams.

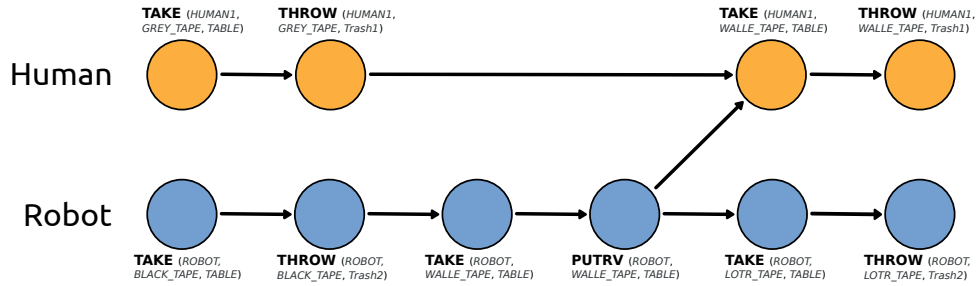


Figure 4.10: A plan produced by HATP with 2 streams

Action costs and social rules A cost and a duration function is associated to each action. The duration function provides a duration interval for the action achievement and is used, in one hand, to schedule the different streams and, in the other hand, as an additional cost function. In addition to these costs, HATP also takes into account a set of social rules. Social rules are constraints aiming at leading the plan construction towards the best plan according to some human preferences. The social rules we have defined so far deal with:

- undesirable state: to avoid a state in which the human could feel uncomfortable;
- undesirable sequence: to eliminate sequences of actions that can be misinterpreted by the human;
- effort balancing: to adjust the work effort of the agents;
- wasted time: used to avoid long delays between the actions of the human partner (figure 4.11);
- intricate links: to limit dependencies between the actions of two or more agents.

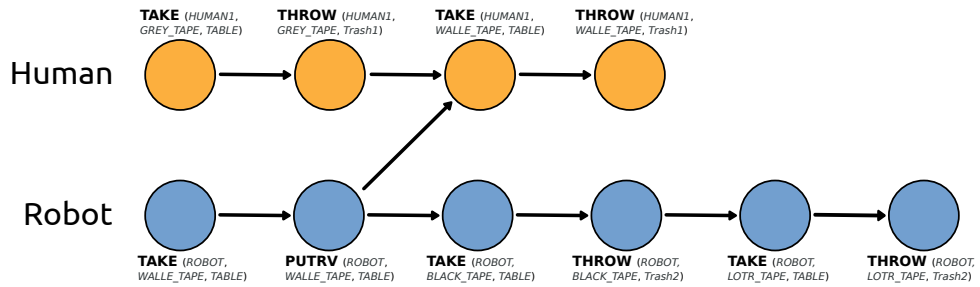


Figure 4.11: Same plan with minimised wasted time.

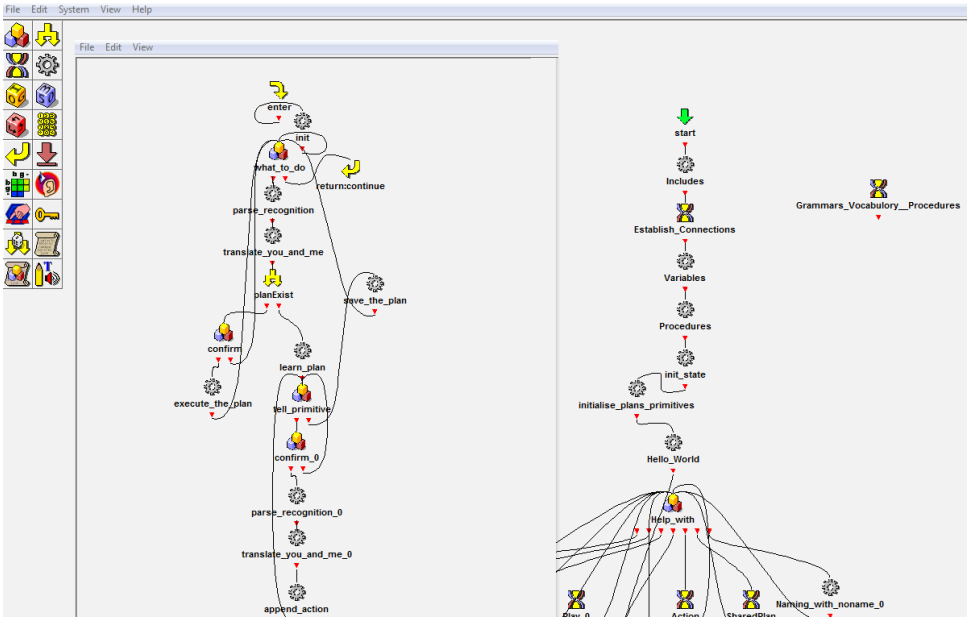


Figure 4.12: Example of graphical scripts produced with the CSLU Toolkit. Behaviours are programmed by building a network of connected "boxes".

Several levels of cooperation By tuning its costs and adapting its social rules, HATP can be used to compute various alternative plans. These plans can be categorised into several levels of cooperation

- helping the human to achieve his goal by acting for him
- sharing concrete resources by handing some objects
- collaboration of the robot and the human by coordinating their actions towards a human-robot joint goal.

4.4.3 Execution control

Depending on experimental setups, the ORO server has been integrated with several distinct execution controllers. We briefly present them here, with some details regarding the integration knowledge base/controller.

CSLU Toolkit The CSLU Toolkit is a rapid application development framework developed at Oregon University. It comprise of a suite of tools that enable exploration, learning, and research into speech and human-computer interaction via a user friendly graphical interface (figure 4.12). The CSLU toolkit has been employed in several experiments of the European FP7 CHRIS project (some are presented at section 6.2.2, others are presented in [70, 71]) to develop scripted verbal interactions with robots.

The CSLU Toolkit can be easily extended with the TCL scripting language, for which bindings with the ORO server exist. This enable users to graphically design interaction experiments that take advantage of the knowledge base and its reasoning infrastructure to add and retrieve concepts.

CRAM CRAM [15] (Cognitive Robotic Abstract Machine) is a RPL-derived framework for rapid development of cognitive robot control programs that is developed at the Intelligent Autonomous Systems at TU Munich.

The integration of ORO is seen as an extension to the robot's belief state that not only contains abstract identifiers of the internal object representation used in plans, but also the semantics and roles of objects in the scenario.

CRAM automatically updates the ORO server whenever an object enters or leaves the field of view with a perception stack based on the COP framework [62].

The *Odd One Out* experiment (section 6.2.2) relies on CRAM for the robot control.

SHARY SHARY [144] is a high level robot control system for cognitive robot interacting with humans, based on the OPENPRS environment (an open-source implementation of PRS Procedural Reasoning System [53]).

A full OPENPRS interface with ORO has been developed, and request and events mechanisms are heavily used by SHARY to monitor and react to changes in the robot environment.

SHARY also produces symbolic facts that are added to the ORO server, including the outcome of actions. This allows to partially expose the internal state of the execution controller, and enhancing the introspective capabilities of the system.

Finally SHARY can also directly interacts with situation assessment and geometric layers (like the SPARK module), which indirectly influences the ORO content. For instance, SHARY can create or delete position hypotheses for currently unperceived objects, which in turn are converted into (*de-facto*) hypothetical symbolic beliefs in the knowledge base.

Section 6.2.3 presents an experiment that demonstrates the integration between SHARY and ORO.

pyRobots `pyRobots` is not a real execution controller: it comprise of a set of Python scripts that ease the construction of complex scripted interaction. `pyRobots` is based on *actions* (high-level tasks like `goto`, `pick` or `give`[6]) that are combined into plans.

ORO server is integrated in `pyRobots` via the ORO Python bindings (presented at section 4.2.1). The action that is currently performed, is maintained up-to-date in the server, and actions are free to store or retrieve facts (for instance, the `pick` task add the fact ⟨`myself hasInRight|LeftHand x`⟩, `x` being replaced by the ID of the object that is taken.

Events can also be used by the application designer.

4.4.4 Integration with natural language processors

Figure 4.13 gives a functional overview of components involved in the situated dialogue grounding.

Verbal interaction with a human is either initiated by the human or by the robot.

In case of human initiated dialogue, two main steps follow: understanding the direct meaning of the sentence (what does the words mean?), and understanding the

[6]The complete list is available online: `http://www.openrobots.org/wiki/pyrobots`.

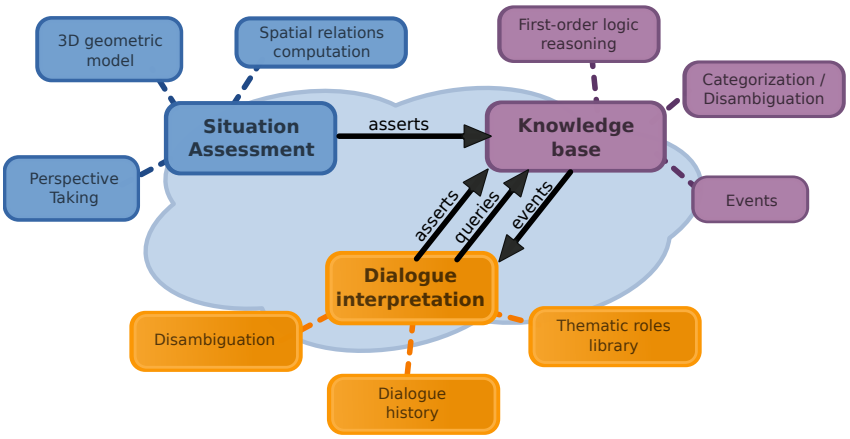


Figure 4.13: Schematic view of the integration of a natural language processor in the architecture

intent of the sentence. The first step usually requires a large amount of queries to the knowledge base to clarify and disambiguate the meaning of the sentence ; the second step, depending on the intent (question, desire, assertions), leads to either other queries (to answer a question) or new assertions.

An interaction initiated by the robot usually follows an event triggered by the controller (or possibly directly by the knowledge base). The dialogue that it initiates then requires the same interaction capabilities with the knowledge base that we mentioned in the previous paragraph.

Studying the integration of natural language grounding with a symbolic knowledge base is one of the focus of the thesis, and the next chapter is entirely dedicated to DIALOGS, a component for situated natural language processing, and its integration with ORO.

Chapter recap

This concludes the chapter 4. In this chapter, we have first developed some technical and implementation-related details, including the software architecture of the ORO server, examples of integration with the Python and C++ API, and an overview of the visualisation and debugging tools.

The chapter then focused on the actual integration of ORO server in existing robotic architecture. We have first presented the software architecture of the decisional layer of service robots at LAAS. We have introduced the SPARK module for geometric reasoning and symbolic situation assessment and underline its perspective taking capabilities.

We have finally presented briefly HATP, a symbolic task planner for human-robot cooperative task planning that takes advantage of the ORO server for planning initialisation, and mentioned four control environment that have been used with ORO.

The next chapter is now dedicated to one specific application of the knowledge base:

situated natural language grounding.

Chapter 5

Knowledge Enabled Situated Natural Language Processing

5.1 Grounding verbal interaction into the robot knowledge

5.1.1 Situated speech acts

A messy kitchen table, covered with knifes, spoons, bowls... Tom is preparing the brownie, with Robi and Roba, its two robots.

" – Robi, give me this bowl", says Tom, looking at the table. The robot smoothly grasps the bowl, and hands it to the human.

What are the prerequisites for such a human sentence — "Robi, give me this bowl" — to be understood by the robot, correctly interpreted in the spatial context of the interaction, and ultimately transformed into an action?

Austin [8] would have at first glance analysed such kind of sentence as a *speech act*, comprising of *locutionary*, *illocutionary* and possibly *perlocutionary* acts: First, we want to understand the direct meaning of the sentence (*locutionary act*): we must acquire the sentence, convert it into a useful syntactic form (quite probably by mean of speech recognition), and understand the semantics of the sentence, *i.e.* What is referred by "*Robi*"? What is "*give*"? What is "*me*"? And "*this bowl*"?

Working in a situated context, we want furthermore to *resolve* these semantics atoms (*i.e.* ground them) in the sensory-motor space of the robot. For instance, "*this*" is a demonstrative pronoun that refers in this context to the object the human is focusing on, whatever *focusing* means: here, we guess Tom is looking at some bowl, which is a possible cue. But it could as well point at something or refer to some previously mentioned concept.

Second, the *illocutionary force*, *i.e.* the *intent* of the utterance as thought by the agent must be extracted, and understood. In our example, Tom obviously wants an action to be performed by the robot. The action parametrisation is conveyed by the semantics attached to the words and the grammatical structures of the sentence. In our example, the type of action is given by the verb "*give*". Assuming the robot has some procedural knowledge (a planning domain and a planner) attached to this symbol, the action

Figure 5.1: Asking the robot to hand over a tape on the table.

type can be considered as grounded for the robot. We can as well understand that the recipient of the action is the human, the performer is the robot itself, and the object acted upon is the bowl. These are the basic *thematic roles* that can be extracted from the sentence that allow to fully ground the action.

Extracting these speech acts and turning them into a content processable by the robot is a difficult challenge in the general case. We base our approach on three distinct, inter-related cognitive functions:

1) Physical environment modelling and *spatial reasoning* (grouped under the term *situation assessment*) are in charge of building and maintaining a coherent model of the physical world. We have presented SPARK in the previous chapter.

2) Knowledge representation and management: we have also already presented the ORO server in the previous chapters. It endows the robot with an active knowledge base that provides a logically sound symbolic model of its beliefs on the world, as well as models for each cognitive agent the robot interacts with.

Used in combination with the situation assessment framework, the robot is thus able to maintain different models of the world, one per agent. This proves an essential feature [112, 65] to enable perspective-aware grounding of natural language, as we will see in next sections.

3) Dialogue input processing, including natural language parsing capabilities, disambiguation routines and interactive concept anchoring. We focused our efforts on three classes of utterance, commonly found in human-robot interaction: *statements* (*i.e.* new facts the human wants to inform the robot), *orders* (or more generically *desires*) and *questions on declarative knowledge* (whose answers do not require explicit planning). This would roughly cover the *representative* (sometimes referred as *assertives*) and *directives* type of illocutionary acts in Searle [117] classification.

5.1.2 Related work

Processing natural language in situated context is already an established research field. We have already mentioned Roy and Reiter [112] that propose cross-modal representation systems, association of words with perceptual and action categories, modelling of context, figuring out the right granularity of models, integrating temporal modelling and planning, the ability to match past (learned) experiences with the current interaction and the ability to take into account the human perspective as the main challenges of situated dialogue processing.

Kruijff et al. provides in [65] an up-to-date survey of literature on situated human-robot dialogue, focusing on formal representation systems, bi-directionality of the interaction and context building. They point as well that, compared to the cognitive psychology community, the "situated AI" community started only recently to take into account agents focus, perspective and temporal projection abilities.

Dialogue processing on real robots have been explored by several teams. Scheutz [22] has contributions regarding natural language processing in an incremental way, and how this enables instant back-channel feedback (like nodding).

Hüwel et al. [52] propose the concept of *Situated Semantic Unit*: these meaning atoms are extracted from sentences and expose semantic links to other units. The parser tries to satisfy these links and rate accordingly the semantic interpretation of the sentence. Used in conjunction with ontologies, their approach offers good robustness to ungrammatical or partial utterances. They validated the approach with an extensive user-study.

Several other systems, already presented at chapter 2, have tackled the challenge: the Tapas system, the GLAIR architecture, GSM or the Ke Jia project.

Compared to these previous contributions, our efforts have two foci: *(1)* integration between language processing and perception of the environment and the humans, from several perspectives; and *(2)* realistic human-robot interactions: real-time processing; open speech; complex, dynamic, partially unknown human environments; fully embodied autonomous robots with manipulation abilities.

We do not claim however any significant contribution to the field of theoritical computational linguists (see [65] for a survey of formal approaches to natural language processing in the robotics field): our main contribution here is the grounding of concepts involved in the human discourse through the robot's own knowledge.

Section 5.2 presents the overall grounding process and section 5.3 proposes an analysis of the processing of three prototypical sentences. Experimental results are presented in the next chapter on the evaluation.

5.2 The Natural Language Grounding Process

As stated above, we process three categories of sentences: *statements*, *desires* and *questions* that can be answered from the declarative knowledge present in the robot knowledge base (a choice similar to the *Behaviour Cycle* in the GLAIR architecture [118]). In our work, the grounding process of the human discourse consists in extracting either the *informational* content of the sentence to produce statements or its *intentional* content (*i.e.* performative value) to collect orders and questions.

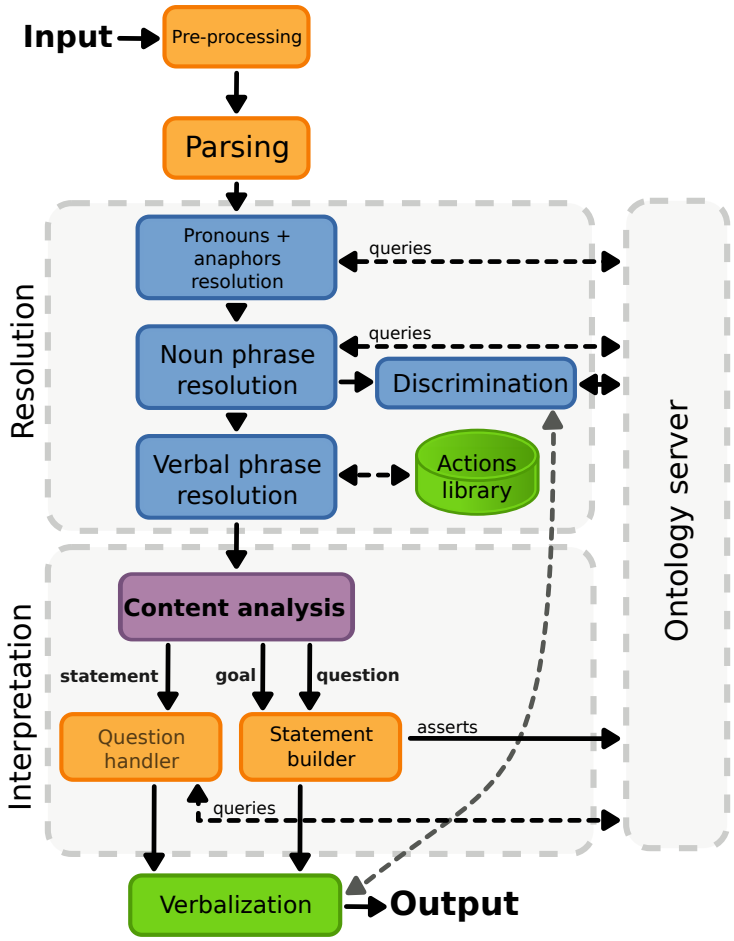


Figure 5.2: The DIALOGS module has four main steps: the parsing, the resolution, the interpretation and the verbalisation.

As shown in Figure 5.2, the DIALOGS module that we have developed, is composed of four main blocks. The user's input is first pre-processed. For instance, *I'm* constructs are expanded into *I am* and then parsed. The parser is a custom-made, rule-based (*i.e.* grammar-free) tool that extracts the grammatical structure from the user's sentence.

Figure 5.3 shows an example of the raw output of the parser for a moderately complex sentence.

The result of the parsing is then sent to the *resolution* module. The processing can be divided again in three steps: *(1)* pronouns and anaphora are replaced by, respectively, the correct speaker ID and the ID of the last object spoken about (extracted from the dialogue history), *(2)* nominal groups are disambiguated and grounded (noun phrase resolution), and *(3)* verbal groups are resolved as well, and their associated thematic roles are retrieved (verbal phrase resolution). Algorithm 5.2.1 describes the overall process (with the subroutine *GenerateDescription* presented in algorithm 5.2.2, page 100). Next section describes specific examples to show how the noun and verbal phrase

```
>> IMPERATIVE
VP: remember (present simple)
    SUBSENTENCE (aim: that)
      NP: I
      VP: want (present simple)
        direct objects:
          NP: you
        secondary VP: give ()
              direct objects:
                NP: my nice blue bottle
              indirect objects:
                NP: me
```

Figure 5.3: Raw output of the DIALOGS parser after processing the sentence: "remember that I want you to give me my nice blue bottle." Nominal groups are not grounded yet.

resolution takes place.

Algorithm 5.2.1: RESOLUTION($sentence, currentSpeaker$)

$\mathcal{G} \leftarrow$ PARSENOMINALGROUPS($sentence$)
for each $g \in \mathcal{G}$
do
- $\mathcal{D} \leftarrow$ GENERATEDESCRIPTION(g) (1)
- $candidates \leftarrow$ ONTOLOGY.FIND($\mathcal{D}$) (2)
- **if** $|candidates| = 0$
  - **then**
    - **output** (Couldn't resolve the group!)
    - **exit**
  - **else if** $|candidates| = 1$
  - **then** $id \leftarrow candidates[0]$ (3)
  - **else**
    - **if** ONTOLOGY.CHECKEQUIVALENT($candidates$)
      - **then** $id \leftarrow candidates[0]$
      - **else** $id \leftarrow$ DISCRIMINATION($candidates, currentSpeaker$)
- REPLACE($g, id, sentence$)

The result of the resolution is then send over to the *interpretation* module that first performs a *content analysis* (what was the intent of the utterance: information, question or desire) and then translate the original sentence into RDF statements (the *statement building* step).

As represented in Figure 5.2, both resolution and interpretation tightly rely on the communication with the knowledge base. All the concepts the robot manipulates are stored in the ontology server and retrieved through logical queries, except for the verbs

that are currently stored in a dedicated library (the *action library* in the diagram).

Algorithm 5.2.2: GENERATEDESCRIPTION($group$)

procedure GENERATEDESCRIPTION($group$)
$noun \leftarrow$ GETNOUN($group$)
if ONTOLOGY.LOOKUP($noun$) $\in (Instances)$ $\begin{cases} id \leftarrow \text{ONTOLOGY.LOOKUP}(noun) \\ \textbf{return } (\mathcal{D} + \langle * \texttt{ sameAs } < id > \rangle) \end{cases}$
else $\mathcal{D} = \mathcal{D} + \langle * \texttt{ type } < noun > \rangle$

$det \leftarrow$ GETDETERMINANT($group$)
if $det \in (possessives)$
then $\mathcal{D} = \mathcal{D} + \langle * \texttt{ isRelatedTo } < possessor > \rangle$
if $det \in (demonstratives)$
then $\begin{cases} \textbf{if } \text{ONTOLOGY.CHECK}(\langle < currentSpeaker > \texttt{ focusesOn } * \rangle) \\ \quad \textbf{then } \mathcal{D} = \mathcal{D} + \langle < currentSpeaker > \texttt{ focusesOn } * \rangle \\ \quad \textbf{else } \mathcal{D} = \mathcal{D} + \text{ANAPHORICMATCHING}() \end{cases}$

$adjs \leftarrow$ GETADJECTIVES($group$)
for each $adj \in adjs$
do $\begin{cases} \textbf{if } adj ==" other" \\ \quad \textbf{then } \begin{cases} id \leftarrow \text{HISTORY.GETMATCHINGCONCEPT}(group)\mathcal{D} = \mathcal{D} + \langle * \texttt{ differentFrom } < id > \rangle \\ \textbf{return } (D) \end{cases} \\ \quad \textbf{else } \mathcal{D} = \mathcal{D} + \langle * \texttt{ hasFeature } < adj > \rangle \end{cases}$

$nounCmplts \leftarrow$ GETNOUNCOMPLEMENTS($group$)
for each $cmplt \in nounCmplts$
do $\mathcal{D} = \mathcal{D} +$ GENERATEDESCRIPTION($cmplt$)

$relativeClauses \leftarrow$ GETSUBORDINATERELATIVECLAUSES($group$)
for each $relative \in relativeClauses$
do $\begin{cases} \mathcal{G} \leftarrow \text{GETNOMINALGROUPS}(relative) \\ \textbf{for each } g \in \mathcal{G} \\ \quad \textbf{do } \mathcal{D} = \mathcal{D} + \text{GENERATEDESCRIPTION}(g) \end{cases}$

return $(\mathcal{D})$

Initial knowledge human_01	*Human input*
⟨banana_01 *type* Banana⟩ ⟨banana_01 *hasColor* yellow⟩	"The yellow banana is big!"
Generated partial statements	*Newly created statements*
⟨**?obj** *type* Banana⟩ ⟨**?obj** *hasColor* yellow⟩ ⇒ ?obj = banana_01	⟨banana_01 *hasSize* big⟩

Figure 5.4: First example: content extraction. "⇒" represents the output of the ontology server.

Algorithm 5.2.3: HISTORY.GETMATCHINGCONCEPT($group$)

procedure HISTORY.GETMATCHINGCONCEPT($group$)
comment: Extract resolved nominal groups from sentences stored in the history
$\mathcal{H} \leftarrow$ HISTORY.GETALLPASTIDS()
comment: The adjective "other" is removed before recursively calling this routine
$\mathcal{G} \leftarrow$ ONTOLOGY.FIND(GENERATEDESCRIPTION($group$))
$candidates \leftarrow \mathcal{H} \cap \mathcal{G}$
if $|candidates| = 0$
then { **output** (Couldn't find another object with the same characteristics!); **exit**
else if $|candidates| = 1$
then $id \leftarrow candidates[0]$
else $id \leftarrow$ DISCRIMINATION($candidates$)
return (id)

5.3 Technical analysis

In order to better understand the overall process of the DIALOGS module and its relation with ORO, we next describe the different steps of the approach based on three examples. In these examples we assume that some initial facts are present in the knowledge base (either from initial common-sense knowledge or through run-time acquisition), both in the robot's own model and in the model of the human. Since the robot tries to ground a human utterance, all queries are sent to the human model in order to interpret it from the human perspective.

5.3.1 Informational Content Extraction

Figure 5.4 shows a first example of human discourse grounding and the extraction of informational content. We assume that the robot knowledge base only contains two initial statements in the human model. The user asserts a new one: "The yellow banana

Initial knowledge human_01	*Human input*
⟨banana_01 *type* Banana⟩ ⟨banana_01 *hasColor* yellow⟩	"Give me the banana."
Generated partial statements	*Newly created statements*
⟨**?obj** *type* Banana⟩ ⇒ ?obj = banana_01	⟨human_01 *desires* sit_a3⟩ ⟨sit_a3 *performedBy* myself⟩ ⟨sit_a3 *actsOnObject* banana_01⟩ ⟨sit_a3 *receivedBy* human_01⟩

Figure 5.5: Second example: processing an order.

is big!". We first want to match the nominal group *The yellow banana* to an already known concept (algorithm 5.2.1), and second to translate the property *is big* into a predicate (hasSize) to state its semantics.

To resolve the nominal group *The yellow banana* a set of partial statements that describe the concept is generated based on the grammatical parsing of the sentence (algorithm 5.2.2). The parsed tree of each nominal group is translated into statements based on a set of rules. In the example, a banana ⟨**?obj** *type* Banana⟩ that is yellow ⟨**?obj** *hasColor* yellow⟩. Based on these partial statements a SPARQL query is sent to the ontology server to retrieve possible instances that match the description (algorithm 5.2.1(2)).

In this first simple case, the concept banana_01 is unambiguously matched (since there is only one possible banana) and returned. and we can add the new information provided by the human, *i.e.* the new statement ⟨banana_01 *hasSize* big⟩, to the human model in the ontology server.

The translation of *yellow* to hasColor yellow is not obvious: in the general case, we associate a adjective to the noun it characterises with the hasFeature predicate (for instance, *The sight is beautiful* would translate to ⟨sight *hasFeature* beautiful⟩). But we can also manually set the predicate associated to a category of adjectives: It is what has been done for the main colours. Another example is the size: for known size adjectives (big, small, etc.), the hasSize predicate is being used.

5.3.2 Intentional Content Through Verb Resolution

The sentence in the first example is built with the state verb *be* at indicative. Let us examine a different example with an action verb at imperative mode (an order): "Give me the banana". The process is described in Figure 5.5.

In order to capture the intentional content of a sentence (for example, an order) we need to retain the semantics of the verb and its complements. *Thematic roles* allow for semantically linking a verb to its complements. There is no general agreement amongst linguists on a comprehensive list of thematic roles. The amount and the granularity of roles varies a lot in the literature [43]. We thus use a small set of them, which matches the relations the robot can actually make sense of (*i.e.* that are modelled in the planner domain). For instance, in the second example, the verb *give* has three thematic roles: performedBy, actsOnObject and receivedBy.

The list of actions the robot can plan for (currently *take*, *place*, *give*, *show*, *hide* and

Verb	Grammatical Role	Thematic Role	Predicate	Range
get	Subject	Agent	`performedBy`	`Agent`
	Direct object	Theme	`actsOnObject`	`Artifact`
put	Subject	Agent	`performedBy`	`Agent`
	Direct object	Theme	`actsOnObject`	`Artifact`
	Indirect object	Recipient	`receivedBy`	`PhysicalSupport`
give	Subject	Agent	`performedBy`	`Agent`
	Direct object	Theme	`actsOnObject`	`Artifact`
	Indirect object	Recipient	`receivedBy`	`Agent`
move	Subject	Agent	`performedBy`	`Agent`
	Direct object	*Theme*	`actsOnObject`	`Artifact`
	Indirect object	Goal	`hasGoal`	`Location`
show	Subject	Agent	`performedBy`	`Agent`
	Direct object	Theme	`actsOnObject`	`Location`
	Indirect object	*Recipient*	`receivedBy`	`Agent`
look	Subject	Agent	`performedBy`	`Agent`
	Indirect object	Goal	`hasGoal`	`Location`

Table 5.1: Main action verbs known to DIALOGS and associated thematic roles. Italics denotes optional roles.

move) along with possible synonyms (for example, *to pick* and *to take* are set as synonyms of *to get*) and their associated thematic roles are stored in a predefined library of actions (table 5.1 and figure 5.2). For each action we identify and store: the role of the subject in the sentence (always `performedBy`); the role of the direct object (for instance, `actsOnObject`); and the role of each of the indirect objects with their optional prepositions (for instance, `receivedBy`)[1]. Moreover, we rely on the ontology to check that each holder of a role is semantically consistent. For instance, the action *Give* must have a manipulable physical item (`Artifact`) as direct object. Thus, if the concept the robot finds for the thematic role `actsOnObject` cannot be inferred to be an artifact, the robot goes back to the human saying it does not understand.

This second example also shows the pronoun reference resolution: "me" is replaced by the id of the current speaker, while "you" is replaced by `myself` (`myself` always represents the robot itself). When present, anaphoras (references to previous concepts like "give me the banana, I like **it**.") are also resolved in the same step.

Once the sentence is completely resolved and translated into a formal representation (a human desire in this example[2]), we store it in the ontology server. The robot's decisional/executive layers can then decide whether to execute the order or not.

Initial knowledge model of `human_01`

⟨`banana_01` *`type`* `Banana`⟩
⟨`banana_01` *`hasColor`* `yellow`⟩
⟨`banana_02` *`type`* `Banana`⟩
⟨`banana_02` *`hasColor`* `green`⟩

Human input

"The banana is good."

Generated partial statements

⟨**`?obj`** *`type`* `Banana`⟩
⇒ `?obj = [banana_01, banana_02]`

Discrimination process

`discriminate([banana_01, banana_02])`
⇒ `?hasColor = [yellow, green]`

Robot output speech

"The yellow one or the green one?"

Human answer

"The green one."

Extended human input

"The green banana is good."

Generated partial statements

⟨**`?obj`** *`type`* `Banana`⟩
⟨**`?obj`** *`hasColor`* `green`⟩
⇒ `?obj = [banana_02]`

Newly created statements

⟨`banana_02` *`hasFeature`* `good`⟩

Figure 5.6: Ambiguity resolution: in this example, "banana" can refer to the yellow banana (`banana_01`) or the green one (`banana_02`). Discrimination routines handle the disambiguation process.

5.3.3 Informational Content Extraction Requiring Clarification

This last example (figure 5.6) shows the resolution of ambiguous concepts. In this case the user refers to "the banana" while two instances of the `Banana` class exist in the ontology. The robot needs to find out to which instance the user is actually referring to. To this end, disambiguation routines (algorithm 3.2.3, page 63) find differences between the instances (in the example, one banana is yellow while the other one is green) and build a sentence through the *verbalisation* module to ask the user a closed question that will help clarify the ambiguity: "Is it yellow or green?" The user's answer is parsed and merged with the previous sentence. The resulting, augmented, sentence ("The green banana is good") goes again through all the interpretation steps. This process is repeated until no ambiguities arise. In the example, the `banana_02` is finally returned.

If no differences can be found, an open question ("give me more information") is send to the human.

Several other strategies are used in parallel to disambiguate concepts without having to ask for more information to the human:

- Which objects are currently visible to the human? If only one of them, then it is probably the one the user is talking about.
- Did a previous interaction involved a specific object that would still be the subject of the current sentence?
- Is the user looking or pointing at a specific object?

Two cases can alter the way the discrimination routines work:

1. If a sentence starts with *Learn that...*, failures during discrimination are interpreted as new concepts, and instead of marking the nominal as not resolved, and new identifier is created and add to the knowledge base.
2. For questions like *Which colour is the bottle?*, the discrimination algorithm can not use the feature *colour* to identify to bottle. The resolution algorithm pass this kind of constraints as a parameter of the discrimination routines.

While no examples involving questions have been detailed, factual *wh-* questions and polar (*yes/no*) questions can be processed in a similar way by DIALOGS. For instance, a question like "What is on the table?" is grounded (to extract the relation `isOn` and to find what *table* refers to) and transformed into the following kind of query: `find ?var [⟨?var isOn table1⟩]`. Answers are converted back to a full sentence by the *verbalisation* module, and uttered to the human.

In section 5.3.1, above, we give an example where the human says "the yellow banana is big". It is assumed in the example that the robot already knows about a banana instance that is yellow. In our experiments, this kind of knowledge was

[1]Note that in example 2, "give me the banana", the pronoun "me" appears before "banana", while it is an indirect complement — "give it **to me**". The parser handles these cases, and correctly identify "me" as an indirect complement.

[2]Orders are here represented as human desires: the human desires a specific new situation.

either hard coded in scenario-specific ontologies (*e.g.* ⟨`banana_01` *type* `Banana`⟩ where `banana_01` is the ID of the banana's tag) or taught to the robot with prescriptive sentences like "Learn that this is a banana" while pointing at the banana's tag. It would be interesting to extend this approach with automatic classifiers (for colour, size, etc.). If the robot later discovers a yellowish and large object, an utterance like "the yellow banana is big" could be used to assert that this object is a banana. A similar approach focused on the combination of visual perception and communication modalities to achieve visual learning has been developed by [142].

Also, while the examples we develop here (and that we illustrate in experiments in the next chapter) are all based on symbols that have a physical meaning, the system deals equally well with abstract, *exo-somatic*, concepts like *Time*, *Event* or *Place*. Demonstrating this in real experiments would be an interesting development.

Chapter recap

This chapter presented the `Dialogs` module, a custom Python application that parse a subset of the English language and semantically ground it into the robot's symbolic knowledge.

The main algorithms have been introduced, and three examples have illustrated how affirmative sentences are converted into new assertions and how orders are analysed with the help of thematic roles.

The processing of ambiguous sentences has also been presented with several disambiguation strategies that take into account the interactors perspectives.

Moving slowly towards the conclusion, the next chapter introduce several paths of evaluation, first from a more theoretical perspective, then through several experimental results.

Chapter 6

Evaluation

The evaluation of our work is split in two main sections. First, a formal summary and analysis of the features of knowledge representation systems that we have presented at section 2.3. We will in particular focus on the ORO framework.

Then, we will provide a detailed presentation of the case-studies and experiments that have been conducted during the four years of the thesis preparation.

These two evaluation facets will naturally lead to the conclusion at the next chapter, where we will draw scientific and technical perspectives for knowledge representation in robotics systems.

6.1 State of the knowledge representation in robotics

This section summarises the various approaches that have been surveyed in the chapter 2 to identify the main research trends and draw some research perspectives from less studied fields.

The table 6.1 sketches the landscape of current research. The main fields of *contribution* for the systems we have surveyed are sorted along the different axis, dimensions of knowledge representation that were identified in chapter 2.

	Category	ARMAR [51]	CAST [48]	GSM [91]	KE JIA [24]	KNOWROB [130]	NKRL [114]	OUR-K [82]	PEIS [29]	ORO [75]
Expressiveness	Logical formalism	TFS	(none)	(none)	ASP	Prolog	NKRL language (FOL + 2nd order extensions)	DL + Horn clauses	CYCL	DL (OWL)
	OWA/CWA				CWA	CWA				OWA
	Modeling uncertainty			++ (stochastic models)		+++ (*ProbCog* [55])	+	+ (*candidate* entities)		
	Meta-cognition					++	+ (transformation rules)			++ (reification, taxonomy walking)
Representation	Space Representation							++		++ (*perspective-aware symbolic locations* [121])
	Time representation			++ (snapshots)			++ (event oriented)	+		
	Actions/Events			++ (events classifiers)		++ (action recognition [17])	+++ (everything is an event)	+++		
	Context						++ (template matching)	++	++ (microtheories)	
	Modality			++ (recursive model)						++ (Theory of Mind [145])
	Self-knowledge					+ (SRDF [68])				
	Memory models								+	+
Reasoning	Standard FOL reasoning				+++	+++	+ (template matching)	+	++	++
	Instantiation and structure alteration	++			++	+ (dynamic instantiation)				++ (TBox alteration)
	Lazy evaluation					+++ (Prolog, computables)				
	Non-monotonic reasoning		++ [45]		+++ [57]					
	Presupposition accommodation			+++						
	Prediction, projection, diagnosis, explanation						++			+ (explanation)
	Task planning				++ [57]	+		+	++	+++ (*HATP*)
	Physics-based reasoning				+	+++ (*naive physics [67]*)				
Acquisition	Cross-modality	+ (pointing gestures)	++	+++ (amodal model)					++ (*ambient intelligence*)	++
	NLP	+++	+++ [64]	+++	+++				+ (template-based)	+++ (*Dialogs* [76])
	Web Resources					+++ (*Web content processing [99]*)				
	Grounding		++	++ (bidirectional)		+++ (*semantic maps [19, 62]*)		+++ (bottom-up)	+++ [85]	++ (*amodal model* [74])
	Intrinsic Motivation		++ [45]							
Integration	...with sensori-motor layers			+		+++ (computables, local geometric models)		+	++ (tuple space)	
	...with executive layers		++ (ubiquitous events)	+		++ (language extensions) [15]		++	++ (tuple space)	++ (semantic events)
	Monitoring and debugging									+ (*remote visualisation*)
	Performances evaluation		++ [47]	++ (Token test)	+	+ [129]				

Table 6.1: Main domains of contribution of current KRS. Italics mean that the feature is implemented as an external module. Main references are given in the table header. When relevant, feature-specific publications have been provided. An empty cell means that either the system has no specific focus on this domain or we could not find relevant literature.

To comment this table, we propose select nine topic based on some of the challenges that McCarthy and Roy have identified (there are listed at section 2.2.1) on the road to natural interaction and, to slightly paraphrase McCarthy, "human-level robots".

6.1.1 Logic formalisms, continuous world

McCarthy underlines the necessity of relying on mathematical logic, as the most fruitful formalism for machine intelligence. This is largely the case. Almost all the systems we have surveyed rely on logical formalisms, in several cases as a mix of declarative languages (often based on Decription Logics) and logical programming languages (Prolog, of course, but not only).

The two exceptions (the CAST knowledge model and GSM) are however also interesting: CAST proposes a pervasive model of knowledge that is seducing from the grounding perspective (but likely suboptimal for deliberative tasks), and GSM encodes knowledge in an amodal (continuous, geometric) model while preserving features that are usually specific of symbolic models (like theory of mind or categorical knowledge).

The interleaving of discreet symbolic models with continuous geometric models remains a major challenge, and techniques vary a lot: besides the CAST proxies and the GSM amodal model, most systems directly extract physical attributes of the environment to insert them in the symbolic model (TAPAS, Ke Jia, OUR-K, PEIS), thus skipping altogether an intermediate geometric model. KNOWROB adopt a top-down approach where local geometric models are set up on-demand to compute symbolic properties like relative locations.

In our approach, ORO has tight (but unidirectional) links with SPARK, the component in charge of geometric modeling and reasoning. Similarly to the GSM approach, SPARK is an amodal model of the environment we can manipulate (3D motion planing, for instance takes place in MHP, the twin brother of SPARK).

6.1.2 Management of uncertainty and approximation

McCarthy also mentions the necessity to deal with *approximate concepts and approximate theories* (that includes representing them and reasoning with them).

The most notable attempt at modeling uncertainty down to the knowledge representation is PROGCOG, an extension of KNOWROB. The theoretical and practical performances (including decidability) remain however difficult to overcome.

The challenge of representing and reasoning under uncertainty has not been tackled in ORO server, at least not *within* the current ORO server knowledge model. Two reasons explain this absence: the decisional architecture developed at LAAS for the robots has not consistent approach to uncertainty representation and management, neither at the *symbol production* level (*i.e.* SPARK or DIALOGS) or at the symbol consumption level (execution control). Thus, no strong incentive pushed us in this direction. Then, the current tools available from the Semantic Web domain to manipulate and reason about knowledge do not provide mature support for uncertainty management. Some effort do exist (like the PRONTO reasoner [63]), but this does not appear to be a major focus in the currently available tools.

6.1.3 Non-monotonic reasoning

McCarthy gives a particular importance to non-monotonic reasoning. Systems like Ke Jia explicitly tackle this issue.

A monotonic system does not theoretically allow for knowledge retraction during the reasoning, which is an important issue in the robotic context where the world model is likely to be often altered. However it is a practical issue only if the reasoning process has to be *continuous* during the whole robot's activity lifespan, which is rarely the case. It is often possible to stop the reasoner, alter the knowledge, and restart the inference process on a new domain.

Non-monotonicity can also be partially dealt with with appropriate time representation and reasoning (the reasoner then only takes into account statements that are set as valid for a given moment).

Finally, probabilistic reasoning also implicitly leads to non-monotonic reasoning, by relying on a continuous description of the state of the world.

Non-monotonicity is however of broader significance to the knowledge representation field, in particular for the representation of common-sense knowledge where *default* representation is currently sorely lacking.

It is also related to what McCarthy calls *elaboration tolerance*: the ability to extend *on demand* the closed domain of interpretation for a given assertion, to take into account new contextual knowledge.

6.1.4 Modeling of contexts

Both Roy and McCarthy mention the importance of formalizing and reasoning about contexts. Many of the KRS we have surveyed mention at some point the context modeling, but no consistent interpretation, let alone theory, of context management has clearly emerged.

Several approaches for building contextualized knowledge have been presented in this thesis: symbolic environment interpretation, perspective taking and independent mental state for each agent, grounded natural language resolution, self-awareness of its own activity. Much remains to be done for a robot to actually identify its current context as well as contexts that may be referred to.

We will further develop this important topic in the conclusion of the thesis.

6.1.5 Reasoning about time, events and actions

While representation of spatial roles (topology, placement, with or without perspective-taking) is well studied and KRS usually integrate with spatial perception and reasoning components, time representation and integration in knowledge systems is less uniform.

The two main tasks that require time reasoning are task planing and sequence recognition (in particular, action recognition). Systems that directly integrate task planing (like OUR-K, KNOWROB) thus have mechanisms (like fluents in KNOWROB) to represent time.

Other approaches include storing snapshots of the knowledge state (like GSM), that is used to move back to past mental states (of the robot or of another agent).

Integration of these techniques with memory mechanisms (forgetting and reinforcement learning) remains to explore.

Temporal modeling is currently absent of ORO, and symbolic and geometric planning are accomplished outside of the main knowledge representation layer. As a matter of fact, knowledge is mostly atemporal in the ORO server: the set of triples stored in the server represents the beliefs of the robot *at the current time*. The robot represents knowledge about the *here and now*. Dealing with time reasoning is delegated to dedicated tools.

However, ORO does provide a simple implementation of a memory mechanism that relies on time: when a statement is attached to a memory profile, the statement is reified and the date of creation is stored. This allows to remove the statement after a period of time that depends on the memory profile.

Statement reification to store timestamps of creation for all statements is technically very easy to do in ORO server, but has not been actually enabled because the performance hit (each reified statement produces four triples instead of one) was not justified by any current use case of ORO server.

Regarding events, several systems (GSM, NKRL, ORO) have adopted an *event-oriented* architecture where conditions (or templates) are used to trigger decisional and execution processes.

A side note about how thematic roles and action models are initialized in ORO: The current implementation relies on a small, predefined set of action verbs that can be recognised from natural language (section 5.3.2). This constraint does not come from the resolution algorithm itself, but rather from the difficulty to automatically extract the thematic roles associated to a verb. An interesting yet easy to implement extension would consist in binding the dialogue processor with the symbolic task planner. The task planner could dynamically provide the list of actions that the robot can process, *i.e.* actions for which the robot can produce a plan. In certain case, it could also validate the understanding of a desire by attempting to plan it.

Also, we could exploit on-line resources like VERBNET [61], which provides a large machine-processable lexicon of English verbs along with their thematic roles.

6.1.6 Grounding in a multi-modal environment

Symbol grounding remains a focus for most of the developers of cognitive architectures for robots, and all the systems we have surveyed implement grounding strategies (...it was an inclusion criteria). Most of the systems adapt their grounding strategies to the sensing modality, so that grounding can not be considered as a single, well delimited process.

Some systems (ORO, GSM) introduce an intermediate step in the grounding process by the mean of an amodal model of the environment that aggregates the perceptions (or suppositions) in a single place. This enables geometric reasoning that takes into account all the perceptual modalities (typically required for spatial perspective taking: we need to know where the humans are looking at, and also where are the objects), and also improve the observability of the system.

We note in addition that the development of what we have called "synthetic sensors" is simplifying the grounding task. The most obvious example is the human tracking

system provided by the low-cost Kinect device, and now used pervasively in the robotics lab. Not only this system segments and computes the pose of humans that enter its field of view, but it also tracks them (including in case they are partially or completely occluded). This kind of high level sensing device does not completely remove the need of grounding one symbolic instance of a human with the physical human, but the task is ways simpler than it used to be.

Many systems also tackle the difficult question of natural language processing. The CAST middleware, in particular, has been used in the European CoSy and CogX projects as knowledge base by linguists [64]. This demonstrates the huge interest for the grounding of verbal interaction within the cognitive robotics community, and we are likely to reach important milestones in this field in the coming years.

Merging more modalities (especially, back-channel communication, deictic gestures and social gazes) also sparks a lot of attention, and is becoming more and more present as symbolic knowledge available to the control layers.

6.1.7 Common-sense

McCarthy actually starts his list by affirming that intelligent systems must be able to *"operate successfully in the common sense informatic situation"*. This question of the *common-sense* is probably one of the toughest because what *common-sense* mean is not very clear at first place (since, by definition, common-sense is, well, common-sense...). We feel, however, that it is related to a diffuse cultural background, and one may even claim that, for a system to acquire common-sense reasoning is equivalent to solve the strong AI challenge (this is at least more or less McCarthy's opinion).

Our knowledge representation systems have a pragmatic and potentially very powerful approach to common-sense: reuse knowledge stored on the Web. While not so many of the systems we have surveyed directly tackle this question (KNOWROB is the only one explicitly working on common-sense reasoning, through physics-based reasoning, integration with Web databases or parsing of semi-structured Web documents), several rely however on Web standards (OWL, OpenCyc) to represent their knowledge. With the development of initiatives like OpenMind that encode with the same standards common-sense facts and rules, there is a strong potential for our robot to gain common-sense knowledge from well-structured online resources in the coming years.

Table 6.2 lists for the systems we have surveyed the current sources of common-sense knowledge.

6.1.8 Learning, representation of experience, introspection

Learning has not been addressed in this work. This is a complex, still emerging field, whose exact meaning and perimeter varies a lot. It was difficult to synthesise in the KRS taxonomy, and we decided to omit it. We did not either really tackled this question in ORO, except in term of knowledge structure alteration based on verbal interaction (this is presented in one case-study, in the next section).

Learning at the level of the knowledge representation system has been explored in conjunction with visual perception systems (mentioned for instance, in the CAST

Project	Common-sense knowledge source
ARMAR/TAPAS	Custom ontology related to the kitchen
CAST Proxies	None
GSM	Predefined categories
Ke Jia	None
KNOWROB	OPENCYC, processed web content, custom OWL-DL ontology, physics simulati
NKLR	None
OUR-K	*A priori* knowledge structure and axioms, custom set of instances
ORO	OPENCYC, custom OWL-DL ontology
PEIS Ecology	RESEARCHCYC

Table 6.2: Underlying common-sense knowledge sources for each project.

project, in [**?**]). KNOWROB, by filling its pool of facts from informations automatically extracted from the Web **??**, can also be considered as a *learning* system (it autonomously acquire knowledge). We lack however a formal study of learning strategies and techniques at the knowledge representation level.

Representation and matching of past experience is a related topic. This ability is a key step for general action recognition, and is of particular importance for the robot to assess the state of the interaction with the human.

While we already mentioned that several systems are able to reason on past states, we are not aware of existing implementation of algorithms to reflect on past experiences.

This is itself related to introspection and meta-cognition: the shift towards explicit knowledge representation exemplified in the nine systems we have presented, has a major impact on the meta-cognitive capabilities of our robots. They can exhibit, manipulate and reason on their internal belief state (what we will call in the conclusion the *cognitive observability*). We have still to discover all the possibilities that are open by this important cognitive ability.

6.1.9 Perspective-awareness

Finally, Roy mentions *the ability to take into account the human perspective*: this cognitive ability, that relates to the representation of different modalities, is present in ORO and GSM. ORO explicits the perspectives in different symbolic mental states while GSM recursively stores models for each agent.

We have already explained how perspective awareness enables advanced cognitive capabilities like a theory of mind, and this is probably the main contribution of this thesis regarding techniques for knowledge representation.

6.2 Experimental evaluation

Experimental validation of our work takes several forms that are presented in this section.

First, it must be noted that our experiments are largely independent from the underlying platform. Since our contributions are mainly at the symbolic reasoning level, we rely on intermediate layers for the back and forth conversion of sensori-motor data to symbols. These intermediate layers are obviously much more dependant on the platform.

We present here three distinct experimental frameworks: first, we present the MORSE simulator (section 6.2.1) that has been developed during the thesis preparation with several features dedicated to human-robot interaction simulation.

Then, several more "traditional" experiments on different robotic platforms are presented (sections 6.2.2, 6.2.3 and 6.2.3). Most of these experiments have been conducted at LAAS-CNRS and in other laboratories involved in the European CHRIS project (*Jido*, *PR2*, *ICub* and *Bert* platforms). Experiments have also been conducted at Munich's IAS laboratory on the *Rosie* platform.

Finally, we report in section 6.2.4 on the theater performance *Roboscopie* that was presented in 2011 at a large general public audience in Toulouse.

6.2.1 Simulation of HRI interaction

The MORSE simulator

The Modular OpenRobots Simulation Engine (MORSE) [33] (figure 6.1) is a open-source tool developed for robotics research. It is a domain independent simulator, where virtual robots can interact with a 3D environment, using sensors and actuators that behave in the same way as their counterparts in the real world. MORSE relies on the advanced 3D (OpenGL shaders) and physics simulation (BULLET physics engine) capabilities of the Blender *Game Engine*, a real-time 3D runtime integrated to the open-source Blender modeling toolkit. This allows for semi-realistic simulation of complex environments.

The MORSE components (sensors and actuators) exchange data with the robotics software via middleware bindings, using a *Software In The Loop* (SAIL) architecture. Middleware supported in the current version include LAAS' Pocolibs library, ROS and YARP, as well as a socket-based raw protocol. This design allows in principle to use the same software in both the real robots and the simulator. Instructions given to the robot are interpreted in the simulator to provide the control of actuators, such as the motion of the robot and its arms. The data from simulated sensors is sent back through the middlewares, *e.g.* exporting the images from cameras, or the positions of the robot, human and other objects of interest.

MORSE provides support for several classes of robots *out of the box*, and allows for easy customization of those, either by composing individual sensors and actuators with empty robot structures directly in the MORSE interface, or through a Python-based script language that permits to conveniently describe robots and simulation scenarii.

Other experiments using simulation have been carried to gather data for HRI [25]. However, these do not involve the actual robot software, and it is another human who takes the role of the robot.

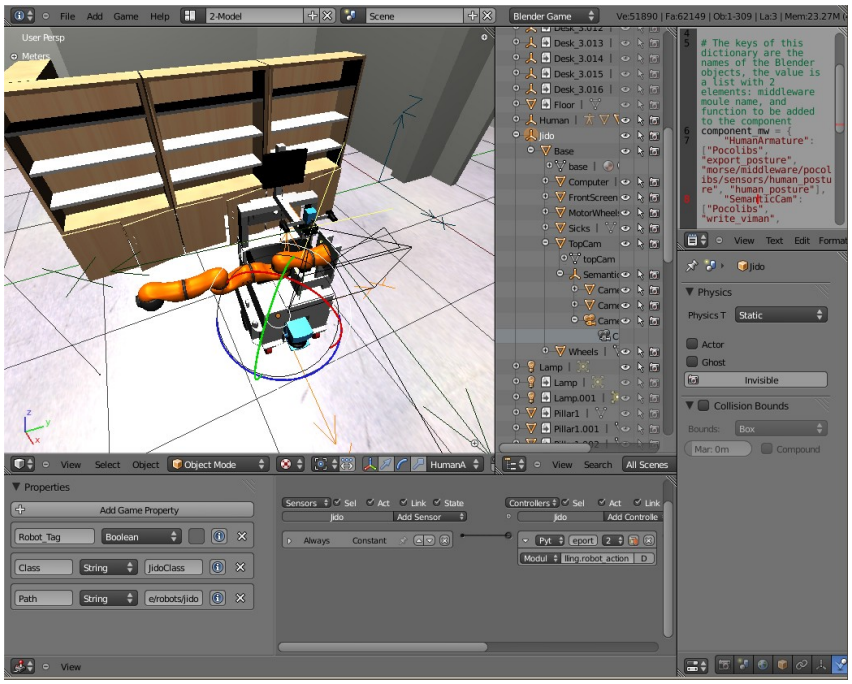


Figure 6.1: Screenshot of the MORSE graphical interface (inside Blender).

Contribution While not directly linked to the main topic of this thesis, I have been deeply involved in the design and development of MORSE: I'm responsible for most of the original software design, and large parts of the core foundations of the project.

HRI specific features

An interactive *human avatar* is available in MORSE. It provides a first-person immersing experience: when started, one can take the role of the human and control it via the keyboard, a WiiMote or through a Kinect device (Fig. 6.2).

When in first-person mode, the user can interact in several ways with the environment. He/she can pick and release objects, can open drawers and cupboards. As any other object, the avatar physically interacts (collision detection) with the surrounding furnitures. MORSE exports the position and posture of the avatar as a global joint state to be used by the real software components of the robot.

MORSE also offers a special sensor that exports abstracted informations of objects visible to the robot (called the *semantic camera*). This sensor typically exports the name, type (glass, table, bottle, etc.), color and location of objects. Since human-robot interaction often involves semantic-rich environments, this abstract sensor simplifies the experiments on such scenarii, by avoiding the added complexity of processing camera images to detect the objects of interest and exploiting the inherent knowledge of the simulated world.

An experimental framework

Due to its nature, MORSE offers two main advantages compared to experiments on a real robot: light-weight deployment and repeatability. MORSE is already used for

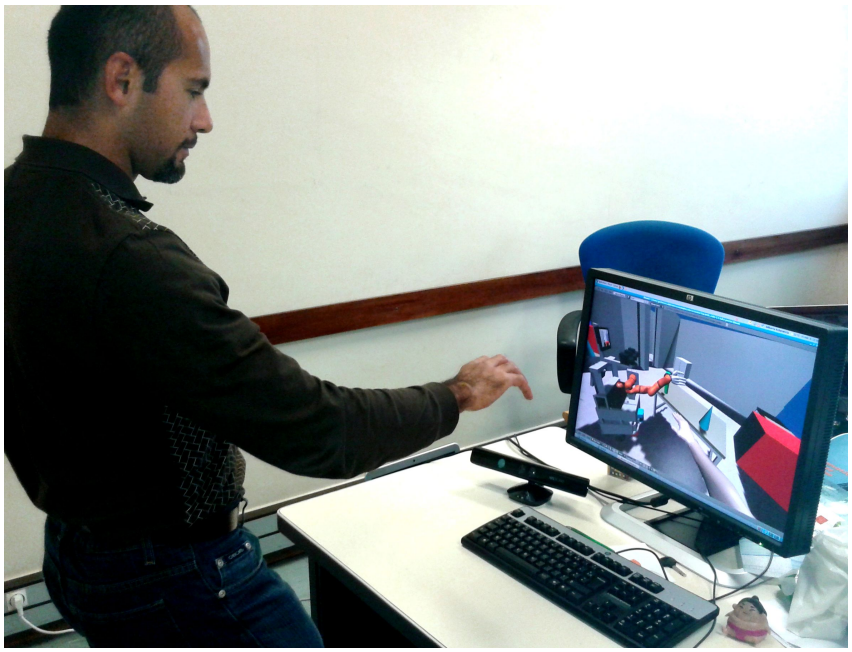


Figure 6.2: The experimental setup with the human avatar controlled from a Kinect.

human-robot interaction both at the LAAS-CNRS and at the Technical University of Munich, Germany.

MORSE is integrated to the LAAS architecture. In particular, both the human posture and the object features are integrated with SPARK, a module dedicated to geometric and temporal reasoning. This module is a key component providing a base of facts such as objects' relative placements, visibility and reachability by the agents present in the scene. It additionally provides a stable state of the world to the motion planners.

6.2.2 Case studies

This section reports on three small experiments conducted in the first half of the PhD thesis preparation. Each of them illustrate one specific aspect of the knowledge base.

The first experiment, *Point & Learn* shows how the structure (*TBox*) of the knowledge base can be altered (in this case expanded) at runtime through pointing interaction.

The second experiment, *Odd One Out* shows how the knowledge model can be used along with the categorisation routines to isolate an "odd" object, given a simple context.

Lastly, the third case study is an implementation of the *Spy Game* where one of the player think of an object and the other one must guess by asking questions.

It must be noted that these three experiments have been implemented on three distinct robotic platforms: the *BERT2* robot from the Bristol Robotics Laboratory (a YARP-based architecture), the *Rosie* robot from the Technical University of Munich (a ROS-based architecture) and the *Jido* robot at LAAS-CNRS (based on the LAAS Pocolibs middleware).

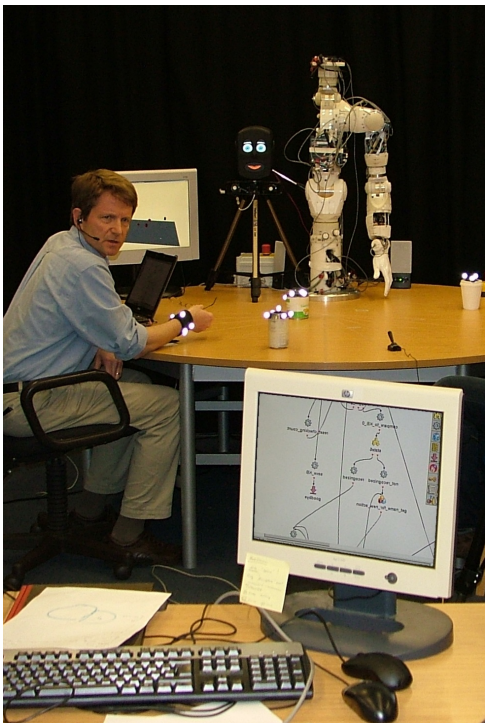


Figure 6.3: Teaching the Bert robot new objects

Knowledge acquisition: Point & Learn

We have implemented a *Point & learn* behaviour on the Bert robot [69] (Figure 6.3): the user shows an object to the robot, and if the robot sees it for the first time, it will ask for its name and type.

The object perception module relies on motion capture (VICON system) to identify and localise objects. A so-called *primitive detection* module is responsible for updating ORO with the list of objects currently seen by the robot as well as their state (moving or not) and their relations to other objects (touching or not). On the other end, a human-robot interface based on the CLSU Toolkit[1] is in charge of speech recognition, speech synthesis and basic natural language processing.

By querying ORO for moving objects, the interface retrieves the object ID that has the focus of attention (last moving object), and asks the human for a name and a type if the object is new. Figure 6.4 reproduces a typical dialog with Bert.

At the end of this sequence, two more RDF statements are added to the robot knowledge base: `[5001 rdfs:label "coffee-cup"]` and `[5001 rdf:type Cup]`.

Due to the limitation of the speech recognition software, only a predefined set of names or types could be recognised, thus preventing the robot to add completely original objects.

Odd one out

The *Odd One Out* scenario extends the *Point & Learn* experiment and completes an ongoing experiment at the IAS laboratory where a robot is asked to list missing items on a

[1] `http://cslu.cse.ogi.edu/toolkit/`

bert	Initializing... *[about 5 sec]* ...What's next?
human	*[moves an object]*
bert	*[does not know the object]* How is it called?
human	coffee-cup
bert	Did you say coffee-cup?
human	yes
bert	Ok. Now I know. What kind of object is coffee-cup?
human	a cup
bert	Did you say cup?
human	yes
bert	So coffee cup is a cup. What's next?

Figure 6.4: Transcript of a chat with the Bert robot

table being set, based on probabilistic reasoning on previously recorded observations.

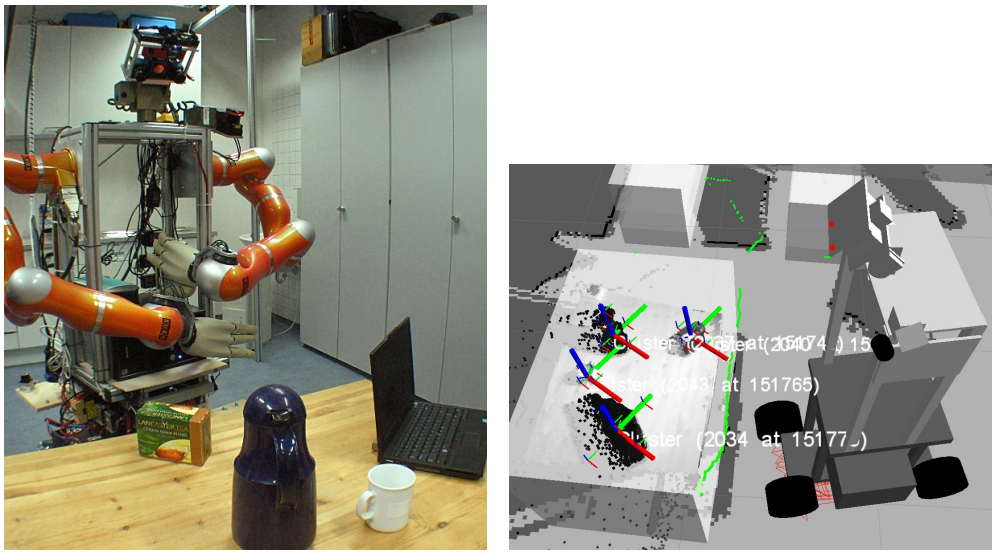
We use ORO to introduce human interactions and common-sense reasoning: the robot picks an unknown object from the table, shows it to the user, and asks about its name and type (Figure 6.5). The user continues to describe the object (through concepts) until a concept known by the robot is given. The learning process starts over again with another unknown object. Once all objects are learned, the robot tells which objects do not belong to a typical breakfast table (*i.e.* objects that are neither food or tableware). The human interacts with the robot through a dedicated XMPP bridge, allowing to chat with the robot with a standard Jabber messaging client. Figure 6.6 corresponds to a chat session with Rosie.

The supervision (CRAM[2] [15]) automatically updates the ORO server whenever an object enters or leaves the field of view (the perception is based on the COP framework [62]). Therefore, the integration of ORO can be seen as an extension to the robot's belief state that not only contains abstract identifiers of the internal object representation used in plans, but also the semantics and roles of objects in the scenario.

By asking in loop the human for the categories of an object until it can connect it to a concept it already knows, the robot accurately anchors perception in its symbolic model and it is able to reason about it. At the end of the experiment, the robot identifies and returns the odd objects for the breakfast table (*i.e.*, in our example, objects that are neither `Tableware` or `Food`).

An unexpected example of what the symbolic reasoning layer brings to more traditional robotic architectures emerged during the *Odd One Out* experiment: the perception routines provided segmented blobs corresponding to objects, along with their colours. The supervision would then feed ORO with the visible objects. At some point, ORO suddenly refused to add an object. What seemed at first a communication bug between modules, was actually the consequence of a consistency check by ORO: Because of bad light conditions, the color recognition was not very reliable, and the same object was set to have two different colours at the same time. That was inferred as impossible by ORO and thus discarded. This kind of logical failure can be used to improve low-level perception results by "closing the loop" with high-level, symbolic knowledge.

[2] CRAM (Cognitive Robotic Abstract Machine) is a RPL-derived framework for rapid development of cognitive robot control programs we currently develop.



(a) (b)

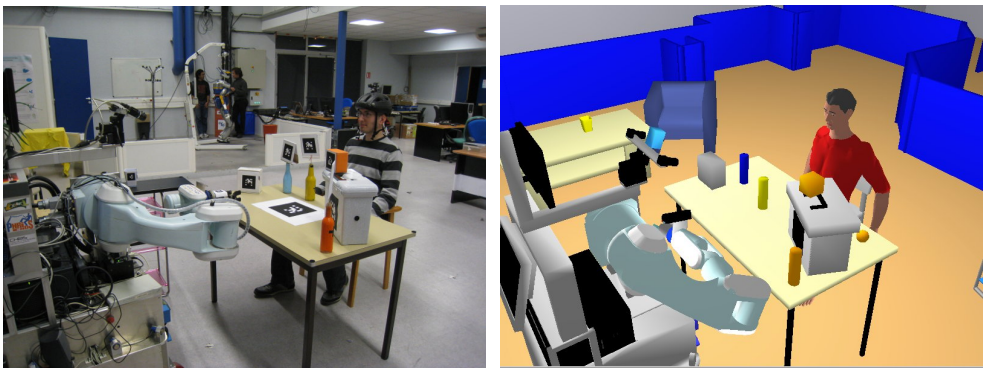
Figure 6.5: (a) Rosie, looking for objects it may know, and (b) viewed in RViz. The clusters of point are given an unique identifier by the perception that allow the supervision create the link between the physical objects and their symbolic representation in ORO.

rosie	Servus! The Bavarian robot is up and fit!
rosie	What is the name of the WHITE CLUSTER on the TABLE?
human	"Rosie's mug" *[Rosie adds* ⟨`OBJ001 rdfs:label "Rosie's mug"`⟩ *stmt]*
rosie	What kind of object is Rosie's mug?
human	Mug
rosie	What is a Mug? *[Rosie adds* ⟨`OBJ001 rdf:type Mug`⟩ *but it does not know the* Mug *concept]*
human	Tableware *[Rosie knows about* Tableware *- belongs to the common-sense ontology. It adds* ⟨`Mug rdfs:subClassOf Tableware`⟩*]*
	(...)
rosie	What is the name of the BLACK CLUSTER on the TABLE?
human	my laptop
rosie	What kind of object is my laptop?
human	Computer
rosie	What is a Computer?
human	Artifact
rosie	Do you want me to learn more objects?
human	no
rosie	These objects are odd: my laptop

Figure 6.6: Transcript of a Jabber session with the robot Rosie. Compared to dialog with Bert (6.4), we see here that the robot anchors the new objects in its already acquired knowledge.

The spy game

This game is based on the traditional children game "I Spy". The idea is to discover the object or concept one of the participants is thinking of by asking questions such as: "Is



(a) (b)

Figure 6.7: Spy game scenario: (a) Real environment and (b) 3D environment model, viewed in MOVE3D.

it green? Is it a machine? Is it on your left?", etc. When playing, children exploit their knowledge about the world while categorising and describing objects through useful discriminants that allow them to find out the answer as fast as possible while including perspective taking abilities [95].

The scenario for this game (Figure 6.7) consists on a face-to-face interaction where the human thinks of an object present in the environment, while the robot queries the human until either discovering the object or giving up, if no object was found. A categorisation example is presented in Figure 6.8. The game starts with the human user giving a first hint (communication is done through a keyboard and screen), allowing the robot to start the search filtering those objects that fulfill this first description. Based on this subset, ORO provides a descriptor (or set of descriptors) that allows a maximum discrimination among objects in the subset. The robot queries the user about the value of the descriptor (or the most discriminant among the set of descriptors) and with this new information, the current subset of objects is filtered again. The process is repeated until either obtaining a single object that fulfills all the descriptor values, or failing (*i.e.* no object found).

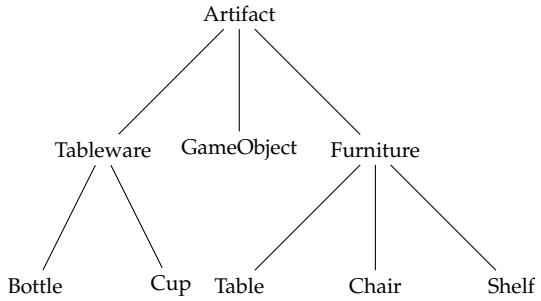


Figure 6.8: Example of object categorisation used in the scenario.

We have integrated the game in the LAAS-CNRS Jido robot [2]. Objects are identified through a tag-based vision approach[3] and we use motion capture for human tracking.

[3]ARToolKit: `http://www.hitl.washington.edu/artoolkit/`

human	It is a tableware.
jido	*[retrieves possible objects: blue-bottle, yellow-bottle, orange-bottle, cup-with-handle]*
	[keeps visible objects: blue-bottle, yellow-bottle, cup-with-handle]
	[obtains discriminants: type, color.]
	Which type of object is: bottle or cup?
human	Bottle.
jido	*[obtains possible objects: blue-bottle, yellow-bottle.]*
	[obtains discriminants: color.]
	What color the object is: blue or yellow?
human	Blue.
jido	*[obtains possible objects: blue-bottle.]*
	The object is the blue-bottle!

Figure 6.9: Example of the robot playing Spy game.

Their descriptions regarding categories (type of object) and features (color, shape) are manually given in advance. Spatial relationships (`front`, `back`, `left`, etc, and `in`, `on` and `next to`) and visibility (only visible objects for both agents can be considered in the game) are automatically computed on-line by the MHP/MOVE3D geometric reasoner and planner [87]. Figure 6.9 shows an example of a round game.

6.2.3 Interaction experiments

Three larger experiments have been also conducted. The first two are focused on knowledge representation and verbal interaction: the "Moving to London" scenario where the human asks for help to find and pack objects, and the "Aperitif Time" scenario (that will be only briefly presented) where the robot tries to prepare a tray with the drinks the human desires. In these experiments, robot actions are limited to simple motions (like head tracking or predefined pick-and-place).

The third experiment, prepared with Mathieu Warnier and Julien Guitton, involves the SHARY execution controller and the HATP symbolic task planner besides ORO and DIALOGS. In this scenario, the human and the robot try to cooperatively remove objects from a table.

First interaction experiment: "Moving to London" scenario

This first experiment is based on the following daily life situation: Tom and Jerry are moving to London, so they are packing things in boxes. The scenario takes places in the living-room, where Jido (our robot) is observing while they move things here and there. To assess the reasoning abilities of the robot they ask Jido for information (entered through keyboard). Ideally, the robot should also perform actions when required (e.g. hand an object when asking "give me..."). However, since it is out of the scope of this work, we do not include any motion from the robot's side.

Perception of objects is done through a tag-based system and humans are detected through motion capture. The robot knowledge base is pre-loaded with the *ORO*

Robot's beliefs about itself (*robot's model*):
⟨videoTape1 *type* VideoTape⟩
⟨videoTape1 *isOn* table⟩
⟨videoTape1 *isVisible true*⟩
⟨videoTape2 *type* VideoTape⟩
⟨videoTape2 *isIn* cardBoardBox⟩
⟨videoTape2 *isVisible true*⟩
Robot's beliefs about Tom (*Tom's model*):
⟨videoTape1 *type* VideoTape⟩
⟨videoTape1 *isOn* table⟩
⟨videoTape1 *isVisible true*⟩
⟨videoTape2 *type* VideoTape⟩
⟨videoTape2 *isIn* cardBoardBox⟩
⟨videoTape2 *isVisible false*⟩

Table 6.3: Robot's beliefs about itself and its human partner.

Commonsense Ontology. We next describe in detail two situations where we can follow the internal robot's reasoning and the interaction with the users.

Implicit disambiguation through visual perspective taking Tom enters the room while carrying a big box (Figure 5.1, page 1). He approaches the table and asks Jido to handle him the video tape: "Jido, can you give me the video tape". The DIALOGS module queries the ontology to identify the object the human is referring to: ⟨**?obj** *type* VideoTape⟩.

There are two video tapes in the scene: one on the table, and another one inside the cardboard box. Thus, the knowledge base returns both: ⇒ `?obj = [videoTape1, videoTape2]`.

However, only one is visible for Tom (the one on the table). Thus, although there is an ambiguity from the robot's perspective (since it can see both video tapes), based on the perspective of its human partner it infers that Tom is referring to the video tape on the table, and not the one inside the box which is not visible from his view. Therefore, non-visible objects are removed obtaining: `?obj =[videoTape1]`.

Since only one object is available, the robot infers that the human refers to it and would eventually execute the command, *i.e.* give it to the human. Alternatively, the robot could first verify with the human if that was the object being referred to or not before proceeding to execute the action. Table 6.3 lists the robot's beliefs about itself and its human partner involved in this situation.

Explicit disambiguation through verbal interaction and gestures In this situation, Jerry enters the living room without knowing where Tom had placed the video tapes. So he first asks Jido: "What's in the box?". Before the robot can answer the question it has to figure out which box Jerry is talking about. Similar to the previous situation, there are two available boxes:

Figure 6.10: Jerry asks Jido for the content of the box by pointing at it.

⟨**?obj** *type* box⟩
⇒ ?obj = [cardBoardBox, toolbox]

However both are visible and the cognitive ambiguity resolution cannot be applied. The only option is to ask Jerry which box he is referring to: "Which box, the toolbox or the cardboard box?" Jerry could now simply answer the question. Instead, he decides to point at it while indicating: "This box" (Figure 6.10). The robot's perception identifies the cardBoardBox as being pointed at and looked at by the human and updates the ontology with this new information using a rule available in the commonsense ontology (**pointsAt**(?ag, ?obj) ∧ **looksAt**(?ag, ?obj) → **focusesOn**(?ag, ?obj)) The DIALOGS module is then able to merge both sources of information, verbal ("this") and gestural to distinguish the box Jerry refers to.

⟨Jerry *pointsAt* carboardBox⟩
⟨Jerry *looksAt* carboardBox⟩
→ ⟨Jerry *focusesAt* carboardBox⟩
⇒ ?obj = [cardBoardBox]

Finally, the DIALOGS queries the ontology about the content of the box and the question can be answered: "Jido-E". Note that the object's label is used instead of its ID. This way we enhance interaction using familiar names given by the users.

⟨**?obj** *isIn* cardBoardBox⟩
⇒ ?obj = videoTape2

At this point Jerry wants to know where the other tape is, and that is exactly what he asks Jido: "And where is the other tape?". In this occasion, the DIALOGS module is able to interpret that Jerry is not referring to the video which they were just talking about, but to the other one:

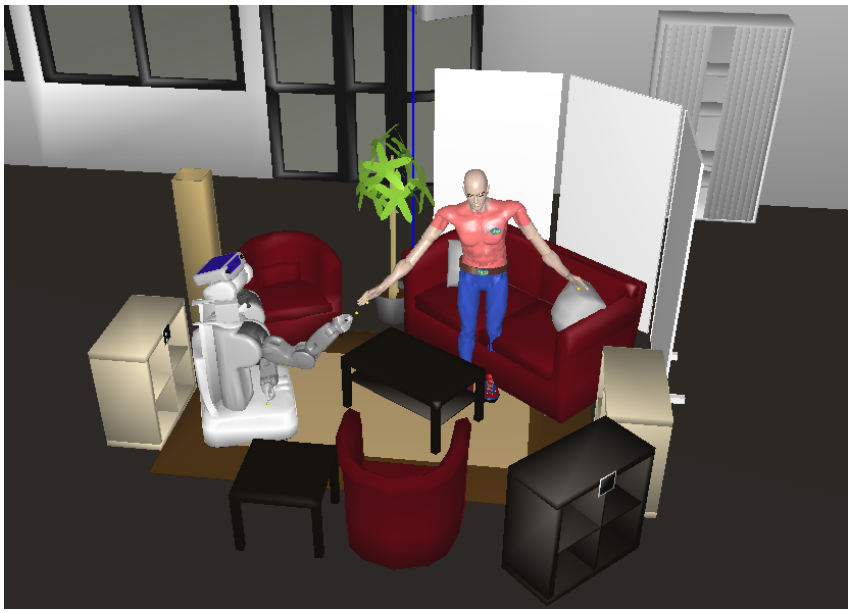


Figure 6.11: The "Living room" setup, as represented by the robot during the experiment.

⟨**?obj** *type* VideoTape⟩
⟨**?obj** *differentFrom* videoTape2⟩
⇒ ?obj = [videoTape1]

Since there is only one possible "other" video (there are only two videos in the scene), it can directly answer Jerry: "The other tape is on the table and next to the toolbox."

⟨videoTape1 *isOn* table⟩
⟨videoTape1 *isNextTo* toolbox⟩

Second interaction experiment: "Aperitif Time"

A similar experiment has been conducted in the appartment environment (figure 6.11) of the LAAS-CNRS with the PR2 robot.

This second experiment is focused on verbal interaction, and the perception is limited to the human tracking with a deported ASUS XTion (similar to the Kinect) sensor. Contrary to the first experiment, objects in the environment are not detected. Their positions are predefined. Pick&Place manipulation tasks are not planned either, and the robot gestures are predefined as well.

Dialogue with the robot goes through a custom Android application running on a touchpad held by the human. This application relies on the Google Speech API for text-to-speech recognition, and communicates exchange messages with the robot via the XMPP/Jabber protocol.

The experiment consists in manipulation of drinks (soda cans, wine minibottles) by verbal commands that take into account the human perspective, and may require discrimination (for instance, "Give me the juice" can refer either to the orange juice or to the apple juice).

Figure 6.12: This experiment led to a video that can be viewed online: `http://www.youtube.com/watch?v=pLz8ifvtoeQ`.

The reasoning and decisional principles are similar to the previous experiment, and are not detailed here. This experiment can be considered as a more mature version of the first one: more objects are involved, speech understanding is vastly improved, the decisional layer is more developed (we use the `pyRobots` environment, briefly presented at section 4.4.3, to react to incoming orders), the robot acts (pick & place, tracking of the human with the head, etc.).

Evaluation The results of these two first experiments is however not fully satisfactory.

- the complexity of the system makes it error prone. Besides, lack of sufficient decoupling between the higher-level components can lead to long restart procedures. In particular, the ORO knowledge base has stability issues (reasoner crashes) that greatly depend on the ontology content in ways that are difficult to predict. As a consequence, these experiments are difficult to set-up and reproduce.

- in the first experiment, the tag-based tracking of object is fragile (sensible to partial occlusions, lighting conditions, etc.), while in the second experiment, the absence of tracking of objects (due in part to the size of cans, too small to stick a tag, in part to the desire to avoid altogether perception issues in the system, and in part to the location of objects, sometimes only partially visible, like the can on the bottom of the picture 6.12) leads obviously to poor robustness of manipulation tasks.

- the high-level behaviour of the robot is also simple (it waits for a spoken goal, executes it, and start waiting again). It was however not the focus of these experiments, and the integration with better high-level control in presented in the next experiment.

Figure 6.13: Snapshot of the film of the "Clean the Table" scenario. The physical situation, the SPARK model, and the current step of the plan can be seen.

It must be also noted that the second experiment has been originally designed as a user-study: a naive, non-expert user was given a list of drinks to ask the robot to prepare for the aperitif. The idea was to compare the time required for the task achievement with several conditions, including use or not of perspective taking. We gave up with this project because of the lack of good enough resilience of the dialogue system to non-expert inputs. For instance ill-formed English sentences or elliptic sentences are not dealt correctly with. A pre-study that was meant to gather example of verbal inputs and involving about fifty users has been however conducted. It provides a valuable database to exercise and further develop the dialogue management system.

Third interaction experiment: "Cleaning the Table"

The third experiment involved a more complex decisional layer where the ORO server was used in conjunction with the HATP symbolic task planner and the SHARY execution controller (the complete software architecture we deployed for this experiment is pictured on figure 4.5, page 82).

In this scenario (figure 6.13), a human and a robot cooperate to remove objects from a table. The robot produces symbolic plans for both itself and the human ([3], see also section 4.4.2) that allow the robot to (verbally) share the task with the human (like "I take the green box and I put it in the trashbin, you take the black video tape and you throw it in the trashbin"). Plans are created based on perceived visibility and reachability of the objects, and the robot also monitors the human activities to track the advancement of the whole plan (this last aspect is presented in [144]).

Figure 6.14 presents a excerpt of the whole task and shows how the different components produce and use symbolic knowledge (note that the plan is actually computed *a priori*. It appears nevertheless on this diagram for better readability).

This experiment also led to a video that can be viewed online: `http://www.youtube.com/watch?v=IODx50uV_k4`.

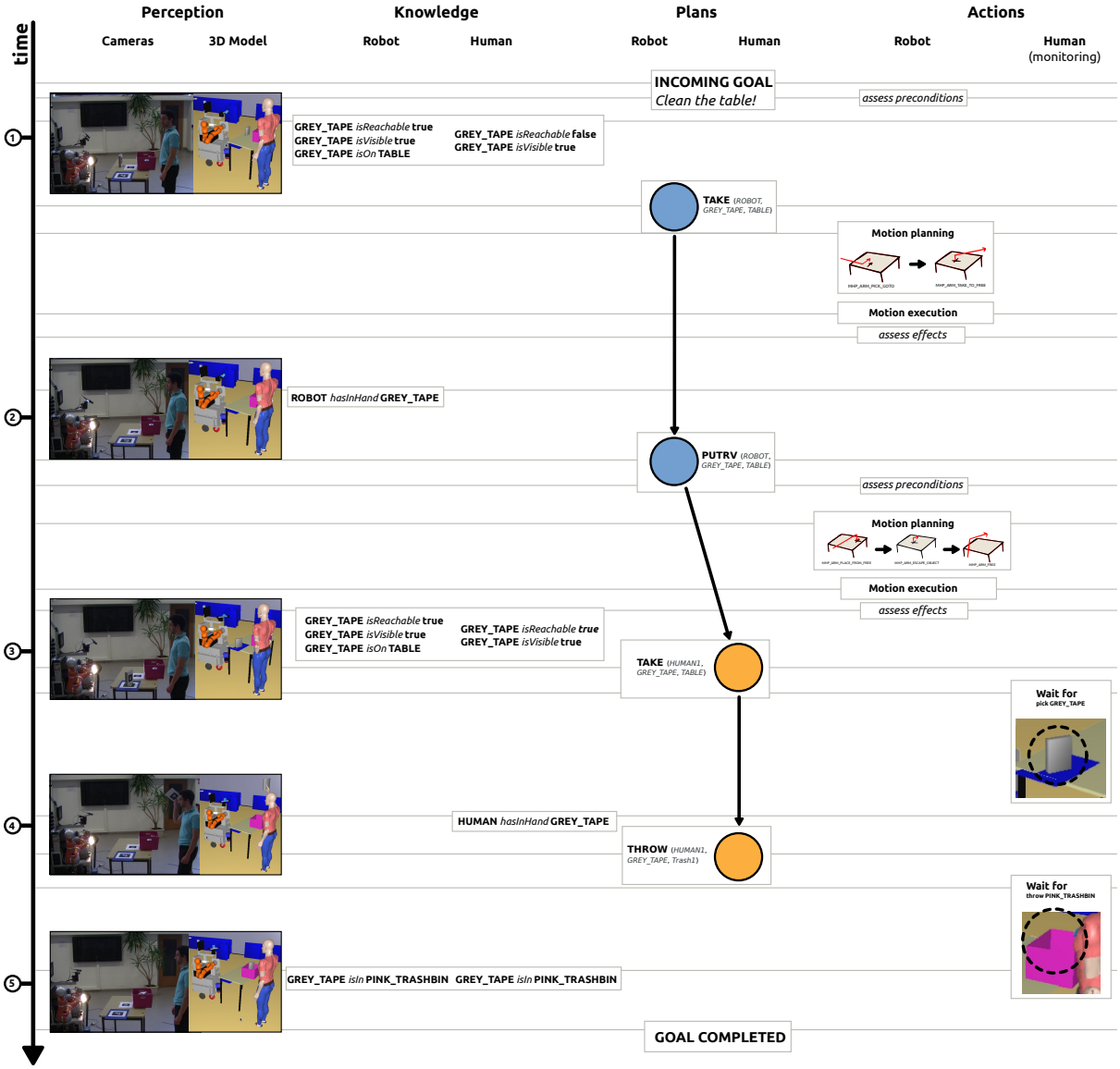


Figure 6.14: This diagram shows a simplified version of the “Clean the Table” scenario timeline.

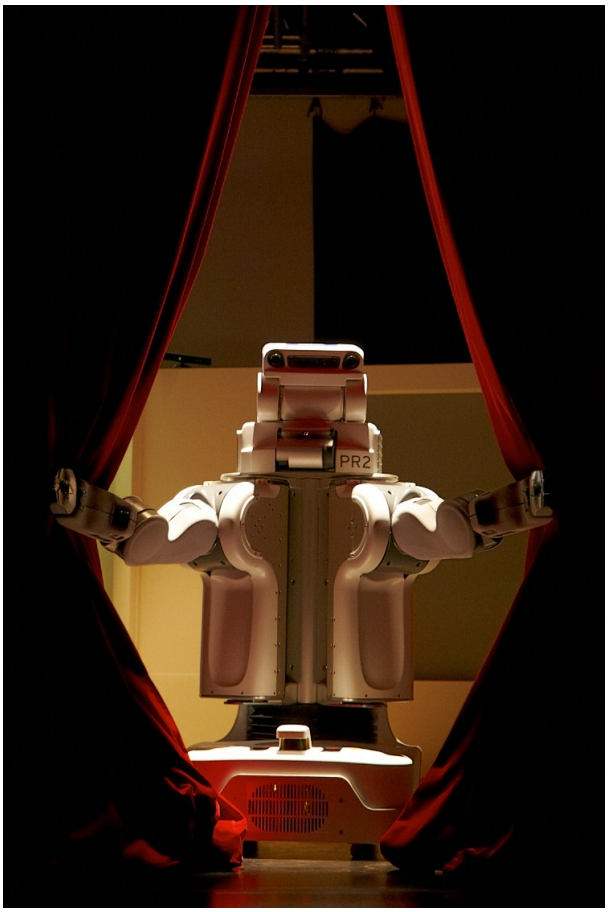


Figure 6.15: The PR2 robot at the beginning of the performance.

6.2.4 The Roboscopie performance

The last experimental setup we have been working on is less usual, since it is a theatre performance, involving one human actor and the PR2 robot.

Theatre with robotic actors is an emerging field, with a few previous published results [20, 83, 90].

On the 14th of October 2011, we performed for a general public audience (over 300 persons) a 18 min long live theatre play, acted by professional actor Xavier Brossard and the LAAS/CNRS PR2 robot. The play was created and directed by Nicolas Darrot, a mixed-media artist from Paris.

The PR2 was programmed in a 2-months course, re-using several software components presented in this thesis, including the 3D environment for situation assessment SPARK, the knowledge base ORO and the natural-language processor DIALOGS.

This section presents the storyline of the play, gives details on the technical side of the project, and underlines some of the significant outcome from the human-robot interaction perspective.

Both a short teaser and the full-length version of the performance are available from a dedicated website, `www.laas.fr/roboscopie`.

Storyline

The play discusses how humans and robots can find a common ground for understanding each other, by living in a kind of frontier world, where real objects are replaced by abstract, disembodied counterparts.

Xavier and PR2 share a white, almost empty, stage. To get the robot to see his world, Xavier must keep being recognised by the human tracking module that lies on the wall, and must stick everywhere 2D barcodes, instead of real objects. The robot can read and identify these barcodes, and while the stage gets covered by the tags, the robot constructs for itself a 3D world with the right-looking objects: a phone, a lamp, a hanger...

While Xavier is drawing by hand more and more of these 2D tags, the robot tries to offer its help. It brings first a bottle of water, then a fan... which blows away all Xavier's code. Angry, Xavier leaves, and PR2 remains alone.

The night comes, and the robot decides to explore the stage, scattered with those barcodes on the ground. On the next morning, Xavier discovers that the robot's 3D model is a mess, full of random objects: an elephant, a boat, a van... Xavier resets the robot model and starts to tidy up the place. The robot decides to help with a trash bin, but suddenly gives up and a new program starts: a home-training session. Xavier starts the exercises, but as the program goes along, the robot looks more and more menacing, up to the point that Xavier shouts "Stop!".

Xavier eventually shows one after the other the objects to the robot, explaining they are all fake, and like the robot, we realize that everything was just an experiment.

Technical overview

The PR2 robot was running softwares developed at the LAAS/CNRS. While the performance tries to picture some of the challenges in the human-robot interaction field, including the needed autonomy of a robot working with humans, the robot was partially pre-programmed for this theatre performance.

Most of the behaviours were coded in Python, relying both on the PR2 ROS middleware and on GENOM, the LAAS own middleware.

What was pre-programmed?

- **General behaviour** While the real perception routines were running (see next section), the robot did not have any mean of synchronization with the human during the play: each sequence was manually started by one of the engineers.
- **Predefined positions** Places on the stage were hard-coded: for instance, the position of the table was known to the robot from the beginning, so was the position of the entrance door, etc.
- **Postures and manipulation tasks** Manipulation tasks (like grasping the fan or the paper bin) were much simplified: the robot would simply open its gripper, and wait for *something* to be detected in its hand. It would then simply close the gripper. Likewise, the robot special postures to enter or leave the stage with an object in hand (required to avoid collision with the door) were all pre-defined.

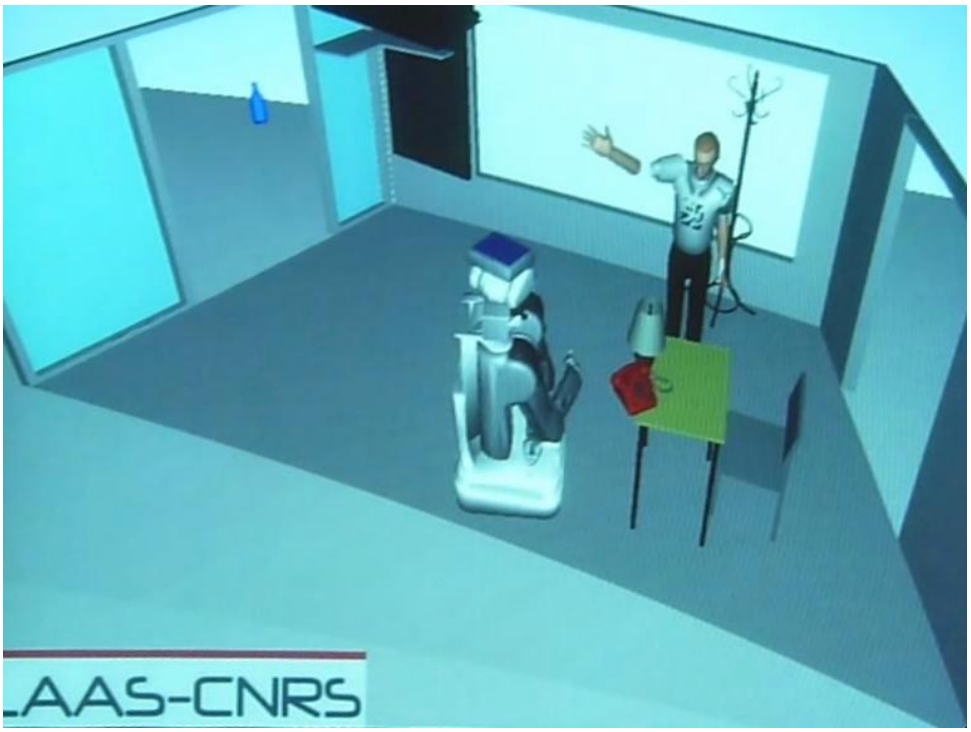


Figure 6.16: The robot build a coherent 3D model of its environment through the SPARK module

- **Speech Understanding** At the end of the play, when Xavier talks to the robot (*Stop!*, *Look at this phone!*, *Everything is fake*, etc.), sentences were manually typed in the system. We could have used speech recognition as we do in the laboratory, but converting speech to its textual version is relatively slow and error prone. So we decided to avoid it on the stage.

  While what Xavier said was actually processed by the robot (see next section), the actions that followed (like looking at the phone, turning the head back to the audience,...) were manually triggered.

What was autonomously managed by the robot?

- **Navigation** All navigation tasks were computed *live* by PR2, using the ROS navigation stack. The main script just tells the robot to go from the engineer desk to the center of the stage for instance. The robot would then find a path that avoid obstacle.

- **Modeling of the environment** The 3D world that is displayed above the stage during the show (figure 6.16) is a live capture of the Move3D and SPARK softwares. These softwares are used daily on the robot to compute trajectories, symbolic locations, visibility and reachability of objects, etc.

  However, during the performance, we deactivated the computation of symbolic facts (like `xavier looksAt jacket`, `RED_PHONE isOn table`,...) which is not reliable enough to be used on the stage.

The 2D barcodes are actually a key perception mechanism for our PR2. They are well identified ARTOOLKIT tags used to identify and localise (both for the position and the orientation) objects surrounding the robot.

Besides, the robot was able to track the human whole-body posture with a Microsoft Kinect sensor and the OPENNI human tracker. In several occasion, the robot automatically tracks the human head or the human hands with this system.

- **Speech Understanding** At the end, Xavier talks to the robot. The textual version of what he said was fed to the system *as it*. Natural language understanding is done by the `Dialogs` module and used extensively the `oro-server` knowledge database to make sense of the word in the current context. The result of the language processing was then added back to the knowledge base and automatically displayed by the `oro-view` OpenGL ontologies viewer.

  Hence, the sentence *"look at this phone"* get translated into symbolic facts: `[human desires action1, action1 type Look, action1 receivedBy RED_PHONE]`. The robot is able to know that <this phone> is indeed the `RED_PHONE` by taking into account what the human focuses on.

  Since the computation of symbolic facts was deactivated, we had to manually add several symbolic facts in a so-called scenario-specific ontology.

Significance for HRI

A first noteworthy achievement of this project from the human-robot interaction point of view is the use and display of the set of research tools developed at LAAS in front of a general audience: while the show had been precisely scripted and rehearsed, the robot was running the same software components we use on a daily basis in the laboratory.

By building the performance storyline on the current, actual state of robotic research, the play also put light on three key questions of today's human-robot interaction: how the human and the robot can understand each others (the robot tries to help but remains intrusive)? how to share and coexist in a common living space? how roles build up between the human and the robot (who dominates)?

Chapter recap

This chapter was focused on the evaluation, from two different perspectives.

First, we have presented a large table that summarises how the main features and requirements of knowledge representation systems identified at chapter 2 are matched by the existing implementations. We have discussed the successes and limits of the current state of the art based on a list of high-level requirements established from McCarthy and Roy proposals towards more advanced artificial cognition.

Amongst the fields that are identified as requiring more research efforts, non-monotonic representations, explaining of the mental state and the mental processes, and context representation represent largely open challenges.

Then, we have presented several experimental frameworks and experiments.

MORSE, a simulator with strong HRI capabilities has been briefly presented. Three case-studies demonstrating the classification and learning capabilities of ORO have been then introduced. Three larger experiments, involving verbal interaction, task planning and execution control have been then presented and explained.

Finally, we have mentioned the Roboscopie theatre performance as an unusual experimental setup, targeted at a general public audience. We see it as a way to materialize and discuss the relationships between the robots and the humans, and to address these questions "in the open", outside of the closed frame of the laboratory.

Chapter 7

Conclusion: On the Road to the Knowledge-Enabled Robot

The time has now come to reach a conclusion to this work.

We have first proposed a systematic study, build as a typology, of the knowledge requirements of modern robotic applications in the context of service robotics and human-robot interaction. About fifty concepts or features that define altogether what knowledge representation means for robotics have been proposed,followed by a review of existing tools and frameworks in the research community.

In a second part, we have presented in depth a particular instantiation of a knowledge representation and manipulation system that we have called *ORO*. Main features, inner workings, algorithms, implementation of this system have been exposed, as well as its integration with several other robot components, for geometric reasoning, task planning or control.

The third part of the study has been focused on the processing of situated dialogue. Our approach and associated algorithms leading to the interactive grounding of unconstrained verbal communication have been presented and illustrated.

One chapter was then dedicated to evaluation. First in term of abstract features: based on several challenges that where identified in the artificial intelligence community, we have examined how the current systems for knowledge management compare. Then in term of experiments and experimental tools, by presenting six case-studies and experiments that illustrate how knowledge can take place in our robots.

7.1 The palpable knowledge

When starting this PhD, we were given *carte blanche* to explore ways to explicit knowledge in our robot architecture, to make it one of the robot's resources in its own right.

The main goal was to transform the knowledge in the robot from some ubiquitous, pervasive, multi-modal and, most importantly, mostly undefined feature of the system into an observable, quantifiable, manipulable resource, what we could call a *palpable* feature.

This transformation, both from the technical point of view (the ORO server, the ontologies, the bindings, etc.), and as a more subtle change in the practises related to

the development of robotic components, is the main contribution of this thesis.

Knowledge is not an abstract concept anymore: it is a set of statements, in most cases directly intelligible to the developers, stored in one place. We can export them, monitor them, review them, question them.

Communication between the robot's module is now conceived in term of what are the *semantics* of the information flows, instead of a simple compatibility of interfaces. When defining the frontiers of a robotic component, we do not think anymore only in term of "is the interface complete and self-contained", but also in term of "is the semantic complete and consistent?". This allow a deeper, more correct modularity: two modules that share the same, well-defined semantic can be confidently exchanged. When we remove or disable a component (the dialogue processing, the geometric reasoning, ...), we know precisely what knowledge will not be available anymore.

We call this new property of our robot, that allows for both qualitative and quantitative analysis of the beliefs, its *cognitive observability*.

It is somewhat related to the idea of *cognitive penetrability* introduced by Pylyshyn [107] in 1989, in the context of the study of possible strong equivalences between computational models and the *psychological reality*:

> [One of the criterion] relies on the assumption that we can identify certain clear cases of phenomenon that should be accounted for at the knowledge level, that is, in terms of the representations alone, rather than in terms of properties of the cognitive architecture. Phenomena that depend in a rational way on subjects' goals, beliefs, and utilities are a case in point. For example in psychophysics we assume that if a measure (such as a threshold) changes systematically as we change the payoffs (that is, the relative cost of errors of commission and of omission), then the explanation of that change must be given at the knowledge level – in terms of decision theory – rather than in terms of properties of sensors or other mechanisms that are part of the architecture. In general showing that certain empirical phenomena are sensitive to goals and beliefs (or what I call *cognitively penetrable*) is prima facie evidence that they should not be attributed to properties of the architecture.

The introduction of an explicit *knowledge level* in our architecture makes it possible to effectively assess the cognitive penetrability of the whole robot behaviours (this is however not new, and traditional BDI architectures would also make this claim).

7.2 Knowledge-oriented architectures

We can give a broader look at the knowledge and the streams of knowledge in our systems. Based on the experience gained while developing and deploying ORO, our ontology-based knowledge server, we have presented how symbolic knowledge could be produced from perception and geometric reasoning in modules like SPARK, a grounded, perspective-aware, geometric reasoner. We have seen how symbolic knowledge could be reused by different control systems and task planners like CRAM, SHARY, PYROBOTS, the CLSU TOOLKIT or HATP and how they take advantage

of semantic abstractions provided by knowledge base. We have also presented the bidirectional integration of DIALOGS, a natural language processor for English, with the knowledge base.

Altogether, these components compose an architecture that we call *knowledge-oriented*:

- Knowledge is explicitly stored in one central and consistent repository of facts, accessible by all modules.
- Knowledge is represented in a strict formalism (OWL statements) and with a clearly defined vocabulary (stated in the common-sense ontology).
- The first two points enable both a loosely-coupled architecture where modules can very easily be removed or replaced by other ones as long as they share the same semantics (modules are defined by the knowledge they produce),
- and a *symbolic* reactive, event-driven approach to supervision. By managing events at the same level as the reasoner, we take full advantage of the inference abilities of ORO to trigger events whose `true` conditions can be inferred.
- Finally, this architecture allows for the combination of very different knowledge modalities in a single homogeneous environment, bringing mutual benefits to components. For instance, the dialogue processing module can perfectly run without any geometric perception, but its disambiguation routines can transparently benefit from it when available (since richer symbolic descriptions of objects are then available).

This architecture moves away from standard layered approaches. Interactions between components are mostly bidirectional and, from the software components point of view, we do not introduce layers of abstraction (we do, however, have access to the lower level modules of the robot to execute actions, but all cognition-related modules reside at the same level). This is especially visible for the dialogue input processing. This component does not simply act as an alternative perceptual input to the symbolic database, but also actively queries previously acquired knowledge to disambiguate and validate the newly created symbolic knowledge.

Our architecture relates but is to be distinguished from *Beliefs, Desires, Intentions* (BDI) architectures. BDI architectures are primarily focused on *practical reasoning*, *i.e.* the process of deciding, step by step, which action to perform to reach a goal (as summarised by Woolridge [146]). The management of the interaction between knowledge (the beliefs) and task and plan representation and execution (the desires and the intentions) is central, and aims at selecting at each step the best subgoal. It becomes then an intention that the robot commits to.

This interaction between knowledge and actions is also central to our approach (as for any cognitive system), but task representation and task execution is not seen as a monolithic, central function: it is one of the activities of the robot, actually split between communication components (that can acquire desires from interaction with agents, amongst other things) and an execution controller that may decide to take an incoming desire into account to create its own internal goals. The controller generates

and controls intentions from these goals with the help of a symbolic task planner, that has also direct access to the knowledge base.

The architecture is not only focused on this workload, and other activities are conducted without being explicitly considered as desires: assessment of the situation and the environment, dialogue (including performative dialogue that can possibly change the internal state of the robot, but does not lead to the creation of desires, like question answering or statement assertion), various background monitoring and recognition tasks, etc.

Regarding the anchoring question, this architecture is bidirectional. The components we described provide a *bottom-up* grounding process: SPARK and DIALOGS constantly build and push new symbolic contents about the world to ORO where it becomes accessible to decisional layers. In parallel, ORO relies on reasoning in a *top-down* way to produce new facts that may trigger in return physical behaviours.

We believe that this *knowledge-oriented* approach has a strong potential not only to enable rich human-robot interaction, but also as a broader approach to information alignment and fusion in complex robotic systems. The versatility of this paradigm could be illustrated by a simple imaginary scenario with a blind robot and a deaf robot. The blind robot does not see (no cameras or alike), but someone can verbally describe a scene to it. On the other hand, the deaf robot has a good vision system, but cannot process verbal input. Without any changes to the software architecture that we described, control modules of both robots could equally perform the same tasks since all the knowledge is abstracted and centralised (note that to actually implement this imaginary situation, the blind robot would of course need *a priori* 3D models of objects talked about to enable planning or pick and place actions, and the deaf robot would require at least some gesture interpretation to understand orders).

This architecture may also contribute to bring closer robotics and psychology: it provides clear entry points to implement some classical psychology tests to robots. For instance, we presented experiments focused on issues related to perspective taking. By explicitly enabling independent modeling of the beliefs of each agent, our architecture is especially well suited to set up cognitive and psychological experiments that involve a theory of mind, such as *False-Belief* experiments, as recently presented in [145].

Knowledge and embodiement

The the experiments that were presented in the previous chapter all illustrate how the robot makes use of its embodied nature to establish a meaningful communication with a human. Mainly, because the robot and the human share the same physical environment and they perceive each other, we are able to create a mutual context.

Sloman, in [124], worried however that the strong focus on embodiment in the robotics community has hindered progress towards natural human-robot interaction by focusing on sub-symbolic physical properties. Our approach has hopefully made clear that, similar to Beetz et al. [15] and many other researches presented in the thesis, we do not consider embodiment *per se* outside of a broader symbolic system, *i.e.* our architecture is not bound to the morphology or the low-level sensori-motor capabilities of a specific agent.

However, we can build a model of the "human point of view" because the robot

perceives the human, and is able to estimate, at least partially, what the human perceives or not. We infer that a human focuses on some object because he/she points at it, looks at it, and besides, the object is visible to him. This relies on the embodied nature of the interaction. In turn, this allows us to understand the meaning of sentences like "Give me that".

We hope that this contribution shows that considering embodiment as the most challenging and fruitful characteristic of robotics in regards to the whole AI community does not contradict with a formal, highly symbolic approach of the representation and decision problems that arise in robotics.

Real-world symbolic reasoning

> Where to find milk? Milk is a subclass of dairy which is itself a subclass of a perishable goods. The usual storage place for perishable goods is the fridge, so the milk is likely to be found in a fridge.

This example of reasoning, quoted from Moritz Tenorth, is a good example of simple yet non-trivial reasoning. As a matter of fact, only very few of such reasoning cases where positively identified in our scenarii and experiments (and consequently implemented as rules in ORO).

The design choices of our architecture partially explain that fact: first, the planning task (which is the prototypical reasoning task) is delegated to a dedicated, external planner. Then, time is not represented in ORO, and consequently no temporal reasoning takes place at this level: action recognition or monitoring are handled by other layers, and the underlying reasoning tasks are not implemented as explicit symbolic rules in the knowledge base.

The experiments we have conducted are also likely to have too simplistic semantics to let complex reasoning needs to emerge. Scenarii with more complex semantics would be desirable to better stress the expressiveness and inference abilities provided by description logics.

Is reasoning at the knowledge level immature or even superfluous, then? Not so: hundreds of trivial (from a human point of view) inferences are continuously produced by the system (translating inheritance relations, domain/range constraints, transitivity, etc.) and encode a large amount of common-sense knowledge that would be tedious, to say the least, to manually assert. These trivial inferences are all the more important that an expressive knowledge representation language is used: when a language like OWL allows to directly represent high-level concepts like partitions, cardinality restrictions, properties' ranges and domains, it leads to a more implicit (because more abstract) description of the vocabulary that in turn requires more underlying reasoning. With the progress in the understanding of the relations between expressiveness and (tractable) satisfiability, along with the progress of reasoners, more and more of the inferences do not need to be explicit anymore, and consequently move behind the scenes.

And we predict that *common-sense encoding* is likely to remain the main application of reasoning in our robotic architectures, where reasoning related to decision making mostly happen outside the knowledge representation system.

7.3 Towards the next generation of knowledge representation systems for robotics

In the chapter 6, we have summarised the current state of the art in knowledge representation, and we have put it in perspective with some mid- or long-term views towards intelligent artificial systems, as expressed by McCarthy and Roy.

So, we can now ask the question: what remains to be invented to get the famous "intelligent" robots we all dream of?

Before writing down the final mark of the thesis, we would like to feed the reflexion on the future of knowledge and knowledge representation for robots (service/companion robots in particular, because they are the ones with the most obvious need of symbolic knowledge for living in complex, interactive, semantically rich environment, but this certainly applies to other robots as well). We will mention three aspects, amongst many others, that we see as both important and difficult questions.

First, one of the directions that seems both critical and under-studied in our community is what we can call *context management* in a broad sense. Proper context management should allow the robot to mentally *move around its own experiences* to place itself in the mental situation where the interpretation of an event, an interaction or a situation makes sense. Cognitive functions like episodic memory, theory of mind, projection, diagnosis and many other can be seen as special cases of a generic context management capability.

Managing context means at least two things: recognising contexts and representing contexts. Depending on what context we talk about, recognising contexts can be relatively easy (who is talking to me? where am I?) to difficult (what past experience does my interactor implicitly refers to?). One of the main problem we see with context identification is that it is a fundamentally *multi-scale* problem: at any moment, several temporal, spatial, social, cultural context co-exist and overlap.

This lead to the second aspect, context representation. Contexts are currently often limited to the current spatial and temporal situation. Some projects model offer the possibility to jump in the past or to switch to another agent's perspective, but in current approaches, selecting a context always basically consists in retrieving a set of beliefs corresponding to a situation, and temporarily replacing the current beliefs by those other ones. This misses the fact that at a given moment, not one but many contexts co-exist at different scales. We do not want to retrieve one monolithic set of beliefs, but instead carefully craft a context from several *atomic* contexts. Techniques for representation of overlapping pools of knowledge largely remain to be developed, as well as efficient algorithms to retrieve (or discard) such context-related pools of knowledge.

The ability to explicitly manage contexts and context switches would endow the robot with a cognitive capability similar to what is known as *context-dependent memory* in cognitive psychology. This is also related to Tulving's *autonoetic consciousness* [135]: the ability to reflect upon its own past or future experiences.

From a technical standpoint, proper context management would mean a transition from a monolithic knowledge base to an more modular architecture, with either multiple (overlapping) models or *facets* (one per agent, one per place, one per period of

time, etc.), or maybe a systematic use of reification to attach to each *atom* of knowledge (the atom is usually the statement. It could maybe be extended to a small set of cohesive statements) one or several contexts. The development of modal logic in practical applications is also an important direction to examine.

Much remain to be done to this regard, starting with a formal analysis of what are the relevant contexts for our robots.

Proper management of inconsistent knowledge is another point that seems of particular interest. Inconsistencies are mostly considered today as errors (modeling issues, perception errors, wrong interpretations of communication, etc.) that prevent, at best, further reasoning, at worst, the use of the knowledge base.

However, from a cognitive point of view, logical inconsistencies are a very valuable source of knowledge by themselves. We have previously evoked the role of cognitive dissonances as an intrinsic motivation factor for knowledge acquisition. This can be generalised to many sources of inconsistencies: detection of faulty perception and setup of alternative perception strategies, start of interactions with other agents to fix a wrong model, etc.

Recognition of inconsistencies between different models (for instance, divergences between the mental state of two agents) could also be a fruitful motivation for action and interaction for the robot.

To this respect, technical handling of inconsistencies also require more research efforts, that include the development of techniques like default logics for robotics.

The systematic study of the relations between symbolic and geometric models are another broad field open to research. While (discrete) symbolic models are usually seen as abstractions of a (continuous) geometric model, this link does not need to be unidirectional. Presupposition accommodation is an example of *a priori* symbolic knowledge that may alter a geometric model. Many more of these bidirectional relations remain to be identified.

One overlooked aspect of these links between symbolic and geometric realms is the temporal reasoning granularity: one of the challenging task for a robot interacting with humans relates to action recognition and prediction. While mid- to longterm action recognition is a well studied field [39, 58, 129], early and fine grained action recognition (like gesture initiation, back-channel communication – nodding, social gaze, ... –, etc.) that are important for smooth interaction, requires geometric reasoning at relatively high temporal resolution that is able to operate within a symbolic context. Viewed from a slightly different perspective, better temporal resolution in the knowledge stack could lead the way to programming paradigms that are massively event-oriented and still semantically grounded.

Our last word will be for the idea embodiment and the research perspectives it opens: the knowledge-oriented architectures that are being build in many robotic research lab around the world have a very specific characteristic compared to the efforts of other research communities working on the question of knowledge by human systems, like in cognitive psychology, or computer-based systems, like in the semantic web community: robots are *embodied computers*. They act and interact in the physical world, and the physical world plays a key role as a communication support. And at the same time, they are computers, with unlimited access to remote sources of knowledge via the Web, either as static database, or through exchanges with other robots.

This dual essence, both as embodied organism and disembodied agent, places the robot at the crossing of two fundamental approaches to knowledge management: either physically and experientially grounded, central, internal to the agent, or on the contrary ungrounded, distributed, pervasive. The robot has this rare privilege of being an intelligent entity that can merge and take advantage of both. The research efforts to achieve this fusion have started, foundations have been laid, much more remains to be explored.

Appendices

Appendix A

Description Logics Semantics

This appendix describes some notations and the naming convention of Description Logics. The content of this page comes from the Wikipedia page on Descriptions Logics[1] and the DL Complexity Navigator [149]. The academic reference on this matter is [9].

ALC Let N_C, N_R and N_O be (respectively) sets of *concept names* (also known as *atomic concepts*), *role names* and *individual names* (also known as *individuals*, *nominals* or *objects*). Then the ordered triple (N_C, N_R, N_O) is the *signature* of the language.

Description Logics are implicitely *Attributive Concept Language with Complements*: $\mathcal{ALC}$. The set of $\mathcal{ALC}$ *concepts* is the smallest set such that:

- The following are *concepts*:
  - $\top$ (*top* is a *concept*)
  - $\bot$ (*bottom* is a *concept*)
  - Every $A \in N_C$ (all *atomic concepts* are *concepts*)
- If C and D are *concepts* and $R \in N_R$ then the following are *concepts*:
  - $C \sqcap D$ (the intersection of two *concepts* is a *concept*)
  - $C \sqcup D$ (the union of two *concepts* is a *concept*)
  - $\neg C$ (the complement of a *concept* is a *concept*)
  - $\forall R.C$ (the universal restriction of a *concept* by a *role* is a *concept*)
  - $\exists R.C$ (the existential restriction of a *concept* by a *role* is a *concept*)

This can be formulated as ALC languages allowing:

- Atomic negation (negation of concept names that do not appear on the left hand side of axioms)
- Concept intersection
- Universal restrictions

[1] `http://en.wikipedia.org/wiki/Description_logic`

- Limited existential quantification

The expressiveness of ACL languages can be extended, following this naming convention:

- $\mathcal{F}$ for support of functional properties,
- $\mathcal{N}$ for cardinality restrictions (`owl:cardinality`, `owl:maxCardinality`, implies $\mathcal{F}$),
- $\mathcal{Q}$ for qualified cardinality restrictions (available in OWL 2, cardinality restrictions that have fillers other than `owl:Thing`, implies $\mathcal{N}$),
- $\mathcal{E}$ for full existential qualification (Existential restrictions that have fillers other than `owl:Thing`),
- $\mathcal{U}$ for concept union,
- $\mathcal{C}$ for complex concept negation,
- $\mathcal{S}$ for role transitivity,
- $\mathcal{H}$ for role hierarchy (subproperties - `rdfs:subPropertyOf`),
- $\mathcal{R}$ for complex role inclusion axioms (reflexivity and irreflexivity; role disjointness, implies $\mathcal{S}$ and $\mathcal{H}$),
- $\mathcal{O}$ for nominals (enumerated classes of object value restrictions - `owl:oneOf`, `owl:hasValue`),
- $\mathcal{I}$ for inverse properties,
- $(\mathcal{D})$ for use of datatype properties, data values or data types.

OWL2 is a $\mathcal{SROIQ}(\mathcal{D})$ languages, which can be written in expansed form as $\mathcal{ALC} + \mathcal{SHRFNQOI}(\mathcal{D})$. This is the expressiveness level of the ORO common-sense ontology.

Appendix B

Task representation in the ontology

This appendix is a specific study of the representation issues that arise when one tries to model actions, pre-conditions and post-conditions in ontologies.

B.1 The challenge of task representation

It has already been presented at chapter 2, managing tasks in a robot includes these kind of cognitive capacities:

1. the ability to infer which tasks could be started at any time,
2. the ability to check that if we do some task, it won't bring the world in an inconsistent state,
3. more generally, the ability to predict the state of the world after some task execution,
4. retrieve tasks that should be started to achieve some result,
5. knowing how long a task lasts.

These capabilities are traditionally deferred to dedicated reasoning systems (planners) that take into account different constraints (time, current activity, agent's desires...) to select tasks and build plans.

The ontology, as built by the robot during its lifetime, is a model of the world that can help the planner. It can efficiently represent some of the knowledge required for planning and task execution:

- Agents state (like, busy, reading, standing, talking...), desires (short or long term goals)
- Agents capacities (technically doable actions)
- Physical world state (location of objects,...)
- Common-sense knowledge (role, usage of objects...)

Note that this knowledge is mostly declarative. Storing of procedural knowledge is more difficult. But we'll come to it later.

Now, what are *tasks* from the ontology point of view?

1. Tasks are **instances** of **actions** that have some purpose. As it, they are instances of `cyc:PurposefulAction`

2. The type of action can be further refined by using sub-classes of PurposefulAction (like `cyc:Movement-TranslationEvent`, `cyc:Reading`... the robot is expected to provide - likely at startup - the list of actions it can achieve depending on its hardware)

3. a `cyc:PurposefulAction` is `cyc:performedBy` one (or several) `cyc:EmbodiedAgent`.

A task may have a specific context as prerequisites and reversely, may imply some new state of the world as post-condition. Since these two aspects are largely symmetric, I will only discuss post-conditions.

Post-conditions are difficult to represent, because the new state of the world they represent may contradict with the current state, thus leading to inconsistencies. It is a fundamentally non-monotonic process.

Imagine the task: *"cracking an egg"*. To build a model of this task, we need somehow to encode the post condition: `the egg shell is broken` [stmt1].

We can as well assume that the state of the egg shell before we crack it is: `the egg shell is not broken` [stmt2].

If we add both these statements `[stmt1]` and `[stmt2]` in the ontology (*i.e.* , we assert them), the ontology becomes inconsistent, and no reasoning is possible.

For people doing planning, there's no real issue there: that's precisely the planner role to **replace** `[stmt2]` by `[stmt1]` when the task is achieved. In this case, the two statements wouldn't *hold* (*i.e.* , be asserted to be true) at the same time, and everything remains consistent.

OPENCYC has another powerful way to deal with *possible worlds*: *micro-theories*. Within a microtheory, a certain set of facts must hold. But not necessary outside. To put it in another way, OPENCYC doesn't generally requires the set of facts to be consistent. Only facts asserted in a specific microtheory must be consistent with each others.

In the egg example, it means that statements 1 and 2 can be asserted at the same time. Two microtheories must be created as well (called `cyc:TaskState` in this case), one that would describe the world **before** the task *"cracking an egg"* (it would be linked to the task by the `cyc:taskPrerequisites` predicate), one that would describe the world **after** the task completion (linked with `cyc:taskToAchieve` predicate).

However the microtheories approach does not belong to the Description Logics (it is not anymore a first order logic system), and thus considerably reduce the practical inference capabilities.

Second problem (that is partially a consequence of the first one): none of the standard semantic tools and languages like OWL, Protege, OWL-API, Jena or Pellet, has a notion of microtheory.

B.2 A simple case: the `Move` task

So in practice, some mitigation must be done.

Since the semantics behind `cyc:TaskState`, `cyc:RealizedTaskState` (an effective, realised state of the world), `cyc:taskPrerequisites`, `cyc:taskToAchieve` and `cyc:taskConstraints` (that expresses specific constraints applied during task execution) are still relevant to us, the question is: how to build a `cyc:TaskState` without `cyc:Microtheory`?

A simple yet "real world" example, extracted from the LAAS' HATP planner task model, helps to illustrate the problem:

```
action Move(Agent ag1, Place p1, Place p2)

   preconditions

     p1 != p2;
     ag1.posTopo == p1;
   ;

   effects
    ag1.posTopo == p2; ;
```

Pre- and post-conditions and the relations between entities (`ag1`, `p1`, `p2`) are easy to understand from this model.

To represent it in the ontology, we can start with the pre-condition `P1 != P2`. We want to create a new kind of `cyc:TaskState` that says: "Two locations are different".

```
DifferentLocationState subClassOf cyc:TaskState
forLocation hasDomain DifferentLocationState
forLocation hasRange cyc:SpatialThing-Localized

precondition1 rdf:type DifferentLocationState
precondition1 forLocation p1
precondition1 forLocation p2
```

`precondition1` is an instance of the state `DifferentLocationState`.

Same thing for an `IdenticalLocationState` and the two instances `precondition2` and `postcondition1`:

```
IdenticalLocationState subClassOf cyc:TaskState
forLocation hasDomain IdenticalLocationState
```

```
precondition2 rdf:type IdenticalLocationState
precondition2 forLocation ag1
precondition2 forLocation p1

postcondition1 rdf:type IdenticalLocationState
postcondition1 forLocation ag1
postcondition1 forLocation p2
```

We can then instantiate the task itself:

```
move1 type Move
move1 performedBy ag1
move1 fromLocation p1
move1 toLocation p2
move1 taskPrerequisites precondition1
move1 taskPrerequisites precondition2
move1 taskToAchieve postcondition1
```

While this model is valid, we observe two issues:

- `DifferentLocationState`/`IdenticalLocationState` conditions, as formalised above, lack expressiveness: we, as robot designers, *decide* that *DifferentLocationState* means that the locations must be disjoint, but this rule is not explicit at the symbolic level. The only explicit constraint is that `DifferentLocationState` or `IdenticalLocationState` involve locations (instances of `cyc:SpatialThing-Localized`). But we don't formally say that these locations must be disjoint or identical.
- The abstract model of the task does not maintain the semantic: it merely says that a `Move` task must have two preconditions, one of kind *different location*, one of kind *identical location* and one post-condition of type *identical location*. But the fact that explicitly the location `p1` and `p2` must be different (and not, for instance, `ag1` and `p1` or even `the moon` and `the sun`) is not kept. We set this semantic at instantiation time, but it seriously reduces the relevance of such a task description for anything useful.

As it, it's difficult to me to see the relevance of such a task model for robotics. The original task model from HATP is ways simpler and easier to read. But let dig a bit futher.

B.3 Fluent-based approach

An alternative approach (used at the Technical University of Munich, for instance) is based on fluents and events.

The idea is to represent the various states of the world with instances of `Holds` and `Occurs` classes, associated to a fluent and a time (or time interval).

With the egg example, `occurs(breaking(egg), 10.2)` (which means that the egg was broken at time 10.2s) would be represented in the ontology as:

```
event231 rdf:type cyc:BreakingEvent
event231 hasObject egg

state746 rdf:type Occurs
state746 fluent event231
state746 occursAt 10.2
```

which relies on the reification of events.

This representation matches situation calculus (or variants like event/fluent calculus) theories and takes explicitly into account the time (either time points or time intervals) and is especially suited for plan representation.

However, causal relations are not addressed in this approach, which limit its practical use to represent tasks in the ontology.

Another issue is the lack of integration with classical first-order logic reasoners: fluents are a kind of statement reification and prevent standard reasoners to make inference on the new state of the world after the conclusion of some action. To put it another way: when a fluent holds, we can not directly infer the consequences of this fluent. Even if the fluent `BrokenEgg` holds, the statement `egg rdf:type cyc:Fractured` is not asserted anyhow, and we couldn't directly query the ontology for the list of broken objects for instance.

B.4 Rules based representations

The expressiveness issue we had with tasks represented as pure OWL statement can be partially addressed with rules. The example below, written in SWRL, is rewrites the `IdenticalLocationState` condition (`DifferentLocationState` is similar):

```
IdenticalLocationState(?state) ∧
forLocation(?state, ?p1) ∧
forLocation(?state, ?p2) ∧
sameAs(?p1, ?p2)
⇒ RealizedTaskState(?state)
```

We can now create a new class, `EligibleAction`, that holds all the actions whose pre-conditions are satisfied.

```
//Define our variables
EmbodiedAgent(?ag1) ∧
SpatialThing-Localized(?p1) ∧
SpatialThing-Localized(?p2) ∧

//Bind the Move action
Move(?action) ∧
performedBy(?action, ?ag1) ∧
fromLocation(?action, ?p1) ∧
toLocation(?action, ?p2) ∧

//Bind the agent position
hasLocation(?ag1, ?p3) ∧

//Check precondition 1
DifferentLocationState(?precondition1) ∧
forLocation(?precondition1, ?p1) ∧
forLocation(?precondition1, ?p2) ∧
RealizedTaskState(?precondition1) ∧

//Check precondition 2
IdenticalLocationState(?precondition2) ∧
forLocation(?precondition2, ?p1) ∧
forLocation(?precondition2, ?p3) ∧
RealizedTaskState(?precondition2)

⇒ EligibleAction(?action)
```

This rule would be triggered by the reasoner if and only if:

1. At least one agent is asserted or inferred (it is bound to `?ag1`),
2. Two other, different locations have been asserted of inferred (`?p1` and `?p2`)[1]
3. At least on instance of the action `Move` exists (`?action`), performed by `?ag1` from `?p1` to `?p2`.
4. `?p3` will be successively bound to every current location of `?ag1`
5. One instance of a `DifferentLocationState` state exists with `?p1` and `?p2` as locations. This state is a *realised state* (*i.e.* , it holds).
6. Symmetrically, one instance of a `IdenticalLocationState` state exists with `?p1` and `?p3` as locations. This state is *realised* as well.

[1]Since OWL and SWRL use the Open World Assumption, it means that `?p1` and `?p2` must be **explicitly** disjoint

If all these conditions hold, then the `?action` is inferred by the reasoner as an `EligibleAction`.

Three important remarks, however:

- One could suggest to *factor* the code with a rule that would say: "For any `Action`, if all pre-conditions are `RealizedState`, then the action is undertakeable". Somethink like:

```
Move(?action) ∧
taskPrerequisites(?action, ?state) ∧
RealizedState(?state)
⇒ EligibleAction(?action)
```

  This is not possible in OWL because of the Open World Assumption (OWA): there is not way to be certain that **all** states are realised because there is no way to know **all** the prerequisite states.

- For post-conditions, it's important to understand that we **can not** *apply* the result of a rule to the ontology (non-monotonicity).

  It means that something like that:

```
Move(?action) ∧
SuccessfullyCompleteAction(?action) ∧
IdenticalLocationState(?postcondition1) ∧
forLocation(?postcondition1, ?p2) ∧
forLocation(?postcondition1, ?p3) ∧
⇒ RealizedState(?postcondition1)
```

  or even simpler:

```
Move(?action) ∧
SuccessfullyCompleteAction(?action) ∧
fromLocation(?action, ?p1) ∧
toLocation(?action, ?p2)
⇒ sameAs(?p1, ?p2)
```

works as expected, but only as long as the action is considered a `SuccessfullyCompleteActi`
It means as well that if we do another movement (`move2`) but `move1` is still a `SuccessfullyCompleteAction`, then we are likely to have an inconsistency (or unwanted inferences): the robot will be at several places at the same time.

And even if we can *add* somehow new statements, we can not *retract* statement at all: it's not possible for instance to complete the post-condition rule to say that the agent is not anymore in `p1`.

- Finally, one could conversely want to use post-condition with rules to know if an action was successful:

```
Move(?action) ∧
performedBy(?action, ?ag1) ∧
hasLocation(?ag1, ?p1) ∧
toLocation(?action, ?p2) ∧
sameAs(?p1, ?p2)
⇒ SuccessfullyCompleteAction(?action)
```

This does not prove that the action was successful - or did even occurred at all. It only says that the world is in a state that match the post-conditions of the action `Move`.

At the end, what should we think of rule-based representations of pre- and post-conditions? We will conclude this study on this question, but for now, it is important to underline the complexity of writing correct SWRL rules in the Description Logics safe space. It can however prove useful to check that the current world state is a context that permit some actions to occur or conversely that matches the expected outcome of some other actions.

B.5 Towards a conclusion

Our initial questions where:

1. How to express *pre- and post-conditions* in the ontology? how the knowledge processing framework could be used to check or enforce them?
2. What *abstract model* of a task could be formalised in the ontology?
3. How could we rely on the ontology to *instantiate* tasks?

Can our knowledge representation system help with dealing with pre- and post-condition? Or, are ontologies and associated reasoning tool relevant for the *legality task*

(are tasks performable?) and the *temporal projection task* (what will be the state of the world after some tasks are performed?), as described by Levesque?

We have seen that standard OWL semantics does not allow for a satisfying way to represent the complex semantic and logic of pre- and post-conditions, even in simple case like the `Move` action.

However, the rule-based approach allows us to effectively represent the full expressiveness of these constraints, at the cost of a complex syntax and lack of generality. Since rule language like SWRL are actually implemented in current reasoners like Pellet, such description of task conditions could be actually suitable for checking both that:

- the robot believes the current world state allows a given action to take place,
- the robot believes that it reached a state where the intended results of an actions are realised (which may mean that either the action was successfully complete or that some environment changes lead to an equivalent state of the world)

Per se this knowledge is useful for the execution control. But it can not replace the need for the planner to reason itself on pre- and post-conditions since the reasoning that the ontology framework provides applies only to the **current** state of the world. Indeed, traditional first-order logic reasoners do not offer convenient ways to build hypothetical world models that would come as the result of some actions (in part because of the monotonic reasoning paradigm).

Moreover, as we already stated, using such rules for post-condition only allows to check if the post-conditions are met. It doesn't prove that the action was achieved. More importantly, none of the various solutions we presented above allow to *apply* post-conditions to the ontology. An external module (likely the execution control) must take care of updating the ontology according to the consequences of the action, adding new statements and removing deprecated facts.

It appears that representing certain *states of the world* as a set of constraints expressed as rules can be relevant. It would allow the robot to be *conscious* that a specific situation occurs. But, to be useful, these `cyc:TaskState` (which can be viewed indeed as *contexts*) must be easy to create "on the fly" by the planner or by the execution controller, either to describe a context required by a new task or on the contrary to express an expected situation. It could be useful to develop to this end a library dedicated to SWRL content generation.

Last aspect: are abstract models of tasks easily and effectively representable in our current, OWL-based, knowledge frameworks?

Brandon Ibach, one of the Pellet developers, summarises the model of the `Move` task this way (for the precondition part):

```
Place
Agent [ hasPosition(Place) ]
Action
    UndertakableAction ← EligibleMove ∧ NonEmptyMove
    Move [ hasAgent(Agent) hasFrom(Place) hasTo(Place) ]
        EligibleMove ← Move(m) ∧ hasAgent(m, a) ∧ hasPosition(a, p) ∧ ha
```

```
sameAs(p, f)
        NonEmptyMove ← Move(m) ∧ hasFrom(m, f) ∧ hasTo(m, t) ∧ different
```

This can be probably considered as an abstract model of the task. This model mixes pure OWL statements with SWRL extensions. It's formally sound, but we doubt of the practical usability of such a model for the planner.

Appendix C

The *Knowledge API*

Refer to section 2.4 for the rationale of the API, as well as a discussion of its limitations.

This presentation of the API is slightly reduced: for brevity, some non-essential parts and many examples have been omitted.

C.1 Conventions

- **Service**: we call *service* the set of all methods defined by this API.
- **Method**: *method* refers to one single remote function offered by the service. The terms function, method, and procedure can be assumed to be interchangeable.
- The **Service provider** or **implementation** is a software that implement the API.
- A **policy** is a (extensible) set of rules that defines in which ways the knowledge must be altered. It is represented as a dictionary of (keys, values). The section C.3 details the content of these policies.

C.2 Statements, partial statements and rules

C.2.1 Resources and literals

All entities in the knowledge base are either **resources** or **literals**. Resources can be classes, predicates or instances.

Resources must start with a letter and can be comprised of symbols [`a-zA-Z0-9_`].

They may be prefixed with a namespace prefix (like in `rdf:type`) or come with a complete namespace URI. In this case, the full URI (the namespace and the resource name) must be enclosed between < and >. If no namespace is provided, it is assumed to be defined in the server default namespace.

Literals should follow the SPARQL syntax for literals[1].

[1] `http://www.w3.org/TR/rdf-sparql-query/#QSynLiterals`

C.2.2 Statements

Two syntaxes are admissible to represent a **statement** (or **axiom**). The general one is a string starting with a predicate `p` followed by a set in brackets of comma-separated arguments:

```
p(a1, ..., an)
```

where `n` is the arity of the predicate. For instance:

```
cutsWith(human0, bred0, knife0)
```

The predicate must be a resource, as well as at least one of the arguments. Other arguments may be either resources or literals.

Infix syntax As a special case, the infix syntax (*subject predicate object*) is acceptable for binary predicates (in this case without commas or brackets):

```
s p o
```

For instance:

```
sky0 hasColor blue
```

A **partial statement** is a statement with a least one unbound member. Unbound members are denoted either by a "*" (an anonymous variable, for instance `* isVisible true`. The Prolog's "_" is also acceptable) or by a string starting with `?` (named variable, for instance {`sees(?ag, obj1), ?ag type Human`}).

Partial statements are masks or patterns, and used as such in methods like `find` or `remove`.

Note that, as in Prolog, each occurrence of "*" or "_" corresponds to a different variable; even within a clause, _ does not stand for one and the same object. Wherever a variable is used only once within a clause, the anonymous variable can (and should) be used to emphasize this fact.

C.2.3 Rules

Rules syntax is based on the human-readable form of the SWRL syntax[2], encapsulated in a string. Atoms must be separated by either a comma, ^ (unicode U+005E) or ∧ (logical AND, unicode U+2227). The head and the tail of the rule are separated either by the two symbols `->` or by the implication symbol →, unicode U+2192.

For instance:

- `Yogurt(?x) -> DairyProduct(?x)` asserts that all instances of type `Yogurt` are also `DairyProduct`,
- `Tableware(?o), eatsWith(?x, ?o), Yogurt(?x) -> Tablespoon(?o)`
- `looksAt(?a, ?o)` → `isVisible(?o)` ∧ `sees(?a, ?o)`

Same rule as above apply for anonymous variables.

[2] `http://www.w3.org/Submission/SWRL/#2.2`

C.2.4 Probabilities

Statements may have a truth probability attached to them, as a float value comprised between 0.0 (impossible fact) and 1.0 (certain fact). If no probability is specified, a probability of 1.0 is assumed.

Two syntaxes are possible: either included in the statement string, separated with a colon:

```
predicate(a1, ...,an):p
```

For example: `Age(EiffelTower, 300):0.54`, which means: the fact that the Eiffel tower is 300 years old holds with a probability of 54%.

Or as a independent float value:

```
[predicate(a1, ...,an), p]
```

For example: `[Age(EiffelTower, 300), 0.54]`

When using the infix syntax for statements, only the second syntax for probabilities is allowed:

`[human1 holds my_cup, 0.87]` (which mean that it is believed with 87% of certainty that the human holds the cup).

`statement` **always** means either a statement alone or a statement with a probability.

C.3 Policies

Knowledge content interacts with the knowledge base through the `revise` method. This method takes as first parameter a set of statements, and as second parameter, a *policy* that specifies what must be done with the statements.

A policy is represented as a set of `(key, value)` pairs whose possible values are presented in table C.1.

Several shortcuts for common operations (like addition of knowledge) are defined and listed below.

C.4 Models

A model is a knowledge container unit (for instance, a RDF tree, a SQL database, etc.). The service provider **may** support several models (for instance, one per agent or one for certain facts, one for uncertain facts, etc.). The actual use and semantic of each model is left to the client.

Each model is identified by a unique alphanumeric id. Most of the API methods can take as parameter a model id. If omitted (`null`) or if the reserved id `all` is used, the method is applied on all models (except otherwise noted, this should be strictly equivalent to call the method on each model separately). Service provider must offer at least one model called `default`, denoting the default robot knowledge storage.

Key	Values	Meaning
`method`	`add` *(default)*	the statements are added to the knowledge base, without ensuring consistency.
	`safe_add`	the statements are added only if they (individually) do not lead to inconsistencies.
	`retract`	the statements are removed from the model. Associated probabilities are discarded.
	`update`	Updates objects of one or several statements in the specified model. If the predicate is not inferred to be *functional* (*i.e.* , it accept only one single value), behaves like `add`.
	`revision` or `safe_update`	Updates objects of one or several statements in the specified model if it does not (individually) lead to inconsistencies. If the predicate is not inferred to be *functional* (*i.e.* , it accepts only one single value), behaves like `safe_add`.
`model`	`all` *(default)*	all existing *models* (section C.4) are impacted by the change.
	a valid model id or a set of valid model id	only the specified model(s) are impacted

Table C.1: Knowledge revision policies.

C.5 Core API

C.5.1 Service management

- `string` **hello**`()`
  - Returns the version number of the service provider. Can be used to check connection status.
  - *Params*: None
  - *Return values*:
    - version number *(string)*
- **load**(*`string` path, [`string` model]*)
  - Loads the content of the specified OWL ontology. If <owl:imports> are specified, the server is expected to honour them.
  - *Params*:

    string the URI of the OWL file. Must be reachable by the server.
    - *opt.* [string] the model ID that receives the OWL content. If omitted, the content is added to all existing models.
  - *Return values*: None
- **save**(*[`string` path], [`string` model]*)
  - Saves the model content to an OWL ontology. Each models are stored in their own namespace (by appending the model id to the default namespace).
  - *Params*:
    - *opt.* [string] the path where to export the OWL file. Must be reachable by the server. If omitted, a default, unique path, is build (the naming scheme is implementation dependent). OWL serialization (XML, turtle, n3...) is implementation dependent.
    - *opt.* [string] the model ID that should be saved. If omitted, all models are dumped.
  - *Return values*: None
- **reset**(*[`string` model]*)
  - Reset a model to its initial state. Note that the initial state may not be an empty set of facts if the service provider loads some initial content at model initialization.
  - *Params*:
    - *opt.* [string] the model ID to reset. If omitted, all models are reset.
  - *Return values*: None
- **special**(*`string` method, [`set`<`string`> parameters]*)

  - This method enables implementation-dependent extensions.
  - *Params*:

    string the name of the method to execute
      * *opt.* [set<string>] method parameters
  - *Return values*: implementation dependent

C.5.2 Managing knowledge

- **revise** (*set<statement>* `statements,` [*policy* `policy`])
  - Alter the knowledge base with one or several statements, following the specified policy.
  - *Params*:

    set<statement> the set of statements to add.
      * *opt.* [policy] the policy to follow. If omitted, the default policy is applied (all statements are added to all models, without checking for consistency).
  - *Return values*: None
- **add** (*set<statement>* `statements,` [*string* `model`])
  - Adds one or several statements to the specified model. Equivalent to `revise` with the policy `"method":"add"`.
  - *Params*:

    set<statement> the set of statements to add.
      * *opt.* [string] the model ID that receives the statements. If omitted, statements are added to all models.
  - *Return values*: None
- **retract** (*set<[statement|partial_statement]>* `pattern,` [*string* `model`])
  - Removes one or several statements to the specified model. Equivalent to `revise` with the policy `"method":"retract"`.
  - *Params*:

    set<statement|partial_statement> the set of statements to remove. If a partial statement is encountered, all statements matching this pattern are removed.
      * *opt.* [string] the model ID where the statements must be removed from. If omitted, statements are removed from each models.
  - *Return values*: None
- **update** (*set<statement>* `statements,` [*string* `model`]))

- Updates objects of one or several statements in the specified model, and only for *functional* predicates (ie, predicates that accept only one value). Equivalent to `revise` with the policy `"method":"update"`.
- *Params*:

  set<statement> the set of statements to update. If a partial statement is encountered, all statements matching this pattern are removed.
  - *opt.* [string] the model ID to update. If omitted, statements are updated in all models.
- *Return values*: None
-

C.5.3 Knowledge retrieval

- **lookup**(*string* concept, [*string* model]))
  - Searches for the given string in the knowledge base, and returns the matching resources, along with their types.
  - *Params*:

    string the string to look for.
    - *opt.* [string] the model ID to look in. If omitted, the string is looked for in all models.
  - *Return values*: A set of pair `[resource_id, [class|instance|object_property|`
- **about**(*string* resource, [*string* model]))
  - Returns the list of asserted (and if available inferred) statements which the resource is part of.
  - *Params*:

    string the resource to look for.
    - *opt.* [string] the model ID to look in. If omitted, the resource is looked for in all models.
  - *Return values*: A set of statements.
- **exist**(*set<[statement|partial_statement]>* pattern, [*string* model]))
  - Checks that the given pattern matches content in the ontology. If statements, returns true if all the statements are present in the knowledge base (asserted or inferred), if partial statements, returns true if at least one statement match the conjunction of the partial statements.
  - *Params*:

    set<[statement|partial_statement >] the pattern to check.
    - *opt.* [string] the model ID to look in. If omitted, the resource is looked for in all models.

  - *Return values*: A boolean.

- **check**(*set<statement>* statements, [*string* model]))
  - Checks that the given statements are consistent with the knowledge base. Statements are not added to the knowledge base.
  - *Params*:

    set<statement> the statements to test.
    * *opt.* [string] the model ID to look in. If omitted, the resource is looked for in all models.
  - *Return values*: A boolean. `true` if statements do not contradict with the knowledge base.

- **find**(*set<string>* named_variables, *set<partial_statement>* pattern, [*set<string>* constraints], [*string* model]))
  - Retrieves ressources given a set of partially defined statements plus optional constraints about this resource. Constraints must follow the SPARQL syntax for filters. `named_variables` defines the set of variables whose bindings are looked for. Probabilities can be retrieved as well.
  - *Params*:

    string the name of the variable to identify, as used in the statements.

    set<partial_statement> pattern build from partial statements.
    * *opt.* [set<partial_statement>] pattern build from partial statements.
    * *opt.* [string] the model ID to look in. If omitted, the resource is looked for in all models.
  - *Return values*: A set of resources.

- **findmpe**(*string* named_variable, *set<partial_statement>* pattern, [*set<string>* constraints], [*string* model]))
  - Retrieves ressources within the *Most Probable Explanation* (*ie* the most likely current state of the world in the given model). For implementation not supporting probabilistic reasoning, `findmpe` is strictly equivalent to `find`. See `find` for details about the use of the request.
  - *Params*:

    string the name of the variable to identify, as used in the statements.

    set<partial_statement> pattern build from partial statements.
    * *opt.* [set<partial_statement>] pattern build from partial statements.
    * *opt.* [string] the model ID to look in. If omitted, the resource is looked for in all models.
  - *Return values*: A set of resources.

C.5.4 Managing models

- **models** `()`
  - Returns the set of current available models. See also models
  - *Params*: None
  - *Return values*: a set of all model ids.
- **addmodel** (*`string`* `id`)
  - Adds a new knowledge model to the knowledge base.
  - *Params*:

    string the model ID to add.
  - *Return values*: None.
- **removemodel** (*`string`* `id`)
  - Removes a knowledge model from the knowledge base.
  - *Params*:

    string the model ID to remove.
  - *Return values*: None.
- **isconsistent** (`[`*`string`* `model]`)
  - Checks that the knowledge base is globally consistent.
  - *Params*:
    - *opt.* [string] the model ID to check. If omitted, all models are checked and `true` is answered only if all models are consistent.
  - *Return values*: A boolean. `true` if the model is consistent.
- **addrules** (*`set<string>`* `rules, [`*`string`* `model])`
  - Add rules to the model. See rules for examples.
  - *Params*:

    set<string> a set of rule to add to the model.
    - *opt.* [string] the model ID that receive the rules. If omitted, rules are added to all models.
  - *Return values*: None.

C.5.5 Taxonomy walking

- **classesof**(*string* instance, [*bool* direct], [*string* model]))
  - Returns the (asserted and if available, inferred) classes of an instance.
  - *Params*:

    string the instance id to look for.
    - *opt.* [bool] if true, only direct types are returned. If omitted, `false` is assumed
    - *opt.* [string] the model ID to look in. If omitted, the classes are searched in all models.
  - *Return values*: A set of resource ids.
- **instancesof**(*string* class, [*bool* direct], [*string* model]))
  - Returns the (asserted and if available, inferred) instances of a class.
  - *Params*:

    string the class id to look for.
    - *opt.* [bool] if true, only direct instances are returned. If omitted, `false` is assumed
    - *opt.* [string] the model ID to look in. If omitted, the instances are searched in all models.
  - *Return values*: A set of resource ids.
- **subclassesof**(*string* class, [*bool* direct], [*string* model]))
  - Returns the (asserted and if available, inferred) sub-classes of a class.
  - *Params*:

    string the class id to look for.
    - *opt.* [bool] if true, only direct sub-classes are returned. If omitted, `false` is assumed
    - *opt.* [string] the model ID to look in. If omitted, the sub-classes are searched in all models.
  - *Return values*: A set of resource ids.
- **superclassesof**(*string* class, [*bool* direct], [*string* model]))
  - Returns the (asserted and if available, inferred) super-classes of a class.
  - *Params*:

    string the class id to look for.
    - *opt.* [bool] if true, only direct super-classes are returned. If omitted, `false` is assumed
    - *opt.* [string] the model ID to look in. If omitted, the super-classes are searched in all models.
  - *Return values*: A set of resource ids.

Appendix D

`oro-server` API

This appendix lists the complete API of `oro-server` as of version 0.8.

Base

- **safeAdd** (*set<string>* `statements`): try to add news statements in long term memory, if they don't lead to inconsistencies (return false if at least one stmt wasn't added).
- **safeAdd** (*set<string>* `statements, string`): try to add news statements with a specific memory profile, if they don't lead to inconsistencies (return false if at least one stmt wasn't added).
- **check** (*set<string>* `statements`): checks that one or several statements are asserted or can be inferred from the ontology
- **checkConsistency** (): checks that the ontology is semantically consistent
- **checkConsistency** (*set<string>* `statements`): checks that a set of statements are consistent with the current model
- **help** (): returns a human-friendly list of available methods with their signatures and short descriptions.
- **getLabel** (`string`): return the label of a concept, if available.
- **lookup** (`string`): try to identify a concept from its id or label, and return it, along with its type (class, instance, object_property, datatype_property).
- **lookup** (`string, string`): try to identify a concept from its id or label and its type (class, instance, object_property, datatype_property).
- **revise** (`set, string`):
- **add** (*set<string>* `statements`): adds one or several statements (triplets S-P-O) to the robot model, in long term memory.
- **add** (*set<string>* `statements, string`): adds one or several statements (triplets S-P-O) to the robot model associated with a memory profile.

- **clear** (*set<string>* `statements`): removes statements in the given set
- **remove** (*set<string>* `statements`): removes one or several statements (triplets S-P-O) from the ontology.
- **update** (*set<string>* `statements`): update the value of a functional property.

Agents

- **checkConsistencyForAgent** (`string`): check the consistency of a specific agent model.
- **safeAddForAgent** (`string, set<string> statements`): try to add news statements to a specific agent model in long term memory, if they don't lead to inconsistencies (return false if at least one stmt wasn't added).
- **safeAddForAgent** (`string, set<string> statements, string`): try to add news statements to a specific agent model with a specific memory profile, if they don't lead to inconsistencies (return false if at least one stmt wasn't added).
- **discriminateForAgent** (`string, set<string> statements`): returns a list of properties that helps to differentiate individuals for a specific agent.
- **findForAgent** (`string, string, set<string> statements`): tries to identify a resource given a set of partially defined statements in an specific agent model.
- **findForAgent** (`string, string, set, set`): tries to identify a resource given a set of partially defined statements and restrictions in an specific agent model.
- **getInfosForAgent** (`string, string`): returns the set of asserted and inferred statements whose the given node is part of. It represents the usages of a resource.
- **listAgents** (`()`): returns the set of agents I'm aware of (ie, for whom I have a cognitive model).
- **lookupForAgent** (`string, string`): lookup a concept in a specific agent model.
- **addForAgent** (`string, set`): adds one or several statements (triplets S-P-O) to a specific agent model, in long term memory.
- **addForAgent** (`string, set, string`): adds one or several statements (triplets S-P-O) to a specific agent model associated with a memory profile.
- **clearForAgent** (`string, set`): removes statements from a specific agent model.
- **removeForAgent** (`string, set`): removes one or several statements. Deprecated. Use clearForAgent instead.

- **save** (`string, string`): exports the cognitive model of a given agent to an owl file. The provided path must be writable by the server.
- **updateForAgent** (`string, set`): updates one or several statements (triplets S-P-O) in a specific agent model, in long term memory.

Administration

- **makeHtmlDoc** (): returns a list of available methods in html format for inclusion in documentation.
- **listMethods** (): returns the list of available methods with their signatures and short descriptions as a map.
- **stats** (): returns some statistics on the server
- **listSimpleMethods** (): returns a raw list of available methods.
- **reset** (): reload the base ontologies, discarding all inserted of removed statements, in every models
- **list** (`string`): lists on the serveur stdout all facts matching a given pattern.
- **save** (): exports the current ontology model to an owl file. The file will be saved to the current directory with an automaticallygenerated name.
- **save** (`string`): exports the current ontology model to an owl file. The provided path must be writable by the server.

Concepts comparison

- **discriminate** (`set`): returns a list of properties that helps to differentiate individuals.
- **getDifferences** (`string, string`): given two concepts, return the list of relevant differences (types, properties...) between these concepts.
- **getSimilarities** (`string, string`): given two concepts, return the list of relevant similarities (types, properties...) between these concepts.

Events

- **registerEventForAgent** (`string, string, string, string, List`): registers an event on a specific agent model. Expected parameters are: agent, type, triggering type, variable, event pattern.
- **registerEventForAgent** (`string, string, string, List`): registers an event on a specific agent model. Expected parameters are: agent, type, triggering type, event pattern.

- **registerEvent** `(string, string, string, List)`: registers an event. Expected parameters are: type, triggering type, variable, event pattern.
- **registerEvent** `(string, string, List)`: registers an event. Expected parameters are: type, triggering type, event pattern.
- **clearEvent** `(string, string)`: remove one specific event from a specific model.
- **clearEventsForAgent** `(string)`: remove all events associated to a specific model.
- **clearEvent** `(string)`: remove one specific event from the main model.
- **clearEvents** `()`: remove all events associated to the main model.

Querying

- **find** `(string, set)`: tries to identify a resource given a set of partially defined statements about this resource.
- **find** `(string, set, set)`: tries to identify a resource given a set of partially defined statements plus restrictions about this resource.
- **getInfos** `(string)`: returns the set of asserted and inferred statements whose the given node is part of. It represents the usages of a resource.
- **query** `(string, string)`: performs one sparql query on the ontology
- **getResourceDetails** `(string)`: returns a serialized ResourceDescription object that describe all the links of this resource with others resources (sub and superclasses, instances, properties, etc.).
- **getResourceDetails** `(string, string)`: returns a serialized ResourceDescription object that describe all the links of this resource with others resources (sub and superclasses, instances, properties, etc.). The second parameter specify the desired language (following rfc4646).

Taxonomy

- **getClassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of asserted and inferred classes of a given individual.
- **getDirectClassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of asserted and inferred direct classes of a given individual.
- **getDirectInstancesOf** `(string)`: returns a map of instance name, label (or instance name, instance name without namespace is no label is available) of asserted and inferred direct instances of a given class.

- **getDirectSubclassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of all asserted and inferred direct subclasses of a given class.
- **getDirectSuperclassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of all asserted and inferred direct superclasses of a given class.
- **getInstancesOf** `(string)`: returns a map of instance name, label (or instance name, instance name without namespace is no label is available) of asserted and inferred instances of a given class.
- **getSubclassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of all asserted and inferred subclasses of a given class.
- **getSuperclassesOf** `(string)`: returns a map of class name, label (or class name, class name without namespace is no label is available) of all asserted and inferred superclasses of a given class.

Appendix E

Main Software Contributions

Preparing a PhD thesis in a robotic laboratory often involved writting quite a lot of code.

During my four years spanned at LAAS-CNRS and TUM's IAS laboratory, I have participated to many projects:

	Project	(Main) language	Total LOC	Contribution (% of total diffs)
ORO	`oro-server`	Java	17.6k	100%
	`oro-view`	Python	1.8k	100%
	`pyoro`	Python	1.8k	100%
	`liboro`	C++	5.3k	100%
	`oro-ros`	Python		100%
	`oro-yarp`	C++	1.0k	100%
NLP	`dialogs`	Python	18.3k	38%
	`android-robot-cmd`	Java	0.8k	100%
	`xmpp-stdio`	Python	0.2k	100%
	`gspeett`	Python	0.3k	100%
CONTROL	`pyrobots`	Python	2.6k	90%
MORSE	`morse`	Python	28.1k	20%
GENOM	`pypoco`	Python	1.5k	100%

The count of lines of code (LOC) includes comments but not the blank lines.

I have also an acknowledged contribution to the ROS meta operating system related to the support of Python 3.

Appendix F

Publications and Dissemination Activities

F.1 Publications as Main Author

F.1.1 International Peer-Reviewed Journals

- Lemaignan, S. and Ros, R. and Sisbot, E. A. Alami, R. and Beetz M., **Grounding the Interaction: Anchoring Situated Discourse in Everyday Human-Robot Interaction**, *International Journal of Social Robotics*, 2011.

F.1.2 Book Chapters

- **(in press)** Lemaignan, S. and Alami, R. and Pandey, A. K. and Warnier, M. and Guitton, J. , **Towards Grounding Human-Robot Interaction** in *Bridges between the Methodological and Practical Work of the Robotics and Cognitive Systems Communities - From Sensors to Concepts* Editors: Amirat, T. and Chibani, A. and Zarri, G. P.. Springer Publishing. 2012.

F.1.3 Conference Articles

- Lemaignan, S. and Ros, R. and Alami, R. and Beetz, M., **What are you talking about? Grounding dialogue in a perspective-aware robotic architecture**, *RO-MAN*. 2011.

- Lemaignan, S. and Ros, R. and Mösenlechner, L. and Alami, R. and Beetz, M., **ORO, a knowledge management module for cognitive architectures in robotics**, *IROS*. 2010.

- Ros, R. and Lemaignan, S. and Sisbot, E. A. and Alami, R. and Steinwender, J. and Hamann, K. and Warneken, F., **Which One? Grounding the Referent Based on Efficient Human-Robot Interaction**, *ROMAN*. 2010. **Best paper award**.

F.2 Publications as Co-Author

F.2.1 International Peer-Reviewed Journals

- **(in press)** Lallée, S. and Pattacini, U. and Lemaignan, S. and Lenz, A. and Melhuish, C. and Natale, L. and Skachek, S. and Hamann, K. and Steinwender, J. and Sisbot, E.A. and Metta, G. and Pipe, T. and Alami, W. and Warnier, M. and Guitton, J. and Warneken, F. and Dominey, P.F., **Towards a Platform-Independent Cooperative Human-Robot Interaction System: III. An Architecture for Learning and Executing Actions and Shared Plans**, *IEEE Transactions on Autonomous Mental Development*, 2012.

F.2.2 Conference Articles

- **(submitted)** Warnier, M. and Guitton, J. and Lemaignan, S. and Alami, R., **When the robot puts itself in your shoes. Explicit geometric management of position beliefs.**, *ROMAN*, 2012.

- **(submitted)** Warnier, M. and Guitton, J. and Lemaignan, S. and Alami, R., **Let's clean the table together**, *IROS*, 2012.

- Alami, R. and Warnier, M. and Guitton, J. and Lemaignan, S. and Sisbot, E. A., **When the robot considers the human...**, *ISRR*, 2011.

- Lallée, S. and Pattacini, U. and Boucher J.D. and Lemaignan, S. and Lenz, A. and Melhuish, C. and Natale, L. and Skachek, S. and Hamann, K. and Steinwender, J. and Sisbot, E.A. and Metta, G. and Alami, W. and Warnier, M. and Guitton, J. and Warneken, F. and Dominey, P.F., **Towards a Platform-Independent Cooperative Human-Robot Interaction System: II. Perception, Execution and Imitation of Goal Directed Actions**, *IROS*, 2011.

- Lallée, S. and Lemaignan, S. and Lenz, A. and Melhuish, C. and Natale, L. and Skachek, S. and van Der Zant, T. and Warneken, F. and Dominey, P.F., **Towards a Platform-Independent Cooperative Human-Robot Interaction System: I. Perception**, *IROS*, 2010.

F.3 Peer-Reviewed Workshops and Other Dissemination Activities

F.3.1 2012

- Echeverria, G. and Karg, M. and Lemaignan, S. and Degroote, A., and Lacroix, S., **MORSE Tutorial**, *EURON*.

- Lemaignan, S. and Gharbi, M. and Mainprice, J. and Herrb, M. and Alami, R., **Roboscopie: A Theatre Performance for a Human and a Robot**, *HRI, peer-reviewed*.

- Lemaignan, S. and Echeverria G. and Karg, M. and Mainprice, M. and Kirsch, A. and Alami, R., **Human-Robot Interaction in the MORSE Simulator**, *HRI, peer-reviewed*.

F.3.2 2011

- Lemaignan, S. and Ros, R. and Alami, R., **Dialogue in situated environments: A symbolic approach to perspective-aware grounding, clarification and reasoning for robots**, *Grounding Human-Robot Dialog for Spatial Tasks - RSS workshop, peer-reviewed*.
- Lemaignan, S. and Sisbot, E. A. and Alami, R., **Anchoring interaction through symbolic knowledge**, *HRI Pioneers, peer-reviewed*.

F.3.3 2010

- Lemaignan, S., **Ontologies et robotique : Quand le robot donne du sens à ce qu'il perçoit**, *DocToMe Seminar*.
- Lemaignan, S., **Knowledge centric architecture for Human-Robot interaction**, *TUM/LAAS Joint Winter Workshop*.
- Lemaignan, S., **Ontologies for HRI**, *Cotesys ROS Fall School*.
- Lemaignan, S., **What can we do with ontologies for HRI?**, *Dagstuhl Seminar on Human-Robot Interaction*.
- Ros, R. and Sisbot, E. A. and Lemaignan, S. and Pandey, A. K. and Alami, R., **Robot, tell me what you know about...?: Expressing robot's knowledge through interaction**, *ICAIR, peer-reviewed*.
- Lemaignan, S., **OpenRobots Ontology: an overview**, *EURON*.

Bibliography

[1] B. Aarts. *English syntax and argumentation*. MacMillan, 1997.

[2] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *The International Journal of Robotics Research*, 17(4):315, 1998.

[3] R. Alami, M. Warnier, J. Guitton, S. Lemaignan, and E. A. Sisbot. When the robot considers the human... In *Proceedings of the 15th International Symposium on Robotics Research*, 2011. To appear.

[4] S. Alili, M. Warnier, M. Ali, and R. Alami. Planning and plan-execution for human-robot cooperative task achievement. In *19th International Conference on Automated Planning and Scheduling*, 2009.

[5] J. Allen. Towards a general theory of action and time. *Artificial intelligence*, 23(2):123–154, 1984.

[6] J. Anderson. *Language, Memory, and Thought*. Lawrence Erlbaum, 1976.

[7] R. Atkinson and R. Shiffrin. Human memory: A proposed system and its control processes. *The psychology of learning and motivation: Advances in research and theory*, 2:89–195, 1968.

[8] J. Austin, J. Urmson, and M. Sbisà. *How to do things with words*. Harvard University Press, 1962.

[9] F. Baader, I. Horrocks, and U. Sattler. *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, chapter Description Logics, pages 135 – 179. Elsevier, 2008.

[10] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[11] A. Baddeley. Working memory. *Current Biology*, 20(4):R136–R140, 2010.

[12] P. Baerlocher and R. Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer: International Journal of Computer Graphics*, 20(6):402–417, 2004.

[13] S. Bail, B. Parsia, and U. Sattler. JustBench: A framework for OWL benchmarking. In P. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Pan, I. Horrocks, and B. Glimm, editors, *The Semantic Web – ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 32–47. Springer Berlin / Heidelberg, 2010.

[14] S. Baron-Cohen, A. Leslie, and U. Frith. Does the autistic child have a "theory of mind" ? *Cognition*, 1985.

[15] M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM — A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[16] M. Beetz, F. Stulp, B. Radig, J. Bandouch, N. Blodow, M. Dolha, A. Fedrizzi, D. Jain, U. Klank, I. Kresse, et al. The assistive kitchen—a demonstration scenario for cognitive technical systems. In *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[17] M. Beetz, M. Tenorth, D. Jain, and J. Bandouch. Towards automated models of activities of daily life. *Technology and Disability*, 22(1-2):27–40, 2010.

[18] S. N. Blisard. Modeling spatial referencing language for human-robot interaction. In *in Proc. IEEE Intl. Workshop on Robot and Human Interactive Communication*, pages 698–703, 2005.

[19] N. Blodow, L. C. Goron, Z.-C. Marton, D. Pangercic, T. Rühr, M. Tenorth, and M. Beetz. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September, 25–30 2011.

[20] C. Breazeal. Towards sociable robots. *Robotics and Autonomous Systems*, pages 167–175, 2003.

[21] C. Breazeal, M. Berlin, A. Brooks, J. Gray, and A. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and Autonomous Systems*, pages 385–393, 2006.

[22] T. Brick and M. Scheutz. Incremental natural language processing for HRI. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, 2007.

[23] B. Carpenter. The logic of typed feature structures. *Cambridge Tracts in Teoretical Computer Science*, 32, 1992.

[24] X. Chen, J. Ji, J. Jiang, G. Jin, F. Wang, and J. Xie. Developing high-level cognitive functions for service robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '10, pages 989–996, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[25] S. Chernova, N. DePalma, E. Morant, and C. Breazeal. Crowdsourcing human-robot interaction: Application from virtual to physical worlds. In *20th IEEE International Symposium in Robot and Human Interactive Communication*, 2011.

[26] H.-Q. Chong, A.-H. Tan, and G.-W. Ng. Integrated cognitive architectures: a survey. *Artificial Intelligence Review*, 2009.

[27] S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.

[28] M. Cox and A. Raja. Metareasoning: A manifesto. *BBN Technical*, 2007.

[29] M. Daoutis, S. Coradeshi, and A. Loutfi. Grounding commonsense knowledge in intelligent systems. *Journal of Ambient Intelligence and Smart Environments*, pages 311–321, 2009.

[30] N. Derbinsky, J. Laird, and B. Smith. Towards efficiently supporting large symbolic declarative memories. *Ann Arbor*, 1001:48109–2121, 2010.

[31] F. DiSimoni. *The token test for children*. Pro. ed, 1978.

[32] P. Dominey and F. Warneken. The basis of shared intentions in human and robot cognition. *New Ideas in Psychology*, 29(3):260–274, 2011.

[33] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular openrobots simulation engine: MORSE. In *Proceedings of the IEEE ICRA*, 2011.

[34] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[35] A. Ferrein and G. Lakemeyer. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*, 56(11):980–991, 2008.

[36] J. Flavell, H. Beilin, and P. Pufall. Perspectives on perspective taking. *Piaget's theory: Prospects and possibilities*, pages 107–139, 1992.

[37] C. Galindo, J. Fernández-Madrigal, J. González, and A. Saffiotti. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008.

[38] M. Gelfond. *Foundations of Artificial Intelligence*, volume 3, chapter Answer sets, pages 285–316. Elsevier, 2008.

[39] M. Ghallab. On chronicles: Representation, on-line recognition and learning. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 597–607. Morgan Kaufmann Publishers, 1996.

[40] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the 12th international conference on World Wide Web*, pages 48–57. ACM, 2003.

[41] J. Gruber. *Studies in lexical relations*. PhD thesis, Massachusetts Institute of Technology, 1965.

[42] S. Gspandl, I. Pill, M. Reip, G. Steinbauer, and A. Ferrein. Belief management for high-level robot programs. In *International Joint Conference on Artificial Intelligence*, Barcelona, Spain, to appear 2011. AAAI Press.

[43] J. P. M. Gutiérrez. *Directed Motion in English and Spanish*, volume 11, chapter Semantic Role Lists. Universidad de Sevilla, 2001.

[44] S. Harnad. The symbol grounding problem. *Phys. D*, 42(1-3):335–346, 1990.

[45] N. Hawes, M. Hanheide, J. Hargreaves, B. Page, H. Zender, and P. Jensfelt. Home alone: Autonomous extension and correction of spatial representations. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

[46] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G. Kruijff, M. Brenner, G. Berginc, and D. Skocaj. Towards an integrated robot with multiple cognitive functions. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 1548. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[47] N. Hawes and J. Wyatt. Benchmarking the influence of information-processing architectures on intelligent systems. In *Proceedings of the Robotics: Science & Systems 2008 Workshop: Experimental Methodology and Benchmarking in Robotics Research*, June 2008.

[48] N. Hawes, M. Zillich, and J. Wyatt. BALT & CAST: Middleware for cognitive robotics. In *In Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication (ROMAN 2007)*, pages 998–1003, 2007.

[49] F. Heintz and P. Doherty. Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent & Fuzzy Systems*, 15(1):3–13, 2004.

[50] F. Heintz, J. Kvarnström, and P. Doherty. Knowledge processing middleware. In S. Carpin, I. Noda, E. Pagello, M. Reggiani, and O. von Stryk, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 5325 of *Lecture Notes in Computer Science*, pages 147–158. Springer Berlin / Heidelberg, 2008.

[51] H. Holzapfel, D. Neubig, and A. Waibel. A dialogue approach to learning object descriptions and semantic categories. *Robotics and Autonomous Systems*, 56(11):1004 – 1013, 2008.

[52] S. Huwel, B. Wrede, and G. Sagerer. Robust speech understanding for multi-modal human-robot communication. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication*, 2006.

[53] F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, 1992.

[54] H. Jacobsson, N. Hawes, G.-J. Kruijff, and J. Wyatt. Crossmodal content binding in information-processing architectures. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*, pages 81–88, New York, NY, USA, 2008. ACM.

[55] D. Jain, L. Mösenlechner, and M. Beetz. Equipping robot control programs with first-order probabilistic reasoning capabilities. In *International Conference on Robotics and Automation*, 2009.

[56] R. Jakobson. *Style in language*, chapter Closing statements: Linguistics and Poetics. T.A. Sebeok, New-York, 1960.

[57] J. Ji and X. Chen. Induction in nonmonotonic causal theories for a domestic service robot. In *Proceedings of the 21st International Conference on Inductive Logic Programming*, 2011.

[58] M. Johnson and Y. Demiris. Perceptual perspective taking and action recognition. *Advanced Robotic Systems*, 2(4):301–308, 2005.

[59] J. Kelleher and G. Kruijff. Incremental generation of spatial referring expressions in situated dialog. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 1041–1048. Association for Computational Linguistics, 2006.

[60] W. G. Kennedy, M. D. Bugajska, A. M. Harrison, and J. G. Trafton. "Like-Me" simulation as an effective and cognitively plausible basis for social robotics. *International Journal of Social Robotics*, 1 (2):181–194, 2009.

[61] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer. A large-scale classification of english verbs. *Language Resources and Evaluation*, 42(1):21–40, 2008.

[62] U. Klank, D. Pangercic, R. B. Rusu, and M. Beetz. Real-time CAD model matching for mobile manipulation and grasping. In *9th IEEE-RAS International Conference on Humanoid Robots*, Paris, France, December 7-10 2009.

[63] P. Klinov. Pronto: A non-monotonic probabilistic description logic reasoner. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, pages 822–826. Springer-Verlag, 2008.

[64] G. Kruijff, M. Janıcek, and P. Lison. Continual processing of situated dialogue in human-robot collaborative activities. In *19th IEEE International Symposium in Robot and Human Interactive Communication*, 2010.

[65] G. Kruijff, P. Lison, T. Benjamin, H. Jacobsson, H. Zender, I. Kruijff-Korbayová, and N. Hawes. Situated dialogue processing for human-robot interaction. *Cognitive Systems*, pages 311–364, 2010.

[66] B. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(1-2):191–233, 2000.

[67] L. Kunze, M. E. Dolha, and M. Beetz. Logic programming with simulation-based temporal projection for everyday robot object manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September, 25–30 2011.

[68] L. Kunze, T. Roehm, and M. Beetz. Towards semantic robot description languages. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May, 9–13 2011.

[69] S. Lallée, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, T. van Der Zant, F. Warneken, and P. F. Dominey. Towards a platform-independent cooperative human-robot interaction system: I. perception. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. To appear.

[70] S. Lallée, C. Madden, M. Hoen, and P. Dominey. Linking language with embodied and teleological representations of action for humanoid cognition. *Frontiers in Neurorobotics*, 2010. submitted.

[71] S. Lallée, U. Pattacini, B. J.D., S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, K. Hamann, J. Steinwender, E. Sisbot, G. Metta, W. Alami, M. Warnier, J. Guitton, F. Warneken, and P. Dominey. Towards a platform-independent cooperative human-robot interaction system: Ii. perception, execution and imitation of goal directed actions. In *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[72] P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, June 2006.

[73] J. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to SOAR, an architecture for human cognition: 2006 update. Technical report, University of Michigan, 2006.

[74] S. Lemaignan, R. Alami, A. K. Pandey, M. Warnier, and J. Guitton. *Bridges between the Methodological and Practical Work of the Robotics and Cognitive Systems Communities - From Sensors to Concepts*, chapter Towards Grounding Human-Robot Interaction. Intelligent Systems Reference Library. Springer Publishing, 2012. To be published.

[75] S. Lemaignan, R. Ros, L. Mösenlechner, R. Alami, and M. Beetz. ORO, a knowledge management platform for cognitive architectures in robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[76] S. Lemaignan, R. Ros, E. A. Sisbot, R. Alami, and M. Beetz. Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. *International Journal of Social Robotics*, pages 1–19, 2011.

[77] A. Leslie. Theory of mind as a mechanism of selective attention. *The new cognitive neurosciences*, pages 1235–1247, 2000.

[78] H. Levesque and G. Lakemeyer. Chapter 23 cognitive robotics. In V. L. Frank van Harmelen and B. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 869 – 886. Elsevier, 2008.

[79] H. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2:159–178, 1998.

[80] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1-3):59–83, 1997.

[81] Z. Li. *Multimodal Human-Robot Interaction in an Assistive Technology Environment*. PhD thesis, Monash University, 2012.

[82] G. Lim, I. Suh, and H. Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man and Cybernetics*, pages 1–18, 2011.

[83] C. Lin, C. Tseng, W. Teng, W. Lee, C. Kuo, H. Gu, K. Chung, and C. Fahn. The realization of robot theater: Humanoid robots and theatric performance. In *ICAR 2009*, pages 1–6. IEEE, 2009.

[84] C. Lörken and J. Hertzberg. Grounding planning operators by affordances. In *International Conference on Cognitive Systems (CogSys)*, pages 79–84, 2008.

[85] A. Loutfi, S. Coradeschi, M. Daoutis, and J. Melchert. Using knowledge representation for perceptual anchoring in a robotic system. *International Journal On Artificial Intelligence Tools: Architectures, Languages, Algorithms*, 17(5):925, 2008.

[86] J. Mainprice, E. A. Sisbot, L. Jaillet, J. Cortes, R. Alami, and T. Simeon. Planning human-aware motions using a sampling-based costmap planner. In *IEEE International Conference on Robotics and Automation*, 2011.

[87] L. Marin, E. A. Sisbot, and R. Alami. Geometric tools for perspective taking for human-robot interaction. In *Mexican International Conference on Artificial Intelligence (MICAI 2008)*, Mexico City, Mexico, October 2008.

[88] V. Mascardi, V. Cordì, and P. Rosso. A comparison of upper ontologies. Technical report, DISI, 2007.

[89] C. Matuszek, D. Fox, and K. Koscher. Following directions using statistical machine translation. In *Proceedings of the International Conference on Human-Robot Interaction*. ACM Press, 2010.

[90] N. Mavridis and D. Hanson. The IbnSina center: An augmented reality theater with intelligent robotic and virtual characters. In *RO-MAN 2009*, pages 681–686. IEEE, 2009.

[91] N. Mavridis and D. Roy. Grounded situation models for robots: Where words and percepts meet. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[92] B. McBride. Jena: A semantic web toolkit. *Internet Computing, IEEE*, 6(6):55–59, 2002.

[93] J. McCarthy. From here to human-level AI. *Artificial Intelligence*, 2007.

[94] G. Metta, P. Fitzpatrick, and L. Natale. Yarp: Yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48, 2006.

[95] H. Moll and M. Tomasello. Level 1 perspective-taking at 24 months of age. *British Journal of Developmental Psychology*, 24(3):603–614, 2006.

[96] Y. Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley Longman Publishing, 1990.

[97] D. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, pages 379–404, 2003.

[98] A. Newell. The knowledge level: presidential address. *AI magazine*, 2(2):1, 1981.

[99] D. Nyga, M. Tenorth, and M. Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *ICRA*, 2009.

[100] A. Nüchter and J. Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915 – 926, 2008.

[101] J. O'Keefe. *The Spatial Prepositions*. MIT Press, 1999.

[102] P. Oudeyer, F. Kaplan, and V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[103] P.-Y. Oudeyer and F. Kaplan. How can we define intrinsic motivation ? In *Proceedings of the 8th International Conference on Epigenetic Robotics*, 2008.

[104] A. Pandey and R. Alami. Mightability maps: A perceptual level decisional framework for co-operative and competitive human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[105] J.-Y. Pollock. *Langage et Cognition*. Presses Universitaires de France, 1998.

[106] D. Premack and G. Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and Brain sciences*, 1(4):515–526, 1978.

[107] Z. Pylyshyn. *Foundations of cognitive science*, chapter Computing in cognitive science, pages 51–91. Cambridge: A Braddford Book, The MIT Press, 1989.

[108] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[109] T. Regier and L. Carlson. Grounding spatial language in perception: An empirical and computational investgation. *Journal of Experimental Psychology*, 2001.

[110] R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken. Which one? grounding the referent based on efficient human-robot interaction. In *19th IEEE International Symposium in Robot and Human Interactive Communication*, 2010.

[111] R. Ros, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken. Solving ambiguities with perspective taking. In *5th ACM/IEEE International Conference on Human-Robot Interaction*, 2010.

[112] D. Roy and E. Reiter. Connecting language to the world. *Artificial Intelligence*, 2005.

[113] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter Chapter 11: Planning and Acting in the Real World, pages 401–437. Pearson, 2009.

[114] L. Sabri, A. Chibani, Y. Amirat, and G. Zarri. Narrative reasoning for cognitive ubiquitous robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop: Knowledge Representation for Autonomous Robots*, 2011.

[115] A. Saffiotti and M. Broxvall. PEIS ecologies: Ambient intelligence meets autonomous robotics. In *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence*, pages 277–281. ACM, 2005.

[116] B. Scassellati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12(1):13–24, 2002.

[117] J. R. Searle. A classification of illocutionary acts. *Language in society*, 5(01):1–23, 1976.

[118] S. Shapiro and J. Bona. The GLAIR cognitive architecture. In *AAAI Fall Symposium Series*, 2009.

[119] P. Singh, T. Lin, E. Mueller, G. Lim, T. Perkins, and W. Li Zhu. Open mind common sense: Knowledge acquisition from the general public. *Lecture Notes in Computer Science*, pages 1223–1237, 2002.

[120] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[121] E. Sisbot, R. Ros, and R. Alami. Situation assessment for human-robot interaction. In *20th IEEE International Symposium in Robot and Human Interactive Communication*, 2011.

[122] E. A. Sisbot, A. Clodic, R. Alami, and M. Ransan. Supervision and motion planning for a mobile manipulator interacting with humans. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, 2008.

[123] A. Sloman. Why symbol-grounding is both impossible and unnecessary, and why theory-tethering is more powerful anyway. Technical report, Research Note No. COSY-PR-0705, 2007.

[124] A. Sloman. Some requirements for human-like robots: Why the recent over-emphasis on embodiment has held up progress. *Creating Brain-like Intelligence*, pages 248–277, 2009.

[125] A. Sloman. Varieties of meta-cognition in natural and artificial systems. In *Workshop on Metareasoning, AAAI*, volume 8, pages 12–20, 2011.

[126] B. Society. Comparative table of implemented cognitive architectures, 2011.

[127] L. Steels. The symbol grounding problem has been solved. so what's next? *Symbols, Embodiment and Meaning. Oxford University Press, Oxford, UK*, 2007.

[128] I. Suh, G. Lim, W. Hwang, H. Suh, J. Choi, and Y. Park. Ontology-based multi-layered robot knowledge framework (OMRKF) for robot intelligence. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[129] M. Tenorth. *Knowledge Processing for Autonomous Robots*. PhD thesis, Technische Universität München, 2011.

[130] M. Tenorth and M. Beetz. KNOWROB - knowledge processing for autonomous personal robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[131] M. Thielscher. A unifying action calculus. *Artificial Intelligence*, 175(1):120–141, 2011.

[132] M. Tomasello, M. Carpenter, J. Call, T. Behne, H. Moll, et al. Understanding and sharing intentions: The origins of cultural cognition. *Behavioral and Brain Sciences*, 28(5):675–690, 2005.

[133] J. Trafton, N. Cassimatis, M. Bugajska, D. Brock, F. Mintz, and A. Schultz. Enabling effective human-robot interaction using perspective-taking in robots. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(4):460–470, 2005.

[134] E. Tulving. How many memory systems are there? *American Psychologist*, 40(4):385, 1985.

[135] E. Tulving. Memory and consciousness. *Canadian Psychology/Psychologie Canadienne*, 26(1):1, 1985.

[136] B. Tversky, P. Lee, and S. Mainwaring. Why do speakers mix perspectives? *Spatial Cognition and Computation*, 1(4):399–412, 1999.

[137] K. Varadarajan and M. Vincze. Ontological knowledge management framework for grasping and manipulation. In *IROS Workshop: Knowledge Representation for Autonomous Robots*, 2011.

[138] F. Varela, E. Thompson, and E. Rosch. *The embodied mind: Cognitive science and human experience*. The MIT Press, 1992.

[139] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *Evolutionary Computation, IEEE Transactions on*, 11(2):151–180, 2007.

[140] K. Von Fintel. What is presupposition accomodation, again? *Philosophical Perspectives*, 22(1):137–170, 2008.

[141] D. Vrandečić. *Handbook on Ontologies*, chapter Ontology evaluation, pages 293–313. Springer, 2009.

[142] A. Vrecko, D. Skocaj, N. Hawes, and A. Leonardis. A computer vision integration model for a multi-modal cognitive system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3140–3147. IEEE, 2009.

[143] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, et al. Roboearth. *Robotics & Automation Magazine*, 18(2):69–82, 2011.

[144] M. Warnier, J. Guitton, S. Lemaignan, and R. Alami. Let's clean the table together. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[145] M. Warnier, J. Guitton, S. Lemaignan, and R. Alami. When the robot puts itself in your shoes. Explicit geometric management of position beliefs. In *Proceedings of the 21st IEEE International Symposium on Robot and Human Interactive Communication*, 2012. Submitted.

[146] M. Woolridge. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.*, chapter Intelligent Agents, pages 27–78. Massachusetts Institute of Technology, 1999.

[147] G. Zarri. NKRL, a knowledge representation tool for encoding the 'meaning'of complex narrative texts. *Natural Language Engineering*, 3(2):231–253, 1997.

[148] G. Zarri. *Representation and Management of Narrative Information: Theoretical Principles and Implementation*. Springer-Verlag New York Inc, 2008.

[149] E. Zolin. Description Logic Complexity Navigator.