Master-Thesis Proposal: A Document-Oriented Robot Memory for Knowledge Sharing and Hybrid Reasoning on Mobile Robots

Frederik Zwilling

August 6, 2016

1 Introduction

As autonomous robots are about to advance into more and more complex and unstructured domains, multiple challenges arise. One of the challenges is allowing them to grasp their environment and make smart decisions based on their knowledge. To solve this, robots have to memorize observations and gained knowledge about the environment and provide this knowledge to reasoning and planning systems. When multiple reasoning and planning systems are used and when multiple robots should work in collaboration, memorized knowledge has to be shared. The aim of the proposed thesis is to design and implement a robot memory that is capable of storing and querying knowledge, sharing it between multiple systems and robots, and allowing hybrid reasoning on it.

Basic requirements of a robot memory are representing, storing, and querying knowledge. With knowledge, we mean the generalization of the Data-Information-Knowledge-Wisdom (DIKW) hierarchy [1]. For example the DIKW hierarchy ranges from point clouds (data) to cluster positions (information), derived object positions (knowledge), and learned probability distributions (wisdom). We want to memorize spatio-temporal knowledge, such as positions with time-stamps, as well as symbolic knowledge (e.g. the room a robot is in). Memorized knowledge is used by planners and reasoners, knowledge based systems (KBS), to reason about the environment and plan actions. KBS are often specialized and have different strengths. For example PDDL based planners find plans achieving a goal, CLIPS reasoners can execute plans and update world models, and motion planners avoid collisions during grasping and driving. For complex tasks and environments, different KBS can be combined to utilize their specialized strengths. However this requires common knowledge (e.g. so that a global planner can use the current world model updated by a reasoner). A general and shared robot memory, as intended in this thesis, can centrally manage the robot's knowledge and supply or generate specialized knowledge bases. This allows planners and reasoners to be closer integrated and have world models consistent to each other because they can utilize shared knowledge and state estimations only have to be done once. Supplying or generating a specialized knowledge base requires filtering the robot memory, when only a part is relevant, and transforming the knowledge into the required form to be usable in the planner or reasoner language. The robot memory simplifies hybrid reasoning by providing spatio-temporal and symbolic representations of knowledge. It is also required to provide a persistent and scalable long time storage. It can also be used for multiple robots sharing common knowledge for proper coordination, which is usually laborious to implement KBS programming languages.

Two main features of the robot memory are *computables* and *event-triggers*. The concept of computables allows computing knowledge on demand instead of storing it staticly in the robot memory. Imagine a query for the distance of two objects. This can be computed rather quickly but storing all distances between each two dynamic objects and keeping them up do date is a lot of unnecessary effort. Moreover computables allows deriving symbolic from spatio-temporal knowledge. Event-triggers should notify components when a previously defined conditions of the robot memory apply. This can be used to keep a world model updated without polling or to wait for complex situations. For example a reasoner executing a production plan could register to be notified when a machine is out of order, what makes the plan is infeasable.

Our contributions are a conceptual and architectural design of a generic robot memory that is capable of efficiently storing arbitrary symbolic or geometric information including spatio-temporal data. As a central component it allows to consistently share knowledge for one among different reasoning and planning systems and for another among multiple robots. Basis for the thesis will be an implementation of the robot memory based on the document-oriented database MongoDB using the Fawkes Robot Software Framework and integrating specific adapters for the CLIPS reasoning and PDDL planning systems. The thesis focuses on two applications, which can profit from the robot memory and will be used for evaluation. On the one hand, the robot memory should be used by logistics robots of the RoboCup Logistics League (RCLL). Here it should share the world model between reasoners running on multiple robots collaborating in a team and provide the shared world model in PDDL for the use in a global planner. On the other hand, the robot memory should be used on a domestic service robot related to the RoboCup@Home league. Here the robot memory is needed as flexible and scalable long-time storage used by different components depending on either spatio-temporal or symbolic representations of stored knowledge.

In the following, the proposal gives an overview over the background of the thesis with the RoboCup application domain and the used software in Section 2. Section 3 presents related work and in Section 4 the approach of the thesis is shown with the goals of the robot memory, it's theoretical foundation, the architecture, and implementation. Evaluation considerations are presented in Section 5. We conclude with a summary and the time-schedule in Section 6.

2 Background

This section presents the background of the proposed thesis. On the one hand, we describe the primary application and evaluation domains the RoboCup Logistics League (RCLL) and the RoboCup@Home league. On the other hand, we present the most important software for the proposed thesis, the robot software framework Fawkes, planners and reasoners, which should benefit from using the robot memory, and the database MongoDB used in the back-end of the robot memory.

2.1 RoboCup

RoboCup is an international robotics competition founded to foster research in the field of robotics and artificial intelligence [17]. It provides standardized problems as a platform to foster and compare research results. Teams from all over the world compete in different leagues to benchmark their robotic systems. RoboCup provides a research test-bed, in which participating teams implement new approaches and make them robust against the

challenges of the real world complexity. Furthermore, the competition leads to comparison and evaluation of different approaches.
RoboCup features a variety of leagues, from soccer leagues in different sizes to rescue robots, each focusing on another aspect or application domain of robotics and artificial intelligence.

2.1.1 RoboCup Logistics League

The *RoboCup Logistics League (RCLL)*[1] is an industry-oriented competition. It tackles the problem of production logistics in a smart factory, where mobile robots have to plan, execute, and optimize the material and production flow between machines to produce and deliver products according to dynamic orders. Each competing team deploys a group of three robots that has to build ordered products autonomously by interacting with *Modular Production Machines (MPS)* and transporting work pieces between these machines. Figure 1 shows an RCLL robot filling a machine that mounts colored rings on work pieces. The robots are based on the Festo Robotino 3 platform, which uses a holonomic drive, and can be extended and programmed by the teams. The robot shown in Figure 1 was built by the Carologistics team and uses a laser range finder for localization, a custom made gripper for handling work pieces, and several cameras for detection of markers and light-signals mounted on the machines [27, 29]. The game is controlled by a software component called *referee box (refbox)* which randomizes the machine placement in the factory and the production orders, communicates with competing robots, controls machines, and awards points.

Figure 1: Robot and MPS used in the RCLL [29]

The game consists of two phases [8]. In the first phase, the *exploration phase*, the robots have to roam the factory to find randomly placed machines, which are used later. In the second phase, the *production phase*, the refbox announces orders, which products have to be produced by the robots. The products are build from colored cups with optionally mounted rings and a colored cap. There are four different machine types. The *base-station (BS)* provides new bases, the colored cups, as raw resource. The *ring stations (RS)* mounts colored rings on a work piece, the *cap-station (CS)* mounts caps to finish a product, and the *delivery-station (DS)* is used to deliver products.

A major challenge of the RCLL lies in the fully autonomous task planning and coordination of the multi-robot system in the dynamic environment with spatial information and time constraints. To cope with these challenges, a multi-robot decision making is necessary and therefore a mechanism to share knowledge about the world model between the robots. This can be done by the robot memory, which also allows combining a global planner for finding plans and a reasoner for plan execution.

2.1.2 RoboCup@Home League

The RoboCup@Home league is a competition about domestic service robots [34]. Robots have to assist humans in a wide variety of everyday tasks. These tasks include serving

[1] RoboCup Logistics League website: `http://www.robocup-logistics.org`

drinks, cleaning, setting tables, guiding or following people, helping in emergency situations, shopping, and cooking. Figure 2 shows an @Home robot in the competition. The goal of the RoboCup@Home league is to foster and benchmark research in the area of domestic service robots, to build a research community and to envision autonomous multi-purpose robots helping humans and especially elder people in the personal life.

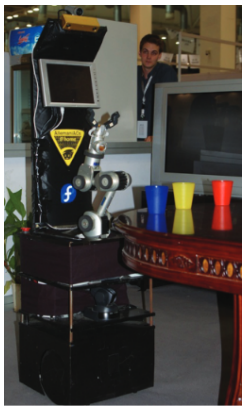

Figure 2: RoboCup@Home robot tidying up [34]

The competition consists of multiple sub-tasks, such as finding and manipulating objects, navigation tasks, remembering persons, wait on tables, acting as nurse, and an open challenge [33]. To solve these challenges, robots need to have a wide variety of abilities, including navigation, object detection and manipulation, speech recognition, and especially human robot interaction, which requires, for example, applying everyday knowledge to incomplete task descriptions given by humans.

Important challenges in the @Home league are acting robustly in a dynamic and only partial observable domestic environment and hybrid reasoning with symbolic and spatio-temporal information. The robot memory can help here because it allows to collect a lot of knowledge about the concrete domestic environment and, for example, log object observations to learn their distribution [22]. Furthermore, it can help with hybrid reasoning by transforming spatio-temporal knowledge into symbolic knowledge with computables.

2.2 Fawkes Robot Software Framework

The basis for the proposed thesis is the robot software framework *Fawkes*[2], which is available as open source [24,28]. It is designed to work with various robots in different domains and follows a component-based software design, which separates functional entities into individual software modules [7]. This enables reusing software solutions for robotic problems, such as localization, perception, reasoning, and behavior execution. Compiled components can be loaded as *plugins* at run-time. Plugin activity is organized in threads that can be run continuously ore hooked into the main-loop of Fawkes to order the execution into a *sense-think-act* cycle. Fawkes softly guarantees loop times by monitoring and eventually pausing and resuming threads running for too long. Features in Fawkes are provided as *aspects*. These are based on aspect-oriented programming [16] and give access to particular features. Threads that need to use a feature can inherit from the corresponding aspect. For example, there are aspects for accessing logging, timing, and specific external libraries [23].

The communication between plugins is realized with a *hybrid blackboard* paradigm. The blackboard lists structured entries called *interfaces*, which contain qualitative and quantitative information. Interfaces can be provided by one plugin at a time, the writer, and read by other plugins, the readers. This allows a flexible communication because the interface is independent from the concrete writer and readers. For example, sensor and actor plugins can be exchanged by simulation plugins [36]. To send commands to an interface writer, e.g. a motor command to the motor controller, readers can send messages to the interface.

As a communication infrastructure between different components, the blackboard has only limited possibility to realize a robot memory. Indeed the blackboard works well

[2] `http://www.fawkesrobotics.org`

```
1 (defrule found-machine
2   ?s <- (state SEARCHING_FOR ?machine)
3   (visible  (name ?machine))
4   (position (name robot1) (translation $?pos))
5   =>
6   (retract ?s)
7   (assert (state IDLE))
8   (printout t "Found machine " ?machine " at " ?pos crlf)
9 )
```

Listing 1: CLIPS rule to change a robots state when the object it searched for is visible.

for providing information and sending commands, though there are limitations for this application. The size and structure of blackboard interfaces is fixed, and thus does not allow to dynamically represent more or other information. Furthermore the blackboard does not support long-time memory, expressive querying, and event-triggers.

2.3 Planners and Reasoners

This subsection gives an overview of planners and reasoners, which are going to benefit from the robot memory. For the variety of these, we give three examples that should use the robot memory to achieve new or improved functionality and to evaluate this thesis. The example knowledge based systems are *CLIPS*, a rule-based production system, the *Planning Domain Definition Language (PDDL)*, which incorporates a family of planning formalisms into a standard programming language, and *Motion Planners* finding collision free actuation plans.

2.3.1 CLIPS Rules Engine

The reasoner currently used by Carologistics in the RCLL to implement the high-level game agent is implemented in the rule-based production system CLIPS [15]. It uses forward chaining based on the Rete algorithm [12] and basically consists of a fact-base, which is a working memory with small pieces of information, a knowledge base with rules and procedures, and an inference engine working with the knowledge base on the fact-base. The pieces of information in the fact-base are called facts and consist of a name and a key-value structure defined in a template for *unordered facts* (e.g. *(position (name robot1) (translation 2.5 1.0 0.0)*) or an ordered list of values for *ordered facts*. An example rule is shown in Listing 1. Rules are composed of an *antecedent* and a *consequent*. The antecedent is the first part of the rule and describes the condition that has to be satisfied before the rule is considered for activation. It consists of a list of patterns that have to be matched by facts in the fact-base. For the antecedent in Listing 1, there have to be fitting `state`, `visible`, and `position` facts with matching constants and variables, which start with question marks. The consequent is written behind the antecedent and defines the procedure to execute when the rule is activated. It usually modifies the fact-base by retracting and asserting facts, for example to exchange the `state` fact. The inference engine checks which rule antecedents are satisfied and puts only them on the *agenda*. Which rule of the agenda is executed, is determined by a conflict resolution strategy such as highest priority or earliest on the agenda. The consequent of the chosen rule is executed and the inference engine checks again. *Functions* encode procedural knowledge, can have side effects, and also be implemented in C++.

The Carologistics RCLL agent implemented in CLIPS represents its knowledge about the world, the *world model*, in its fact-base. It chooses actions to take by using a technique

called *Incremental Reasoning* [25]: Whenever the robot is idle, it searches for the next best task and executes it. The coordination of the robot team includes synchronizing their world model and resource allocation. This coordination is currently implemented in CLIPS rules using Google Protocol Buffers (Protobuf) messages. Alternatively, this could also be done by using a distributed robot memory that contains the world model in a distributed database. This would be advantageous because the database already provides efficient synchronization and the implementation of the synchronization in CLIPS increases the agent's complexity.

2.3.2 Planning Domain Definition Language (PDDL)

PDDL is a standardized language for planning problems [20]. It allows modeling the nature and behavior of a domain as well as the *actions* possible to perform in a *domain description*. With an additional *problem description*, which defines the problem to solve, it can be given to a PDDL planner that typically searches for a totally or partially ordered sequence of actions, the *plan*, to solve the problem. Actions consist of preconditions and effects. There are various versions and extensions of PDDL which introduce new concepts and features. For example, PDDL 2.1 [14] adds numeric values and continuous actions. This allows finding plans in continuous environments where, for example, distances and travel times matter. PDDL has the advantage of being able to find complete and optimal or efficient plans depending on the model. However, this comes with the drawbacks of high computational effort for larger domains and the problem of adopting to events during plan execution. For example when the domain is not fully observable, the robot could sense changes in the environment or fail executing an action. This would require notifying the planner, incorporating the changes in PDDL and re-planning. To solve this problem, the robot memory should allow combining PDDL with an execution in CLIPS. Then PDDL generates an efficient and complete plan with coarse actions and CLIPS executes the plan action for action, while monitoring the execution and updating the world model according to perceived changes. This world model is written into the robot memory so that PDDL can access the world model with all changes applied. This way the strengths of both PDDL and CLIPS can be combined.

2.3.3 Motion Planners

Another kind of widely used planners in robotics are geometric motion planners. They usually plan the motion of a robotic arm or the path of a driving robot to reach a certain goal position where an object should be grabbed or the robot wants to go to. Thus they use geometric knowledge in contrast to logical reasoners and planners focusing on symbolic knowledge. A motion planner needs to find a trajectory from the initial position to the goal position, taking joints of the arm into account and avoiding collisions with objects in the environment. A widely used motion planner framework is MoveIt! which is available for ROS [5]. Another motion planner framework already integrated into Fawkes is OpenRave [10]. These planners could benefit from using a robot memory by accessing and sharing information about objects and obstacles in the environment and by informing other planners and task execution components about the outcome of a motion plan (e.g. about the cause that prohibited the motion). This especially requires the robot memory to represent hybrid knowledge in the form of continuous positions and time related events.

```
{
  type: "position",
  name: "robot1",
  translation: {x:2.5, y:1.0, z:0.0},
  rotation: {x:0.0, y:0.0, z:0.0, w:1.0},
  timestamp : ISODate("2016-05-19T15
      :26:34.466Z")
}
```

Listing 2: MongoDB document representing the position of a robot

```
db.positions.find(
  {
    type: "position",
    name: "robot1",
    timestamp : {"$gt":
      ISODate("2016-05-19T15
          :26:34.000Z")}
  })
```

Listing 3: MongoDB query yielding the document in Listing 2

2.4 MongoDB

MongoDB is a *document-oriented* database that fits particularly well for the application in this thesis. It provides many useful features, such as flexible data structures, aggregation, and distribution over multiple machines, which would take a large effort to implement manually. It has proven as powerful and scalable data storage [6, 26] and is widely used. In contrast to relational databases, document-oriented ones store entities called *documents* consisting of key-value pairs. Listing 2 shows such a document. The different values can be accessed by their key and can have different types, such as strings, numbers, complex objects (e.g. dates) and nested sub-documents (e.g. translation containing 3D coordinates). In contrast to relational databases, there is no predefined schema that enforces the structure of documents. However, documents with similar structure are usually grouped into *collections*. Therefore the documents inserted in a collection implicitly define the structure during run-time. This allows a flexible use of the database because applications are not limited to a fixed schema and can, for example, add key-value pairs during run-time when additional information needs to be stored. Queries in MongoDB are similar to documents because the arguments of query functions also use document structure. An example query is shown in Listing 3. It searches for all documents in the collection `positions` that have matching values for `type` and `position`, and are up to date, thus with a timestamp greater than the given one. Multiple key-value pairs in a query are conjunctive. Values can be checked and compared as in the example with a variety of operators such as *not*, *or*, *equals*, *greater than*, etc. These queries can be evaluated very quickly, especially when *indexing* is used, a database feature that manages indexes for a combination of document keys. For more complex queries, it is possible to add arbitrary JavaScript functions with the *where* operator to check if documents match the query. Furthermore, it is possible to aggregate results using the *Map-Reduce* paradigm [9], which maps resulting documents to intermediate documents which are then reduced until the end-result is found. For example this allows querying for the nearest visible object by getting all visible objects, mapping their position to a distance and reducing them to the nearest one by dropping more distant ones when comparing them. The higher expressiveness of these queries comes with the price of slower computation. However, the computational effort can be reduced by formulating queries smartly. Many usual queries, e.g. Listing 3, can be formulated without using *where* operators. Furthermore, complex queries can be nested to filter documents first with a fast query and execute the computationally costly query only on the remaining documents. An especially useful feature of MongoDB that allows sharing knowledge between multiple robots is *replication*. Replication is a process that distributes a database onto multiple machines. One MongoDB instance called *Master* initially copies the database to other instances called *Slaves*. Afterwards, all modifications to the master database, which are stored in a separate collection called *operations log (oplog)*, are forwarded to the slave to keep them synchronized and consistent. Read only queries can be computed by Master and

Slave instances. Modifying queries are forwarded to the Master to guarantee consistency. MongoDB can also group multiple instances into a *Replica Sets*, which automatically manages the master election when necessary. Using replication, multiple robots can robustly share a common and consistent world model. Robot specific knowledge can be kept in a separate and not replicated collection to avoid unneeded communication.

MongoDB is especially well suited for this thesis because of its performance and scalability compared to other document-oriented databases with similar features. A possible alternative to MongoDB is the document-oriented database CouchDB [2]. CouchDB focuses more on distributed Systems providing a system called Multi-Master Replication, which allows writing to all replication instances. Conflicts can be solved similar to a version control system, thus multiple conflicting versions can be kept in parallel for later revision. However, MongoDB has shown to scale much better than CouchDB and other document-oriented databases [19].

3 Related Work

This section gives an overview of the related work of the proposed thesis starting with the knowledge processing systems KnowRob and OpenRobots Ontology, which provide similar functionality. Afterwards we present the Generic Robot Database based on MongoDB and Fawkes, and Open-EASE, a combination of KnowRob and the Generic Robot Database.

3.1 KnowRob

KnowRob is an open source knowledge processing system for cognition-enabled robots [30, 31]. It is designed to store knowledge about the environment and set it into relation to common sense knowledge to understand vaguely described tasks, such as "set the table". For representation and inference of knowledge, KnowRob uses Description Logic and approaches of the semantic web. Each small piece of information is represented as a *Resource Description Framework (RDF) triple* (e.g. $rdf(robot, holding, cup)$) that are composed of a subject, predicate, and object. A set of these RDF triples in a network can compose commonsense knowledge based on the *Web Ontology Language (OWL)*. It uses *ontologies*, which can roughly be described as directed graphs setting objects or classes of objects into relation. The graph can be represented with multiple RDF triples, where the subject and object are nodes and the predicate is a directed edge from the subject to the object. In these ontologies, it is possible to represent common sense knowledge, such as "milk is-a perishable", "refrigerator storage-place-for perishable", and "refrigerator is-a box-container". Thus a robot asked to bring milk could infer that the milk is stored in the refrigerator, which needs to be opened first since it is a box-like container. To be able to interface with perception, KnowRob uses a concept called *virtual knowledge base*. It allows computing needed information on demand instead of storing everything providently. Thus special queries can be forwarded to other components that compute the answer efficiently. The concept inspired the usage of computables in this thesis. The implementation and interface of KnowRob is based on the logic programming language Prolog, what makes representing RDF triples and inference with ontologies intuitive. To understand the robots environment, large and detailed ontologies are necessary. KnowRob features acquiring and connecting these from various sources, such as online common sense databases, the cloud-based robot infrastructure RoboEarth [35] and websites for deriving object information from internet shops and action recipes from how-to websites giving step by step instructions. However, it has been found that this acquired knowledge needs a lot

of manual extension and verification before it can be used [32]. Encyclopedic knowledge often lacks action information needed by the robot. Furthermore large ontologies limit the performance of KnowRob because of the Prolog implementation and general complexity. In contrast to the robot memory intended in this thesis, KnowRob focuses on common sense reasoning instead of an efficient and flexible on-line back-end. It does not support sharing knowledge between multiple robots and has a too strong focus on data only usable for reasoning components. Although it can also represent spatio-temporal knowledge, we have performance and scalability concerns with Prolog handling larger data-sets (e.g. for storing locations of found object over long time to learn the distribution where they can be found).

3.2 OpenRobots Ontology (ORO)

The OpenRobots Ontology (ORO) is an open source knowledge processing framework for robotics [18]. It is designed to enable human-robot interaction by featuring commonsense ontologies similar to KnowRob and modeling the human point of view. It also uses RDF triples and OWL ontologies. The triple storage is implemented in Jena, a Java toolkit for RDF, and the inference with Pellet, a Java OWL reasoner. Besides knowledge storing, querying and reasoning, ORO features a modular architecture between the back-end storage and front end socket server for querying. These modules add features such as events that notify external components about changes. Another module adds representations of alternative perspectives that should allow understanding the point of view of other agents. For example when a human asks a robot to bring the cup, there are two cups on a table and the robot should infer which cup is meant by knowing that the other cup is occluded from the human's point of view. Furthermore ORO features categorization to find differences between objects (e.g. to ask if the blue cup or the red cup was meant in a vague task description) and memory profiles for distinguishing long-term knowledge and short-term knowledge that is removed when the lifetime of a fact expires. Although these are useful features and we want to realize the concepts of events and memory profiles in this thesis, we do not use ORO as a basis. Due to its focus on human-robot interaction and commonsense reasoning, it is not suitable for representing large amounts of data for non-reasoning components because all data would have to be stored in RDF triples and processed by the OWL reasoner. Furthermore, it does neither support synchronizing a part of the knowledge with other robots nor knowledge computable on demand.

3.3 Generic Robot Database with MongoDB

There already is a generic robot database developed with MongoDB as data storage [26]. It is used to log data of a robot middleware (Fawkes and ROS) and allows, for example, fault analysis and performance evaluation. To achieve this, it taps into the messaging infrastructure in ROS and the blackboard interfaces in Fawkes to store the data one to one utilizing the flexibility of the schema free MongoDB. The logged data can later be queried by the robot or a developer. This allows better evaluation and fault analysis because the data would otherwise be disposed and logging of the whole Data-Information-Knowledge-Wisdom (DIKW) hierarchy [1] enables retracing processes, such as from point clouds (data) to cluster positions (information) to object positions (knowledge) combined from multiple clues to learning results (wisdom). This work already implemented a basic connection to MongoDB in Fawkes for storing documents and executing queries, which can be used in this thesis. It also showed that MongoDB is a good choice because it can handle querying a large database and writing a lot of data efficiently while being flexible how and

which kind of data is being represented. However this work lacks some important concepts needed for a robot memory as intended in this thesis. It is intended as logging facility and not as working memory holding a world model for multiple planners and reasoners. Therefore updating and querying mechanisms for planners and reasoners are missing as well as triggers, a knowledge computable on demand, multi-robot synchronization, a suitable front-end, ad integration into knowledge based systems.

3.4 Open-EASE

Open-EASE is a knowledge processing service for robots and AI researchers [3]. It is based on and combines KnowRob [30] and the Generic Robot Database with MongoDB in ROS [26]. It is designed as a web service accessible by robots via a socket connection and by humans via a web browser. It uses KnowRob for commonsense reasoning and storing the world model of a robot. The Generic Robot Database is used to log data about robots or humans performing manipulation tasks. This data can then be accessed by using the virtual knowledge base concept to learn from human demonstration and analyze faults. The focus of Open-EASE also is on providing the recorded data and access to KnowRob to humans using the web interface. This allows using Open-EASE as eLearning tool for students, who can explore and experiment with the data and system on the robot. Furthermore it fosters reproducing experiment results (e.g. for reviews) and visualizing them. Although the system is accessible from multiple users and robots, which can query the same Open-EASE instance, it is focused on single robot data and non collaborative processes because each user operates in his private container with individual knowledge base. This makes sense in the context of student exercises, but is not suitable for multi-robot systems or multiple knowledge based systems sharing knowledge. If the system is also usable or being extended for cooperating robots or planners exchanging knowledge over the web service is not mentioned in current publications or on the project website[3].

4 Approach

This section presents the approach of the proposed thesis. It starts with the goals we want to achieve with the robot memory and a theoretical foundation of the robot memory and how knowledge is transferred into a knowledge based system (KBS). It follows an overview of the architecture, which describes in which components the robot memory is organized and how they interact with each other. Afterwards, we describe how the implementation is planned, especially for the robot memory and interfaces for planners, reasoners and other components.

4.1 Goals

This subsection presents the design goals of the robot memory:

Flexible Storage and Retrieval: The robot memory has to provide a service for storing, updating, removing, and querying knowledge so that different components can use the robot memory to provide knowledge, use it as a source, or collaborate using the same memory. It should be flexible enough to store any kind of data, information, knowledge, and wisdom of the DIKW hierarchy in any structure. For example the robot memory should be able to store a world model, perception results and even large datasets, such as remembered sights of objects.

[3] `http://www.open-ease.org/`

Consistent Knowledge Sharing Between Knowledge Based Systems: The robot memory has to allow multiple components to collaborate by consistently sharing a common database. This should especially be used to share a common world model between different kinds of KBS. This allows combining them with their specialized strengths (e.g. generating plans in PDDL and executing them in CLIPS). It avoids inconsistencies because the basic knowledge used by the KBS is generated by the robot memory and thus only one state estimation is necessary. For example world model changes detected and stored in the robot memory by CLIPS are made available to PDDL.

Special Views for Different Components: Depending on the application of different components using the robot memory, different views on the stored data is necessary. For example a planning components needs another part of the stored data, than a component learning distributions of object sights. Thus component-specific filters are necessary. Furthermore, different components need the stored data in different formats (e.g. depending on the used programming language). Keys or identifiers might be named differently and have to be translated.

Knowledge Computation on Demand: The robot memory has to provide knowledge on demand. This is knowledge that is not stored in the robot memory directly, but computed when needed. It allows accessing knowledge through the robot memory that is otherwise impracticable to compute and store continuously and for all possible queries. For example when a component queries the distance of two objects, it is better to compute the result on demand than to store all distances between each two objects in the memory and to keep them updated. Components using the robot memory should be able to provide knowledge computation functions (*computables*) to the robot memory. This adds a lot of versatility and expandability.

Shared Knowledge for Multi-Robot Systems: The robot memory should be able to distribute parts of the knowledge over multiple robots, which can then share a common world model necessary for collaboration. The concept is similar to multiple planners sharing a world model on the same machine with the difference that the components on a multi-robot system work simultaneously and the additional difficulty of message loss and keeping consistency. Such a distributed robot-memory is necessary in many multi-robot domains, especially in the RoboCup Logistics League (RCLL). Distributing the robot memory adds additional requirements and challenges, especially robustness (e.g. in the case of robots dropping out), consistency of the memory, and accessibility so each robot can use the robot memory without large latices.

Spatio-Temporal Grounding: It is often desirable to ground stored knowledge based on location and time because this is the prerequisite for many applications, such as spatio-temporal clustering and learning distributions (e.g. at which day-times objects can be found at which positions [22]). Thus the robot memory has to be able to store knowledge tagged with spatio-temporal information and consider this information while querying.

Event Triggers: The robot memory should be able to notify components about relevant changes in the memory. These notifications are called *event-triggers* and should automatically be sent after a component registers an event it wants to be notified about. Events include appearances of new instances of a specific knowledge type, modifications and deletions of it, as well as sentences becoming true or queries returning documents. The balance of expressiveness versus performance still needs to be found. An example use case, in which event triggers are useful, is updating reasoner predicates.

Persistent Storage: To profit from long-time knowledge and to operate robots over

long time as necessary in intended domains, such as the domestic service, the knowledge needs to be stored persistently. This way, it is not lost when restarting a robot or repeating an experiment depending on knowledge stored in the robot memory.

Human Interface and Visualization: An important factor in the development of a robotic systems is the accessibility of underlying components for easier debugging and modeling. Therefore the robot memory needs to provide an interface for developers to introspect and modify the state represented in the robot memory. A visualization would also be desirable to analyze spatio-temporal data efficiently.

4.2 Theoretical Foundation

In this subsection, we lay the theoretical foundation of the robot memory and the integration into knowledge based systems. For this proposal, we focus on the integration into PDDL. The formal definition of the robot memory describes how knowledge is represented and depends on the definition of computables. Because the robot memory is document oriented, we first define documents and their key-value pairs. This definition is derived from the specification of the Binary JavaScript Object Notation (BSON)[4] used in MongoDB. Because documents can be nested, we define them by induction. Unnested documents are defined as follows:

1. **Keys** are strings $\mathcal{K} := \Sigma^*$, where Σ contains all valid characters.
2. **Atomic values** $\mathcal{V}_0$ are constants containing integers, floating point numbers, and strings.
3. **Unnested key-value pairs:** $\mathcal{P}_0 := \mathcal{K} \times \mathcal{V}_0$
4. **Unnested documents** are sets of key-value pairs with unique keys and thus included in the power set of $\mathcal{P}_0$:
   $\mathcal{D}_0 := \{d \in \mathbb{P}(\mathcal{P}_0) | \forall (k,v), (k',v') \in d, k \neq k' \vee (k,v) = (k',v')\}$

Values and documents with a nesting depth up to n with $n >= 1$ can be defined as follows:

1. **Values:** $\mathcal{V}_n := \mathcal{V}_{n-1} \cup \mathcal{D}_{n-1}$
2. **Key-value pairs:** $\mathcal{P}_n := \mathcal{K} \times \mathcal{V}_n$
3. **Documents:** $\mathcal{D}_n := \{d \in \mathbb{P}(\mathcal{P}_n) | \forall (k,v), (k',v') \in d, k \neq k' \vee (k,v) = (k',v')\}$

This yields the set of all **finitely nested documents** $\mathcal{D} = \cup_{n \in \mathbb{N}} \mathcal{D}_n$. Infinite nesting and documents containing themselves are not considered because they can not properly be mapped into most KBS formalisms.
A **database** is a set of documents $\mathcal{DB} \subset \mathcal{D}$. **Queries** are represented by documents $q \in \mathcal{D}$ and yield a set of documents $r \subset \mathcal{DB}$ as result when executed in a database according a specification (e.g. of MongoDB [6]).
A **computable** f is a function mapping a query document to a set of computed documents

$$f : \mathcal{D} \to \mathbb{P}(\mathcal{D}).$$

With these preliminaries, we can now define a **robot memory** as tuple of a database $\mathcal{DB}$ and a set of computables $\mathcal{C}$:

$$\mathcal{RM} = (\mathcal{DB}, \mathcal{C})$$

[4] `http://bsonspec.org/spec.html`

The set of documents **memorized** by the robot memory consists of the underlying database and all computable documents:

$$mem(\mathcal{RM}) = DB \cup \bigcup_{f \in \mathcal{C}} f(\mathcal{D})$$

To integrate the robot memory into PDDL, we need to define the **mapping functions** map_d between documents and predicates, and map_v between values and terms. As prerequisite, the connection between key-names in documents and predicates, functions, and their attributes needs to be defined. Let $\mathcal{R}$ be the **set of predicate symbols** and $\mathcal{F}$ be the **set of function symbols**, then

$$name_{predicate} : \mathcal{R} \to \mathcal{K} \text{ and } name_{function} : \mathcal{F} \to \mathcal{K}$$

map to the strings used in document keys (e.g. $name_{predicate}(\text{at}) = \text{"at"}$). The functions

$$name_{func-atr} : \mathcal{R} \times \mathbb{N} \to \mathcal{K} \text{ and } name_{pre-atr} : \mathcal{F} \times \mathbb{N} \to \mathcal{K}$$

map the attributes in the predicate or function to a key name (e.g. $name_{pre-atr}(\text{at}, 1) =$ "object" and $name_{pre-atr}(\text{at}, 2) =$ "room").
Now we can define the mapping of values in documents to terms as follows. Values that are no sub-documents can be mapped directly to the corresponding constant:

$$map_f(v) = v, v \in \mathcal{V} \setminus \mathcal{D}$$

Otherwise the value is a sub-document, which is mapped to a function term:

$$\begin{aligned} map_f(d) &= f(map_f(v_1), ..., map_f(v_n)), \text{ iff } f \text{ is a n-array function in } \mathcal{F}, \\ &\quad (\text{"}function\text{"}, name_{function}(f)) \in d, \forall i \in \{1..n\}(name_{func-atr}(f, i), v_i) \in d \\ map_f(d) &= nil_f, otherwise \end{aligned}$$

where nil_f is a new constant used for not mappable documents. Similarly documents can be mapped to predicates as follows:

$$\begin{aligned} map_p(d) &= p(map_f(v_1), ..., map_f(v_n)), \text{ iff } p \text{ is a n-array predicate in } \mathcal{R}, \\ &\quad (\text{"}predicate\text{"}, name_{predicate}(p)) \in d, \forall i \in \{1..n\}(name_{pre-atr}(p, i), v_i) \in d \\ map_p(d) &= nil_p, otherwise \end{aligned}$$

For example is

$$\begin{aligned} &map_p(\{(\text{"}predicate\text{"}, \text{"}at\text{"}), (\text{"}object\text{"}, \text{"}cup\text{"}), (\text{"}room\text{"}, \text{"}kitchen\text{"})\}) \\ &\quad = at(map_f(\text{"}cup\text{"}), map_f(\text{"}kitchen\text{"})) = at(cup, kitchen). \end{aligned}$$

This theoretical foundation of the robot memory and the mapping of queried documents into predicates is the basis for the generation of PDDL problem descriptions. In future work, this can also be used to formalize the integration of the robot memory into Golog, but this exceeds the scope of the proposed thesis.

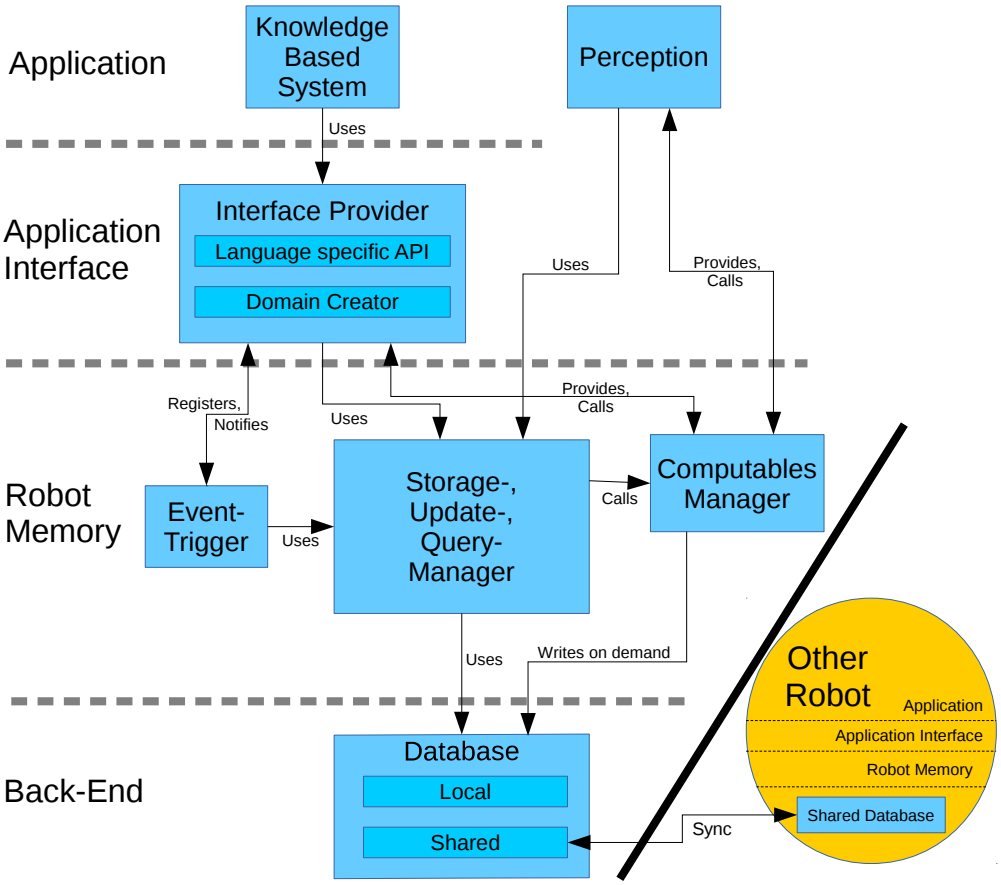

Figure 3: Software architecture of the Robot Memory

4.3 Architecture

This subsection presents the architecture of the robot memory and explains its design considerations. It shows how the intended functionality is organized in smaller components and how these components interact with each other. The architecture contains standard concepts of software engineering, most importantly a layered architecture, functional abstraction, and data abstraction [4] to allow reusability and adaptability to changing requirements and use cases.

Figure 3 shows the architecture of the robot memory and is explained in the following. The architecture organizes all components into four layers: Back-End, Robot Memory, Application Interface, and Application. The Back-End contains a database for storing knowledge and executing queries. There is a private part and a shared part distributed between multiple robots using the same architecture. This distribution can utilize the replication features of the database. The Robot Memory-layer contains the central functions that distinguish the robot memory from a pure database. It contains the Storage-, Update-, and Query-manager, which is an intermediate component that realizes most of the robot memory's functionality by modifying and enhancing queries, storage, and update requests before applying them to the database. If a query asks for knowledge provided by a computable, the manager forwards it to the computable manager, which manages which computable is provided by which application and requests the computation on demanded. The component responsible for event-triggers receives and manages registrations of applications that want to be notified in the case of an event. It uses the Query-manager to check if events happen and notifies applications accordingly. Additionally the Storage-, Update-, and Query-manager can initialize the robot memory (e.g. to setup the world

model for a new RCLL game). Between the robot memory and a planner or reasoner components is the Application Interface layer. For each KBS, it includes a Interface Provider component, which makes the function of the robot memory available to the KBS and the special programming language used. It is not necessary for components that can directly use the C++ interface of the robot memory. The responsibilities of the interface provider also include transforming query results in the appropriate format used by the KBS and eventually renaming key-names if they have be mapped to match the names used in the KBS model. Because languages and concepts of knowledge based systems can be very different, each KBS needs a separate Interface Provider. The Interface Provider can also contain a domain creator, which creates an initial state (e.g. a problem definition or initial fact base) from a template and additional knowledge resulting from queries. Finally the Application layer contains all components using the robot memory either through the Interface provider or directly through the robot memory interface. Application components can also register event-triggers and provide computation functions for computables. The sources and consumers of knowledge stored in the robot memory are the applications using it. The ordinary data flow starts in the application requesting storage, modification, or querying of knowledge (over the language specific interface). The Storage-, Update-, and Query-manager forwards the request enhanced with management information to the database. For queries, the result is transferred back on the reversed path of the query request. For computables, the only difference in the data flow is the knowledge computation before executing the query in the database. The Storage-, Update-, and Query-manager additionally initiates the knowledge computation via the Computables Manager, which forwards the query to the application providing the specific computable. The application then computes and stores the result in the robot memory, where it is queried afterwards.

4.4 Implementation

This subsection covers the considerations for the implementation. It is separated according to the layered architecture. Both application layers are represented by the planner and reasoner part, because all other application components (e.g. for perception) can simply use the C++ interface of the robot memory.

4.4.1 Back-End

The back-end is realized with MongoDB. It acts as storage of the robot memory, and executes queries. The representation of stored knowledge utilizes the document structure of MongoDB with key-value pairs and nested sub-documents. This allows a very flexible storage of knowledge-pieces in the structure induced by the application and expandable by robot memory information.

An example document stored in the back-end is shown in Listing 4. It contains knowledge about a bottle of milk as it could be needed by a planner with additional information where it should be stored later. Additionally the document contains meta information needed by the robot memory (e.g. if the document should be stored persistently or removed at restart and when it should be dropped because it's use-time has expired). The MongoDB back-end also manages the distribution

```
{
  _robot_memory_info:
  {
    persistent: true,
    decay_time: ISODate(
      "2016-05-19T 23:50:00.000Z")
  },
  type: "object info",
  name: "milk_1",
  position: {x:2.5, y:1.0, z:0.0},
  storage_place: "refrigerator"
}
```

Listing 4: Representation of a knowledge piece in the back-end

of knowledge between multiple robots. A part of the robot memory is only locally relevant. The other part that should be shared has to be set up as MongoDB replica set as already described in Section 2.4. This ensures that efficient networking code is used. The other problems of a distributed database, such as consistency, master election, and synchronization are solved by MongoDB using replication. The back-end also contains the operations log (oplog) of MongoDB, a separate collection containing a list of changes to the database with a timestamp. This can be accessed by the robot memory to analyze changes for event-triggers.

4.4.2 Robot Memory

The robot memory middle-layer implements most functions exceeding a typical database. A central question is which query language should be used between applications and the robot memory because this determines the expressiveness and has a large impact on the performance. We choose to use the query language of MongoDB as it is already used between the robot memory and the back-end. This has many advantages compared to using other query languages, such as SQL, SPARQL, XQuery [21], and JSONiq [13]. For the application view, the query language of MongoDB is a good choice because it is an intuitive query language, has been proven as efficient for usual and well designed queries, and is also highly expressive when using additional JavaScript functions or the MapReduce paradigm [6, 26]. For the robot memory, it requires no translation before application on the database and is very flexible for extending and modifying queries because queries are structured as documents with key-value fields and can be nested or executed in sequence. Furthermore, MongoDB queries can easily be parsed (e.g. from a string) by using the MongoDB C++ API. The resulting object can be analyzed and modified for example to add key-value pairs or to check if computables are queried.

This provides the starting point for the Storage-, Update-, and Query-manager. When adding new documents, the `robot_memory_info` sub-document can be added to store additional meta information. When querying documents, this information can be removed by using a filter. To detect queries for computables, the manager can analyze the fields of the query to check if there is a computable provided for it. For example when a query contains the field `type:"distance"` and there is a application that provides computation functions for distances between two objects, the query can be forwarded with the additional parameters, the two objects. When some fields are missing (e.g. only one object is given) the application may have to compute all distances, so that the query can afterwards be executed on the set of results (e.g. to find the nearest one with some property). To execute the query as a usual query with arbitrary filters, aggregation, and functions, the computation function writes the the result into a separate collection the query can be executed on with MongoDB. To lower the computational effort of processing the same query frequently it would be possible to add time-bounded caching for computed knowledge. We have verified this concept of computables using a prototype, which executes queries from interface messages either on the MongoDB directly or in case of computables about other interfaces, generates the knowledge from the blackboard on demand and then executes the query.

To implement event-triggers, a registration function needs to be provided that takes a notification function and a query to define the event. This query can check the oplog first whether there are changed documents with relevant key-value pairs and afterwards execute a more complex query on the database. The registration also specifies whether the callback function is called if the query result changes, returns no document, or is not empty. Whether event-triggers on computables perform well, has to be evaluated. It

```
1 (rm-query "database.collection"
2           (str-cat "{type:'order', end-time:{$gt:" ?gametime "}}"))
```

Listing 5: CLIPS function to execute a query

would be possible to allow events only on knowledge in the database or to compute the knowledge and check for the event in certain intervals while monitoring the computation time.

4.4.3 Planner/Reasoner

The implementation on the application layer includes the development of the Interface Provider and the usage of the robot memory in the planning language. As an example for the various planners and reasoners, we focus here on CLIPS. The Interface Provider for CLIPS can be realized as CLIPS-feature in Fawkes as it was already done for providing CLIPS access to the blackboard, Protobuf-messaging and the navgraph. The CLIPS *robot-memory feature* will be implemented in C++ and provides the CLIPS environment with functions that call C++ functions of the robot-memory feature. For example Listing 5 would be the CLIPS function, which queries all orders that have not ended yet, by creating the query using string concatenation to fill in the current game-time. The result would be a list of pointers to document objects represented as instances of CLIPS templates. Similarly there can be functions for registering events and providing computables.

The Domain Creator for CLIPS can be implemented by creating the initial fact-base with a static list of facts and additional facts resulting from a set of queries. For example, a domain creator implemented in the CLIPS language could execute the query function in Listing 5 and assert all orders as facts into the fact base.

CLIPS could use event-triggers to be notified when there is a new PDDL plan in the robot memory that should be executed or when the world model has changed in the robot memory and thus facts in CLIPS have to be updated.

5 Evaluation

This section covers the evaluation of the proposed thesis. We will evaluate the robot memory qualitatively in both application domains, the RCLL and RoboCup@Home league, and quantitatively.

The qualitative evaluation analyzes how well the robot memory can be used, how useful it is, and where difficulties are. It also includes analyzing the expressiveness and convenience of representing and querying knowledge, the integration into KBS languages, and the shared access to the robot memory across different applications and robots. To perform the qualitative evaluation, we will implement prototypes using main features of the robot memory. These prototypes act as proof of concept without implementing complete domain solutions which would extend the scope of the thesis. The general storage and query features of the robot memory will be evaluated in the RCLL context with a CLIPS agent that stores the world model in the robot memory and keeps it up to date. This shows how well the robot memory can be integrated in a reasoner. In a second step, the world model should be synchronized between a multi-robot team to evaluate how well the shared robot memory works in a distributed system. This includes an analysis of the network throughput and robustness to package loss. The planner specific view of the robot memory will be evaluated with a PDDL planner in the RCLL domain. The robot-memory-PDDL

interface should generate a PDDL problem description from a world model stored in the robot memory. The resulting plan should be stored in the robot memory and queryable by CLIPS. To evaluate event-triggers, we will register events depending on changes of the RCLL world model that need to be updated in the local CLIPS agent. Furthermore we want to try more complex events, which could be used to trigger actions in the @Home domain (e.g. the space occupied by objects in the dishwasher is high enough to start it). This should show how useful and expressive registered events can be. These prototypes should show if the robot memory allows knowledge sharing between multiple KBS. A related thesis, which is in preparation, wants to utilize PDDL for global task planning and CLIPS for execution to achieve more efficient multi-robot behavior in the RCLL. It could benefit from using the robot memory and yield further evaluation results.

To further evaluate how well the robot memory works in a hybrid reasoning context with motion planners and symbolic planners, we will also develop a RoboCup@Home related prototype using computables. Here the robot memory should hold knowledge usable by a motion planner and transform it into symbolic knowledge usable by a reasoner (e.g. if the robot's coordinates are stored and a reasoner asks for the room the robot is in). Thus a computable has to compute the symbolic knowledge from the spatio-temporal one or vice versa. Quantitatively, computables have to be evaluated by comparing the times between giving the query and receiving the results with and without computables taking the computation time, either on demand or on changing data.

To analyze how the query computation time increases with increasing query expressiveness and database scale, we want to implement a simple tidying up scenario. The robot memory models a scene with current object placements and object storage positions in increasing amount. The queries start simple (e.g. where is the storage place for the red cup) and get more complex (e.g. which objects are not at their storage position). Furthermore, the CPU and memory usage of the robot memory have to be analyzed as well as the size of the database on the hard drive.

6 Conclusion

In the proposed master thesis, we develop the conceptual and architectural design of a generic robot memory and implement it in Fawkes with MongoDB. The envisioned robot memory is a generic, capable, and efficient long-time storage for knowledge, which is the basis for memorizing a complex environment and thus for reasoning in it. The robot memory is a knowledge sharing system, on the one hand, for multiple knowledge based systems and, on the other hand, for multiple robots working collaboratively. Therefore it allows a consistent and tight integration of planners and reasoners while avoiding multiple state estimations. The robot memory is a hybrid reasoning tool that allows storing spatio-temporal knowledge and transforming it on demand into symbolic knowledge.

In contrast to existing knowledge processing systems, such as KnowRob and ORO, the concept of the robot memory decouples the storage of knowledge and reasoning on it and is based on a document-oriented representation. As we want to show in the evaluation of the thesis, this allows more efficient and scalable querying and tighter integration of multiple KBS and robots teams.

The architecture of the robot memory builds on top of a database system. It contains components for dispatching and enhancing queries and storage requests, for knowledge computed on demand, and for event-triggers notifying about changes. There are interface components, which provide the functionality of the robot memory for KBS by giving access in the used languages and transforming query results into the appropriate form.

	Weeks																									
Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Robot Memory Prototype																										
Storage-Update-Query Execution																										
Visualization,Human Interface																										
CLIPS Interface																										
Computables Infrastructure																										
Perception Connection																										
RCLL @Home Domain																										
Multi-Robot Knowledge Sharing																										
PDDL Interface																										
Event-Trigger																										
@Home Domain																										
Motion Planner Interface																										
Computables for Hybrid Reasoning																										
Evaluation																										
Qualitative																										
Quantitative																										
Writing																										

Table 1: Gantt Chart of the thesis time schedule

Applications using the robot memory can store and query knowledge, provide computables for other components, and use event triggers.

Challenges of the thesis include the conceptual tuning of the document-oriented knowledge representation and querying to achieve a balance between expressiveness and computation time. Further challenges are the design of computables in a document-oriented system to allow on demand computation of knowledge in an efficient and still powerful way, and the robust knowledge sharing in a distributed multi-robot system.

The main part of the thesis focuses on the robot memory as back-end of a robot system and foundation for future applications, such as a thesis in preparation about global PDDL planning in the RCLL with CLIPS as executive. Nevertheless we will use the RCLL and RoboCup@Home application domains to implement prototypes utilizing the robot memory and working as proof of concept for qualitative evaluation and basis for quantitative evaluation. For these two application domains, we will connect the robot memory to CLIPS, PDDL, and various perception components.

6.1 Schedule

The time-schedule of the thesis is shown in Table 1. The first part focuses on developing a robot-memory prototype with the CLIPS interface, infrastructure for computables, and fundamental connection to perception components. This prototype is then iteratively improved to implement all other features needed in the RCLL and in RoboCup@Home domain. The robot-memory itself and both applications are then evaluated and the rest of the time is spent on the written report.

References

[1] Russell L Ackoff. From Data to Wisdom. *Journal of applied systems analysis*, 16(1):3–9, 1989.

[2] J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: the definitive guide.* " O'Reilly Media, Inc.", 2010.

[3] Michael Beetz, Moritz Tenorth, and Jan Winkler. Open-EASE. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1983–1990. IEEE, 2015.

[4] Frank Buschmann, Kevin Henney, and Douglas C Schmidt. *Pattern-oriented software architecture, on patterns and pattern languages*, volume 5. John wiley & sons, 2007.

[5] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ROS topics]. *Robotics & Automation Magazine, IEEE*, 19(1):18–19, 2012.

[6] Kristina Chodorow. *MongoDB: the definitive guide*. O'Reilly, 2013.

[7] Mark Collins-Cope. Component based development and advanced OO design. Technical report, Technical report, Ratio Group Ltd, 2001.

[8] RCLL Technical Committee. RoboCup Logistics League: Rules and Regulations 2016. `http://www.robocup-logistics.org/rules`, 2016.

[9] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[10] Rosen Diankov and James Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[11] Christopher D. Ford (writer) and Jake Schreier (director). *Robot & Frank*. Samuel Goldwyn Films, 2012.

[12] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1), September 1982.

[13] Ghislain Fourny et al. *Jsoniq the SQL of NoSQL*. Citeseer, 2013.

[14] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 2003.

[15] Joseph C. Giarratano. *CLIPS Reference Manuals*, 2007. `http://clipsrules.sf.net/OnlineDocs.html`.

[16] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-Oriented Programming*. Springer, 1997.

[17] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The Robot World Cup Initiative. In *1st Int. Conference on Autonomous Agents*, 1997.

[18] Seéverin Lemaignan, Raquel Ros, Lorenz Mösenlechner, Rachid Alami, and Michael Beetz. ORO, a knowledge management platform for cognitive architectures in robotics. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3548–3553. IEEE, 2010.

[19] Yishan Li and Sathiamoorthy Manoharan. A performance comparison of SQL and NoSQL databases. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 15–19. IEEE, 2013.

[20] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language. Technical report, AIPS-98 Planning Competition Committee, 1998.

[21] Jim Melton. SQL, XQuery, and SPARQL: what's wrong with this picture. In *Proc. XTech*, 2006.

[22] Deebul Nair. Predicting Object Locations using Spatio-Temporal Information by a Domestic Service Robot: A Bayesian Learning Approach. Master thesis proposal, Hochschule Bonn-Rhein-Sieg University of Applied Sciences, March 2016.

[23] Tim Niemueller. Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao. Master's thesis, RWTH Aachen University, Knowledge-Based Systems Group, April 2009.

[24] Tim Niemueller, Alexander Ferrein, Daniel Beck, and Gerhard Lakemeyer. Design Principles of the Component-Based Robot Software Framework Fawkes. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 300–311. Springer, 2010.

[25] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In *AAAI Spring Symposium 2013 - Designing Intelligent AI*, 2013.

[26] Tim Niemueller, Gerhard Lakemeyer, and Siddhartha S. Srinivasa. A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[27] Tim Niemueller, Sebastian Reuter, Daniel Ewert, Alexander Ferrein, Sabina Jeschke, and Gerhard Lakemeyer. The Carologistics Approach to Cope with the Increased Complexity and New Challenges of the RoboCup Logistics League 2015. In *RoboCup Symposium – Champion Teams Track*, 2015.

[28] Tim Niemueller, Sebastian Reuter, and Alexander Ferrein. Fawkes for the RoboCup Logistics League. In *RoboCup Symposium 2015 – Development Track*, 2015.

[29] Tim Niemueller, Frederik Zwilling, Gerhard Lakemeyer, Matthias Löbach, Sebastian Reuter, Sabina Jeschke, and Alexander Ferrein. *Industrial Internet of Things: Cybermanufacturing Systems*, chapter Cyber-Physical System Intelligence – Knowledge-Based Mobile Robot Autonomy in an Industrial Scenario. Springer, 2017. (to appear).

[30] Moritz Tenorth and Michael Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. Part 1: The KnowRob System. *Int. Journal of Robotics Research (IJRR)*, 2013.

[31] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence*, 2015.

[32] Moritz Tenorth, Ulrich Klank, Dejan Pangercic, and Michael Beetz. Web-Enabled Robots. *Robotics & Automation Magazine, IEEE*, 18(2):58–68, 2011.

[33] Loy van Beek, Kai Chen, Dirk Holz, Mauricio Matamoros, Caleb Rascon, Maja Rudinac, Javier Ruiz des Solar, and Sven Wachsmuth. RoboCup@Home 2015: Rule and regulations. `http://www.robocupathome.org/rules/2015_rulebook.pdf`, 2015.

[34] Thomas Wisspeintner, Tijn Van Der Zant, Luca Iocchi, and Stefan Schiffer. Robocup@home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, 10(3), 2009.

[35] Oliver Zweigle, René van de Molengraft, Raffaello d'Andrea, and Kai Häussermann. RoboEarth: connecting Robots worldwide. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pages 184–191. ACM, 2009.

[36] Frederik Zwilling, Tim Niemueller, and Gerhard Lakemeyer. Simulation for the RoboCup Logistics League with Real-World Environment Agency and Multi-level Abstraction. In *RoboCup Symposium*, 2014.