D5.1 MARIO Ontology Network

Project Acronym:	**MARIO**
Project Title:	**Managing active and healthy aging with use of caring service robots**
Project Number:	**643808**
Call:	**H2020-PHC-2014-single-stage**
Topic:	**PHC-19-2014**
Type of Action:	**RIA**

D5.1

Work Package:	WP5	
Due Date:		M18
Submission Date:		31/07/2016
Start Date of Project:		01/02/2015
Duration of Project:		36 months
Organisation Responsible of Deliverable:		CNR
Version:		1.0
Status:		final
Author name(s):	Luigi Asprino, Aldo Gangemi, Stefano Nolfi, Andrea Giovanni Nuzzolese, Nico Pisanelli, Valentina Presutti, Alessandro Russo	
Reviewer(s):	reviewers	
Nature:	[✓ R – Report] ☐ P – Prototype ☐ D – Demonstrator ☐ O – Other	
Dissemination level:	[✓ P – Public] ☐ CO – Confidential, only for members of the consortium (including the Commission) ☐ RE – Restricted to a group specified by the consortium (including the Commission Services)	
Project co-funded by the European Commission within the Horizon 2020 Programme (2014-2020)		

Revision history

Version	Date	Modified by	Comments
0.1	20/05/2016	Luigi Asprino and Alessandro Russo	First draft, document structure
0.2	30/05/2016	Valentina Presutti	Revise of document structure
0.3	07/06/2016	Luigi Asprino	Added draft about background
0.4	12/06/2016	Luigi Asprino	Added draft about MARIO Ontology Network (MON)
0.5	20/06/2016	Andrea Giovanni Nuzzolese	Added draft about the Knowledge Management System (KMS)
0.6	24/06/2016	Valentina Presutti	Draft review
0.7	05/07/2016	Luigi Asprino and Andrea Giovani Nuzzolese	Document revision according to Valentina Presutti's review
0.8	13/07/2016	Luigi Asprino and Andrea Giovani Nuzzolese	Revised the description of ontologies and the architecture of the KMS
0.9	13/07/2016	Luigi Asprino and Andrea Giovani Nuzzolese	Document revision according to Mario internal quality control procedure (reviewers: PASSAU, NUIG and IRCSS)
1.0	28/07/2016	Valentina Presutti	added Intro and Conclusion

Executive Summary

This document presents the Mario Ontology Network and its related management software framework.

MARIO is an assistive robot that has to support a set of knowledge-intensive tasks aimed at (i) helping patients affected by dementia to feel more autonomous and less lonely, (ii) supporting carers in their activity to assess the patient's cognitive condition. Examples of knowledge-intensive tasks are the Comprehensive Geriatric Assessment (CGA) and the triggering of appropriate entertainment activities. MARIO has also to address a number of behavioural tasks such as to drive the patient to a specific location (e.g. the bathroom) and identifying searched objects (e.g. keys).

In order to enable this tasks MARIO features a set of *abilities* implemented by pluggable software components. MARIO abilities, when executed, contribute to and benefit from a common knowledge base. For example, MARIO ability to accompany a patient to the bathroom retrieves the information needed about the physical environment from MARIO knowledge base and stores the information that such an event happened at a certain date/time in the same knowledge base. Another example is the ability to entertain the patient by playing music, which retrieves music tracks from the knowledge base and stores liking feedback about them in order to reuse such knowledge in future executions. As for the CGA the associated ability retrieves questions to be posed to the patient from the knowledge base and stores the obtained answers and all associated relevant metadata.

This examples shows the need to provide MARIO abilities with a common knowledge base able to cover all relevant *knowledge areas* as well as mechanisms for organising, accessing, storing and interacting with such knowledge base (i.e. MARIO background knowledge). This requirement is fulfilled by the MARIO Knowledge Management System, which consists of:

- A set of interconnected and modularised ontologies, i.e. the MARIO Ontology Network (MON), which is meant to model all *knowledge areas* that are relevant for MARIO abilities.
- A set of software interfaces that provide abilities with high level access to MON and its associated knowledge base. Such interfaces are automatically generated by a software framework, named *Lizard*, that allows to smoothly update them when some ontology change occurs.
- A reasoning component that supports the production of inferred knowledge.

Contents

List of Figures

List of Tables

Introduction

This deliverable presents the *Mario Ontology Network* (MON) and its related management software. The main objective of the work here described is to provide MARIO robots with the needed infrastructure to organise, query and interpret their background knowledge. MARIO background knowledge consists of: lexical knowledge (e.g. natural language lexica and linguistic frames), domain knowledge (e.g. cga, personal sphere), environmental knowledge (e.g. physical locations and maps), sensor knowledge (e.g. RFID, life mesures), and metadata knowledge (e.g. entity tagging).

The key outcome of this work is a set of networked ontology modules and a set of software components. The networked ontology modules are identified by the name MON, while a software framework named “Lizard” provides a middleware between MARIO abilities, i.e. software applications that implement MARIO behaviour and capabilities, and MARIO background knowledge allowing its manipulation.

1.1 Work Package 5 Objectives

The key objectives of Work Package 5 can be summarised as follows:

- providing MARIO robots with the means for creating, organising, querying and reasoning over a background knowledge base (e.g. to store/retrieve user’s personal information) - Task 5.1
- providing MARIO with the ability to process natural language input, to recognise relevant location-aware requests and to react appropriately (e.g. to understand if the user wants to know where she is now located and to provide a correct answer) - Tasks 5.2-5.4
- providing MARIO with the ability to evaluate and possibly reuse the sentiment polarity of natural language expressions (e.g. to assign a sentiment score to a user’s feedback about a certain activity and reuse it to decide whether to re-propose or discard that activity later) - Task 5.3

This deliverable addresses Task 5.1.

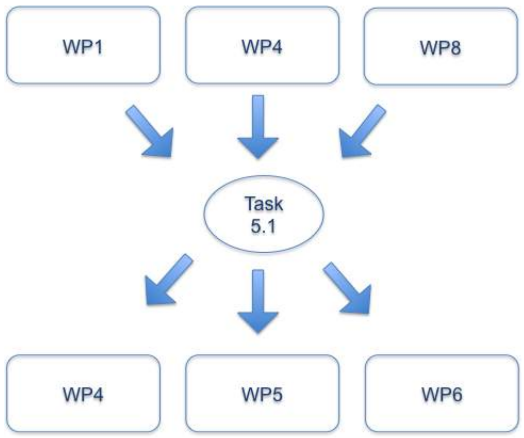


Figure 1.1: Input-output relations between Task 5.1 - the outcome of which is the subject of this report - and other work packages in the project.

1.2 Purpose and Target Group of the Deliverable

This report provides details about the ontologies that are currently available in MON and its supporting software. Its target is the group of developers of MARIO's abilities, who can leverage the outcome of this work by reusing MARIO knowledge base and the available functionalities that support its manipulation. In other words, developers of MARIO's abilities can rely on MON for querying and storing knowledge and on Lizard for accessing and reusing it.

1.3 Relations to other Activities in the Project

The work described in this deliverable has been mainly affected by Deliverable 1.2 "Mario Functionalities and Requirements", and Deliverable 1.3 "Data Management and System Architecture". These deliverables constituted the key input for this work together with the ongoing work of Task 4.1 "Customization of the comprehensive geriatric assessment (CGA) approach to the service robot context", Task 8.1 "Pilot Planning, Management and Coordination" and the other tasks in Work Package 5 and Work Package 6. The output of this deliverable and its further extensions[1] is input to the other tasks in Work Package 4, 5 and 6. This input-output relations are summarised in Figure 1.1.

[1]Notice that the work on MON is still continuing and will accompany the whole duration of Work Package 6 and 8 in order to ensure that MARIO knowledge base will address and adapt to all emerging requirements.

1.4 Document Outline

The document is organised as follows: Chapter 2 provides some background on Semantic Web technologies and ontology design methodologies that have been used for producing the output of this work. It also provides some guidelines for the used notation. Chapter 3 discusses the knowledge areas that are covered by the Mario Ontology Network. These knowledge areas have been identified by analysing MARIO functionalities and requirements (Task 1.2). The chapter also provides details, i.e. ontology design methodologies, specific requirements and design choices, about the ontology modules that are currently available in MON. Chapter 4 presents the software framework that supports the usage of MON and depicts the implemented software architecture.

1.5 About MARIO

MARIO addresses the difficult challenges of loneliness, isolation and dementia in older persons through innovative and multi-faceted inventions delivered by service robots. The effects of these conditions are severe and life-limiting. They burden individuals and societal support systems. Human intervention is costly but the severity can be prevented and/or mitigated by simple changes in self-perception and brain stimulation mediated by robots.

From this unique combination, clear advances are made in the use of semantic data analytics, personal interaction, and unique applications tailored to better connect older persons to their care providers, community, own social circle and also to their personal interests. Each objective is developed with a focus on loneliness, isolation and dementia. The impact centres on deep progress toward EU scientific and market leadership in service robots and a user driven solution for this major societal challenge. The competitive advantage is the ability to treat tough challenges appropriately. In addition, a clear path has been developed on how to bring MARIO solutions to the end users through market deployment.

Background

MARIO's Knowledge Base System relies on Semantic Web technologies including languages, tools and methodologies at the state-of-the-art in knowledge management system. This section provides an overview of these technologies.

2.1 Semantic Web Languages: RDF, OWL and SPARQL

The Semantic Web is an extension of the Web aims at providing a common framework that allows data to be shared and reused across application boundaries. Standardisation for Semantic Web is under the care of World Wide Web Consortium (W3C). The W3C standards for the Semantic Web mainly include: XML, RDF(S), OWL and SPARQL. Figure 2.1 shows the semantic web stack and provides an overview of the standard technologies recommended by the W3C.

2.1.1 Extensible Markup Language (XML)

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a both human-readable and machine-readable format. An XML document consists of a properly nested set of open and close tags, where each tag can have a number of attribute-value pairs. Crucial to XML is that the vocabulary of the tags and their allowed combinations is not fixed, but can be defined per application of XML. In the Semantic Web context, XML is being used as a uniform data-exchange format thus providing a common syntax for exchange data across the web.

2.1.2 Resource Description Framework (RDF)

Resource Description Framework (RDF)[1] is a W3C recommendation originally designed as metadata model, it has being used as a general framework for modelling information. The basic construction in RDF is the triple <subject, preficate, object>. The subject denotes

[1]RDF, W3C Recommendation `https://www.w3.org/TR/rdf11-concepts/`

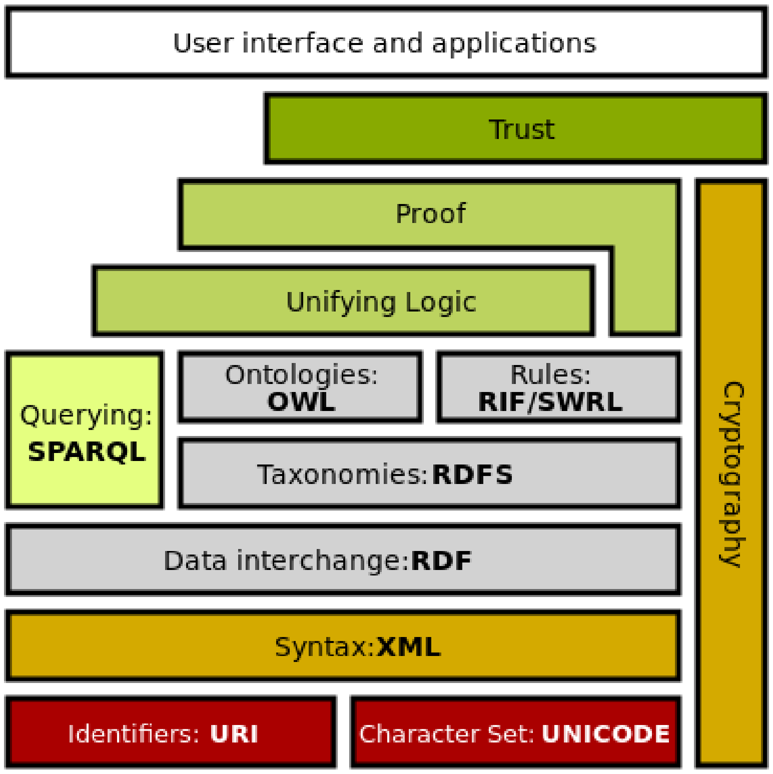


Figure 2.1: The Semantic Web stack.

a resource and the predicate expresses a relationship between the subject and the object (which can be a value or another resource). For example, a way for representing the notion "The author of *War and Peace* is *Leo Tolstoy*" is

```
:War_and_Peace :author :Leo_Tolstoy
```

where `:War_and_Peace` and `:Leo_Tolstoy` are the Uniform Resource Identifiers (URIs) of two resources representing respectively the book titled "War and Peace" and the writer "Leo Tolstoy", and `:author` is the URI of the predicate "author" which is used to connect a book to its author. It is easy to see that an RDF model can be seen as a graph where nodes are values or resources and edges are properties. Several common serialisation formats of RDF are in use, including: TURTLE[2], RDF/XML[3], N-Triples[4].

RDF Schema (RDFS)[5] provides a data-modelling vocabulary for RDF data. RDFS is an extension of RDF aims at providing basic elements for structuring RDF resources. It allows to define: Classes, Properties, Datatypes and Hierarchies for both classes and properties.

[2]TURTLE, `https://www.w3.org/TR/turtle/`
[3]RDF/XML, `https://www.w3.org/TR/rdf-syntax-grammar/`
[4]N-Triples, `https://www.w3.org/TR/n-triples/`
[5]RDFs, W3C Recommendation `https://www.w3.org/TR/rdf-schema/`

2.1.3 Web Ontology Language (OWL)

The Web Ontology Language (OWL)[6] is a semantic markup language for defining, publishing and sharing ontologies on the World Wide Web. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called *ontology*. OWL is part of the Semantic Web stack (see Figure 2.1) and it is complementary to XML, RDF and RDFS:

- *XML* provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents;
- *RDF* is a datamodel for resources and relations between them. It provides a simple semantics for this datamodel;
- *RDFS* is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalisation-hierarchies of such properties and classes;
- *OWL* adds constructs for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. ”exactly one”), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

2.1.4 SPARQL

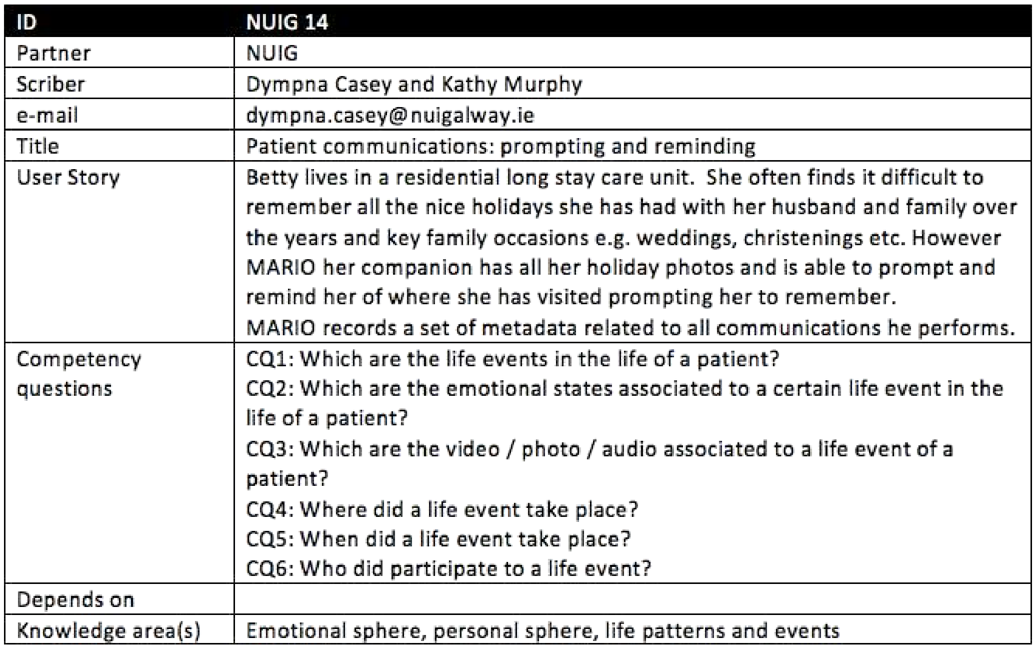
SPARQL[7] is a query language for retrieving and manipulating data store in RDF format. Most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable (denoted by a question mark). A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph. For example, the following SPARQL query retrieves pairs *book* together with its author *author*:

```
SELECT ?book ?author WHERE {?book :author ?author}
```

2.2 Pattern-based Ontology Design

The notion of “pattern” has proved useful in design, as exemplified in diverse areas, such as software engineering. Under the assumption that there exist classes of problems that can be solved by applying common solutions (as has been experienced in software engineering), it is suggested to support reusability on the design side specifically. To this end Ontology Design

[6]OWL, W3C Recommendation `https://www.w3.org/TR/owl-ref/`
[7]SPARQL, W3C Recommendation `https://www.w3.org/TR/rdf-sparql-query/`



ID	NUIG 14
Partner	NUIG
Scriber	Dympna Casey and Kathy Murphy
e-mail	dympna.casey@nuigalway.ie
Title	Patient communications: prompting and reminding
User Story	Betty lives in a residential long stay care unit. She often finds it difficult to remember all the nice holidays she has had with her husband and family over the years and key family occasions e.g. weddings, christenings etc. However MARIO her companion has all her holiday photos and is able to prompt and remind her of where she has visited prompting her to remember. MARIO records a set of metadata related to all communications he performs.
Competency questions	CQ1: Which are the life events in the life of a patient? CQ2: Which are the emotional states associated to a certain life event in the life of a patient? CQ3: Which are the video / photo / audio associated to a life event of a patient? CQ4: Where did a life event take place? CQ5: When did a life event take place? CQ6: Who did participate to a life event?
Depends on	
Knowledge area(s)	Emotional sphere, personal sphere, life patterns and events

Figure 2.2: An example of collected uses-story.

Patterns (ODPs) have been proposed as modeling solutions to recurrent ontology design problems. ODPs are modeling components that can be used as basic building blocks of an *ontology network*. eXtreme Design (XD) is an ontology design methodology that supports the pattern-based approach. In MARIO we adopted XD as methodology for the ontology design and we extensively reused ODPs. Sections 2.2.1 and 2.2.2 briefly introduce ODPs and XD, respectively.

2.2.1 Ontology design patterns

Ontology Design Patterns (ODPs) [20] is an emerging technology that favors the reuse of encoded experiences and good practices. ODPs are modeling solutions to solve recurrent ontology design problems. They can be of different types including: (i) *logical*, which typically provide solutions for solving problems of expressivity e.g., expressing n-ary relations in OWL; (ii) *architectural*, which describe the overall shape of the ontology (either internal or external) that is convenient with respect to a specific ontology-based task or application e.g. a certain DL family; (iii) *content*, which are small ontologies that address a specific modeling issue, and can be directly reused by importing them in the ontology under development e.g., representing roles that people can play during certain time periods; (iv) *presentation*, which provide good practices for e.g. naming conventions.

2.2.2 eXtreme Design

eXtreme Design (XD) [48, 6, 49] is a family of methods and associated tools, based on the application, exploitation, and definition of ontology design patterns (ODPs) for solving ontology development issues. XD principles are inspired by those of the agile software methodology called eXtreme Programming (XP). The main idea of agile software development is to be able to incorporate changes easily, in any stage of the development. Instead of using a waterfall-like method, where you first do all the analysis, then the design, the implementation and finally the testing, the idea is to cut this process into small pieces, each containing all those elements but only for a very small subset of the problem. XD is test-driven, and applies the divide-and-conquer approach as well as XP does. Also, XD adopts pair design, as opposed to pair programming. The main principles of the XD method can be summarised as follows:

- **Customer involvement and feedback.** The customer should be involved in the ontology development and its representative should be aware of all parts of the ontology project under development. Interaction with the customer representative is key for favoring the explicit expression of the domain knowledge.
- **Customer stories and Competency Questions.** The ontology requirements and its tasks are described in terms of small stories by the customer representative. Designers work on those small stories and, together with the customer, transform them in the form of Competency Questions [25] (CQs). CQs will be used through the whole development, and their definition is a key phase as the designers have the challenge to help the customer in making explicit as much implicit knowledge as possible. At the beginning of the task T5.1 we asked all the partners to provide their stories. The template for providing the stories is shown in Figure 2.2. The fields "*Partner*", "*Scriber*", "*e-mail*" were used for asking further clarification about the story. The *Title* field helped for a better understanding the main focus of the story. The "*Priority*" field was used to choose the stories to treat first. The allowed values were *High*, *Medium* and *Low*. "*Depends on*" allowed to specify a link between two stories. For example, if a story was too long, it could be split into two stories and this field allowed one to express the dependency. The last field "*Knowledge area(s)*" was used for associating the story with one or more knowledge areas which the story belonged to. The customer stories collected together with the resulting Competency Questions can be retrieved on-line[8]. Other competency questions have been extracted by analysing domain documents, such as those used for effectuating a Comprehensive Geriatric Assessment (CGA) of a patient.
- **Content Pattern (CP) reuse and modular design.** A development project is characterised by two main sets: (i) the *problem space* composed of the actual modelling issues that have to be addressed during the project which are called "Local Use Case" (LUC); (ii) the *solution space* made up of reusable modelling solutions, called "*Global Use Case*" (GUC), representing the problem that a certain ODP provides a solution for.

[8]http://etna.istc.cnr.it/mario/D5.1/.

If there is a CP's GUC that matches a LUC it has to be reused, otherwise a new module is created. An analysis of the possible strategies for reusing CP is provided by [49].

- **Collaboration and Integration.** Collaboration and constant sharing of knowledge is needed in a XD setting, in fact similar or even the same CQs and sentences can be defined for different stories. When this happens, it means that these stories can be modelled by reusing a set of shared CPs.
- **Task-oriented design.** The focus of the design is on that part of the domain of knowledge under investigation that is needed in order to address the user stories, and more generally, the tasks that the ontology is expected to address.
- **Test-driven design.** A new story can be treated only when all unit tests associated with it have been passed. An ontology module developed for addressing a certain user story associated to a certain competency question, is tested e.g. (i) by encoding in the ontology a sample set of facts based on the user story, (ii) defining one or a set of SPARQL queries that formally encode the competency question, (iii) associating each SPARQL query with the expected result, and (i) running the SPARQL queries against the ontology and compare actual with expected results.

2.3 Graphical notation

The Ontology Definition Metamodel (ODM)[9] is a standard adopted in 2006 that defines a set of UML metamodels and profiles for development of RDF and OWL. The UML profiles in the ODM specification adapt UML notations to provide a form of visual representation for RDF and OWL. In order to improve the effectiveness of the communication between domain experts and ontology engineers an extendion of the ODM OWL profile was provided by [48], who introduced a stereotype for component diagrams that enables the representation of Ontology Design Patterns. In this document we use this extended ODM notation for providing a graphical for representation of the ontology network and its modules.

[9]Ontology Definition Metamodel (ODM), `http://www.omg.org/spec/ODM/`

MARIO Ontology Network

3.1 MARIO Ontology Network Knowledge areas

The MARIO Ontology Network (MON) is composed of different ontologies that cover different knowledge areas that are relevant to MARIO in order to make it a cognitive agent able to support older patient affected by dementia. The knowledge areas were identified by analysing the use cases emerged from the system specification carried on for Pilot 1 in the context of the Work Package 1 [4]. These uses cases mainly describe actions and behaviours featuring the MARIO robots. Nevertheless, they also provide us with detailed descriptions about the nature of the knowledge that the robot should deal with in order to perform and select actions and behaviours, respectively. Hence, we highlighted, for each use case, the knowledge domains required to address such a use case. This process was driven by the identification of the *competency questions* [25] from the textual descriptions of the use cases. We remark that in knowledge engineering the competency questions are commonly identified as the requirements that an ontology has to address. Thus, the knowledge domains emerged from the competency questions we collected, i.e. the knowledge domains identify the topics that specific competency questions answers to. Finally, we gathered a set of top-level knowledge areas by iteratively generalising the knowledge domains by adopting a method similar to the Gronded Theory [55], which is a method often used in Social Sciences to extract relevant concepts from unstructured corpora of natural language resources (e.g., texts, interviews, or questionnaires).

Table 3.1 reports the association of the knowledge areas we identified so far with the use cases analysed in [4]. Each knowledge area is reported along with a brief description that explains its applicability.

Table 3.1: Knowledge areas and their association with the use cases identified in WP1.

Knowldge area	Description	Related use cases
Personal sphere	People information, information about relationship among people, contacts etc.	**UC3.1.1.5** Capture and load personal data for the user
		UC3.1.1.12 Set up users
		UC3.1.3.2 Assist the user with information about people

		UC3.1.6.2 CGA: Question User about Family
Life events and patterns	Information about everyday events, memories, scheduling, plans etc.	**UC3.1.3.1** Add Events
		UC3.1.3.4 Help the user carry out a sequence of actions
		UC3.1.3.5 Inform the user about events
		UC3.1.3.6 Suggest things the user can do
		UC3.1.6.3 CGA: Question user about Daily Living activity
		UC3.1.6.8 Monitor the daily pattern of the user
Social and multimedia content	Online social network community, multimedia content such as photos, videos, movies, documents	**UC3.1.1.1** Choose and pre-load Games for the User
		UC3.1.1.2 Choose and pre-load Music for the User
		UC3.1.1.3 Choose and pre-load Videos for the User
		UC3.1.2.2 Play Music for the User
		UC3.1.2.3 Play a Game with the User
		UC3.1.2.4 Read a text to the User
		UC3.1.2.5 Show a video to the User
Environment	Information about rooms, furnitures, objects etc.	**UC3.1.1.8** Name Locations and rooms on the map
		UC3.1.1.11 Map Operating Environment
		UC3.1.3.3 Assist the user with information about place and locations
		UC3.1.4.1 Approach the user
		UC3.1.4.2 Identify the User in the Immediate Area
		UC3.1.4.3 Search for the User in the Operating Environment
		UC3.1.6.9 Record where the user goes and what they do during the day

Health sphere	Information about living patterns, health patterns, vital signs, anything related to CGI and MPI	**UC3.1.6.1** CGA: Assess the user when using a Telephone
		UC3.1.6.2 CGA: Question User about Family
		UC3.1.6.3 CGA: Question user about Daily Living activity
		UC3.1.6.4 Question User to Establish Emotional State
		UC3.1.6.5 Carry out a CGA assessment on the user
		UC3.1.6.6 Generate Health reports for the care staff
		UC3.1.6.7 Monitor the Health of the user
		UC3.1.6.8 Monitor the daily pattern of the user
		UC3.1.6.9 Record where the user goes and what they do during the day
		UC3.1.8.1 Ask the user a series of questions to establish facts about them, or examine their heath or how they are feeling.
Emotional sphere	Information about emotions, sentiments, interests, opinions related to people etc.	**UC3.1.6.4** Question User to Establish Emotional State
		UC3.1.7.3 Identify and remember what the user likes
		UC3.1.7.4 Show the User some items from the generic reminiscence store that match their era.
		UC3.1.7.5 Show the User some items from their personal reminiscence store.
		UC3.1.8.1 Ask the user a series of questions to establish facts about them, or examine their heath or how they are feeling.
Open knowledge	Speech-derived data, web-extracted data etc.	**UC3.1.3.5** Inform the user about events
		UC3.1.3.6 Suggest things the user can do

		UC3.1.8.1 Ask the user a series of questions to establish facts about them, or examine their heath or how they are feeling.
Regulatory sphere	Information about norms, rules, social habits etc.	**UC3.1.3.4** Help the user carry out a sequence of actions
		UC3.1.3.6 Suggest things the user can do
MARIOcpetion	Information about MARIO abilities, MARIO functionalities, applications MARIO is able to run, actions MARIO is able to do etc.	This knowledge area is orthogonal to all the use cases

3.2 Ontology design methodology

The MON is designed by following best design practices and pattern-based ontology engineering methods aimed at extensively re-using Ontology Design Patterns (ODPs) [20]. According to [20], ODPs are modeling solutions that can be re-used in order to solve recurrent ontology design problems. Hence, they enable design by-reuse methodologies. For example, ODPs can be re-used by means of their specialisation or composition according to their types and to the scope of the new ontology that is going to be modelled. The design methodology that we followed is based on the eXtreme Design [6, 48] (cf. Section 2.2.2). The eXtreme Design (XD) is an agile design methodology developed in the context of the NeON project[1]. XD is inspired by the eXtreme Programming (XP). In fact, like XP, it emphasises incremental development and recommends pair development, test driven development, refactoring, and a divide-and-conquer approach to problem-solving [17]. Additionally, XD uses competency questions as a reference source for the requirement analysis and associates ODPs with generic use cases in the solution space. The problem space is composed of local use cases that provide descriptions of the actual issues. Use cases represent problems that ODPs provide solutions to. Both global and local use cases are competency questions expressed in natural language. The separation of use cases and the way in which the latter are expressed makes possible to match local use cases against global use cases. This matching conveys suitable ODPs to be exploited for solving modelling problems. The XD methodology is implemented by the XD tool that is available as a plug-in for both the NeON platform and TopBraid composer[2]. Recently [26] proposed an extension[3] of XD that enables its usage in

[1]http://www.neon-project.org/nw/.
[2]http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/
[3]https://github.com/hammar/webprotege.

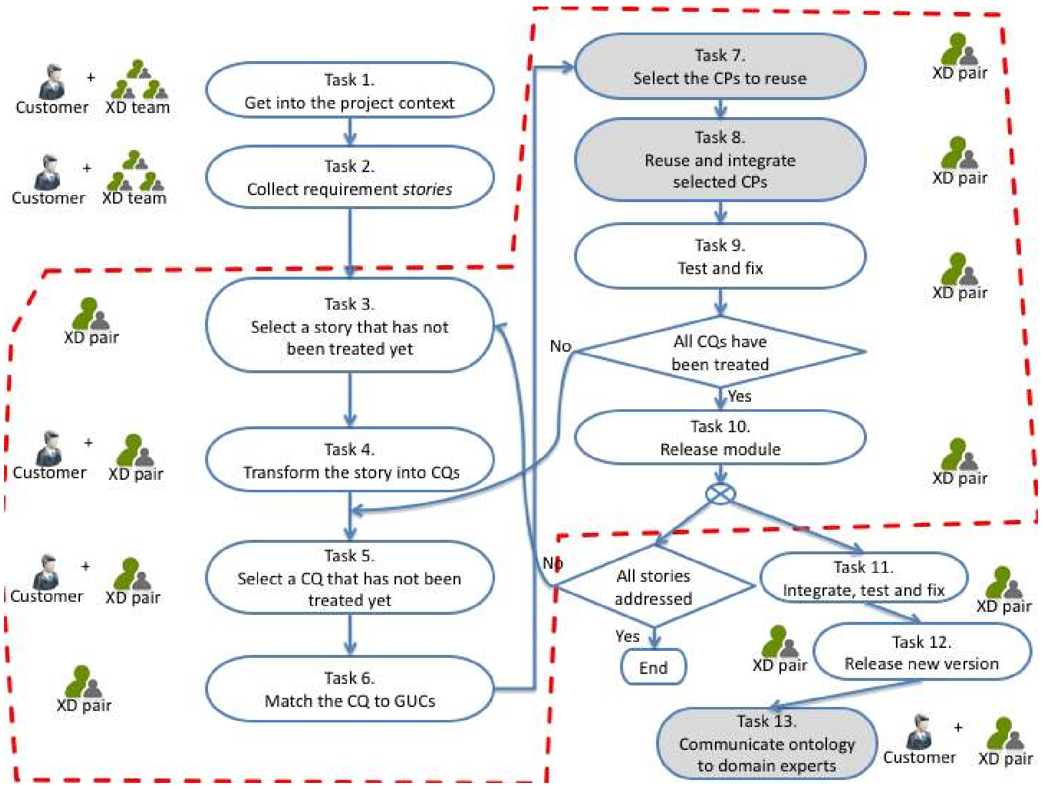

Customer
XD team
Task 1. Get into the project context
Task 2. Collect requirement stories
Task 3. Select a story that has not been treated yet
Task 4. Transform the story into CQs
Task 5. Select a CQ that has not been treated yet
Task 6. Match the CQ to GUCs
Task 7. Select the CPs to reuse
Task 8. Reuse and integrate selected CPs
Task 9. Test and fix
All CQs have been treated
No
Yes
Task 10. Release module
All stories addressed
End
Task 11. Integrate, test and fix
Task 12. Release new version
Task 13. Communicate ontology to domain experts
XD pair

Figure 3.1: The XD workflow as extended in [49]

the WebProtégé[4]. This extensions allows to combine the benefit of the agile ontology development process provided by XD and a modern collaborative ontology engineering platform consisting of WebProtégé.

Figure 3.1 shows the XD workflow we extended in [49]. Such an extension mainly focuses on providing ontology engineer with clear strategies for ontology re-use. Ontology re-use models can be classified based on (i) the type of reused ontology (e.g. foundational, top-level, ontology design patterns, domain ontologies), (ii) the type of reused ontology fragment (e.g. individual entities, modules, ontology design patterns, arbitrary fragments), (iii) the amount of reused axioms (e.g. import of all axioms, of only axioms in a given neighbourhood of an entity, of no axioms), (iv) and the alignment policy (e.g. direct reuse of entities, reuse via equivalent relations such as `rdfs:subClassOf` and `owl:equivalentClass`). For the development of the MON we configured XD in order to have an *indirect re-use* of ontology design patterns and alignments. ODPs are used in this case as templates. At the same time, the ontology guarantees interoperability by keeping the appropriate alignments with the external ODPs, and provides extensions that satisfy more specific requirements. The alignment axioms may be published separately from the core of the ontology. If the ontology needs further extensions, the same approach must be followed in order to avoid the introduction of dependencies. With this type of re-use, the potential impact of possible changes in the external ODP is minimised. In fact, if incoherences or changes should occur in the external

[4]http://webprotege.stanford.edu/.

ODP (which is rather unlikely to happen) then the redesign process would very simple. The ontology signature and axioms would remain unchanged, as incoherences or changes would be resolved by simply removing or revising the alignment axioms.

3.3 MON modules

Figure 3.2 shows the top level of the MON. The top level is identified by the OWL ontologies connected to the MARIO ontology (i.e., mario.owl[5]) by green arrows. The arrows, independently from their colors, represent `owl:imports` axioms. Each ontology module connected to the top level MARIO ontology identifies a knowledge area as introduced in Section 3.1 and is itself a broader conceptualisation built on top of more specific ontology modules. Such modules are, in turn, built by means of re-use of ontology design patterns according to our ontology design methodology. In next sections we detail the ontology modules of the MON (i.e., framester, cga, affordance and tagging) that we developed so far and are relevant to the Pilot 1. Other ontology modules are part of our ongoing work and will be developed, released and assessed by next tasks part of the Work Package 5 (i.e. T5.2 - Robot Raeading and Listening component and T5.3 - Robot Sentiment Aalysis) and the Work Package 6 (i.e. T6.1 - Development of MARIO's behavioural capabilites and T6.2 - Development of MARIO's social skills), respectively. Nevertheless, we have already identified ontology modules that are not currently implemented in the MON and that are part of our ongoing work, as aforementioned, for next tasks. Namely, those modules are:

- the Computer-based Patient Record (CPR) Ontology[6], which provides a modelling solution for describing patient-related records;
- DogOnt[7], which provides a suitable solution for describing domotic devices, the home environmenthow and how architectural elements and furniture are placed inside the home;
- the Emotion Ontology (MFOEM)[8] Emotion Ontology for representing phenomena such as emotions and moods;
- the Middle Layer Ontology for Clinical Care (MLOCC)[9] that guarantees the exploitation of well-founded and formalised medical concepts;
- the RECommendations Ontology (RECO)[10] that provides domain-independent means to describe user profiles and preferences in a coherent and context-aware way;

[5]The top level is available at `http://www.ontologydesignpatterns.org/ont/mario/mario.owl`.
[6]`http://purl.org/cpr/0.9`.
[7]`http://elite.polito.it/ontologies/dogont.owl`.
[8]`http://www.ontobee.org/ontology/MFOEM`.
[9]`http://www.ifomis.org/chronious/mlocc`.
[10]`http://purl.org/reco`.

- the Symptom Ontology (SYMP)[11] that provides means to describe symptoms associated with deaseases.

In next sections we detail the ontology modules that have been implemented so far and are part of the MON accordingly.

3.3.1 Framester

Many resources belonging to different domains are now being published on-line using Linked Data principles to provide easy access to structured data on web. This includes many linguistic resources that are already part of Linked Data, but made available mainly for the purpose of being used by NLP applications.

When dealing with robot understanding, we need to integrate knowledge about the robot's physical context, linguistic knowledge, and knowledge about the world in general. Designing a knowledge base for an assistive robot can be therefore considered a direct application of the Linked Data paradigm, which provides interoperability across existing Linked Open Data (world's background knowledge), linguistic knowledge, user's knowledge and robot's sensor knowledge. In order to create an adequate amount of background knowledge, and to make it accessible to the robot, Framester[12], a large "cloud" of linguistic and factual data has been created which stands on the shoulders of a flexible and cognitively-sound theory of human sense making, i.e. Frame Semantics [14].

Framester is intended to work as a knowledge graph/linked data hub to connect lexical resources, NLP results, linked data, and ontologies. It is bootstrapped from existing resources, notably the RDF versions of FrameNet [41], WordNet, VerbNet, and BabelNet, by interpreting their semantics as a subset of (a formal version of) Fillmore's frame semantics [14], and semiotics [16], and by reusing or linking to off-the-shelf ontological resources including OntoWordNet, DOLCE-Zero, Yago, DBpedia, etc.

State of the art

Two of the most important linguistic linked open data resources are WordNet [13] and FrameNet [1]. They have already been formalised as semantic web resources, e.g. in OntoWordNet [18], WordNet RDF [57], FrameNet RDF [41], etc. FrameNet allows to represent textual resources in terms of Frame Semantics. The usefulness of FrameNet is limited by its limited coverage, and non-standard semantics. An evident solution would be to establish valid links between FrameNet and other lexical resources such as WordNet , VerbNet [30] and BabelNet [39] to create wide-coverage and multi-lingual extensions of FrameNet. By overcoming these limitations NLP-based applications such as question answering, machine reading and understanding, etc. would eventually be improved. Within MARIO these are important require-

[11] http://purl.obolibrary.org/obo/symp.owl.
[12]

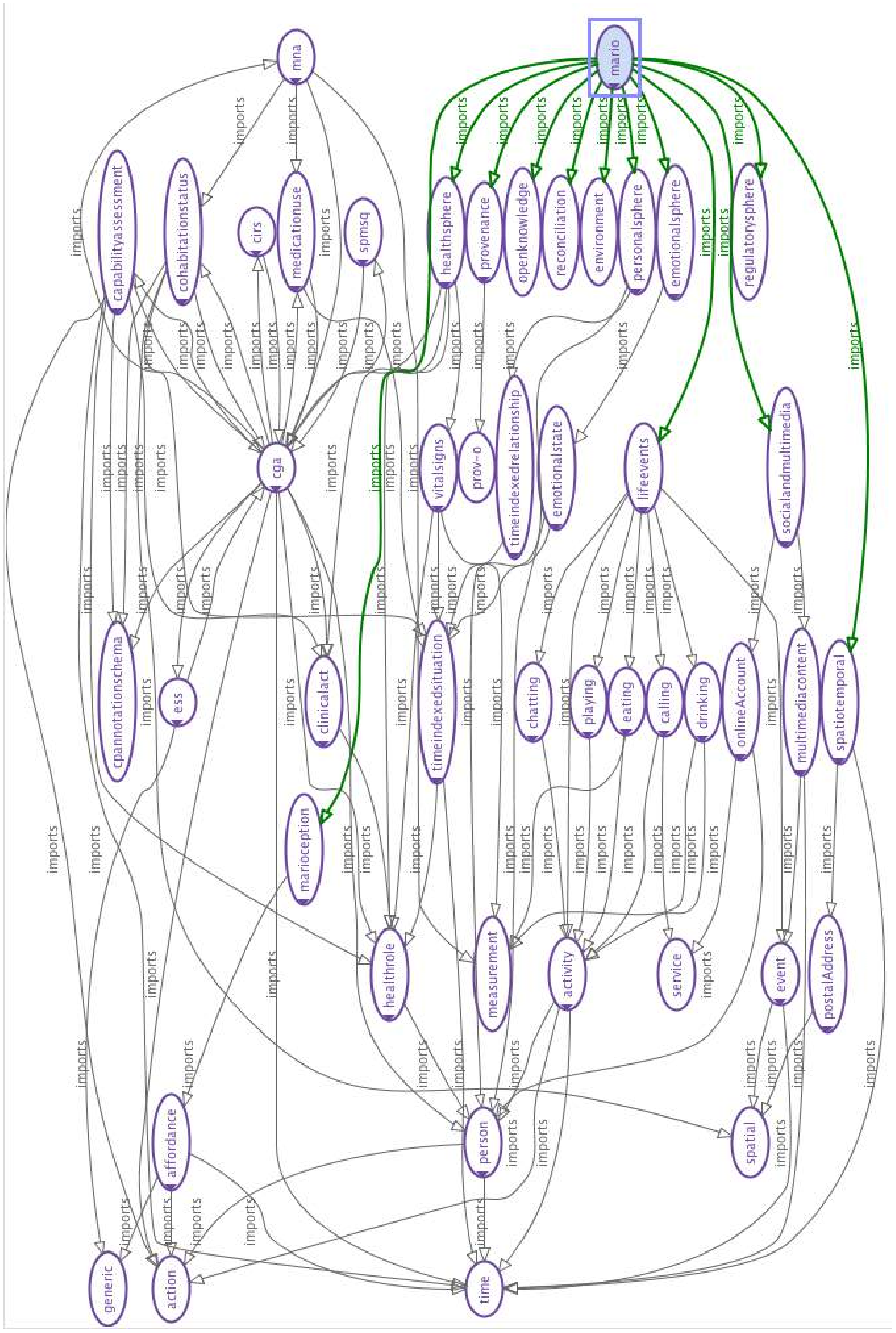


Figure 3.2: Top level o the MON

ments, hence we developed Framester: a frame-based ontological resource acting as a hub between e.g. FrameNet, WordNet, VerbNet, BabelNet, DBpedia, Yago, DOLCE-Zero, and leveraging this wealth of links to create an interoperable *predicate space* formalised according to frame semantics [14], and semiotics [16]. Data designed according to the predicates in the predicate space created by Framester result to be more accessible and interoperable, modulo alignments between specific entities or facts.

The closest resources to Framester are FrameBase [51] and Predicate Matrix [32].

FrameBase is aimed at aligning linked data to FrameNet frames, based on similar assumptions as Framester's: full-fledged formal semantics for frames, detour-based extension for frame coverage, and rule-based lenses over linked data. However, the coverage of FrameBase is limited to an automatically learnt extension (with resulting inaccuracies) of FrameNet-WordNet mappings, and the alignment to linked data schemas is performed manually. Anyway, Framester could be combined with FrameBase (de)reification rules so that the two projects can mutually benefit from their results.

Predicate Matrix is an alignment between predicates existing in FrameNet, VerbNet, WordNet, and PropBank. It does not assume a formal semantics, and its coverage is limited to a subset of lexical senses from those resources. A RDF version of Predicate Matrix has been created in order to add it to the Framester linked data cloud, and (ongoing work) to check if those equivalences can be reused in semantic web applications.

Ontology Description

Framester uses the D&S (Descriptions and Situations) knowledge pattern [17], which allows to distinguish the reification of the intension of a predicate (a *description*) from the reification of the extensional denotation of a predicate (a *situation*). A description d can define or reuse *concepts* $c^1, ..., c^n$ that can be used to *classify* entities $e^1...e^m$ involved in a situation s that is expected to be compatible with d. For those reasons, D&S perfectly fits the core assumptions of Fillmore's frame semantics, by which a frame is a schema for conceptualising the interpretation of a natural language text (and beyond), its denotation (a frame occurrence) is a situation, and the elements (or semantic roles) of a frame are aspects of a frame, which can be either obligatory, optional, inherited, reused, etc. Furthermore, D&S [17] takes into account a semiotic theory to integrate linguistic and formal semantics. It can therefore support additional frame semantics assumptions such as *evocation* and *semantic typing*.

As described in [41], several recipes can be designed to interpret FrameNet frames and frame elements as OWL classes, object properties, or punned individuals. Both FrameBase and Framester make use of the basic recipe that interprets frames as classes and frame elements as properties. However, Framester goes deeper in providing a two-layered (intensional-extensional) semantics for frames, semantic roles, semantic types, selectional restrictions, and the other creatures that populate the world of lexical resources. The two-layered representation is based on the Descriptions and Situations pattern framework [17], and exploits OWL2 punning, so enabling both (intensional) navigation in the linked lexical datasets, and the reuse of lexical predicates as extensional classes or properties. The main assumptions for Framester knowledge graphs are as follows:

1. A frame is a *multigrade intensional predicate* [42] $f(e, x_1, ..., x_n)$, where f is a first-order relation, e is a (Neo-Davidsonian) variable for any *eventuality* or *state of affairs* described by the frame, and x_i is a variable for any *argument place*, which could admit several *positions* in case multiple entities are expected to be classified in a place.
2. OWL2 punning allows to represent a frame as either a class $f \sqsubseteq \texttt{framester:Situation}$ (a subclass of the framester:Situation class, having situations as instances) or as an individual $f \in \texttt{framester:Frame}$ (an instance of the framester:Frame class).
3. Any word or multiword can evoke a frame: this is represented by means of a property chain that connects a word entity to a (punned) frame.
4. *Frame Projections* include any projections of a frame relation. Assuming frame semantics, each meaning consists of activated frames, whose formal counterparts are multigrade intensional predicates. When only some aspect of that frame is considered, it can be formalized as a (typically unary or binary) projection of a frame relation. Semantic role as well as co-participation relation are binary projections of a frame.
5. A *frame occurrence* (a situation denoted by text or data) $s \in f$ is an instance of f and the entities $\{e, x_1...x_n\}$ involved in a situation are individuals.

Due to the expressivity limitations of OWL, some refactoring was needed to represent frame semantics: frames are represented as both classes and individuals, semantic roles and co-participation relations as both (object or datatype) properties and individuals, selectional restrictions and semantic types as both classes and individuals, situations and their entities as individuals. Frames and other predicates are represented as individuals when a schema-level relation is needed (e.g. between a frame and its roles, or between two frames), which cannot be represented by means of an OWL schema axiom (e.g. subclass, subproperty, domain, range, etc.).

Figure 3.3 gives an overview of Framester modules and interlinks.

Example

Figure 3.4 shows how the D&S pattern framework (descriptions, situations, classification patterns) is at the basis of Framester representation of the example predicate `G_suit`. Related notions from WordNet (wn:), BabelNet (bn:), FrameNet (fe:), DBpedia (dbr:), and DOLCE-Zero (dul:) make linguistic and factual data linked by using Framester ontology (fschema:) and data (framester:), and OWL logic (owl:, rdfs:). In practice, this automated integration allows to represent any data coming from different resources (i.e. not only those depicted, but all those that are associated with them in the Linked Open Data cloud) in a homogeneous and logically rigorous way. That would include instances of G-suits, knowledge about

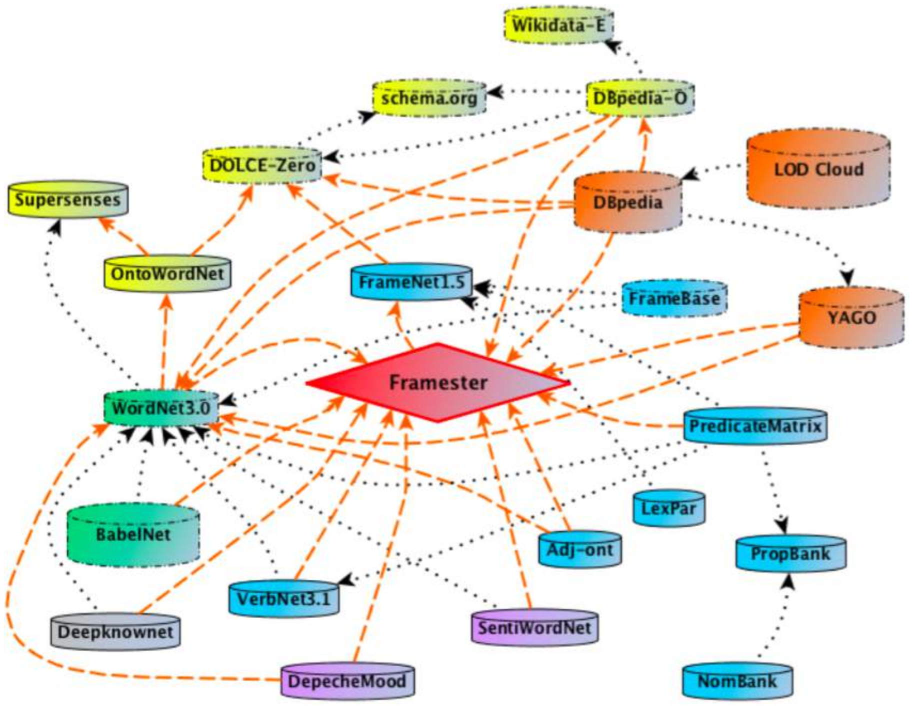


Figure 3.3: Framester: frame-based lexical linked data. The set of resources that compose Framester and the interlinks between them.

G-suits, places where G-suits are produced or used, the frames (e.g. Clothing) that include G_suit as a participant, multilingual versions of the predicates and entities associated with G_suits, etc.

3.3.2 CGA ontology

The evaluation of elderly patients represents a complex universe difficult to evaluate and manage as a unique body. The Comprehensive Geriatric Assessment (CGA) represents one of the most used and validated approaches to evaluate the elderly subjects. It is defined as a "multidimensional interdisciplinary diagnostic process focused on determining a frail older person's medical, psychological and functional capability in order to develop a coordinated and integrated plan for treatment and long term follow up." [52]. It can be roughly viewed as a set of tests aiming at assessing the medical, psychological and functional capability of a person.

One of the objectives of the MARIO project is to take advantage of the continuous monitor of a subject so to facilitate its assessment, to improve the communication inside the team who is in charge of the care of the subject, and to manage and check rehabilitation plans. In the context of MARIO project a customised CGA has been defined with the following instruments: i) *Activities of Daily Living* (ADL) [28] and *Instrumental Activities of Daily Living* (IADL) [33] for evaluating functional disabilities in the daily living; ii) Short-Portable

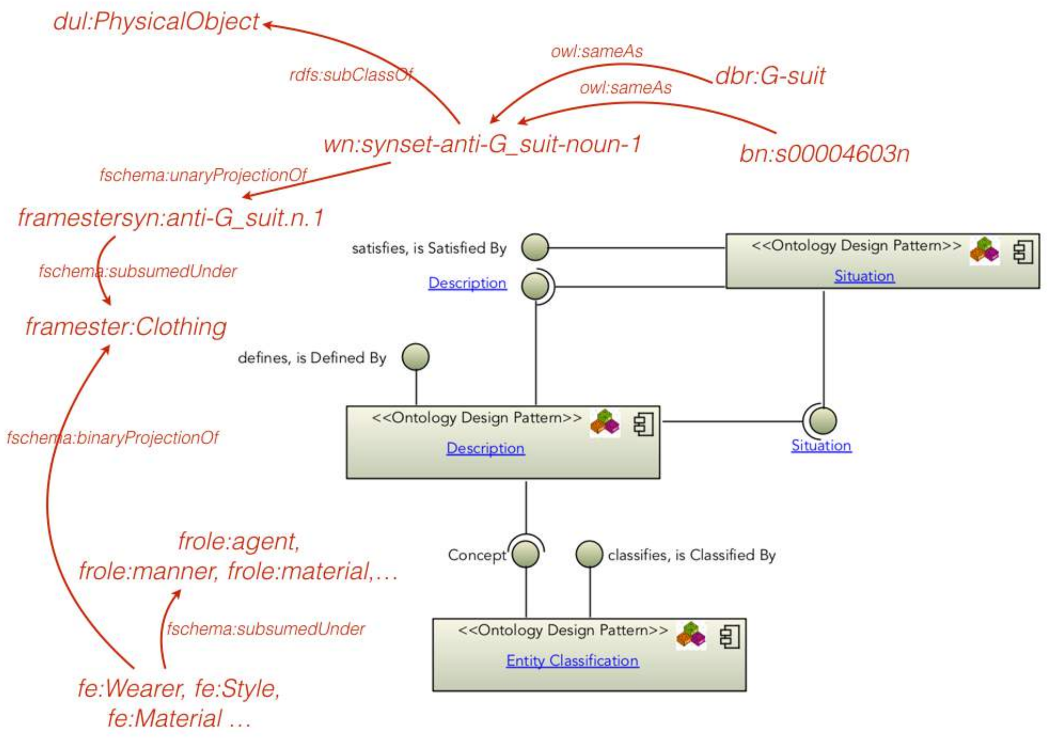

dul:PhysicalObject
rdfs:subClassOf
owl:sameAs
dbr:G-suit
owl:sameAs
wn:synset-anti-G_suit-noun-1
bn:s00004603n
fschema:unaryProjectionOf
framestersyn:anti-G_suit.n.1
fschema:subsumedUnder
framester:Clothing
fschema:binaryProjectionOf
frole:agent,
frole:manner, frole:material,...
fschema:subsumedUnder
fe:Wearer, fe:Style,
fe:Material ...
satisfies, is Satisfied By
Description
<<Ontology Design Pattern>>
Situation
defines, is Defined By
<<Ontology Design Pattern>>
Description
Situation
Concept
classifies, is Classified By
<<Ontology Design Pattern>>
Entity Classification

Figure 3.4: An example of Framester representation for the concept `G_suit`.

Mental Status Questionnaire (SPMSQ) [46] for assessing the cognitive status for dementia screening; iii) *Mini-Nutritional Assessment* (MNA) [58] for assessing the nutritional status; iv) *Exton-Smith scale* (ESS) [5] for evaluating the risk of pressure sores in patients at high risk of immobilisation or bed-ridden; v) *Comorbidity Illness Rating Scale* (CIRS) [12] for carefully evaluating the the comorbidities; vi) Evaluation of medication use for assessing the appropriateness of prescriptions, and the risk for adverse drug reactions.

The contribution of the CGA ontology is twofold. On the one hand, the ontology supports the execution of the assessment by providing a reference model for storing test information (such as questions, expected answer etc.). On the other hand, it allows to store the data resulting from the patient's assessments.

State of the art

Medicine is one of the first fields that employed ontologies in knowledge-base systems [47]. Ontologies have been used to create an unified medical language [7, 19], to build a computer based patient record[13], to allow the representation of clinical narratives [56], to support professional decisions in the life-cycle of home care treatments [50], to an ontology-driven

[13]CPRO, `http://ontohub.org/bioportal/CPRO.owl`

Table 3.2: Ontology modules imported/reused by the CGA ontology.

Namespace prefix	Namespace
cga	http://www.ontologydesignpatterns.org/ont/mario/cga.owl#
coh	http://www.ontologydesignpatterns.org/ont/mario/cohabitationstatus.owl#
ca	http://www.ontologydesignpatterns.org/ont/mario/capabilityassessment.owl#
spmsq	http://www.ontologydesignpatterns.org/ont/mario/spmsq.owl#
ess	http://www.ontologydesignpatterns.org/ont/mario/ess.owl#
cirs	http://www.ontologydesignpatterns.org/ont/mario/cirs.owl#
mna	http://www.ontologydesignpatterns.org/ont/mario/mna.owl#
action	http://www.ontologydesignpatterns.org/ont/mario/action.owl#
clinicalact	http://www.ontologydesignpatterns.org/ont/mario/clinicalact.owl#
time	http://www.ontologydesignpatterns.org/ont/mario/time.owl#

adaptive medical questionnaire [8] and so on. To the best of our knowledge it does not exist any ontology able to represents the results of an execution of our customisation of the CGA. Some ontologies have been proposed for supporting the (general) medical assessment process [9, 3]. This ontologies define high-level concepts for representing medical assessment. The CGA ontology follows their approach and specialises the high-level concepts where needed.

Ontology description

The choice of customising approach to the Comprehensive Geriatric Assessment impedes the re-use of an off-the-shelf ontology for the CGA. The requirements of the ontology have been directly derived from the form template[14] used by physicians during the assessment of a Physician's Working Diagnosis (PWD). The competency questions extracted by analyzing these documents can be found at on-line[15].

The ontology is modular. It contains (to be precise, imports) a module for each test included in our customisation of the CGA. The modules composing the CGA ontology (together with the specification of their namespace definition) can be found in Table 3.2.

The submodules addressing specific tests specialise the CGA ontology on the basis of the specific requirements of the test, e.g. the CGA ontology defines the class `cga:GeriatricAssessment` and the ontology addressing ADL and IADL specialises that class with `ca:CapabilityAssessment`.

The Figure 3.5 shows the UML class diagram of the CGA ontology. As in [3], a patient assessment (i.e. `cga:GeriatricAssessment`) is an action having as participant the assessed `healthrole:Patient` and an `action:Agent`[16] who make the assessment. The agent making the assessment can be either a `healthrole:Physician` or another kind of agent (e.g. MARIO). In order to represent the *description* of how the assessment is to

[14]http://etna.istc.cnr.it/mario/D5.1/CGA.pdf.
[15]http://etna.istc.cnr.it/mario/D5.1/CQ-CGA.pdf.
[16]Since `cga:GeriatricAssessment` specialises the class `action:Action`

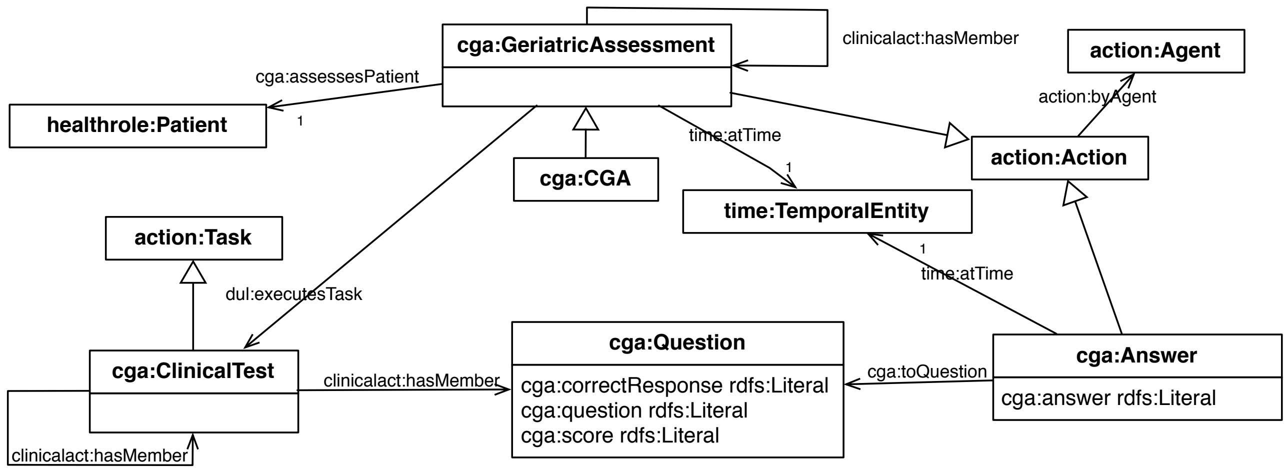


Figure 3.5: The UML class diagram of CGA ontology.

be executed, we implemented the Ontology Design Pattern *Task Execution*[17]. The action `cga:GeriatricAssessment` executes a `cga:ClinicalTest` which provides a "*description*" of how the assessment has to be executed. A `cga:ClinicalTest` can be composed of other clinical tests or some `cga:Question`. Furthermore the CGA ontology allows to store information about the answers (i.e. `cga:Answer`) a patient provides to reply a question.

Example

The Frame 3.1 shows an example of usage of the CGA ontology. The resource `:CGA` represents the CGA (intended as "*test*"), whereas the resource `:CGA-20160617` represents the actual execution of `:CGA`, performed on 17 June 2016, for assessing the patient `:Freddy`. In this example `:CGA` is composed only of the test `:SPSMQ` which represents a questionnaire containing only of the question (i.e. `:Q1-SPMSQ`) "Who is the president now?". `:Q1-SPMSQ` defines the conditions under which the answer provided by the patient has to be considered correct (i.e. "Requires only the correct last name") and the score to be given if the patient properly responds.

The actual execution of the CGA effectuated by the agent `:DrRossi` for assessing the patient `:Freddy` is represented by `:CGA-20160617`. Within `:CGA-20160617` the Short Portable Mental Status Questionnaire is performed, i.e. `:SPMSQ-20160616`. Freddy's answer (i.e. `:Answer-Freddy-Q1-20160616`) to question Q1 is "Mattarella". The answer has been considered correct by the `:DrRossi` who gave the score "1" to it. The answer's assessment effectuated by the `:DrRossi` is represented by `:Answer-Freddy-Q1-20160616-Assessment`.

[17] Aldo Gangemi, Task execution ODP `http://www.ontologydesignpatterns.org/cp/owl/taskexecution.owl`

```
:CGA a  cga:ClinicalTest ;
  clinicalact:hasMember cga:SPMSQ .

:SPMSQ a cga:ClinicalTest ;
  clinicalact:hasMember cga:Q1−SPMSQ .

:Q1−SPMSQ a cga:Question ;
  cga:correctResponse ”Requires only the correct last name”@en ;
  cga:question ”Who is the president now?”@en ;
  cga:score ”1” .

:CGA−20160617 a cga:ComprehensiveGeriatricAssessment ;
  action:byAgent :DrRossi ;
  cga:executesTask :CGA ;
  clinicalact:hasMember :SPMSQ−20160616 ;
  cga:assessesPatient :Freddy .

:SPMSQ−20160616 a spmsq:ShortPortableMentalStatusQuestionnaire ;
  action:byAgent :DrRossi ;
  cga:executesTask :SPMSQ ;
  clinicalact:hasMember :Answer−Freddy−Q1−20160616−Assessment ;
  cga:assessesPatient :Freddy .

:Answer−Freddy−Q1−20160616 a cga:Answer ;
  action:byAgent :Freddy ;
  cga:toQuestion :Q1−SPMSQ ;
  cga:answer ‘‘Mattarella’’ .

:Answer−Freddy−Q1−20160616−Assessment a spsmq:AnswerAssessment ;
  score ‘‘1’’;
  action:byAgent :DrRossi ;
  cga:derivedFrom :Answer−Freddy−Q1−20160616 .
```

Frame 3.1: An example of usage of the CGA ontology and the SPMSQ ontology serialised in TURTLE language

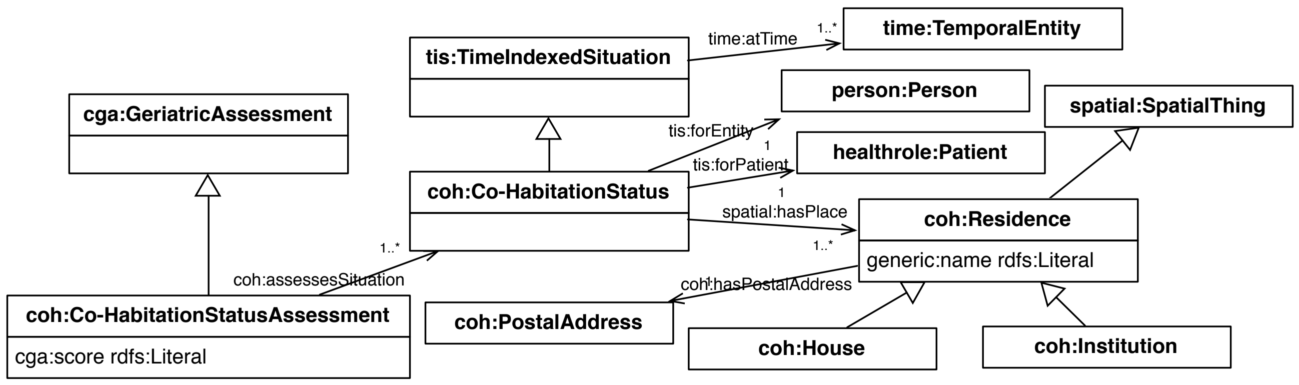


Figure 3.6: The UML class diagram of the Co-Habitation status ontology.

CQ1	Does the patient live alone or with relatives/nurse or in an institution?
CQ2	Which is the address of the place where a patient lives?
CQ3	Which is the name of the place where a patient lives?
CQ4	Which is the assessment for a certain co-habitation status of a certain patient?

Table 3.3: Competency questions answered through the Co-Habitation status ontology.

CGA ontology modules

In this section we provide an overview of the ontology modules composing the CGA ontology.

Co-Habitation Status. The aim of the "*Co-Habitation Status*" ontology is to provide a reference model for representing the habitation status of a patient so to allow assessment on it. The ontology, shown in Figure 3.6, answers to the competency questions reported in Table 3.3. The prefix `coh:` is associated with the namespace `http://www.ontologydesignpatterns.org/ont/mario/cohabitationstatus.owl#`. The ontology implements the ODP *Time Indexed Situation*[18] to represent the `coh:Co-HabitationStatus` of a `healthrole:Patient` at a given time. `coh:Co-HabitationStatus` can be seen as a n-ary relation connecting i) a patient, ii) its `coh:Residence` (an `coh:House` or an `coh:Institution` characterised by a name and a `coh:PostalAddress`), iii) and possibly some cohabitants. A `coh:Co-HabitationStatus` is assessed by a `coh:Co-HabitationStatusAssessment` which assigns a score to it.

Medication Use. The ontology module "*Medication Use*" enables to keep track the medication use of a patient thus allowing to make an assessment on it. The ontology, shown in Figure 3.7, answers to the Competency Question in Table 3.4. The namespace prefix `medicationuse` is associated with `http://www.ontologydesignpatterns.org/ont/mario/medicationuse.owl#`.

[18]Aldo Gangemi, Time Indexed Situation ODP, `http://ontologydesignpatterns.org/cp/owl/timeindexedsituation.owl`

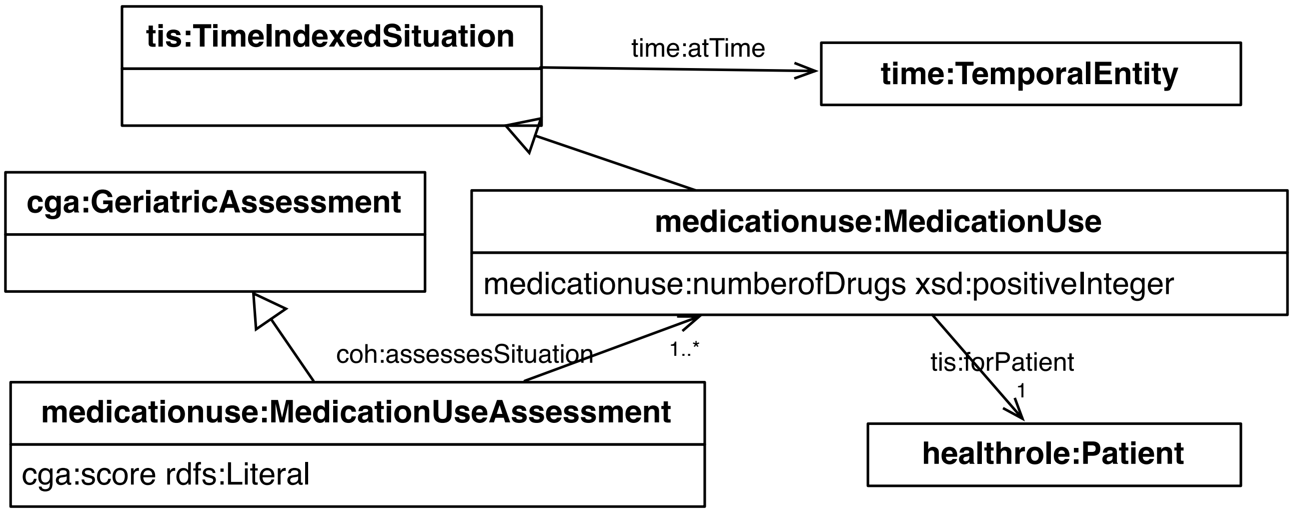


Figure 3.7: The UML class diagram of the Medication Use ontology.

CQ1	Which is the assessment for a certain medication use of a certain patient?
CQ2	How many drugs does a patient use?

Table 3.4: Competency questions answered through the Medication Use ontology.

The *Medication Use* is a *Time Indexed Situation* [18] involving (i) the Patient targeted of certain treatments; (ii) the number of medications used by him; (iii) and the time period in which he takes the medications. A `medicationuse:MedicationUse` is assessed by a `medicationuse:MedicationUseAssessment` which assigns a score to it.

Capability assessment. The ontology module "*Capability assessment*" allows to store into a knowledge base the results of an execution of both *Activities of Daily Living* (**ADL**) and *Instrumental Activities of Daily Living* (**IADL**). The Figure 3.8 shows the UML class diagram of the Capability Assessment ontology. The ontology allows to answer the competency questions listed in Table 3.5. The prefix `ca:` is associated with the namespace `http://www.ontologydesignpatterns.org/ont/mario/capabilityassessment.owl#`.

The *Capability Assessment* ontology defines two `cga:GeriatricAssessment`, i.e. the `ca:IndexAssessment` and `ca:CapabilityAssessment`. The former allows to represent the assessment (e.g. an execution of ADL or IADl) made on the basis of a set of capability assessments (e.g. Bathing, Dressing etc.). Therefore a `ca:IndexAssessment` specifies the set of `ca:CapabilityAssessment` on which the assessment is made and the resulted total score (`cga:score`). The latter, `ca:CapabilityAssessment` is used to evaluate a patient's capability in performing the activities of daily living. A `ca:CapabilityAssessment` is an action used to associate a certain patient with a certain `ca:CapabilityLevel` at a certain time. The `ca:CapabilityAssessment` could be derived from a `cga:Answer` that the patient provide to reply to a `cga:Question` (e.g. *Do you need assistance for bathing? - No*). `ca:CapabilityLevel` is used to define the capability levels for a certain capability to assess.

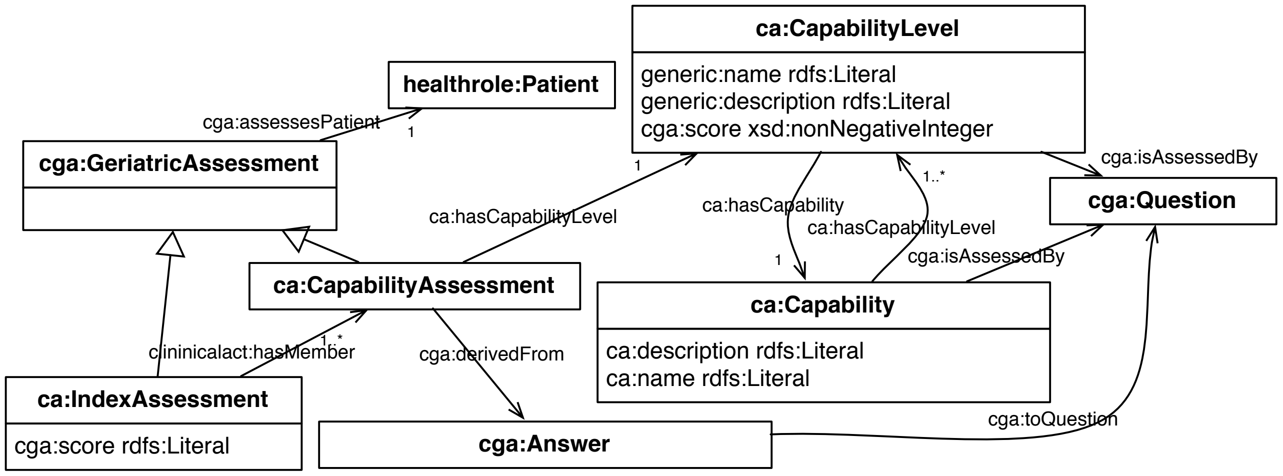


Figure 3.8: The UML class diagram of the Capability Assessment ontology.

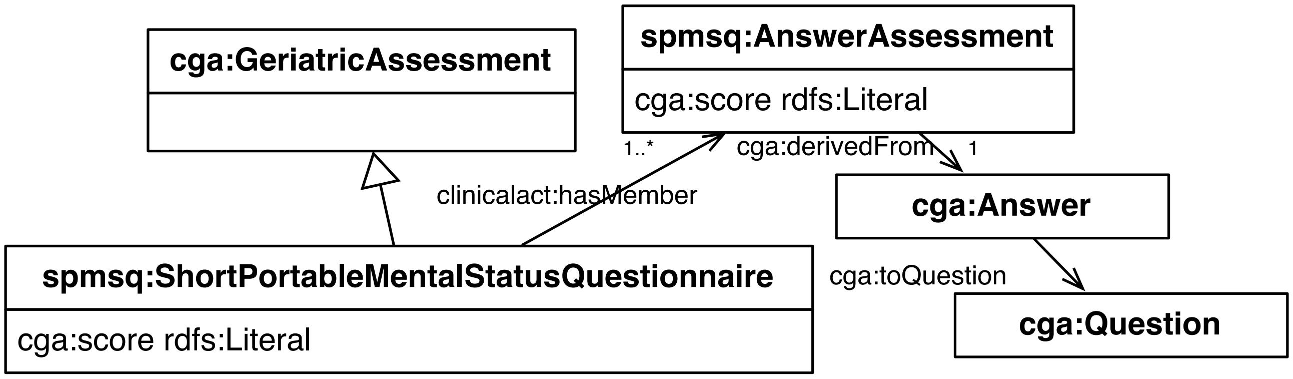


Figure 3.9: The UML class diagram of the SPMSQ ontology.

Each `ca:CapabilityLevel` is characterised by the `ca:Capability` (e.g. *Bathing*) it refers to, its `generic:name` (e.g. *Receives no assistance*), a `generic:description` (e.g. *gets in and out of tub by self if tub is usual means of bathing*) and a `cga:score` to give if the patient being assessed shows that level of capability (e.g. 1).

Short Portable Mental Status Questionnaire (SPMSQ). The *SPMSQ ontology* allows to store the results of a *Short Portable Mental Status Questionnaire* into the Knowledge Base. The Figure 3.9 shows the UML diagram of the ontology. The CQs answered by the SPMSQ ontology are listed in Table 3.6. The prefix `spmsq:` is associated with the namespace `http://www.ontologydesignpatterns.org/ont/mario/spmsq.owl#`.

An individual of type `spmsq:ShortPortableMentalStatusQuestionnaire` is created when the questionnaire for assessing the mental status of a patient is terminated. This instance provides the total result of the SPMSQ (i.e. `cga:score`) scored by the assessed Patient. Each `cga:Answer` of the Patient being assessed is associated with a `spmsq:AnswerAssessment` providing the evaluation of that answer. `spmsq:ShortPortableMentalStatusQuestionnaire`

CQ1	Which are the capabilities needed to assess the ADL/IADL?
CQ2	Which is the ADL/IADL question used to assess the capability level of a certain patient?
CQ3	Which is the description of a certain capability level?
CQ4	Which is the score in ADL/IADL associated with a certain capability level?
CQ5	Which is the score of a correct answer of a certain question?
CQ6	Which is the total score of the ADL/IADL questionare of a certain patient at a certain time?
CQ7	Which was the level of capability of the activities of daily living (such as, bathing, dressing, toileting, transferring, controlling urination and bowel movement, feeding) of a patient at a certain time?
CQ8	Which was the level of capability of the instrumental activities of the daily living (such as, using a telephone, shopping, preparing food, housekeeping, laundering, getting means of transportation, having responsibility of own medications, managing finances) of a patient at a certain time?

Table 3.5: Competency questions answered through the Capability Assessment ontology.

CQ1	Which is the total score of the SPMSQ at a certain time?
CQ2	Which is the score associated with an answer of the SPMSQ?

Table 3.6: Competency questions answered through the SPMSQ ontology.

aggregates all the `spmsq:AnswerAssessment` and provides the sum scored answering to the individual questions. An example of usage of the SPMSQ ontology is shown in the Frame 3.1.

Exton-Smith Scale (ESS). The *ESS ontology* allows to store into the KB the results of an evaluation of a Patient through the Exton-Smith Scale. The UML class diagram of the ESS ontology is shown in Figure 3.10. The CQs answered through the ESS ontology are listed in Figure 3.7. The namespace prefix `ess:` is associated with the value `http://www.ontologydesignpatterns.org/ont/mario/ess.owl#`.

The *Exton-Smith Scale* aims at evaluating the pressure sores risk of a Patient. `ess:Exton-SmithScaleAssessment` represents an execution of this assessment. It associates the Patient that is being assessed with a `cga:score` and with the observed patient conditions that induced to this score. Moreover, `ess:Exton-SmithScaleAssessment` associates (through `ess:hasPressureSoresRisk`) a Patient with a `ess:PressureSoresRisk`. An individual of type `ess:PressureSoresRisk` represents a level of pressure sores risk on the ESS scale (e.g. "*Score 16-20: minimum risk*"). A `ess:PressureSoresRisk` is characterised by a `generic:name` (e.g. "*minimum risk*") and the interval (i.e. `cga:scoreMin` and `cga:scoreMax`) of values associated with the pressure sores risk (e.g. *16-20*). The object property `ess:hasCondition` is used to associate an `ess:Exton-SmithScaleAssessment` with the current `ess:PatientCondition` observed in the assessed Patient. The ontology defines five different types of patient condition that are evaluated by the ESS: (i) the `ess:PatientGeneralCondition`; (ii) the

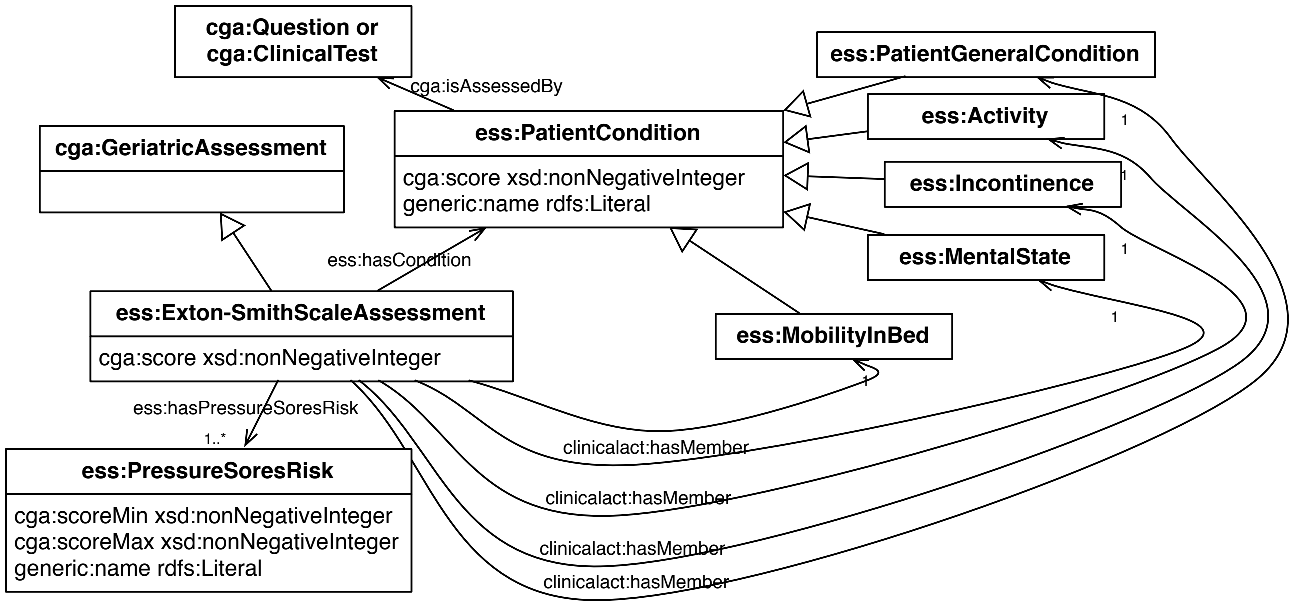

cga:Question or cga:ClinicalTest
cga:isAssessedBy
ess:PatientGeneralCondition
cga:GeriatricAssessment
ess:PatientCondition
cga:score xsd:nonNegativeInteger
generic:name rdfs:Literal
ess:Activity
ess:Incontinence
ess:MentalState
ess:hasCondition
ess:Exton-SmithScaleAssessment
cga:score xsd:nonNegativeInteger
ess:MobilityInBed
ess:hasPressureSoresRisk
1..*
ess:PressureSoresRisk
cga:scoreMin xsd:nonNegativeInteger
cga:scoreMax xsd:nonNegativeInteger
generic:name rdfs:Literal
clinicalact:hasMember
clinicalact:hasMember
clinicalact:hasMember
clinicalact:hasMember
1
1
1
1
1

Figure 3.10: The UML class diagram of the ESS ontology.

`ess:MentalState`; (iii) the `ess:Activity`; (iv) the `ess:Incontinence`; and (v) the `ess:MobilityInBed`.

Each `ess:PatientCondtion` has a `generic:name` (e.g. "*Doubly incontinent*") and a `cga:score` (e.g. 1) that can contribute to the result of the `ess:Exton-SmithScaleAssessment`.

Cumulative Illness Rating Scale (CIRS). The *CIRS ontology* allows to store the results of an evaluation of a patient with the *Cumulative Illness Rating Scale*. The UML class diagram of the CIRS ontology is shown in Figure 3.11. The CQs answered through the ESS ontology are listed in Figure 3.8. The namespace prefix `ess:` is associated with the value `http://www.ontologydesignpatterns.org/ont/mario/cirs.owl#`. The ontology aims at evaluating the illness severity of the patient's biological system (e.g. *cardiovascular system*, *respiratory system* etc.). Individuals of the class `cirs:BiologicalSystem` are characterised by a `generic:name` (e.g.*respiratory system*) and a `generic:description` (e.g. *lungs, bronchi, trachea*). `cirs:BiologicalSystemAssessment` represents an action aiming at assessing a `cirs:BiologicalSystem`. `cirs:BiologicalSystemAssessment` assigns at a certain time a `cirs:CIRSRating` (e.g. *Moderate*,3) to a `cirs:BiologicalSystem`. A `cirs:CIRSRating` has a `generic:name` (e.g. *Moderate*) and a `cga:score` (e.g. 3), which may contribute to the illness severity score and to the comorbidity index. An instance of `cirs:CIRSAssessment` evaluates the illness severity score and the comorbidity index of a Patient at certain time. It indicates (through `clinicalact:hasMember`) the `cirs:BiologicalSystemAssessment` used to compute these two scores.

Mini Nutritional Assessment (MNA). The *Mini Nutritional Assessment* (MNA) aims at evaluating the nutritional status of a patient at a certain time. The *MNA ontology* allows to

CQ1	How was the condition of patient at a certain time? Was it Bad, Poor, Fair or Good?
CQ2	How was the mental state of a patient at a certain time? Was it Stuporosous, Confused, Apathetic or Alert?
CQ3	Which is the range of ESS scores associated with the high/medium/low risk of sores?
CQ4	Which is the score associated with a certain patient activity level/condition/mental state/incontinence level/mobility?
CQ5	Which was the patient's activity level at a certain time? Did s/he stay in the bed all the day? Did s/he need of a chairfast? Did s/he walk with help? Or, was s/he ambulant?
CQ6	Which was the patient's incontinence level at a certain time? Was s/he double incontinent? Was s/he usually incontinent of urine? Was s/he occasionally incontinent? Or, wasn't s/he incontinent?
CQ7	Which was the patient's mobility in bed at a certain time? Was s/he immobile, very limited, slightly limited, or full?
CQ8	Which was the patient's score at the exton-smith scale (ESS) at a certain time?

Table 3.7: Competency questions answered through the ESS ontology.

CQ1	Which is the name of a rating in the CUMULATIVE ILLNESS RATING SCALE (C.I.R.S.)?
CQ2	Which was the patient's illness rating for a biological system at a certain time?
CQ3	Which was the patient's COMORBIDITY INDEX (CIRS-CI) at a certain time?
CQ4	Which was the patient's ILLNESS SEVERITY SCORE (CIRS-IS) at a certain time?

Table 3.8: Competency questions answered through the CIRS ontology.

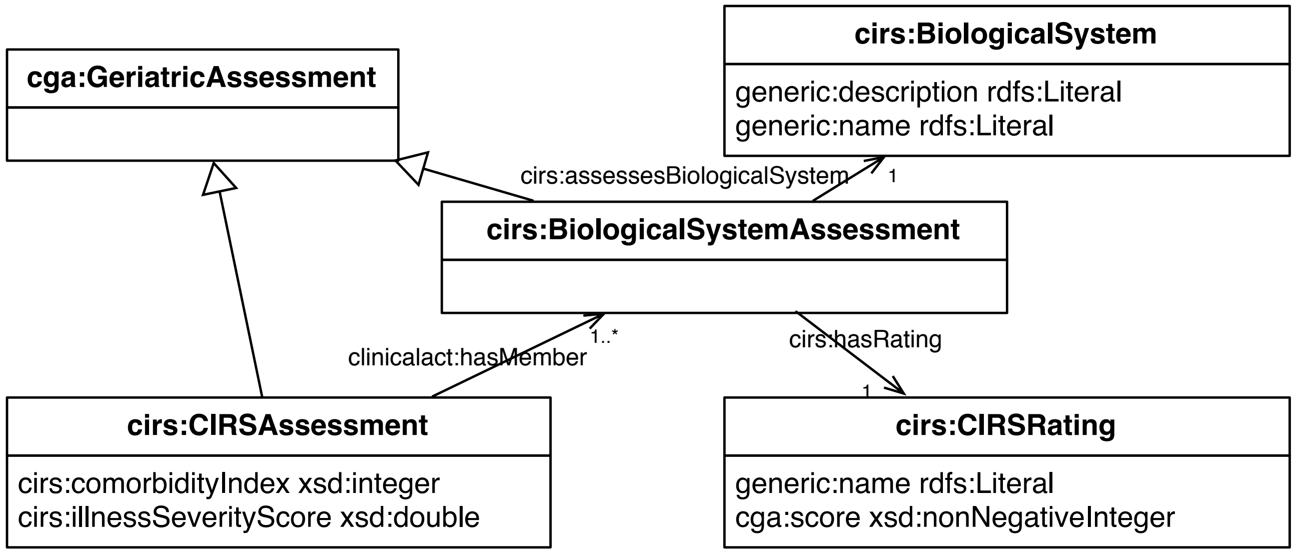

cirs:BiologicalSystem
generic:description rdfs:Literal
generic:name rdfs:Literal
cga:GeriatricAssessment
cirs:assessesBiologicalSystem
1
cirs:BiologicalSystemAssessment
1..*
clinicalact:hasMember
cirs:hasRating
1
cirs:CIRSAssessment
cirs:comorbidityIndex xsd:integer
cirs:illnessSeverityScore xsd:double
cirs:CIRSRating
generic:name rdfs:Literal
cga:score xsd:nonNegativeInteger

Figure 3.11: The UML class diagram of the CIRS ontology.

store the results of the Mini Nutritional Assessment into the Knowledge Base. The UML class diagram of the MNA ontology is shown in Figure 3.12. Refer to the OWL file for the CQs. The namespace prefix `mna:` is associated with the value `http://www.ontologydesignpatterns.org/ont/mario/mna.owl#`. The result of a Mini Nutritional Assessment is represented by `mna:MiniNutritionalAssessment` that associates a certain Patient with the score (i.e. `cga:score`) of the assessment and the resulting Malnutrition Indicator Score (represented by `mna:MNARating`). An `mna:MNARating` has a `generic:name` (e.g. *well-nourished*) and a quantitative value indicating the range of MNA score it refers to (e.g. a Patient is considered well-nourished if her score in the MNA is greater than 24). A `mna:MiniNutritionalAssessment` is made on the basis of other four assessments: (i) the `mna:AnthropometricAssessment`; (ii) the `mna:GeneralAssessment`; (iii) the `mna:DietaryAssessment`; and (iv) the `mna:SelfAssessment`. Each of these assessment is made evaluating other assessments. For instance, the `mna:AnthropometricAssessment` is made on the basis of the `mna:BMIAsssessment`, the `mna:CalfCircumferenceAssessment`, the `mna:MACAssessment` and the `mna:WeightLossAssessment`. Each assessment defines its rating scale, e.g. the rating scale of `mna:BMIAsssessment` is `mna:BMIRating`.

3.3.3 Tagging ontology

Tagging has become a key feature of the todays social media. A tag is a label (precisely, a free-word keyword) that is attached to someone or something for many purposes: identifying, categorising, commenting, voting, reacting etc.

The aim of the tagging ontology is to represent a tagging action, i.e the action performed by an agent that attaches a label or something with a well-defined semantics (eg. a concept or a frame etc.) to some entity. In the context of the MARIO project tags will be associated

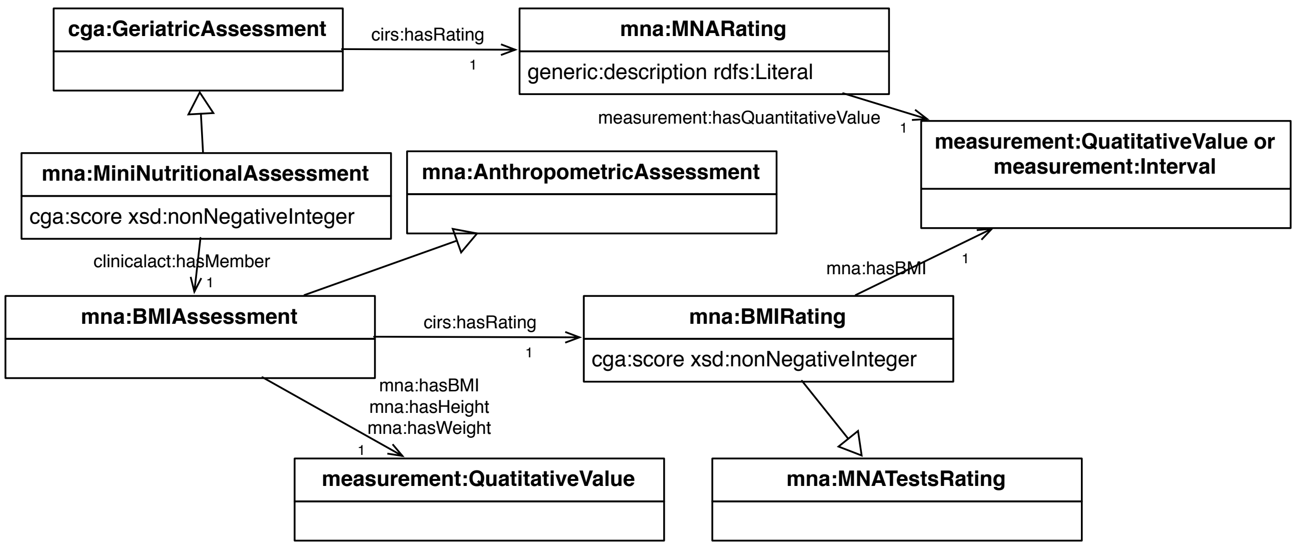

cga:GeriatricAssessment
cirs:hasRating
1
mna:MNARating
generic:description rdfs:Literal
measurement:hasQuantitativeValue
1
measurement:QuatitativeValue or measurement:Interval
mna:MiniNutritionalAssessment
cga:score xsd:nonNegativeInteger
mna:AnthropometricAssessment
clinicalact:hasMember
1
mna:hasBMI
1
mna:BMIAssessment
cirs:hasRating
1
mna:BMIRating
cga:score xsd:nonNegativeInteger
mna:hasBMI
mna:hasHeight
mna:hasWeight
1
measurement:QuatitativeValue
mna:MNATestsRating

Figure 3.12: The UML class diagram of the MNA ontology.

with multimedia contents (e.g. photos, videos, audios and so on.), events (e.g. festivals, birthdays, daily events etc.) or personal memories (which can be considered as a particular kind of events in a person's life). Hence, the tagging ontology provides a reference model for representing tags in MARIO and can be used for interacting with patients in a variety of tasks that requires remembrances, e.g., stimulating the patient's memory, entertaining the patient with multimedia contents associated with particular moments of her life, etc.

State of the art

Several ontologies [35, 31, 34] and Ontology Design Patterns[19] have been proposed to conceptualise the tagging action so far. [29] surveyed the state of the art in the tagging ontologies. In the most of them the tagging concept is represented as reified n-ary relationship between the tagger (the agent who gives the tag), the tag (often a keyword taken from a folksonomy), the entity tagged, and the date when the action happened. Some of them tried to associate the tag with its meaning [35], to express the polarity of the tagging [19] [31] or to attempt to treat tagging as a vote [20].

Ontology description

The ontology we propose does not deviate from the state of the art significantly. However, our ontology allows MARIO to use not only simple free-word keywords, but also individuals with a well defined semantics or even complex named graphs. More in general, the designed ontology allows to answer the competency questions reported on Table 3.9.

[19]Aldo Gangemi, http://ontologydesignpatterns.org/cp/owl/tagging.owl
[20]http://info.slis.indiana.edu/ dingying/uto.owl

CQ1	*Which is the tag associated with a certain photo or events?*
CQ2	*Who has given a tag to a certain entity?*
CQ3	*Which is the entity associated to a certain tag?*

Table 3.9: Competency questions answered through the Tagging ontology.

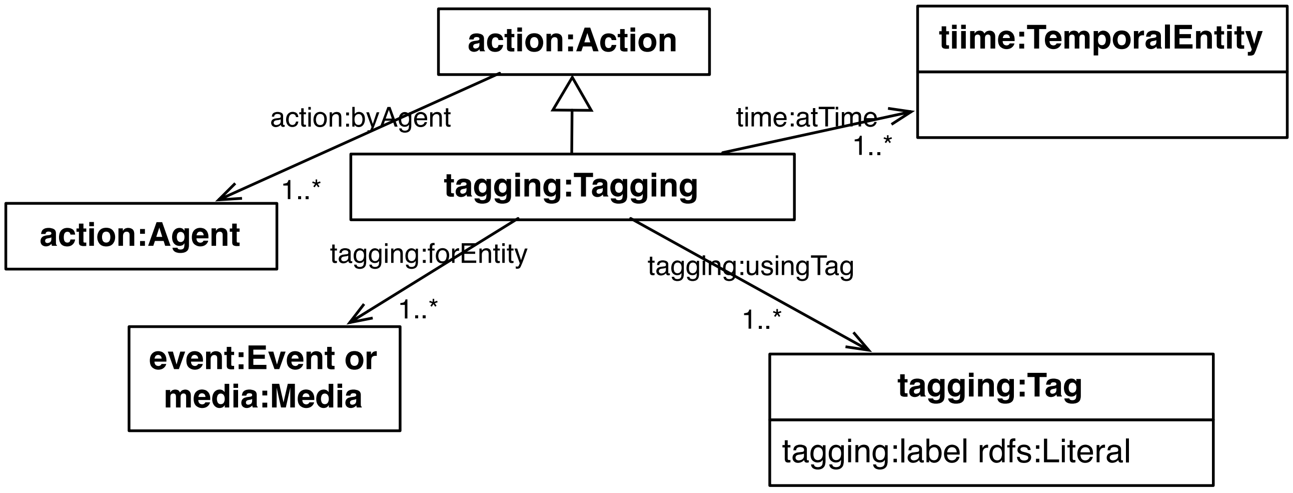


Figure 3.13: The UML class diagram of tagging ontology.

The figure 3.13 shows the ontology diagram. The class `tagging:Tagging`[21] is a reification of the relationship between the agent who made the tag, the tag itself and the entity tagged. It can be also viewed as an action whose agent is the tagger and the patient is the entity tagged. An individual of `tagging:Tagging` has to define *at least* an agent that performs the action (`action:byAgent` property), an entity target for the tagging action (`action:forEntity` property), the tag used (`tagging:usingTag` property) and the date time when this action is performed (`time:atTime` property).

The class `tagging:Tag` represents any object that can be used to identify, categorize, describe or comment the entity being tagged. Being object of a the predicate `tagging:hasTag` implies to be a Tag, therefore a richer description such as a frame, a named graph, or a FRED [23] graph can be used as tag for an entity. Furthermore, the datatype property "`rdfs:label`" can be used to associate an individual of Tag with the text it represents.

Currently, an individual of `tagging:Tagging` can be connected either to an individual of the class `event:Event` or to an individual of the class `media:Media` (such as photos, videos, audios etc.). A multimedia content (i.e., an individual of the class `media:Media`) can be associated with an event by the object property `event:hasEvent`, e.g. a video taken at a birthday party. Finally, we defined a property chain to ensure that the tags of an event are inherited by all multimedia content connected to it, e.g. if the birthday party has the tag "Birthday", then the video associated to it inherits the tag "Birthday".

[21]The prefix `tagging` is associated with the namespace `http://www.ontologydesignpatterns.org/ont/mario/tagging.owl`. Refer to Table 3.10 for the namespaces of the imported ontology modules.

Namespace prefix	Namespace
action	http://www.ontologydesignpatterns.org/ont/mario/action.owl
time	http://www.ontologydesignpatterns.org/ont/mario/time.owl
event	http://www.ontologydesignpatterns.org/ont/mario/event.owl
media	http://www.ontologydesignpatterns.org/ont/mario/media.owl

Table 3.10: Ontology modules imported/reused by the Tagging ontology.

```
:January_20_1971 a time:TemporalEntity .

:John_51st_birthday a event:Event ;
   spatial:hasPlace :Piper_Club ;
   time:atTime :January_20_1971 .

:John_picture a media:Image .

:Piper_Club a spatial:SpatialThing .

:tag_John_51st_birthday a tagging:Tag .

:tagging_John_51st_birthday a tagging:Tagging ;
  tagging:forEntity :John_51st_birthday , :John_picture;
  tagging:usingTag :tag_John_51st_birthday .
```

Frame 3.2: An example of usage of the tagging ontology serialized in TURTLE language

Example

MARIO might use the tagging ontology for annotating a picture about the birthday of his patient John. This picture was takes at John's 51st birthday on January 20 1971. Thus, the tagging ontology can be used for tagging the picture with this knowledge. Frame 3.2 shows the RDF graph resulting from the usage of tagging ontology for the previous example.

3.3.4 Affordance ontology

In robotics, *behaviour selection* (also called behaviour arbitration) is the process of deciding which action to execute at each point of time. For the sake of simplicity, most implemented systems use a built-in fixed priority ordering of behaviours, i.e. the agent's control strategy is embedded into a collection of preprogrammed condition-action pairs. This strategy, called *purely reactive*, has proven effective for a variety of problems that can be completely specified at design-time [37]. However, it is inflexible at run-time due to its inability to store new information in order to adapt the robot's behaviour on the basis of its experience. Moreover, the burden of predicting all possible input states and choosing the corresponding output actions

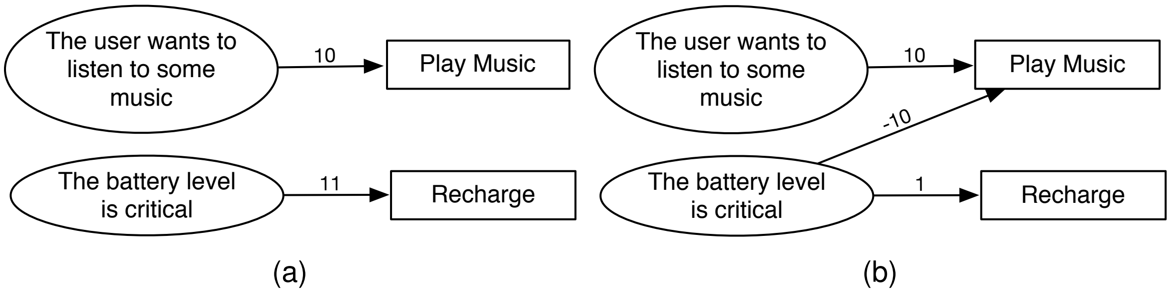

The user wants to listen to some music
10
Play Music
The battery level is critical
11
Recharge
(a)
The user wants to listen to some music
10
Play Music
-10
The battery level is critical
1
Recharge
(b)

Figure 3.14: Two equivalent action-selection schemes.

is completely left to the designer.

Behaviour-based approaches to action selection can be considered as an extension of purely reactive strategy. These approaches are related to the concept of *affordance*. The notion of affordance has been introduced by Gibson [24] who devised a theory of how animals perceive opportunities for action. Gibson called these opportunity *affordance*. He suggested that the environment offers the agents (people or animals) opportunities for action. For instance, a door can have the affordance of “openability”. These action opportunities are latent in the environment and independent from individual’s ability to recognize them, but affordances are always dependent on agent’s capability. For example, to a thief an open window can have an affordance of “climbing”, but not so to an infant who is not tall enough to reach the window.

MARIO uses a behaviour-based approach to action selection which relies on both the notion of affordance devised by Gibson [24] and the proposal of Pattie Maes [36]. The overall behavioural capacity of MARIO is decomposed into a set of behaviours (or *Tasks*). Each behaviour is associated with states of the world (that we call *situation*) in which the task could be activated together with a so-called *affordance strength*. The affordance strength indicates how much a given behaviour is relevant for a certain situation. The behaviour selection mechanism is implemented by the MARIO’s *Task Manager*. This component is responsible for detecting the active situations and consequently activating the Task having the highest activation level. The activation level of a task is a dynamic value computed by summing the affordance strengths of the active situation associated with the task.

The classical notion of affordance suggests that the physical objects (e.g. a door) offer the opportunity of performing an action (e.g. open). We claim that, not only in physical objects, but also *complex situations* (e.g. the user want to listen to some music) afford actions (e.g. play music). A complex situation can be seen as the fullfilment at a certain time of certain conditions. These conditions may involve: temporal aspects (e.g. lunchtime may afford the task *remember the user to take the pills*), the perception of certain physical objects, the receiving of a command (e.g. *I want to listen to some music*), or, even the existence of certain state-of-affairs (e.g. the situation *the user is sitting on a chair for a long while* may afford the task *entertain the user*).

Consider the action-selection scheme shown in Figure 3.14a. The situations are represented by ovals, whereas tasks are rectangles. In this model only two situations are relevant for choosing the task to execute. The first situation is active when *the user wants to listen to some music*. This situation could be activated either when MARIO receives a vocal command (e.g. “Mario, I want to listen to some music”) or when the user presses the button “Music” on the touchscreen. The second situation is activated when the battery level becomes critical and the robot needs to recharge. Intuitively, when the battery level is critical the robot should stop any other activity and go to the recharge station. With an action selection scheme this behaviour could be achieved by giving to the task *Recharge* the greatest affordance strength. For instance, a possible lifecycle of the system could be:

time^0: The system starts. There is no active situation. The activation level for both the tasks is 0.

time^1: The user touches the button *Music* on the touchscreen and the task manager activates the situation: “The user wants to listen to some music”. The activation level of the the task *Play Music* is 10, whereas the activation level of the task *Recharge* is 0. The task manager starts the task *Play Music*.

time$^{2...9}$: The situation “The user wants to listen to some music” is still active, and the Task manager keeps active the task *Play Music*.

time^{10}: The task *Play Music* is still active. The battery level becomes critical. The task manager activates the situation “The battery level is critical”. The activation level of the the task *Play Music* is 10, whereas the activation level of the task *Recharge* is 11. The task manager stops *Play Music* and starts *Recharge*.

It is worth noticing that the same behaviour could be achieved by an infinite number of action selection schemes, i.e. any other scheme in which the activation level of *Recharge* is always[22] greater than *Play Music*. For instance, the action-selection scheme in Figure 3.14b induces the same behaviour of that in Figure 3.14a.

Remark.

For explanatory reasons, the above discussion made the implicit assumptions that the robot can execute only one task at a time and it chooses the task with the highest activation level. The strategy for choosing the task to start/stop depends on the actual implementation of the task manager. The task manager could realize more sophisticated mechanisms overcoming these assumptions. This document aims at proposing a model for supporting the task manager in deciding the behaviour to execute. The definition of the actual strategy for selecting the task to execute is out of the scope of this document.

[22] *Always* means for any set of active situations.

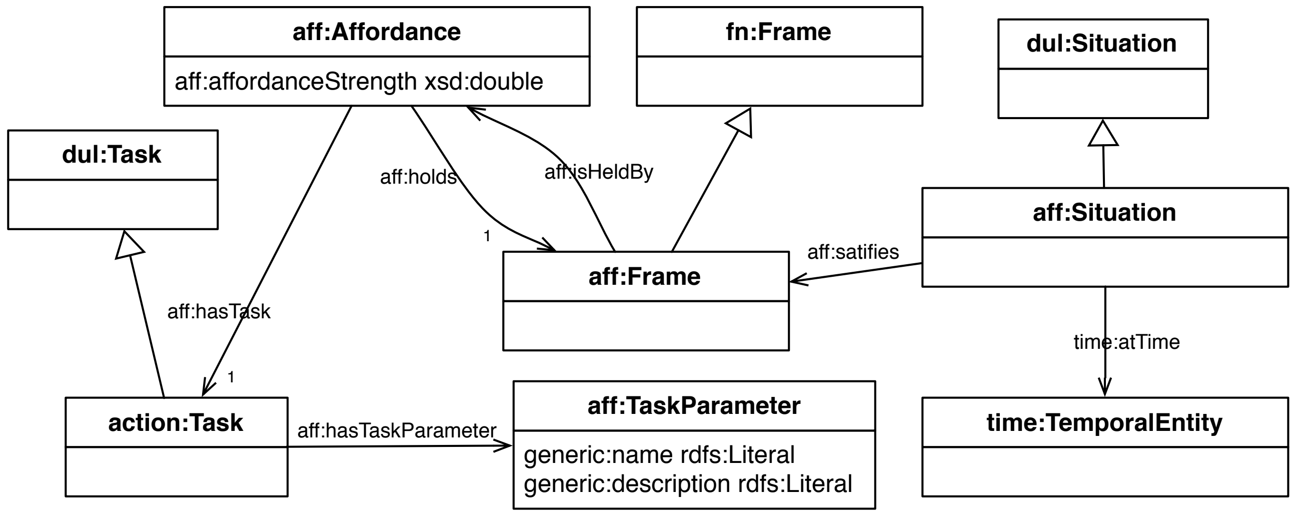


Figure 3.15: The UML class diagram of the Affordance ontology

CQ1	Which is the strength of an Affordance?
CQ2	Which tasks are afforded in a certain situation?
CQ3	How should an agent behave in a certain situation?
CQ4	Which are the parameters involved in certain task?

Table 3.11: Competency questions answered through the Affordance ontology.

State of the art

There exist few examples of ontologies conceptualizing the idea of affordances. In literature, the notion of affordance has been seen as relation between the environment and the agent [54] or as qualities of objects in environment taken with reference to an observer [45, 44, 53]. Our approach is closer to the characterization proposed by Stoffregen [54], albeit we abstract the notion environment to a more general concept of *situation*, i.e. a situation embeds all the environment's characteristics perceived by the robot and possibly other conditions (e.g. involving time, the receiving of a commands etc.).

Ontology description

The proposed pattern relies on the descriptions and situations ODP[23] [17], combined with a frame-based representation scheme [40]. Figure 3.15 shows the UML class diagram of the ontology.

Table 3.11 reports the competency questions that identify the requirements associated with the modelling of the affordance ontology.

[23]Aldo Gangemi, Description and Situation ODP `http://ontologydesignpatterns.org/wiki/Submissions:DescriptionAndSituation`

Ontology	Namespace prefix	Namespace
Action	`action:`	`http://www.ontologydesignpatterns.org/ont/mario/action.owl`
Time	`time:`	`http://www.ontologydesignpatterns.org/ont/mario/time.owl`
FrameNet	`fn:`	`http://www.ontologydesignpatterns.org/ont/framenet/tbox/`
DOLCE	`dul:`	`http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#`

Table 3.12: Ontology modules imported or re-used by the Affordance ontology.

The ontology modules imported or re-used by the Affordance ontology are those reported in Table 3.12[24].

The name space prefix `aff` is associated with the value `http://www.ontologydesign-patterns.org/ont/mario/affordance.owl#`.

Affordances are represented as individuals of the class `Affordance`, which is modelled as a n-relation connecting:

- a class of situations that represents states of the world (i.e., any individual of the class `Frame`). This relation is expressed the object property `holds`;
- an agent's behaviour (aka a task), which is any individual of the class `action:Task`. This relation is expressed by the object property `hasTask`.
- a quantity that indicates how much a a behaviour is relevant for the the occurrence of a certain frame. This relation is expressed by the datatype property `affordanceStrength`, whose range is `xsd:double`.

According to [21], the intended meaning of a frame (represented in our ontology by the class `Frame`) can be summarised as a small-sized and richly interconnected structure, used to organize our knowledge, as well as to interpret, process or anticipate information. Frames identify classes of situations and have been investigated in linguistics by Fillmore [15], in AI by Minsky [38], and more recenty in the Semantic Web [21, 40]. We modelled the class `Frame` as a sub-class of `fn:Frame`[25], which is a class re-used from the OWL version [40] of FrameNet [2].

Situations are states of the world fulfilling certain conditions. These conditions may involve: temporal aspects, the perception of physical entities, the receiving of a command or the existence of certain state-of-affairs. Following the Description and Situation ODP we made a basic distinction, between a `Frame` (or description) and a `Situation`, which is a frame occurence. The class `Situation` is modelled as sub-class of the class `dul:Situation`[26] that is re-used from DOLCE Ultra-lite [22]. Any individual of `Situation` is modelled as a time indexed situation, i.e., a state of the world anchored to a certain time point (e.g. at 11am

[24]The Framenet ontology can be found at `http://www.ontologydesignpatterns.org/ont/framenet/tbox/schema.owl`

[25]The prefix `fn:` stands for the namespace `http://www.ontologydesignpatterns.org/ont/framenet/tbox/`.

[26]The prefix `dul:` stands for the namespace `http://www.ontologydesignpatterns.org/ont/DUL.owl#`.

the user expresses the willingness to listen to jazz music). We re-used the time-indexed situation ODP[27] for modelling time constraints for situations. Hence, a `Situation` is related to `time:TemporalEntity`[28] that allows to represent the notion of time either as a time interval (i.e., any individual of the class `time:TimeInterval`), which has a start and an end temporal instant, or an instant itself (i.e., any individual of the class `time:Instant`) that is associated with temporal values by means of the datatype property `time:inXSDDataTime` whose range is `xsd:dateTime`[29].

Affordance ontology models agent's behaviours as tasks. Those tasks are represented as individuals of the class `action:Task` that can be parameterised by specific parameters represented as individuals of the class `TaskParameter`. The relations between tasks and task parameters are expressed by the object property `hasParameter`. For example, a certain task "Play music" can be associated with a parameter "Jazz" that specifies the genre of the music to play.

Example

Frame 3.3 shows the RDF graph resulting from the usage of affordance ontology for representing the situation depicted in figure 3.14b. The example describes the status of the knowledge base at the time point 1 of the example used in the section 3.3.4 to introduce the mechanism of affordance. `:UserWantsToListenToSomeMusic` represents the situation where a `:user` request to listen to some music of a particular `:genre`. `:BatteryInCritalLevel` represents the situation where the `:batteryLevel` of a certain `:agent` is critical. `:affordancePlayMusicBatteryCritical`, `:affordancePlayMusicUserWantsToListenToSomeMusic` and `:affordanceRechargeBatteryCritical` represents the three affordance relations depicted in figure 3.14b as arrows. `:sitTime1` represents the situation at time 1. `:actPlayAtTime1` is the action carried out by `:MARIO` to cope with the situation `:sitTime1`.

[27] `http://www.ontologydesignpatterns.org/cp/owl/timeindexedsituation.owl`.

[28] The prefix `time:` stands for the namespace `http://www.ontologydesignpatterns.org/ont/mario/time.owl#`.

[29] The prefix `xsd:` stands for the namespace `http://www.w3.org/2001/XMLSchema#`.

```
:UserWantsToListenToSomeMusic a aff:Frame;
fn:hasFrameElement :genre, :user.

:genre a fn:FrameElement, aff:TaskParameter .

:BatteryInCritalLevel a aff:Frame;
fn:hasFrameElement :batteryLevel, :agent .

:batteryLevel a fn:FrameElement .

:agent a fn:FrameElement .

:user a fn:FrameElement .

:PlayMusic a action:Task;
aff:hasParameter :genre .

:Recharge a aff:Task.

:affordancePlayMusicBatteryCritical a aff:Affodance ;
aff:affordanceStrength "-10"^^xsd:double ;
aff:holds :BatteryInCritalLevel ;
aff:hasTask :PlayMusic  .

:affordancePlayMusicUserWantsToListenToSomeMusic a aff:Affodance ;
aff:affordanceStrength "10"^^xsd:double ;
aff:holds :UserWantsToListenToSomeMusic ;
aff:hasTask :PlayMusic  .

:affordanceRechargeBatteryCritical a aff:Affodance ;
aff:affordanceStrength "1"^^xsd:double ;
aff:holds :BatteryInCritalLevel ;
aff:hasTask :Recharge  .

:sitTime1 a aff:Situation ;
:user :Freddy ;
:genre :Jazz ;
aff:satisfies :UserWantsToListenToSomeMusic

:actPlayAtTime1 a action:Action ;
action:byAgent :MARIO ;
action:executes :PlayMusic.
```

Frame 3.3: An example of usage of the affordance ontology serialized in TURTLE language

Mario Knowledge Management System

MARIO is an intelligent robot that is able to address a variety of knowledge-intensive tasks required to provide efficient support to patients affected by dementia. Examples of knowledge-intensive tasks are the Comprehensive Geriatric Assessment (CGA), the triggering of specific entertainment activities according to as much specific patient's needs or emotional statuses (e.g., MARIO decides to play the favorite patient's song if it recognises that the patient is sad), the behavioural tasks (e.g., take the patient to the bathroom), the multi-modal interaction with the patient that includes understanding and speech capabilities, etc. Enabling these tasks means to provide MARIO with mechanisms for organising, accessing, storing and interacting with knowledge. These mechanisms should be accessible by any software components part of the MARIO architecture that actually implements intelligent tasks. Hence, the Knowledge Management System (KMS) is the framework that provides MARIO with such capabilities.

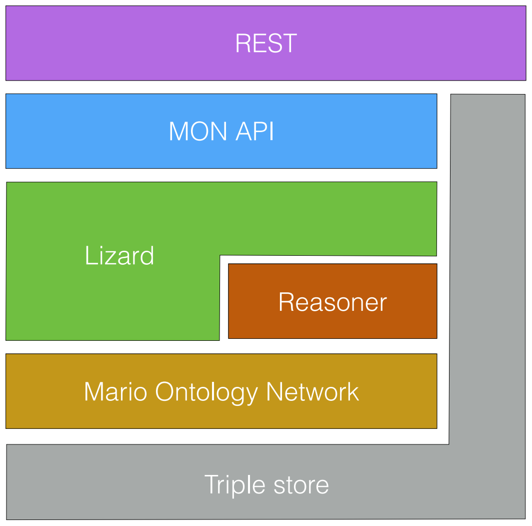


Figure 4.1: Informal architecture of the Mario Knowledge Management System.

Figure 4.1 shows an informal graphical representation of the KMS. The KMS is composed by the following parts:

- a triple store that serves as physical storage for the knowledge managed by MARIO.

The knowledge is expressed by using RDF as reference standard.

- an ontology network (i.e., MON described in Chapter 3) that organises the knowledge as a set of interlinked ontologies;
- a reasoner that provides reasoning capabilities to the KMS. These reasoning capabilities allows MARIO to infer new knowledge by using the axiomatisations provided by the ontologies of the MON and both knolwedge available from the triple store and that coming as streams from external devices (e.g., sensors, kinect) and software components;
- a modular set of software components, called Lizard, that is designed to provide a framework that autamatically generates a middleware API between the ontology network and the external systems part of the MARIO architecture. Those external systems are meant to interact with the ontology network in order to carry on specific knowledge-intensive tasks (e.g., Comprehensive Geriatric Assessment). Hence, Lizard is responsible for enabling the efficient and transparent access to the ontology network. More details can be found in Section 4.1;
- a modular set of software components that are generated and managed by Lizard dynamically. These software components implement the MON Application Programming Interface (API) and provide programmatic access to the knowledge stored in the triple store and organised according to the MON;
- a REST layer that enable to access MON API via HTTP requests.

4.1 Lizard

In recent years a number of frameworks have been proposed to foster the adoption of Semantic Web (SW) technologies in software development. Examples are Apache Jena[1] [11], the OWL API[2] [27] or rdf4j[3] (previously known as Sesame [10]). These frameworks provide the basic building blocks on top of which it is possible to design and implement complex systems that rely on RDF, OWL and SPARQL. However, they require developers with extensive knowledge of SW technologies and Knowledge Engineering. This knowledge is mandatory in order to use these frameworks effectively. Hence, we designed a software architecture that tackles the challenging goal of easing the software development of knowledge-aware systems by filling the gap between SW technologies and architectural and programming styles that are more familiar to software engineers and developers. Addressing such a goal is relevant to MARIO as it provides the robotic framework with a Knowledge Management System (KMS) that hides to other components the technicalities associated with the way it organises, represents, stores and exchanges knowledge.

[1] `https://jena.apache.org/`
[2] `http://owlapi.sourceforge.net/`
[3] `http://rdf4j.org/`

Our framework is called Lizard and is as a component of the Mario Knowledge Management System. Lizard is an Object-RDF mapper, such as SuRF[4] or ActiveRDF[5] [43]. An Object-RDF mapper is a system that exposes the RDF triple sets as sets of resources and seamlessly integrates them into the Object Oriented paradigm. However, differently from existing systems Lizard provides a RESTful layer that exposes Object Oriented paradigm by using the REST architectural style over HTTP. This is an added value in our scenario as the software parts of MARIO are developed in different languages that communicate via HTTP. Additionally, Lizard embeds an Access Control Management System (ACMS) that enables the setup of specific access policies in order to guarantee the access (either in read or write mode) to specific knowledge areas only to a set of allowed entities/systems. For example, an application that performs some entertaiment activity (e.g. play music) would not be allowed to access knowledge about the Continuous Geriatric Assessment (CGA). Hence, the ACMS allows Lizard to deal with some important data management aspects regarding data access, security and data privacy.

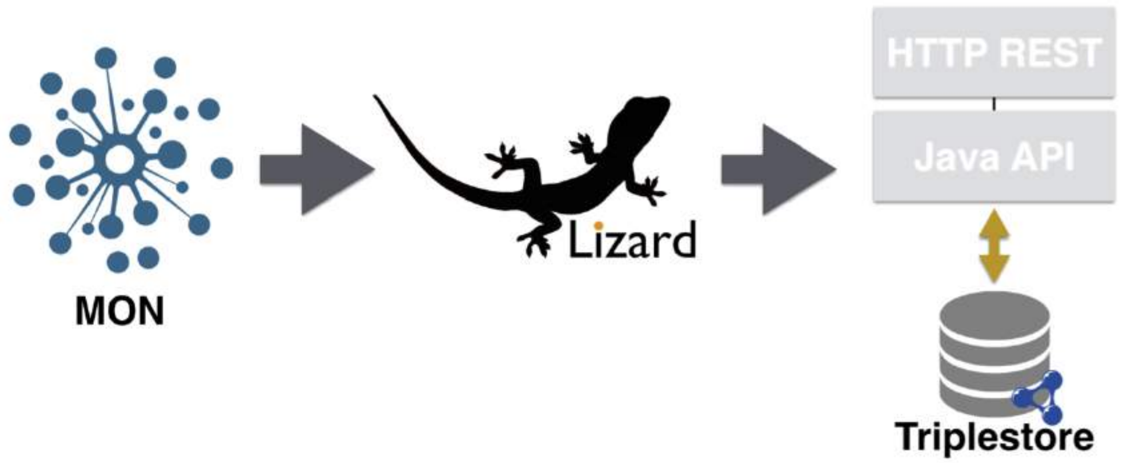


Figure 4.2: High-level diagram presenting the intuition behind Lizard.

Figure 4.2 provides the reader with an high level diagram that informally introduces the basic intuition behind Lizard. Basically, Lizard dinamically generates Java and HTTP REST API from the MARIO Ontology Network (MON). Those API reflects the semantics of MON and allows transparent access to the knowledge base. This means that a client application (i.e., any component of the MARIO robotic framework) can access the knowledge base without any prior knowledge of the ontologies used within the KMS. For example, this avoids client applications to deal with OWL and RDF or to interact with a knowledge base by means of SPARQL queries.

4.1.1 Requirements

The main requirements of Lizard are the following:

[4] `https://pythonhosted.org/SuRF/`

[5] `https://github.com/ActiveRDF/ActiveRDF`

R1. **Generation of procedural API modelled from an OWL ontology**. Lizard has to produce procedural API starting from an OWL ontology. This requirements allows Lizard to generate Java API that follows the semantics expressed by a source ontology. Hence, Lizard allows the Knowledge Base Management System to bind software API to ontological artifacts.

R2. **Generation of HTTP REST API modelled from an OWL ontology**. Lizard has to produce HTTP REST API starting from an OWL ontology. This requirements allows Lizard to generate a REST API that follows the semantics expressed by a source ontology. Hence, such an ontology can be accessed as a service by external components via HTTP requests based on commons GET, POST, PUT and update. Providing access to the MON via HTTP REST is a central requirement as REST over HTTP has been chosen as the reference architectural style in MARIO. With respect to this Lizard allows the Knowledge Base Management System to bind HTTP REST API to ontological artifacts.

R3. **Dynamic adaptivity of produced API to the MARIO Ontology Network (MON)**. Lizard has to adapt the API it produces and exposes (both Java and REST) according to any change occurring in any part of the MON. By change we mean any addition, deletion or update of the MON ranging from a single axiom to a whole ontology. This requirement is crucial in order to keep valid at runtime the binding between the API produced by Lizard and the ontologies connected by the MON.

R2. **programmatic access to the Knowledge Base of MARIO via Java**. The Java API produced by Lizard has to grant access to the Knowledge Base (KB) to other compoents of the MARIO architecture. The KB consists of a set of datasets expressed as RDF triples, which in turn are modelled according to the semantics expressed by the ontologies of the MON and stored in a triplestore (e.g., Virtuoso[6] or Apache Jena TDB[7]). The access via Java allows MARIO to perform operationg such as: query the KB, fetch specific piece of knowledge expressed as RDF, update the KB, and delete RDF from the KB.

R3. **programmatic access to the Knowledge Base of MARIO HTTP REST**. The REST API produced by Lizard has to grant access to the Knowledge Base (KB) to other compoents of the MARIO architecture. The access via REST has to reflect all the operations enabled via Java, i.e., query the KB, fetch specific piece of knowledge expressed as RDF, update the KB, and delete RDF from the KB.

R4. **transparent access (with respect to client application) to the MON**. Other components of MARIO has to be unaware about how the Knowledge Base Management System (KBMS) models the ontologies of the MON and stores RDF triples in the KB. This requirement allows the KBMS to hide ontology design choices and spefic implementation details to other MARIO components that want to interact with knowledge.

[6] `http://virtuoso.openlinksw.com/`.

[7] `https://jena.apache.org/documentation/tdb/`.

4.1.2 Modules and use cases

The requirements presented in Section 4.1.1 are addressed by different modules of Lizard. In fact, Lizard is designed as a modular software architectural that is composed of the following modules:

- Lizard core that provides the KBMS with core functionalities. These core functionalities include the creation and updated of API from MON;
- Lizard API Manager that is responsible of the life-cycle management of created API. Namely, this modules can activate, stop and resume any module that represents API generated by Lizard core;
- Ontology API that is any module identified by the API generated by Lizard core and managed by the Lizard API Manager.

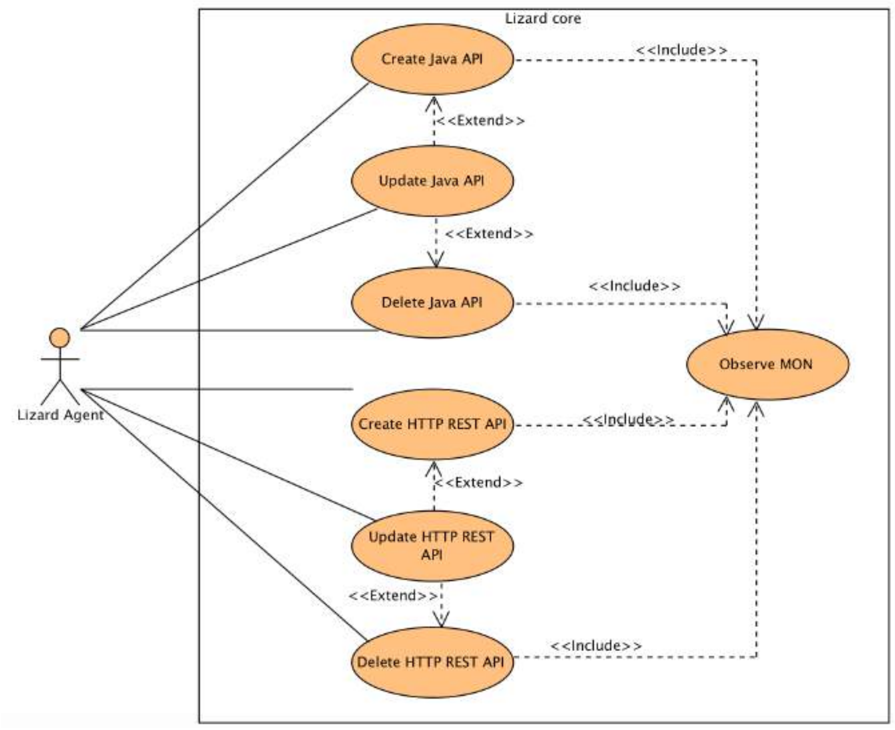


Figure 4.3: UML diagram representing the use cases associated with Lizard core.

Figure 4.3 reports the use cases associated with the Lizard core module. These use cases are graphically represented by using the UML notation, i.e. the use case diagram. The following tables provide detailed descriptions about these use cases.

Use Case ID	UC-1
Title	Create Java API

	UC-1
Description	Lizard core creates a Java API from an OWL ontology of the MON. The API reflects the semantics of the ontology and can be used to access the KB programmatically.
Parent	
Includes	UC-3
Actor(s)	Lizard Agent
Goal	To create a Java API.
Trigger	An ontology is added to the MON.
Preconditions	
Minimal Post conditions	The API has been generated.
Success Post conditions	The API can be loaded into the KBMS.

Use Case ID	UC-2
Title	Create HTTP REST API
Description	Lizard core creates a HTTP REST API from an OWL ontology of the MON. The API reflects the semantics of the ontology and can be used to access the KB via HTTP requests.
Parent	
Includes	UC-3
Actor(s)	Lizard Agent
Goal	To create a HTTP REST API.
Trigger	An ontology is added to the MON.
Preconditions	The Java API have been created.
Minimal Post conditions	The HTTP API are generated.
Success Post conditions	The HTTP API can be loaded into the KBMS.

Use Case ID	UC-3
Title	Observe MON
Description	Lizard core constantly observes the MON in order to trigger an appropriate action corresponding to a MON's change.
Parent	
Includes	
Actor(s)	Lizard core
Goal	To observe changes occurring to the MON.
Trigger	Some change occurs in the MON.

	UC-3
Preconditions	
Minimal Post conditions	The change is detected.
Success Post conditions	An event associated with a detected change is propagated.

Use Case ID	UC-4
Title	Delete Java API
Description	Lizard core deletes a Java API if it is not required anymore.
Parent	
Includes	UC-3
Actor(s)	Lizard Agent
Goal	To handle the request of deletion of a JAVA API associated with one of ontology of the MON.
Trigger	Some change occurs in the MON.
Preconditions	
Minimal Post conditions	The deletion request is received and propagated the Lizard API Manager.
Success Post conditions	The Lizard API Manager returns a reply that notifies the successful deletion.

Use Case ID	UC-5
Title	Delete REST API
Description	Lizard core deletes a REST API if it is not required anymore.
Parent	
Includes	UC-3
Actor(s)	Lizard Agent
Goal	To handle the request of deletion of a REST API associated with one of ontology of the MON.
Trigger	Some change occurs in the MON.
Preconditions	
Minimal Post conditions	The deletion request is received and propagated the Lizard API Manager.
Success Post conditions	The Lizard API Manager returns a reply that notifies the successful deletion.

Use Case ID	UC-6
Title	Update Java API
Description	Lizard core update a Java API when a update of the MON is detected. An update is performed in terms of (i) the deletion of the previous version of the API and (ii) creation of the new version of the API.
Parent	UC-1, UC-4
Includes	
Actor(s)	Lizard Agent
Goal	To handle the request of update of a Java API associated with one of ontology of the MON.
Trigger	Some change occurs in the MON.
Preconditions	
Minimal Post conditions	The update request is received and propagated the Lizard API Manager.
Success Post conditions	The Lizard API Manager returns a reply that notifies the successful update.

Use Case ID	UC-7
Title	Update HTTP REST API
Description	Lizard core update a Java API when a update of the MON is detected. An update is performed in terms of (i) the deletion of the previous version of the API and (ii) creation of the new version of the API.
Parent	UC-1, UC-4
Includes	
Actor(s)	Lizard Agent
Goal	To handle the request of update of a Java API associated with one of ontology of the MON.
Trigger	Some change occurs in the MON.
Preconditions	
Minimal Post conditions	The update request is received and propagated the Lizard API Manager.
Success Post conditions	The Lizard API Manager returns a reply that notifies the successful update.

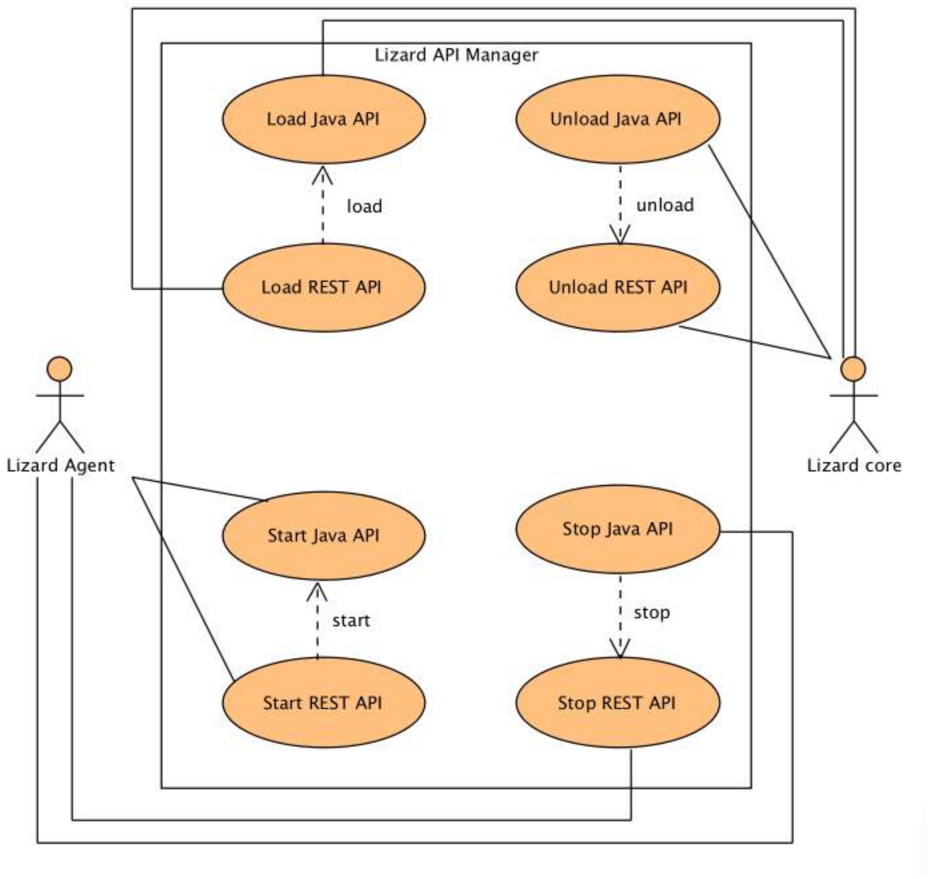


Figure 4.4: UML diagram representing the use cases associated with Lizard API Manager.

Figure 4.4 shows the use case diagram associated with the Lizard API Manager module. These use cases address the life-cycle management of the API generated from ontology artifacts part of the MON. The following tables provide detailed descriptions about these use cases.

Use Case ID	UC-8
Title	Start Java API
Description	Lizard API Manager activates a Java API that has been previously loaded by Lizard core.
Parent	
Includes	
Actor(s)	Lizard Agent
Goal	A Java API is activated and can be used in order to access the knowledge managed by the Knowledge Management System.
Trigger	
Preconditions	The Java API that is object of the activation has been previously created.
Minimal Post conditions	The Java API is activated.

UC-8	
Success Post conditions	The Java API is active and running.

Use Case ID	UC-9
Title	Start REST API
Description	Lizard API Manager activates a REST API that has been previously loaded by Lizard core.
Parent	
Includes	UC-8
Actor(s)	Lizard Agent
Goal	A REST API is activated and can be used in order to access via HTTP the knowledge managed by the Knowledge Management System.
Trigger	
Preconditions	The REST API that is object of the activation has been previously created.
Minimal Post conditions	The REST API is activated.
Success Post conditions	The REST API is active and running. This means that alswo the corresponding Java API has been activated successfully

Use Case ID	UC-10
Title	Stop Java API
Description	Lizard API Manager deactivates a Java API that was active.
Parent	
Includes	UC-11
Actor(s)	Lizard Agent
Goal	A Java API is deactivated and cannot be used anymore to access the knowledge managed by the Knowledge Management System.
Trigger	
Preconditions	The Java API that is object of the deactivation has been previously activated.
Minimal Post conditions	The Java API is deactivated.
Success Post conditions	The Java API is deactivated and cannot be used anymore. If a REST API exists and is active for the same ontology artifact, then also this REST API is deactivated.

Use Case ID	UC-11
Title	Stop REST API
Description	Lizard API Manager deactivates a REST API that was active.
Parent	
Includes	
Actor(s)	Lizard Agent
Goal	A REST API is deactivated and cannot be used anymore to access via HTTP the knowledge managed by the Knowledge Management System.
Trigger	
Preconditions	The REST API that is object of the deactivation has been previously activated.
Minimal Post conditions	The REST API is deactivated.
Success Post conditions	The REST API is deactivated and cannot be used anymore.

Use Case ID	UC-12
Title	Load Java API
Description	A Java API is loaded into the Lizard API Manager. Hence, the Lizard API Manager can begin the management of its life-cycle.
Parent	
Includes	
Actor(s)	Lizard core
Goal	To load a Java API previously created by Lizard core.
Trigger	
Preconditions	The Java API has been previously created.
Minimal Post conditions	The Java API is loaded.
Success Post conditions	The Java API is loaded and its life-cycle can be managed by the Lizard API Manager.

Use Case ID	UC-13
Title	Load REST API

	UC-13
Description	A REST API is loaded into the Lizard API Manager. Hence, the Lizard API Manager can begin the management of its life-cycle.
Parent	
Includes	UC-12
Actor(s)	Lizard core
Goal	To load a REST API previously created by Lizard core.
Trigger	
Preconditions	The REST API has been previously created and the corresponding JAVA API activated.
Minimal Post conditions	The REST API is loaded.
Success Post conditions	The REST API is loaded and its life-cycle can be managed by the Lizard API Manager.

Use Case ID	UC-14
Title	Unload Java API
Description	A Java API is unloaded from the set of APIs managed by the Lizard API Manager. Hence, the Lizard API Manager cannot manage its life-cycle anymore. If the Java API is active, then it is first deactivated. If also a REST API exists, then such a REST API is unloaded.
Parent	
Includes	UC-15
Actor(s)	Lizard core
Goal	To unload a Java API previously loaded.
Trigger	
Preconditions	The Java API has been loded.
Minimal Post conditions	The Java API is unloaded.
Success Post conditions	The Java API is ubloaded and its life-cycle cannot be managed by the Lizard API Manager anymore.

Use Case ID	UC-15
Title	Unload REST API
Description	A REST API is unloaded from the set of APIs managed by the Lizard API Manager. If the REST API is active, then it is first deactivated.
Parent	

UC-15	
Includes	
Actor(s)	Lizard core
Goal	To unload a REST API previously loaded.
Trigger	
Preconditions	The REST API has been previously loaded.
Minimal Post conditions	The REST API is unloaded.
Success Post conditions	The REST API is unloaded and its life-cycle cannot be managed by the Lizard API Manager anymore.

4.2 Mario Ontology Network API

The Mario Ontology Network (MON) API is composed by the set of API whom Lizard dynamically generates. It is meant to address requirement R4, i.e., transparent access (with respect to client application) to the MON. Hence, the MON API is designed to provide software modules that allow client applications to query, retrieve, store, update and delete knowledge from the KMS witouht any knowledge of the MON and the triple store. Figure 4.5 shows the use case diagram of the MON API expressed by using the UML notation.

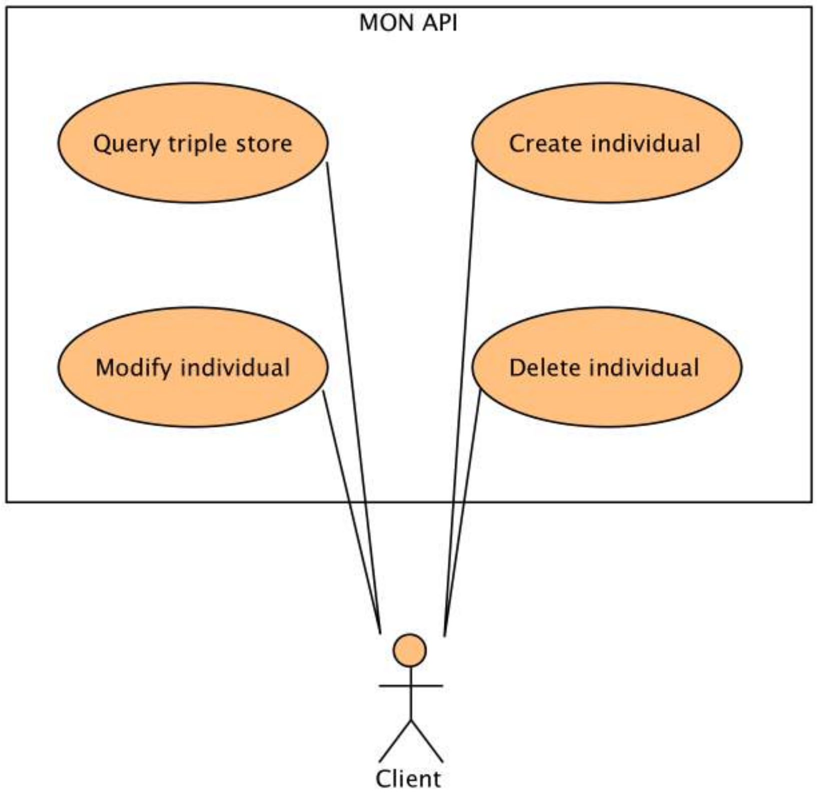


Figure 4.5: UML diagram representing the use cases associated with the MON API.

Use Case ID	UC-16
Title	Query triple store
Description	The specific MON API provides mechanisms that allows to query the kwnoledge available in the triple store according to its formalisation provided by one or more corresponding ontologies of the MON.
Parent	
Includes	
Actor(s)	Client
Goal	To fetch desired knowledge from the triple store.

UC-16	
Trigger	
Preconditions	
Minimal Post conditions	The knowledge is retrieved.
Success Post conditions	The knowledge is returned to the client actor.

Use Case ID	UC-17
Title	Create individual
Description	The specific MON API provides mechanisms to create new individuals that populate the MON.
Parent	
Includes	
Actor(s)	Client
Goal	To populate the MON with new individuals.
Trigger	
Preconditions	
Minimal Post conditions	A new individual can be created in the MON.
Success Post conditions	A new individual is created in the MON.

Use Case ID	UC-18
Title	Delete individual
Description	The specific MON API provides mechanisms to delete existing individuals that populate the MON.
Parent	
Includes	
Actor(s)	Client
Goal	To remove individuals from the MON.
Trigger	
Preconditions	
Minimal Post conditions	An individual can be deleted from the MON.
Success Post conditions	An individual is deleted from the MON.

Use Case ID	UC-19
Title	Update individual
Description	The specific MON API provides mechanisms to update existing individuals that populate the MON.

UC-19

Parent	
Includes	
Actor(s)	Client
Goal	To update individuals of the MON.
Trigger	
Preconditions	
Minimal Post conditions	An individual can be updated from the MON.
Success Post conditions	An individual is updated from the MON.

4.3 Reasoner

The reasoner is the software component of the MARIO KMS that provides methods for executing basic reasoning services. An example of reasoning service that this component should provide is the mechanism for enabling the behaviour selection in MARIO. As previously introduced in Section 3.3.4, behaviour selection (also called behaviour arbitration) is the process of deciding which action to execute at each point of time. We designed an ontology, i.e. the Affordance ontology, that is part of the MARIO Ontology Network and enalbes the representation of the knowledge associated with the notion of affordance as introduced by Gibson [24]. Basically, the affordance tells MARIO what family of situations allows it to perform certain actions. These family of situations (called *frames*) are known a priori by MARIO (e.g. the patient is bored), but the occurrencies of these frames in the real world are not and they should be recognised (e.g. the patient yawns repeatedly at a certain time of the day she is not usually sleepy). The recognition of situations in the real world and their disambiation with respect to a set of known frames is called reconciliation. The reconciliation is a reasoning task. Similarly, the selection of the action to perform when a certain frame is recognised is a reasoning task as well.

Another kind reasoning is dynamic data fusion coming from different sources (e.g. sensors, background knowledge, lingustic system, etc.). The dynamic data fusion would rely on consistency checking as well as rule-based inference to integrate and interpret hetorgeneous knowledge homogeneously in order to avoid inconsistencies with respect to the MARIO Ontology Network.

Currently, the reasoner is under development and we are designing its core on top of the Apache Jena Inference. It will provide the KMS with the the following facilities:

- consistency checking;
- knowledge reconciliation;
- enrichment, i.e. inferring new facts;

- rule-based inference.

We plan to finalise the implementation of the reasoning component by the release of next deliverables.

4.4 Architecture

In this section we provide a formal description of the software modules of the architecture introduces in Figure 4.1 at the beginning of Chapter 4. We make use of the UML notation to formalise the architecture. Namely, Figure 4.6 shows the component diagram describing the software architecture of the KMS. The component diagram presents the following software components:

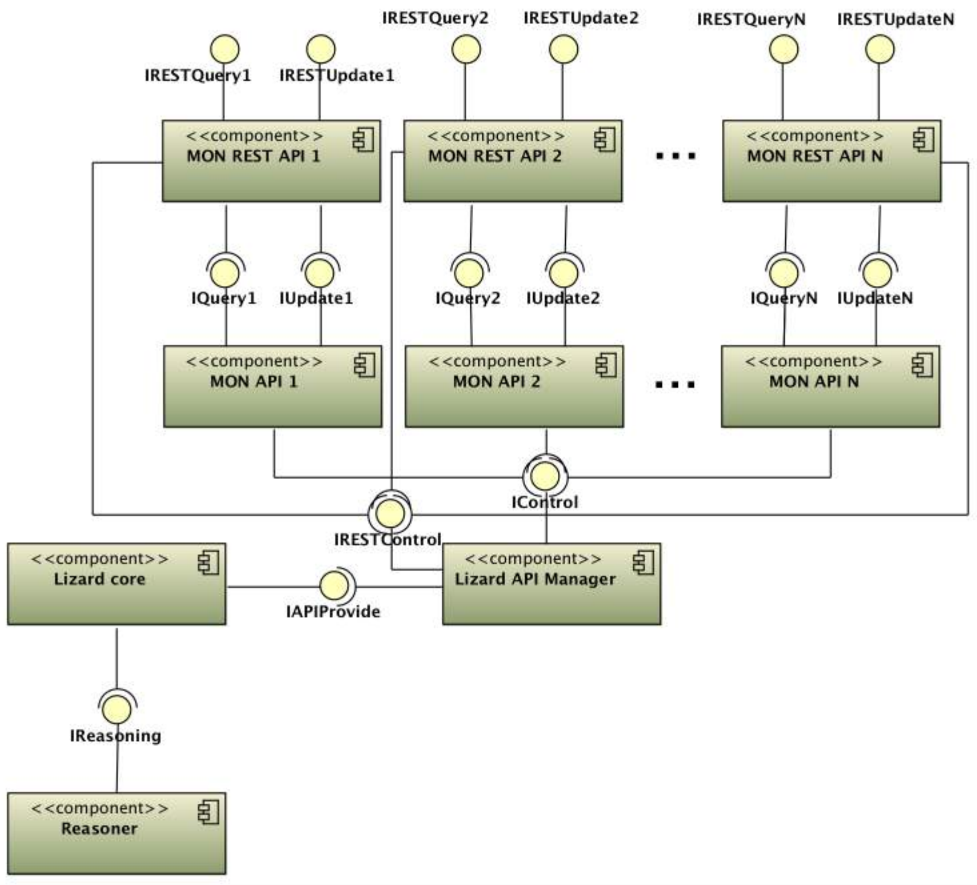


Figure 4.6: UML component diagram representing the software architecture of the MARIO KMS.

- Reasoner, which implements the reasoning facilities described in Section 4.3. It exposes the interface `IReasoning` that can be used by the other components of the KMS to access the reasoning services;

- Lizard core, which implements the core functionalities that address the use cases 1-7. It interacts with the reasoner component by using the interface `IReasoning` and exposes the interface `IAPIProvide` the allows other components to access the software API generated from ontologies of the MON;

- Lizard API Manager, which implements the functionalities that address the use cases 9-15. It accesses the API generated by Lizard core by means of the interface `IAPIProvide` and exposes the interfaces `IControl` and `IRESTControl`. These interfaces allow the Lizard API Manager to manage the life-cycle (i.e., load, unload, activate and deactivate) of the Java and REST API, respectively. We remark that these APIs are automatically generated from ontological artifacts by Lizard core.

- MON API that is any software implementation of an API generated from ontological artifacts. A MON API is controlled by Lizard API Manager by means of the inteface named `IControl` and exposes the interfaces `IQuery` and `IUpdate` that provide programmatic access to the knowledge base with query and update facilities (cf. use cases 15-19);

- MON REST API that is any RESTful implementation of an API generated from ontological artifacts. A MON API is controlled by Lizard API Manager by means of the inteface named `IControl` and exposes the interfaces `IRESTQuery` and `IRESTUpdate` that provide HTTP access to the knowledge base with query and update facilities (cf. use cases 15-19). Basically, a MON REST API is a HTTP REST wrapper around a MON API. In fact, a MON REST API interacts with its corresponding MON API by means of the interfaces `IQuery` and `IUpdate` exposed by the latter.

Bibliography

[1] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. “The Berkeley FrameNet Project”. In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, COLING-ACL ’98, August 10-14, 1998, Université de Montréal, Montréal, Quebec, Canada. Proceedings of the Conference.* Ed. by Christian Boitet and Pete Whitelock. Morgan Kaufmann Publishers / ACL, 1998, pp. 86–90. URL: `http://aclweb.org/anthology/P/P98/P98-1013.pdf`.

[2] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. “The Berkeley FrameNet Project”. In: *Proc. of the 17th international conference on Computational linguistics*. Montreal, Quebec, Canada, 1998, pp. 86–90. DOI: `http://dx.doi.org/10.3115/980845.980860`.

[3] Bénédicte Batrancourt et al. “A core ontology of instruments used for neurological, behavioral and cognitive assessments”. In: *Proceedings of the 6th International Conference on Formal Ontology in Information Systems FOIS*. (Toronto, Canada). 2010, pp. 185–198.

[4] David L. Bisset. *D1.1 MARIO System Specification for Pilot 1*. Tech. rep. MARIO cosortium, Dec. 2015.

[5] MR Bliss, R McLaren, and AN Exton-Smith. “Mattresses for preventing pressure sores in geriatric patients.” In: *Monthly Bulletin of the Ministry of Health and the Public Health Laboratory Service* 25 (1966), p. 238.

[6] Eva Blomqvist et al. “Experimenting with eXtreme Design”. In: *Proceedings of the 17th International Conference on Knowledge Engineering and Management by the Masses (EKAW)*. (Lisbon, Portugal). Ed. by Philipp Cimiano and Helena Sofia Pinto. Vol. 6317. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-16438-5_9. Springer, 2010, pp. 120–134.

[7] Olivier Bodenreider. “The unified medical language system (UMLS): integrating biomedical terminology”. In: *Nucleic acids research* 32.suppl 1 (2004), pp. D267–D270.

[8] M. M. Bouamrane, A. Rector, and M. Hurrell. “Gathering Precise Patient Medical History with an Ontology-Driven Adaptive Questionnaire”. In: *Proceedings of 21st IEEE International Symposium on Computer-Based Medical Systems*. 2008, pp. 539–541.

[9] Matt-Mouley Bouamrane, Alan Rector, and Martin Hurrell. “Development of an ontology for a preoperative risk assessment clinical decision support system”. In: *Proceedings of 22nd IEEE International Symposium on Computer-Based Medical Systems*. 2009, pp. 1–6.

[10] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. “Sesame: A generic architecture for storing and querying rdf and rdf schema”. In: *Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*. Sardinia, Italia: Springer, 2002, pp. 54–68.

[11] Jeremy J Carroll et al. “Jena: implementing the semantic web recommendations”. In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. New York, NY, USA: ACM, 2004, pp. 74–83. DOI: 10.1145/1013367.1013381.

[12] Yeates Conwell et al. “Validation of a measure of physical illness burden at autopsy: the Cumulative Illness Rating Scale”. In: *Journal of the American Geriatrics Society* 41.1 (1993), pp. 38–41.

[13] Christiane Fellbaum, ed. *WordNet: an electronic lexical database*. MIT Press, 1998.

[14] Charles J Fillmore. “Frame semantics and the nature of language”. In: *Annals of the New York Academy of Sciences* 280.1 (1976), pp. 20–32.

[15] C.J. Fillmore. “The Case for the Case”. In: *Universals in Linguistic Theory*. Ed. by E. Bach and R. Harms. New York: Rinehart and Winston, 1968.

[16] Aldo Gangemi. “What’s in a Schema?” In: *Ontology and the Lexicon: A Natural Language Processing Perspective*. Cambridge, UK: Cambridge University Press, 2010, pp. 144–182.

[17] Aldo Gangemi and Peter Mika. “Understanding the Semantic Web through Descriptions and Situations”. In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE, Proceedings*. (Catania, Sicily, Italy). Ed. by Robert Meersman, Zahir Tari, and Douglas C. Schmidt. Vol. 2888. Lecture Notes in Computer Science. DOI:10.1007/978-3-540-39964-3_44. Springer, 2003, pp. 689–706. ISBN: 3-540-20498-9.

[18] Aldo Gangemi, Roberto Navigli, and Paola Velardi. “The OntoWordNet Project: Extension and Axiomatization of Conceptual Relations in WordNet”. In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*. Ed. by Robert Meersman, Zahir Tari, and Douglas C. Schmidt. Vol. 2888. Lecture Notes in Computer Science. Springer, 2003, pp. 820–838. ISBN: 3-540-20498-9. DOI: 10.1007/978-3-540-39964-3_52. URL: http://dx.doi.org/10.1007/978-3-540-39964-3_52.

[19] Aldo Gangemi, Domenico Pisanelli, and Geri Steve. “Ontology integration: Experiences with medical terminologies”. In: *Proceedings of the 1st International Conference on Formal Ontology in Information Systems FOIS*. (Rome,Italy). Vol. 46. 1998, pp. 98–94.

[20] Aldo Gangemi and Valentina Presutti. “Ontology Design Patterns”. In: *Handbook on Ontologies, 2nd Edition*. Ed. by Steffen Staab and Rudi Studer. International Handbooks on Information Systems. DOI:10.1007/978-3-540-92673-3_10. Berlin, Germany, 2009, pp. 221–243.

[21] Aldo Gangemi and Valentina Presutti. “Towards a Pattern Science for the Semantic Web”. In: *Semantic Web* 1.1-2 (2010), pp. 61–68. DOI: 10.3233/SW-2010-0020. URL: http://dblp.uni-trier.de/db/journals/semweb/semweb1.html#GangemiP10.

[22] Aldo Gangemi et al. “Sweetening ontologies with DOLCE”. In: *Knowledge engineering and knowledge management: Ontologies and the semantic Web*. Vol. 2473. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2002, pp. 166–181. DOI: 10.1007/3-540-45810-7_18.

[23] Aldo Gangemi et al. “Semantic Web Machine Reading with FRED”. In: *Semantic Web* Preprint.Preprint (2016), to appear.

[24] James Gibson. “The theory of affordances”. In: *Perceiving, acting, and knowing: Toward an ecological psychology* (1977), pp. 67–82.

[25] Michael Grüninger and Mark S. Fox. In: *Benchmarking – Theory and Practice*. Ed. by Asbjørn Rolstadås. IFIP Advances in Information and Communication Technology. DOI: 10.1007/978-0-387-34847-6_3. Boston, MA, USA: Springer, 1995. Chap. The Role of Competency Questions in Enterprise Engineering, pp. 22–31.

[26] Karl Hammar. “Ontology Design Pattern Property Specialisation Strategies”. In: *Proceedings of Knowledge Engineering and Knowledge Management - 19th International Conference (EKAW)*. (Linköping, Sweden). Ed. by Krzysztof Janowicz et al. Vol. 8876. Lecture Notes in Computer Science. DOI:10.1007/978-3-319-13704-9_13. Springer, 2014, pp. 165–180. ISBN: 978-3-319-13703-2.

[27] Matthew Horridge and Sean Bechhofer. “The owl api: A java api for owl ontologies”. In: *Semantic Web* 2.1 (2011), pp. 11–21.

[28] Sidney Katz et al. “Studies of illness in the aged: the index of ADL: a standardized measure of biological and psychosocial function”. In: *Jama* 185.12 (1963), pp. 914–919.

[29] Hak Lae Kim et al. “The state of the art in tag ontologies: a semantic model for tagging and folksonomies”. In: *Proceedings of the International Conference on Dublin Core and Metadata Applications*. (Berlin, Germany). Ed. by Jane Greenberg and Wolfgang Klas. Metadata for Semantic and Social Applications. 2008, pp. 128–137.

[30] Karin Kipper, Hoa Trang Dang, and Martha Stone Palmer. “Class-Based Construction of a Verb Lexicon”. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.* Ed. by Henry A. Kautz and Bruce W. Porter. AAAI Press / The MIT Press, 2000, pp. 691–696. ISBN: 0-262-51112-6. URL: http://www.aaai.org/Library/AAAI/2000/aaai00-106.php.

[31] Torben Knerr. *Tagging ontology-towards a common ontology for folksonomies*. 2006. URL: https://tagont.googlecode.com/files/TagOntPaper.pdf.

[32] Maddalen Lopez de Lacalle, Egoitz Laparra, and German Rigau. “Predicate Matrix: extending SemLink through WordNet mappings”. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, May 26-31, 2014.* Ed. by Nicoletta Calzolari et al. European Language Resources Association (ELRA), 2014, pp. 903–909. URL: http://www.lrec-conf.org/proceedings/lrec2014/summaries/589.html.

[33] MP Lawton and ELMNE M BRODY. “Assessment of older people: self-maintaining and instrumental activities of daily living.” In: *Nursing Research* 19.3 (1970), p. 278.

[34] Freddy Limpens et al. “NiceTag Ontology: tags as named graphs”. In: *Proceeding of the 1st International Workshop on Social Networks Interoperability SNI*. 2009.

[35] Steffen Lohmann, Paloma Dıaz, and Ignacio Aedo. “MUTO: the modular unified tagging ontology”. In: *Proceedings of the 7th International Conference on Semantic Systems I-SEMANTICS*. (Graz, Austria). Ed. by Chiara Ghidini et al. ACM International Conference Proceeding Series. ACM, 2011, pp. 95–104. ISBN: 978-1-4503-0621-8. DOI: 10.1145/2063518.2063531. URL: http://doi.acm.org/10.1145/2063518.2063531.

[36] Pattie Maes. “The Dynamics of Action Selection”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*. Detroit, Michigan: Morgan Kaufmann Publishers Inc., 1989, pp. 991–997.

[37] Maja J. Mataric. “Behaviour-based control: examples from navigation, learning, and group behaviour”. In: *Journal of Experimental & Theoretical Artificial Intelligence* 9.2-3 (1997), pp. 323–336. DOI: 10.1080/095281397147149.

[38] M. Minsky. “A Framework for Representing Knowledge”. In: *The Psychology of Computer Vision*. Ed. by P. Winston. McGraw-Hill, 1975.

[39] Roberto Navigli and Simone Paolo Ponzetto. “BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network”. In: *Artificial Intelligence* 193 (2012), pp. 217–250.

[40] Andrea G. Nuzzolese, Aldo Gangemi, and Valentina Presutti. “Gathering Lexical Linked Data and Knowledge Patterns from FrameNet”. In: *Proceedings of the sixth international conference on Knowledge Capture (K-CAP)*. (Banff, AB, Canada). Ed. by Mark A. Musen and Óscar Corcho. DOI:10.1145/1999676.1999685. ACM, 2011, pp. 41–48. ISBN: 978-1-4503-0396-5.

[41] Andrea Giovanni Nuzzolese, Aldo Gangemi, and Valentina Presutti. “Gathering lexical linked data and knowledge patterns from FrameNet”. In: *Proceedings of the 6th International Conference on Knowledge Capture (K-CAP 2011), June 26-29, 2011, Banff, Alberta, Canada*. Ed. by Mark A. Musen and Óscar Corcho. ACM, 2011, pp. 41–48. ISBN: 978-1-4503-0396-5. DOI: 10.1145/1999676.1999685. URL: http://doi.acm.org/10.1145/1999676.1999685.

[42] Alex Oliver and Timothy Smiley. “Multigrade predicates”. In: *Mind* 113.452 (2004), pp. 609–681.

[43] Eyal Oren, Benjamin Heitmann, and Stefan Decker. “ActiveRDF: Embedding Semantic Web data into object-oriented languages”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6.3 (2008), pp. 191–202.

[44] Jens Ortmann and Desiree Daniel. “An ontology design pattern for referential qualities”. In: *Proceedings of the 10th International Semantic Web Conference (ISWC)*. Springer. Bonn, Germany, 2011, pp. 537–552.

[45] Jens Ortmann et al. “An egocentric semantic reference system for affordances”. In: *Semantic Web* 5.6 (2014), pp. 449–472.

[46] Eric Pfeiffer. “A short portable mental status questionnaire for the assessment of organic brain deficit in elderly patients”. In: *Journal of the American Geriatrics Society* 23.10 (1975), pp. 433–441.

[47] Domenico M Pisanelli. *Ontologies in medicine*. Vol. 102. IOS Press, 2004.

[48] Valentina Presutti et al. “eXtreme Design with Content Ontology Design Patterns”. In: *Proceedings of the Workshop on Ontology Patterns (WOP)*. (Washington, DC, USA). Ed. by Eva Blomqvist et al. Vol. 516. CEUR Workshop Proceedings. CEUR-WS.org, 2009.

[49] Valentina Presutti et al. “The role of Ontology Design Patterns in Linked Data projects”. In: *Proceedings of the 35th International Conference on Conceptual Modeling (ER 2016)*. Gifu, Japan: Springer, 2016, to appear.

[50] David Riaño et al. “An Ontology for the Care of the Elder at Home”. In: *Proceedings of the 12th Conference on Artificial Intelligence in Medicine AIME*. (Verona, Italy). Ed. by Carlo Combi, Yuval Shahar, and Ameen Abu-Hanna. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 235–239.

[51] Jacobo Rouces, Gerard de Melo, and Katja Hose. “FrameBase: Representing N-Ary Relations Using Semantic Frames”. In: *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*. Ed. by Fabien Gandon et al. Vol. 9088. Lecture Notes in Computer Science. Springer, 2015, pp. 505–521. ISBN: 978-3-319-18817-1. DOI: `10.1007/978-3-319-18818-8_31`. URL: `http://dx.doi.org/10.1007/978-3-319-18818-8_31`.

[52] Laurence Z Rubenstein et al. “Impacts of geriatric evaluation and management programs on defined outcomes: overview of the evidence”. In: *Journal of the American Geriatrics Society* 39.S1 (1991), 8S–16S.

[53] Robert Shaw, Michael T Turvey, and William Mace. “Ecological psychology: The consequence of a commitment to realism”. In: *Cognition and the symbolic processes* 2 (1982), pp. 159–226.

[54] Thomas A Stoffregen. “Affordances as properties of the animal-environment system”. In: *Ecological Psychology* 15.2 (2003), pp. 115–134.

[55] Anselm L. Strauss and Juliet M. Corbin. *Basics of Qualitative Research: Techniques and Procedures for developing Grounded Theory*. DOI: 10.4135/9781452230153. Sage Publications Inc., 1998.

[56] Cui Tao et al. “A semantic web ontology for temporal relation inferencing in clinical narratives”. In: *Proceedings of the Annual Symposium of American Medical Informatics Association AMIA*. (Washington DC, USA). 2010.

[57] Mark Van Assem, Aldo Gangemi, and Guus Schreiber. “Conversion of WordNet to a standard RDF/OWL representation”. In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC?06), Genoa, Italy*. 2006, pp. 237–242.

[58] Bruno Vellas et al. “The Mini Nutritional Assessment (MNA) and its use in grading the nutritional state of elderly patients”. In: *Nutrition* 15.2 (1999), pp. 116–122.