Intel Serv Robotics (2016) 9:63–77
DOI 10.1007/s11370-015-0185-y

ORIGINAL RESEARCH PAPER

Cloud based centralized task control for human domain multi-robot operations

Rob Janssen[1] · **René van de Molengraft**[1] · **Herman Bruyninckx**[2] · **Maarten Steinbuch**[1]

Received: 10 February 2015 / Accepted: 10 August 2015 / Published online: 11 September 2015

Abstract With an increasing number of assistive robots operating in human domains, research efforts are being made to design control systems that optimize the efficiency of multi-robot operations. As part of the EU funded RoboEarth project, this paper discusses the design of such a system, where a variety of existing components are selected and combined into one cohesive control architecture. The architecture's main design principle stems from Radestock's 'separation of concerns', which dictates the separation of software architectures into four disjunct components; coordination, configuration, communication and computation. For the system's coordinating component a Golog based planning layer is integrated with a custom made execution module. Here, the planning layer selects and parametrizes abstract action plans, where the execution layer subsequently grounds and executes the involved actions. Plans and plan related context are represented in the OWL-DL logics representation, which allows engineers to model plans and their context using first-order logic principles and accompanying design tools. The communication component is established through the RoboEarth Cloud Engine, enabling global system accessibility, secure data transmissions and the deployment of heavy computations in a Cloud based computing environment. We desire these computations, such as kinematics, motion planning and perception, to all run on the Cloud Engine, allowing robots to remain lightweight, the instant sharing of data between robots and other algorithms and most importantly, the reuse of these algorithms for a variety of multi-robot operations. A first design of the system has been implemented and evaluated for its strengths and weaknesses through a basic, but fundamental real-world experiment.

Note to Practitioners The work described in this article is part of the EU funded RoboEarth project, in which knowledge repositories and intelligent services are developed to operate service robots world wide. The centralized task controller described here can be regarded as one of these intelligent services, that realizes a time-optimal allocation of multiple robots. This work shows significant similarities with the (not publicly available) *ubiquitous task controller* designed by Ha, where our work uses a more expressive action language. The general use of our system from a human end-user point of view is to request for a list of tasks, for which an abstract plan and according parameterizations are selected by the system. The selected plan is subsequently parsed into a Golog based planning interpreter, that binds available robots and all the other parameters (such as objects and locations). The parametrized actions involved in the plan are mapped onto computational algorithms and robots in a time-optimal manner, and subsequently executed by interfacing with them through a service(http)-based interface. What should be noted is that the executing robots themselves serve merely as sensor-actuator platforms, leaving all the computations to be performed on the Cloud. Our opinion is that this deployment advances the opportunity for calculation optimization and the instant reuse of updated task knowledge, e.g. object locations, action plans, and robot capabilities on a global level.

✉ Rob Janssen
robjanssen80@gmail.com

[1] Department of Mechanical Engineering, Eindhoven University of Technology, Den Dolech 2, 5600 MB Eindhoven, The Netherlands

[2] Department of Mechanical Engineering, University of Leuven, Celestijnenlaan 300, Heverlee, 3001 Leuven, Belgium

Keywords Service robots · Knowledge engineering · Centralized control · Cloud computing

1 Introduction

In modern day robotics research, cognitive robots are widely investigated as assistive technologies in everyday activities for tasks that are either too boring, strenuous or dangerous to be performed by humans, see Fig. 1.

(a) (b) (c)

Fig. 1 Examples of modern day robot assistants. Precise Path Robotics RG3 lawnmower robot (**a**), Panasonic Hospi delivery bot (**b**) and Csiro LHD mining robot (**c**)

Currently, most of these robots are assumed to operate individually and autonomously, i.e., each robot receives a private task request, collects raw sensor data of the nearby environment, processes this sensor data through on-board algorithms, deliberates on which action to perform next, and subsequently executes this action through its actuators. With the dawn of high-bandwidth communication technologies, it is now however possible to communicate sensor information in runtime. It has furthermore become possible, to store and process this vast amount of information in big-data storage facilities and cloud based computing platforms. These technologies pave the way for centralized task control; one intelligent, knowledge driven system that receives task requests and sensor information on a global level, combines, processes and reasons with this information and based on the outcome, controls thousands of connected robots through a multi-robot control algorithm. As opposed to robots calculating, reasoning and operating individually, such a system has the following advantages:[1]

- It allows robots to cooperatively perform tasks by spreading duties, in an omniscient and optimized manner;
- It allows task required knowledge, such as world information, to be collected on a global scale and therefore being most complete and up-to-date. Furthermore, it allows newly obtained knowledge to be instantaneously reused in subsequent task requests, without first having to be distributed to all agents;
- It allows data averaging and data outlier removal, as information is perceived by multiple sources (e.g. the state or existence of an object);
- It allows faulty robots to be replaced instantaneously by similarly capable ones. Additionally, these robot replacements do not need to be individually programmed, as all control configurations and computational algorithms reside on the Cloud;
- The algorithms used for computations can be centrally upgraded, as opposed to having to upgrade these algorithms on all robots individually;
- The required computational bandwidth can be allocated more efficiently, as there will be less computational down time, i.e. when robots are on stand by, turned off or temporarily taken out of order;
- Sensor information stored in databases can be easily backed up (by database mirrors), as opposed to retrieving that same information from the robots individually;
- Sensor information can be secured more efficiently, as none of it remains on the robot and is therefore not locally accessible;
- It reduces the energy consumption required for computations, as any energy consuming overhead (such as motherboards, cooling, storage) can be decreased more effectively when applied centrally.

The design concept of centralized task control for service robots aligns with the concept of ubiquitous robot networks [36]; a distributed control and sensing architecture that enables interoperability between heterogeneous systems with different hard- and software capabilities. These capabilities may vary from purely virtual, information retrieval services, to real world perception and manipulation activities. In this context, earlier work by Sgorbissa created the artificial ecosystem [58], where a service robot improved its navigation capabilities by autonomously interacting with nearby environmental agents, such as doors, elevators and navigational beacons. Later, Safiotti et al. developed a similar system called the PEIS ecology [56], where a deliberative layer was added that supervised the allocation of robots and environmental agents, hereby checking for plan feasibility and allowing plans to be instantiated in a more flexible, i.e. non-static manner. A survey paper by Mastrogiovanni [45] describes these methods in more detail, and an application scenario is sketched, hereby weighing the pros and cons of each individual architecture. Extended work by Mastrogiovanni elaborates on the scheduling of tasks under real-time constraints [46], and the inclusion of ontologies to represent task related context [44].

The use of logical languages and structured representations, such as ontologies, is crucial when desiring to express and reason with the large amounts of data such as those found in the service robot domain. In the work of Lemaignan [38] information inference using an online server is presented through a practical demonstration. Lim [40] subsequently establishes a multi-level representation of robot and environment knowledge, where Bayesian inference and heuristics are used to have a robot complete an under-informed task. In recent work of Rockel [55] a robot deployed system called RACE was developed, that elaborates on knowledge inference, by enabling the system to learn new concepts from semantically labeled experiences.

[1] In line with the advantages of Cloud Computing identified in [2].

To extend these investigations in to the multi-robot domain and to use them for a wide spectrum of heterogenous systems, interoperability between robots has been addressed in the work of Juarez [34], where Semantic Web [6] representations are adopted for the unified embodiment of sensor, service and actuator topologies. The work of Ha [28] incorporates these web-based representations into the ubiquitous robot network, and identical to the automated composition of web-services as proposed by Sirin [59], adds the Hierarchical Task Network SHOP2 [4] for the automated composition of multi-agent tasks. Their experiment describes task planning for cooperative activities between a mobile robot, a temperature sensor and actuated window blinds. For plan composition, each of these agents was modeled as an abstract web service based on the OWL-S [43] representation of tasks, originally designed to describe document or procedure oriented invocations of services on the Semantic Web.

Although the work of Ha establishes a centralized task planning architecture as described above, they conclude that scalability issues will arise, as in a real world application of their system ad hoc networks with invocable services are expected to be added dynamically. They also conclude that safety and privacy issues may arise as their communication framework is fully transparent. Offloading of computationally intense algorithms and the segregation of task related knowledge into separate repositories has in their work not been discussed and most importantly, as their work is closed-source and not publicly available, its implementation nor its design principles can be used and advanced upon by the widely established, open-source robotics community.

Our work therefore focuses on the design of such a centralized control framework, which is required to be scalable and secure, allows deployment of computationally intense algorithms and task related knowledge bases on a Cloud based computing and storage environment, and integrates with existing software design efforts of the open-source robotics community.

1.1 Outline

The following section will describe the design of the system, and how a main software design principle together with a list of system requirements leads to a basic component layout. The adoption of existing components will be motivated, and which alterations or custom additions are required. This section will be followed by an experimental use-case, that describes a first basic demonstration of the system. Conclusions will follow in the end, together with a section on future work required to make the system as general and user friendly as intended.

2 System design

2.1 Main design paradigm

The architecture's main design paradigm stems from Radestock's 'separation of concerns' [52], which dictates the separation of software systems into four disjunct components; coordination, configuration, computation and communication, with each component having its own responsibilities within a virtual system boundary. This segregated design approach is different from the classical design of sequential software programs, where component boundaries are often non-transparent and entangled. Coordination orchestrates the activities that a system has to undertake to achieve its desired goals or accomplish its instructed task. Configurability allows an engineer to develop generic functions, which can be reused for different activities. These functions are called computations, which transform incoming data to output, and are cascaded according to the action sequence of the desired task. Finally, the communication component establishes the data transfer within the system. See Fig. 2 for a general architecture layout.

Based on the commanded instruction, the coordination component requests for an action diagram (or plan) from the configuration database. This plan is subsequently executed, where generic computations are invoked according to the cascaded (or parallelized) action sequence of the plan. Parameterizations of these computations are determined by activity specific configuration files, which upon execution, are retrieved from the configuration database. To devise a concrete system realization from the above software component layout, a set of system requirements needs to be defined.

2.2 System requirements

The requirements of the system are (1) to serve as a centralized task controller for a variety of robot platforms, where (2) robot allocation is optimized through the central controller. As it is desirable that the involved robot platforms remain lightweight by reducing on-board com-

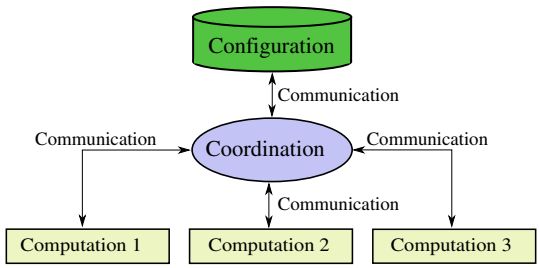


Fig. 2 Radestock general architecture layout

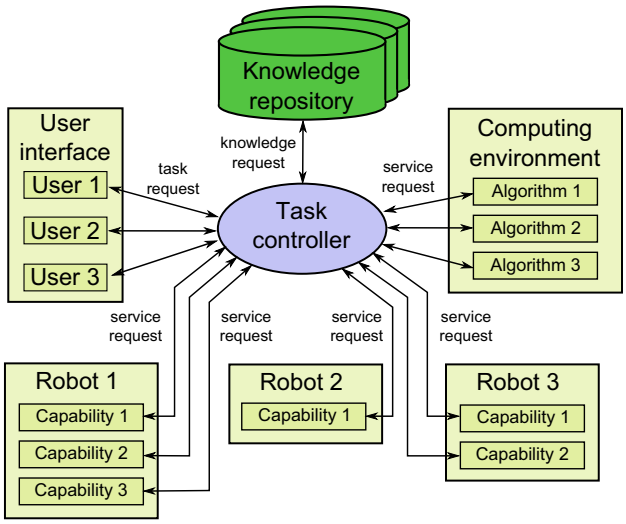


Fig. 3 Basic component diagram of system requirements

puting requirements, (3) computational algorithms are to be deployed in a central computing environment, instead of running locally on the robots. Furthermore, as the execution of tasks in human domains requires vast amounts of task related knowledge, such as binary models for perception and logic based plan representations, (4) the system is required to have fast and direct access to a knowledge repository that is capable of storing and retrieving this information securely.

2.3 Basic component diagram

A basic component diagram that maps onto Radestock's component structure of Fig. 2, and that aligns with above requirements, is sketched in Fig. 3

For this architecture, the computational functions consist out of robot sensing and actuating capabilities, user task requests, and algorithms. To coordinate the invocations of these functions a task controller is designed, which as such is central to the system. Configurations of the system, that provide the ability to reuse the same underlying architecture for a variety of tasks, are stored in a knowledge repository. System communications are performed in a request-response message style format, which is explained in Sect. 3.1.

3 Component implementation

3.1 Communication framework

A key aspect of the component diagram sketched in Fig. 3, is the communication framework that establishes message type protocols and component interfaces. For robotic applications there are several proprietary frameworks available, such as Microsoft Robotics Studio [33] and We-bots [48]. However, in-line with recent advances in mainstream robotics research, it is preferable to target a communication framework that supports direct user contributions through open-source software development licensing [10]. Amongst several open-source available middle-wares, such as Player [23], Urbi [5] and Orocos [11], ROS [51] can be regarded as currently the most widely used and supported middle-ware. Reasons for its popularity are the vast amount of supported packages and libraries (currently over 3000), interface support for five commonly used programming languages (C++, Python, Octave, Lisp and Lua), its peer-to-peer communication approach and its thin messaging layer. This messaging layer currently supports over 400 different message types, such as point-cloud, image, diagnostic and joint-state information.

Earlier attempts have been made to use ROS as a globally accessible data communication and storage mechanism in the DAvinCi project [3], where the Apache Hadoop Map Reduce [17] Framework was used as a data storage and computing environment for the Fast-Slam algorithm [49]. However, the architecture established in the DAvinCi project used only a single computing environment. It constitutes no security protocols, leaving the computing containers and their corresponding data (e.g. secure site navigation maps) fully transparent to the outside world. Furthermore, although the DAvinCi project was built upon the ROS open-source messaging framework, the project by itself is not publicly available.

A secure, ROS based communication framework that is publicly available, is the RoboEarth Cloud Engine, or Rapyuta [31]. Rapyuta is a well-documented communication framework, and is developed as part of the effort for global communication and knowledge reuse in the RoboEarth project [61]. As a Platform As A Service (PAAS) framework [15], Rapyuta offers the possibility to:

- Deploy one or more secured (through mandated user privileges) computing environments, which subsequently allow robots to offload their computational algorithms;
- Creating a scalable computing architecture, by allowing containers to be launched in parallel (as opposed to Google's App Engine [57]);
- Push information from server to robot, through the use of Web-Sockets [62] and serialized Java-Script Object Notation (JSON) messages;
- Communicate messages between multiple ROS masters over the same connection (as opposed to ROS-bridge [16]).

3.2 Knowledge repository

An added advantage of using Rapyuta is the direct integration with the RoboEarth knowledge repository. This

repository contains logical representations of tasks and task related information, such as class and instance descriptions of robots, objects and environments [60]. It furthermore contains binary data, such as object models and navigation maps. The RoboEarth knowledge repository stores its logical data in a Sesame database [9], which can be queried through the SeRQL query language [8]. Binary data is stored in the Apache Hadoop File-system [64], allowing efficient and distributed data storage. Both types of knowledge are linked through a relational database, and can be accessed through either a web interface (by humans), or through a RESTful API [54] (by software agents) called `re_comm`.[2]

The logical knowledge stored in the RoboEarth knowledge repository is encoded in the web ontology language OWL [50], or more specifically, in its language variant OWL Description Logics (OWL-DL). OWL-DL provides in maximum expressivity, but remains decidable. This allows the language to be used in most modern day reasoning tools, such as Pellet, Racer, Fact++ or in theorem proving languages, such as Prolog and SQL. A good read on the advantages of using OWL-DL in robotics can be found in the work of Hartanto [29]. For the centralized task planning architecture as proposed in this work, several OWL-DL type classifications been made. These types will be discussed in Sect. 3.4.

3.3 Task controller

The central component in the proposed architecture is the task controller. As typically described in robot control literature [1], the goal of a task controller is twofold:

- It needs to identify if incoming user requests can be performed, by attempting to select a logical course of actions based on the available resources (planning);
- It needs to perform the selected course of actions by interacting with the required resources (execution).

3.3.1 Planning

There are several goal-based methods available for planning, such as STRIPS [22] and planning graphs [7]. However, as these planning methods are proven to be very ineffective for large-scale human domains [12], this work targets the Hierarchical Task Network (HTN) planning approach as described in the work of Erol [20]. This approach allows a more efficient search, by using full or partially pre-designed plans as a plan heuristic. Compared to STRIPS planning, HTN planning can speed up planning by many orders of magnitude, e.g. in polynomial time versus in exponential time [21].

In RoboEarth, pre-designed plans are called 'action recipes' [42], and identical to HTN methods and operators, RoboEarth action recipes describe primitive tasks that are directly executable, and composite tasks that are composed of other composite or primitive tasks. In our system, a distinction is made between primitive tasks that can be performed on a robot (either a sensor or actuator task), and primitive tasks that can be performed by one of the computational algorithms (which are all deployed on the Cloud Engine).

Planning commences by parsing the action recipes into a planning language, that can be interpreted by a planning algorithm. As the action recipes are stored and expressed in OWL-DL (see Sect. 3.4), a planning language and accompanying algorithm are selected that represent full OWL-DL expressivity. As HTN planning is typically based on PDDL [24] propositional logic, it misses expressivity compared to the description logics semantics of OWL-DL. A language that is capable of expressing description logics semantics is the situation calculus [47], a high-level, first-order plan execution language with some limited second-order features. An accompanying language implementation, that accommodates both planning and plan execution, can be found in Hector Levesque's Golog [39].

A first implementation of Golog lacked certain features required for task planning in human domains, such as the ability to plan with concurrent actions, exogenous events or sensed input. Extensions of Golog have therefore been made, such as ConGolog [25], which plans for tasks with concurrent actions and exogenous events, and IndiGolog [26], which executes plans iteratively based on sensed input. A recent successor of IndiGolog that allows planning for multi-agent systems, is called MIndiGolog [35]. As this work focuses on the planning of human oriented tasks for multiple robots, MIndiGolog will be used as the foundational planning implementation.[3]

As MIndiGolog is a Prolog implementation, task planning occurs by the theorem proving property of Prolog (depth-first search). A basic example of a MIndiGolog composite procedure for placing an object at a certain location is given in Listing 1.

```
proc(placeObjAtLoc(Agt,Obj,Loc),
     has(Agt,Obj) // at(Agt,Loc)
     : placeObj(Agt,Obj,Dest)
     : releaseObj(Agt,Obj)).
```

Listing 1 MIndiGolog example of placing an object at a location. The 'has' and 'at' predicates are used as preconditions, and the actions 'placeObj' and 'releaseObj' are primitive actions. The symbols '//' and ':' indicate control procedures for respectively 'sequential' and 'in-parallel', see [35].

A valid plan solution for this procedure can be obtained through the following domain axiomatization:

[2] http://wiki.ros.org/re_comm.

[3] An example of a MIndiGolog planning domain axiomatization for multiple agents baking a cake, can be found at http://www.rfk.id.au/ramblings/research/thesis/.

```
agent(amigo_1).
object(sprite_1).
location(table_1).
has(amigo_1, sprite_1).
at(amigo_1, table_1).
```

Listing 2 Domain axiomatization that leads to a plan solution. The class restrictions (agent, object,location) are induced to prevent type mix-ups. See Listing 4 how and why these class restrictions are enforced.

Execution of the plan is subsequently performed by the Pro log query *placeObjAtLoc(amigo_1,sprite_1,table_1),S0,S)* where 'S0' is the initial state as given in Listing 2 and 'S' is the final state. Predicate 'do' is used in Golog to start the planning process, see [35] for its implementation details.

If in the example domain axiomatization of Listing 2 two agents would have been defined instead of one, e.g., *agent(amigo_1)* and *agent(amigo_2)*, or another procedure for 'placeObjAtLoc' is found, two valid plan solutions will be found. Identical to the publicly available HTN planner SHOP2 [4], a time optimal choice is then made, by accumulating the durations of all involved primitive actions, and choosing the plan solution with the least amount of total time. In the case of two identical robots, the robot with the shortest path length to the object will be selected. This path length is calculated by the 'compute:Path' algorithm, as it is used in the 'Navigate' task, see Fig. 12.

3.3.2 Execution

After a viable (and optimal) plan is selected by the Prolog theorem prover, each of the involved subtasks is decomposed until a set of primitive actions is found that can be iteratively executed. Execution is performed by running the Prolog planning layer in an asynchronous thread with a custom made executive module (coded in Python), that upon receiving an action, consults the knowledge base for action parameters (a process called grounding). Because this thread, and therefore the optimization between plans, runs continuously, a change in parameters (such as an object being relocated), can immediately lead to the selection of a new plan or a different parameter binding.

The Python executive module executes the primitive actions based on the grounding knowledge, and reports the success of the actions back to the Prolog planning layer. Depending on the success of the action, the Prolog planner returns the following primitive action to be performed, or searches for another plan if the action was unsuccessful. The executive module is written in Python, as it allows fast development cycles, type introspection and has native bindings for all ROS message types.

Figure 4 depicts the integration between planning and execution as an activity diagram, in which the order of steps is as follows:

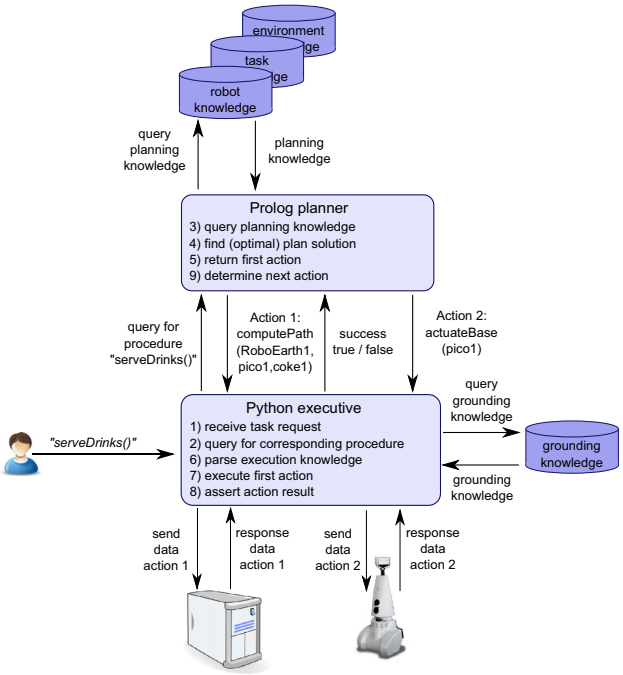


Fig. 4 The integration between planning and execution

1. A (typed) task is received at the executive;
2. The executive forwards this task to the planner;
3. The planner queries the knowledge base for plans that implement this task;
4. The planner tries to find a plan solution and if multiple solutions found, selects the time-optimal one;
5. The planner returns the first primitive action of this plan; together with the accompanying parameter bindings;
6. The executive queries the knowlegde base for grounding knowledge on the action and its parameters;
7. The executive performs the first action, by interacting with the relevant module (either sensor, actuator, or computation);
8. The executive returns the success of the action to the planner;
9. The planner determines the next action.

3.4 Knowledge representations

As the developed architecture intends to instantiate and execute parametrized plans for diverse algorithms and robot platforms, targeting human domain operations, the knowledge that is required for planning and execution consists out of:

- Robot knowledge, that logically describes the capabilities of a robot, such as the ability of using laser, arms or base,
- Environment knowledge, that describes the environment the robots are operating in,

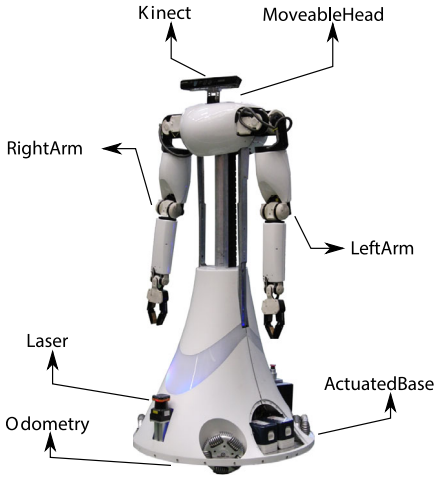


Fig. 5 Topological layout of Eindhoven University robot class 'Amigo'

- Task knowledge, that contains abstract representations of primitive and composite tasks,
- Grounding knowledge, that describes how primitive actions are executed by a robot or algorithm.

3.4.1 Robot knowledge

The robot knowledge base describes for each connected robot what robot class it belongs to, and what the capabilities of that class are. These capabilities are either sensors, such as a Kinect or laser, or actuators, such as arms or base. As an example, Fig. 5 depicts a topological layout of the TU/e Amigo robot class, where each module can be seen as one 'robot capability'.

The corresponding robot description for an instance of the 'Amigo' robot class is given in Listing 3.

```
<robot:Amigo rdf:ID="Amigo_1"/>
<owl:Class rdf:about="robot.owl#Amigo">
  <robot:hasSensor
    rdf:resource="robot.owl#Kinect"/>
  <robot:hasSensor
    rdf:resource="robot.owl#Laser"/>
  <robot:hasSensor
    rdf:resource="robot.owl#Odometry"/>
  <robot:hasActuator
    rdf:resource="robot.owl#ActuatedBase"/>
  <robot:hasActuator
    rdf:resource="robot.owl#LeftArm"/>
  <robot:hasActuator
     rdf:resource="robot.owl#RightArm"/>
  <robot:hasActuator
    rdf:resource="robot.owl#MoveableHead"/>
</owl:Class>
```

Listing 3 Robot description for 'Amigo_1', instance of robot class 'Amigo'.

The robot descriptions are used to match robot capabilities against task required components. This capability matching is performed in the planning language, by adding 'hasSensor' or 'hasActuator' predicates as pre-conditions in the according primitive action. An example, related to the primitive action 'placeObject' from Listing 1, is given in Listing 4.

```
prim_action(placeObject(Agt,Obj,Dest)) :-
    agent(Agt),object(Obj),location(Dest),
    hasActuator(Agt,rightArm);
    hasActuator(Agt,leftArm).
```

Listing 4 Robot capability matching.

3.4.2 Environment knowledge

General information about the world, such as environment and object properties, but also existence of class instances, is contained in the environment knowledge base. Examples are for instance the designated storage, dispose and serve locations for drinks. In Listing 5, an example is given for instance 'Sprite_1' of class 'Sprite' (subclass of 'Drink').

```
<environment:Sprite rdf:ID="Sprite_1"/>
<owl:Class rdf:ID="Sprite">
 <rdfs:subClassOf
  rdf:resource="environment.owl#Drink"/>
</owl:Class>
<owl:Class rdf:ID="environment.owl#Drink">
 <environment:hasStorageLocation
  rdf:resource="environment.owl#Refrigerator"/>
 <environment:hasDisposeLocation
  rdf:resource="environment.owl#TrashBin"/>
 <environment:hasServeLocation
  rdf:resource="environment.owl#People"/>
</owl:Class>
```

Listing 5 Abstract knowledge description of a 'Sprite'.

3.4.3 Task knowledge

For tasks, the abstract representation is built upon an existing OWL extension for processes on the Semantic Web, namely OWL-S [43] (formerly named DAML-S). Identical to the RoboEarth action recipes, OWL-S processes can be either one of two things;[4] a primitive.[5] process, which is directly executable, or a composite process, which describes the execution order for other composite or primitive processes. The execution order in composite tasks is dictated by the use of control procedures, such as while-do, split-join, if-then-else, sequential and parallel.[6] For the evaluation of logical conditions, OWL-S adopts the Semantic Web Rule Language (SWRL) [30], which combines OWL with RuleML, a Semantic Web standard for the evaluation of conditional

[4] A third type simple process exists, but as this is an abstraction of a composite process it will not be considered here.

[5] Formally called an 'atomic' process in the OWL-S technical description.

[6] Extensions to the MIndiGolog domain language have been made in this work, as it natively does not support many OWL-S control constructs, such as any-order, split and repeat-while.

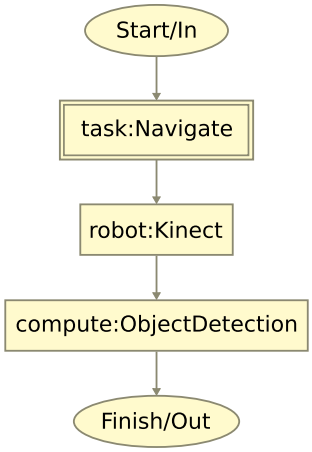


Fig. 6 Protégé design of OWL-S control procedure for detecting an object

expressions. As such, SWRL is used within OWL-S for the evaluation of control procedure conditions (such as if-then-else), and for process preconditions (such as for robot capability matching).

With the Protégé OWL-S modeling tool [18], control procedures can be easily developed by the visual overview and directly imposed logical constraints, see Fig. 6.

Primitive processes are indicated with single, and composite processes with double rectangles. A distinction is made between primitive processes that are executed on a 'robot' platform (if that robot has the proper capability), or executed by a computational algorithm running on the RoboEarth Cloud Engine (indicated by the 'compute' namespace). Process inputs and outputs, such as 3D point-clouds or joint state information, are not visualized on this level of modeling. These are solely represented in the grounding knowledge of each process, see the following section.

3.4.4 Grounding knowledge

OWL-S allows the process-flow modeling of primitive and composite processes on an abstract level, in a description logics representation. This allows logic based reasoning algorithms, such as planners and schedulers, to use these representations as planner building blocks. These abstract representations however, do not describe how processes are actually executed, which is called grounding. For this, primitive processes require information on implementation, interfacing, parametrization and communication. In the proposed architecture, this information is represented by the grounding knowledge, which is composed of an ontology describing process types, messages, parameters and communication channels. The OWL-S ontology by itself provides in a grounding representation suitable for invoking web services through the Web Service Definition Language (WSDL) [13]. WSDL provides in a concrete realization of abstract

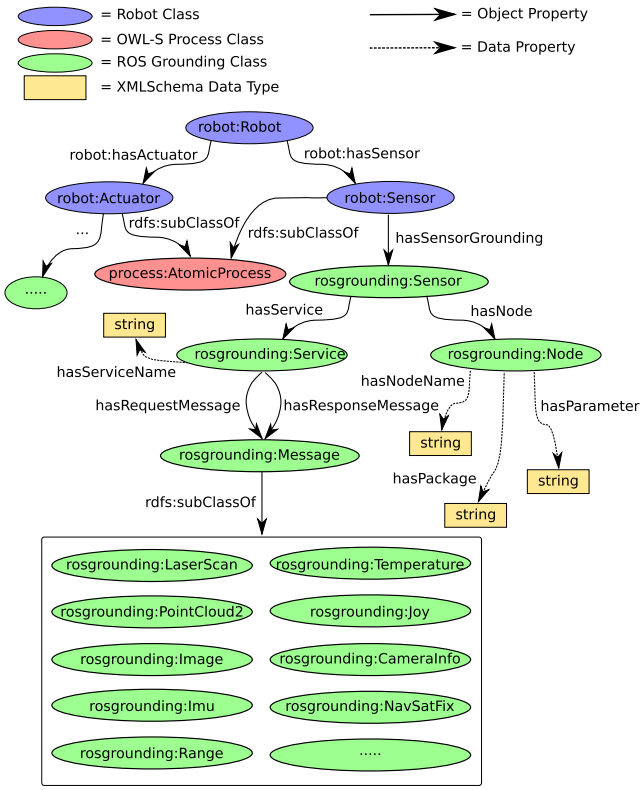


Fig. 7 Example part of the ROS grounding ontology for a robot sensor

operations and messages, which can be either document or procedure oriented, and interfaced through either SOAP, HTTP, GET/POST or MIME. As this work targets the execution of tasks on ROS enabled platforms however, a ROS message-type grounding ontology is specifically developed for this purpose, see Fig. 7.

Grounding ontologies for different middle-wares, such as Urbi or Orocos, are in general also possible to design, but that is beyond the scope of this work. An example OWL snippet of a ROS-grounded task for reading out laser scan messages is given in Listing 6.

```
<owl:Class rdf:about="robot.owl#Laser">
 <rosgrounding:hasSensorGrounding>
  <rosgrounding:Sensor rdf:ID="ReadLaser">
   <rosgrounding:hasService>
    <rosgrounding:Service
     rdf:ID="ReadLaserService">
     <rosgrounding:hasServiceName
      rdf:datatype="XMLSchema#string">/laser
     </rosgrounding:hasServiceName>
     <rosgrounding:hasResponseMessage
      rdf:resource="rosgrounding.owl#LaserScan"/>
    </rosgrounding:Service>
   </rosgrounding:hasService>
  </rosgrounding:Sensor>
 </rosgrounding:hasSensorGrounding>
</owl:Class>
```

Listing 6 ROS grounding snippet for reading out laser scanner messages.

The complete grounding ontology is used for both the grounding of primitive processes on real robots, and for the grounding of (primitive) computational processes running in the computing environment on the RoboEarth Cloud Engine.

3.5 ROS component model

Section 3.3.2 describes all processes being invoked by the executive, as opposed to standard ROS architectures, where nodes are programmed to communicate individually upon the completion of an internal calculation. As this is effective for single-robot architectures, it impedes scalability to larger multi-robot architectures. Furthermore, this type of software entanglement is an open invitation to in-code parameterizations that are specific for the application at hand, impeding component reuse. This work therefore uses an altered ROS node component model, that aligns with Radestock's design principles used in Sect. 2.1. Each algorithm now executes a generic, single computation (or single robot process) which is parametrized based on the configuration parameters (such as the HUE colorspace of a coke bottle, used for e.g. perception) found in the grounding ontology. As stated in Sect. 3.4.4, process interfacing is based on the ROS service-call protocol. A time-line sketch of this interfacing is depicted in Fig. 8.

3.6 Component deployment

Based on the above mentioned component details, Fig. 9 depicts a concretized component layout of Fig. 3.

4 Experimental use-case

To validate the functionality of the proposed system, and to identify the systems strengths and weaknesses, an experimental use case has been devised. The experiment entails two robots, the Eindhoven University Amigo and Pico, see Fig. 10, which are given the task of serving and cleaning up drinks at a 'cocktailparty' in the Eindhoven University robotics lab. The goal of the experiment is to validate the desired interactions between components, and to make a first assessment on where the system or one of its internal components needs to be improved upon.

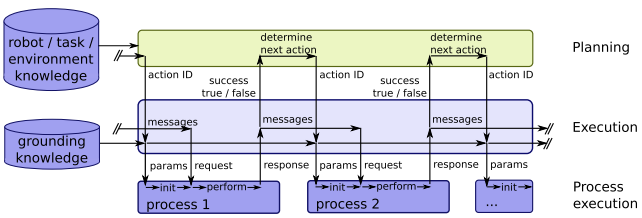


Fig. 8 ROS component model and process interfacing

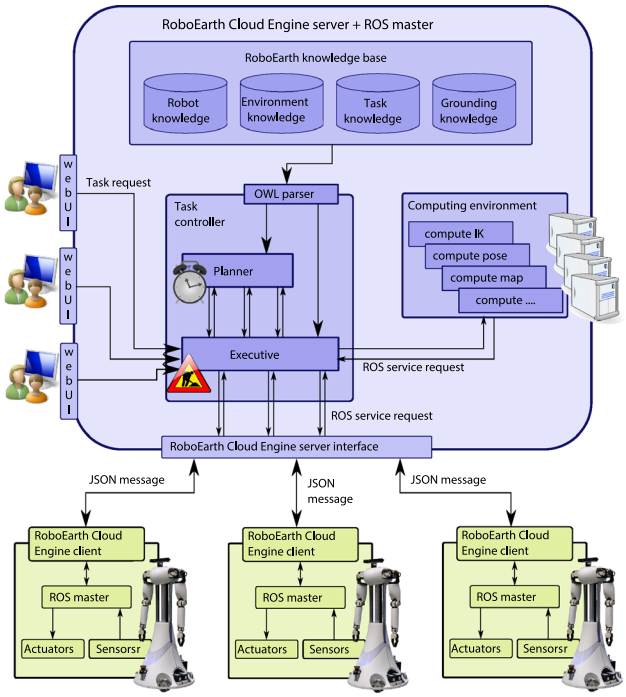


Fig. 9 Concretized component layout of Fig. 3

Fig. 10 TU/e Amigo (*left*) and Pico (*right*)

4.1 Experiment description

The top-level task for the 'cocktailparty' task[7], depicted in Fig. 11, consists out of 4 main subtasks;

- 'TakeOrder',
  - If an order is received, 'ServeDrink',
- 'FindEmptyDrink',
  - If an empty drink is found, 'CleanupDrink'.

[7] Designed with the Protégé OWL-S editor plug-in.

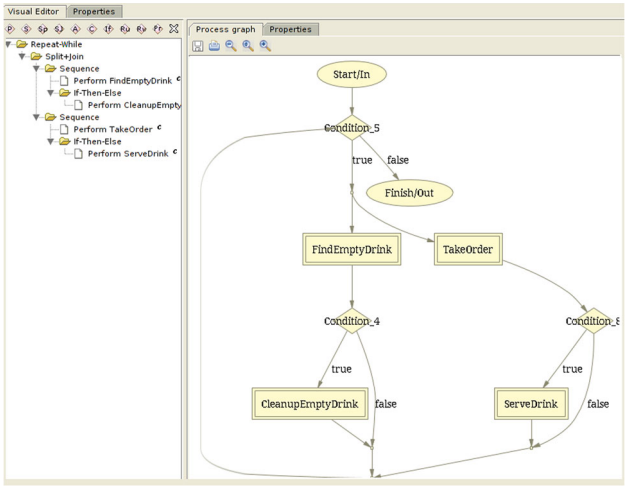
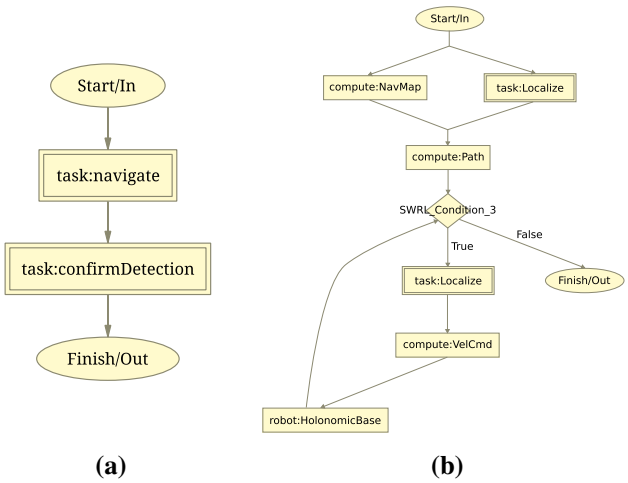


Fig. 11 Top level control flow for the 'cocktailparty' task



(a) **(b)**

Fig. 12 Control flow for 'FindEmptyDrink'(**a**) and subtask 'Navigate'(**b**)

The experiment described here will focus on one subtask, namely 'FindEmptyDrink', see Fig. 12.

In this experiment, the robot capabilities for both Amigo and Pico are identical:

- **hasSensor**(amigo_1,Kinect)
- **hasSensor**(pico_1,Kinect)
- **hasSensor**(amigo_1,Laser)
- **hasSensor**(pico_1,Laser)
- **hasActuator**(amigo_1,Base)
- **hasActuator**(pico_1,Base)

The accompanying services running on the robots are depicted in Table 1.

Furthermore, six generic algorithms have been launched on the Roboearth Cloud Engine, for which their inputs, outputs and parameterizations are listed in Table 2:

- **LocMap:** computes a nav_msgs/OccupancyGrid used for localization.
- **NavMap:** computes a nav_msgs/OccupancyGrid used for navigation.
- **Path:** computes a nav_msgs/Path from location parameters A and B (A and B are bound by the planner to initial robot and drink locations obtained from the 'environment' knowledge base).
- **Pose:** computes a geometry_msgs/PoseStamped that indicates the current position of the robot.
- **Detection:** detects an object, parameterized by its HUE values [53]. Successful detection is forwarded through the boolean 'success' return value.
- **VelCmd:** computes velocity commands, based on desired path and current pose. Returns true for success only if the final point in the path is reached.

4.2 Simulator

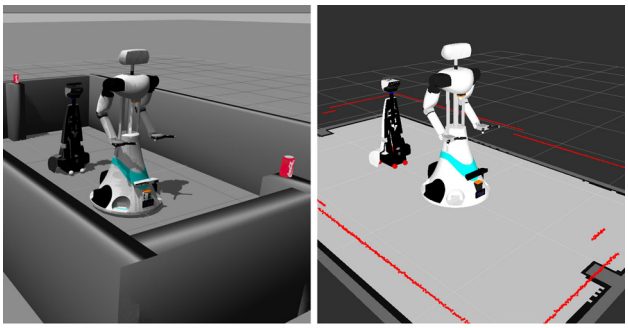
To allow a fast development cycle and initial parameter tuning (such as the parameters for maximum robot velocities and object HUE values), a test environment has been devised in the ROS Gazebo simulator. In this test environment, the two robots are spawned together with two 'empty' drinks. The goal is to execute the 'FindEmptyDrink' plan as it is devised in Protége, where the planner binds the parameters of this task (i.e. which robot searches at which location), in such a

Table 1 Services running on the robots

Service	Inputs	Parameters	Outputs
Kinect	None	None	senMsgs/Image (depth)
			senMsgs/CamInfo
			senMsgs/Image (color)
			Bool success
Laser	None	None	senMsgs/LaserScan
			Bool success
Base	geoMsgs/Twist	None	Bool success

Table 2 Generic algorithms launched on RoboEarth Cloud Engine

Service	Inputs	Parameters	Outputs
LocMap	None	locMap.yaml	navMsg/OccupancyGrid
			Bool success
NavMap	None	navMap.yaml	navMsgs/OccupancyGrid
			Bool success
Path	navMsgs/OccupancyGrid	robot(x,y,θ)	navMsg/Path
		target(x,y,θ)	Bool success
Pose	navMsgs/OccupancyGrid	robotFrame	geoMsg/PoseStamped
	senMsgs/LaserScan		Bool success
	geoMsgs/PoseStamped		
Detection	senMsgs/PointCloud	HUE.yaml	Bool success
	senMsgs/CamInfo		
VelCmd	navMsgs/Path	maxVelLin	geoMsgs/Twist
	geoMsgs/PoseStamped	maxVelAng	Bool success



(a) **(b)**

Fig. 13 Gazebo simulator (**a**) and Rviz visualizer (**b**)

Fig. 14 Real world experiment initial positions

way that a time optimal plan is instantiated. See Fig. 13 for a snapshot of the instantiated plan being executed.

4.3 Real world

In the real-world version of the experiment, RoboEarth client interfaces are deployed on the robots, and the experiment is conducted in a real lab environment, see Fig. 14.

(a) **(b)**

Fig. 15 Real world experiment final positions

Table 3 Packet sizes on client 'amigo_1' (in bytes)

Service	Sent	Received
Laser	4136	0
Base	1	48
Kinect	2150929	0

Figure 15 shows the two robots reaching their final positions, where the 'empty' drinks are positively detected.[8]

For this experiment the packet size per service on 'amigo_1' has been recorded, see Table 3.[9]

Furthermore, planner data has been logged, hereby showing the following incremental plan execution:

do [NavMap(amigo_1,"tue_lab")] at **time** 19.02
do [NavMap(pico_1,"tue_lab")] at **time** 19.02
do [LocMap(amigo_1,"tue_lab")] at **time** 20.41

[8] A video of the experiment can be found at http://youtu.be/4jCGcRs6GZI.

[9] As composed plans and hardware components are identical to both robots, the communication data for 'amigo_1' is assumed to be identical to the communication data of 'pico_1'.

```
do [LocMap(pico_1,"tue_lab")] at time 20.43
do [Laser(amigo_1)] at time 21.98
do [Laser(pico_1)] at time 22.04
do [Pose(amigo_1)] at time 22.19
do [Pose(pico_1)] at time 22.25
do [Path(amigo_1,coke_1)] at time 23.98
do [Path(pico_1,coke_2)] at time 24.03
do [LocMap(amigo_1,"tue_lab")] at time 25.43
do [LocMap(pico_1,"tue_lab")] at time 25.55
do [Laser(amigo_1)] at time 26.9
do [Laser(pico_1)] at time 26.98
do [Pose(amigo_1)] at time 27.05
do [Pose(pico_1)] at time 27.17
do [VelCmd(amigo_1)] at time 31.86
do [VelCmd(pico_1)] at time 31.96
do [Base(amigo_1)] at time 33.29
do [Base(pico_1)] at time 33.52
do [LocMap(amigo_1,"tue_lab")] at time 34.89
do [LocMap(pico_1,"tue_lab")] at time 34.94
....
do [Kinect(amigo_1)] at time 57.59
do [Detection(coke_1)] at time 58.71
do [Kinect(pico_1)] at time 61.04
do [Detection(coke_2)] at time 62.45
```

Listing 7 Planner log.

4.4 Results

What can be concluded from this log, is that the time between two consecutive localization steps (between the second and third 'LocMap(Agt,Env)' calculation), is approximately 9.46 s, and hence, has an update frequency of ∼0.1 Hz. This is a lot lower than native ROS localization components, such as AMCL,[10] which typically run at 20–40 Hz. This is caused primarily by the service based interface, which can be considered much slower than AMCL's topic based interface.

Combining the planner log from Listing 7 with the packet sizes displayed in Table 3, results in an average data transfer rate of 442 Bps (Bytes/second) for one combined localization and navigation step (based on the earlier concluded update rate of 0.1 Hz). If the update frequency of the service interface can be improved, and communication updates can be scaled up to a rate of 30 Hz (comparable to that of AMCL), an average data transfer rate of 130 KBps (KiloBytes/second) will be obtained.

For dynamic look-and-move visual servoing applications [32], that typically require point-cloud and image update rates of ∼30 Hz [14], the transferring of data requires a significantly larger amount of communication bandwidth. If the Kinect service is called at 30 Hz, this will result in a data transfer rate of 61.5 MBps (MegaBytes/second). For current wireless router protocols, such as wireless B,G and N, these speeds can not be achieved, as their maximum data transfer rates under normal conditions are 1.4, 6.8 and 31 MBps, respectively. This means that for visual servoing purposes, the currently used interfacing methods are not suitable.

Computational efforts have also been logged on both client robots, and CPU usage does not exceed 4 % during navigation (both are Intel I5 Quad-Core processors). Only when the Kinect service is called upon, CPU usage increases temporarily to 170 % (distributed over 2 cores).

5 Conclusions and future work

The work described in this article presents a centralized control architecture, used for the time-optimal allocation of multiple robots. A first experiment is conducted that, although in a very first basic form, indicates the functional success of this first implementation and the usability of such an architecture. For future work, there are however a few enhancements that can be made to the current implementations and design choices.

A first remark points towards the presented concept of 'centralized' control. Although our intention is to perform all computations on the Cloud, for the reasons given in the Introduction, the algorithms used for high-rate motion control (with >1000 Hz update rates) still run on-board of the robots themselves. The reason for this is that existing WiFi communication protocols are not yet capable of providing the required real-time performance guarantees hat these controllers demand [27]. Therefore in that sense, the architecture is not fully centralized, i.e. some control 'decisions' are still being made locally on the robot. We assume however, that with future improvements of WiFi real-time capabilities such as those proposed in [63], even high-rate motion controllers can be deployed on the Cloud, and that also their functional implementations (such as those used for position, velocity and force control) can be run there and successfully reused for a multitude of connected platforms and applications.

The goal based feature of HTN planning is currently not used, but can be implemented by mapping the user request onto a parametrized goal state, as opposed to mapping it onto a parametrized task, as is done currently. In the current experiment, there also was no optimization between multiple plans, as there was only one available plan. The experiment nevertheless demonstrates the optimization over parameters, as it contained two robots and two drinks. Optimization over plans is demonstrated identically, if multiple plans are added that represent the same task. Adding multiple plans can also be used to describe similar tasks for robot classes that require different flowcharts, such as for e.g. 'navigation'. Identification of the correct plan based on robot capabilities can be accomplished by using the 'capability matching' feature as described in Sect. 3.4.1, which can be applied to not just primitive actions, but also to composite tasks as a whole.

For this robot capability matching a first prototype was included, that allows a straightforward modeling of required

[10] http://wiki.ros.org/amcl.

components in OWL-S as static action pre-conditions. Annotating these properties based on the capabilities found on a robot, was achieved by manual design of the 'robot' knowledge ontology. These robot descriptions can however also be derived from robot configuration files already deployed on the robot, such as those used for arm- and base-navigation. Work in this direction has been done by Kunze, in the Semantic Robot Description Language (SRDL) [37].

The RoboEarth knowledge repository, as it is used now, merely serves as a storage container for static pre-designed OWL-DL descriptions on tasks, and task related class information (such as robots, objects, environments). Currently, these descriptions can be queried for, but not dynamically altered. An addition to the system can be to improve upon the stored descriptions, either by learning new semantically annotated concepts such as done in [55] or by generating them, through the use of goal based planning methods. By using either one of these methods, new descriptions can be created and inferior or outdated ones (e.g. tasks that by the lack of a better comparable task are considered optimal) can be updated or discarded entirely.

What should be further noted, is that the current interface to the robot components and the computational algorithms running on the RoboEarth Cloud Engine using ROS services, is not as fast as required for certain procedures. As can be seen in the OWL-S process flow for 'Navigate' (Fig. 12b), is that with every cycle the subtask 'Localize' is called upon, which computes the robot's pose by requesting its laser-scanner data. For standard ROS navigation architectures this laser-scanner data is broadcasted over ROS-topics at 30 Hz, whereas with the service call implementation only a rate of approximately 0.1 Hz can be achieved. This results in having to stick with very slow movements of the robot (<1 cm/s for translational, and <0.01 rad/s for angular speeds), as otherwise its recursive location estimation will 'get lost'. As concluded in Sect. 4.4, these low update rates combined with high bandwidth requirements for the Kinect data, make this system currently impractical for dynamic look-and-move visual servoing applications. With the rise of new Wireless protocols, such as 802.11ac, new bandwidth standards will be achieved that go beyond 160 MBps. Together with a redesign of how the current service call interface is made will make it possible to communicate the required data in real-time

In addition to a change in the current service call interfacing, faster execution times can be achieved by caching the initialization phase of a primitive action if the same input parameters are used. If called upon an algorithm which is deterministic, even the final outcome can be cached, and computed significantly faster using e.g. a look-up table. These look-up tables are then subsequently also stored and expanded within the existing RoboEarth database.

With respect to user friendliness a point of remark should be pointed towards the OWL-S editor plug-in for Protégé, that was used to model the required control procedures. The plug-in shows quite some stability issues, resulting in frequent Protégé crashes and unresolvable modeling anomalies. This translates to a low user friendliness of the editor, and therefore needs to be solved by inspecting and debugging the plug-in. A secondary desire here, could be to upgrade the plug-in to be used in the current Protégé version (4.3) as this version, as opposed to version 3.5 used for the plug-in, supports easier modeling of classes, object properties and cardinality restrictions.

The return code of a primitive action was currently set to a boolean true or false value, indicating the success or failure of an action. However, to invoke dedicated error recovery tasks capable of dealing with specific action failures, return codes identical to those already used in several open-source arm and base navigation libraries,[11] can be used. As the planning layer and the execution module run in a continuous thread, see Sect. 3.3.2, these recovery behaviors can be invoked directly upon receiving of an error return code. Identical to the selection of regular plans, also the selection for error recovery plans can be optimized with respect to time, and can include other robots if necessary. An example can be the re-opening of a door by a nearby robot, if a person or other entity accidentally closed it.

A final point that needs to be addressed, is the lack of dynamic state representations in the current architecture. Although the robot position was dynamically updated in the task planning component as an internal state variable, currently there is no mechanism available to track objects over time, and to associate incoming measurements with previously identified objects. For this a world model representation using object tracking and data association algorithms, such as the ones described by Elfring [19] and Safiotti [41], should be promising additions.

Acknowledgments The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement number 248942 RoboEarth.

References

1. Alami R, Chatila R, Fleury S, Ghallab M, Ingrand F (1998) An architecture for autonomy. Int J Robot Res 17(4):315–337
2. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, Zaharia M (2009) Above the clouds: a berkeley view of cloud computing. Tech Rep, Berkeley

[11] See http://docs.ros.org/api/moveit_msgs/html/msg/MoveItErrorCodes.html.

3. Arumugam R, Enti VR, Liu B, Wu X, Baskaran K, Foong FK, Kumar AS, Kang DM, Goh WK (2010) Davinci: a cloud computing framework for service robots. In: IEEE international conference on robotics and automation, pp 3084–3089
4. Au TC, Ilghami O, Kuter U, Murdock JW, Nau DS, Wu D, Yaman F (2011) Shop2: an htn planning system. CoRR
5. Baillie JC (2004) Urbi: towards a universal robotic body interface. In: Humanoids, IEEE, pp 33–51
6. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43
7. Blum A, Furst ML (1997) Fast planning through planning graph analysis. Artif Intell 90(1–2):281–300
8. Broekstra J, Kampman A (2006) Serql: an rdf query and transformation language. Semantic Web and Peer-to-Peer, pp 23–39
9. Broekstra J, Kampman A, van Harmelen F (2002) Sesame: a generic architecture for storing and querying RDF and RDF Schema. In: International Semantic Web Conference, Springer Verlag, Sardinia, Italy, Lecture Notes in Computer Science, vol 2342, pp 54–68
10. Brooks A, Kaupp T, Makarenko A, Williams S, Oreback A (2005) Towards component-based robotics. In: IEEE international conference on Intelligent Robots and Systems, pp 163–168
11. Bruyninckx H (2001) Open robot control software: the orocos project. In: IEEE international conference on robotics and automation, pp 2523–2528
12. Bylander T (1994) The computational complexity of propositional strips planning. Artif Intell 69:165–204
13. Chinnici R, Moreau JJ, Ryman A, Weerawarana S (2007) Web services description language (wsdl) version 2.0 part 1: Core language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626. <!– Wrong Number: [l]20 –>
14. Cho H (2003) Opto-mechatronic systems handbook: techniques and applications. CRC Press, Abingdon
15. Cohen B (2013) Paas: new opportunities for cloud application development. IEEE Computer 46(9):97–100
16. Crick C, Jay G, Osentoski S, Jenkins OC (2012) Ros and rosbridge: roboticists out of the loop. In: Yanco HA, Steinfeld A, Evers V, Jenkins OC (eds) HRI, ACM, pp 493–494
17. Dean J, Ghemawat S (2004) Mapreduce: simplified data processing on large clusters. OSDI, p 13
18. Elenius D, Denker G, Martin D, Gilham F, Khouri J, Sadaati S, Senanayake R (2005) The owl-s editor—a development tool for semantic web services. In: Gmez-Prez A, Euzenat J (eds) ESWC, lecture notes in computer science, vol 3532. ESWC, Springer, Berlin, pp 78–92
19. Elfring J, Van Den Dries S, Van De Molengraft MJG, Steinbuch M (2013) Semantic world modeling using probabilistic multiple hypothesis anchoring. Robots Auton Syst 61(2):95–105
20. Erol K (1996) Hierarchical task network planning: formalization, analysis, and implementation. PhD Thesis, University of Maryland at College Park, College Park, MD, USA, uMI Order No. GAX96-22054
21. Erol K, Hendler J, Nau DS (1996) Complexity results for htn planning. Ann Math Artif Intell 18(1):69–93
22. Fikes RE, Nilsson NJ (1971) Strips: a new approach to the application of theorem proving to problem solving. Tech Rep 43R, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, sRI Project 8259
23. Gerkey B, Vaughan R, Howard A (2003) The player/stage project: tools for multi-robot and distributed sensor systems. In: 11th international conference on advanced robotics (ICAR 2003), Coimbra, Portugal
24. Ghallab M, Howe A, Knoblock C, Mcdermott D, Ram A, Veloso M, Weld D, Wilkins D (1998) PDDL—the planning domain definition language
25. de Giacomo G, Lespérance Y, Levesque HJ (2000) Congolog, a concurrent programming language based on the situation calculus. Artif Intell 121(1–2):109–169
26. Giacomo G, Lespérance Y, Levesque HJ, Sardina S (2009) IndiGolog: a high-level programming language for embedded reasoning agents. Multi-agent programming. Springer, Berlin, pp 31–72
27. Gupta RA, Chow MY (2010) Networked control system: overview and research trends. IEEE Trans Indu Electron 57(7):2527–2535
28. Ha YG, Sohn JC, Cho YJ, Yoon H (2007) A robotic service framework supporting automated integration of ubiquitous sensors and devices. Inf Sci 177(3):657–679
29. Hartanto R (2011) A hybrid deliberative layer for robotic agents—fusing DL reasoning with HTN planning in autonomous robots, lecture notes in computer science, vol 6798. Springer, Berlin
30. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) Swrl: a semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium
31. Hunziker D, Gajamohan M, Waibel M, DAndrea R, (2013) Rapyuta: the RoboEarth cloud engine. Proceedings of the IEEE international conference on robotics and automation (ICRA). Karlsruhe, Germany, pp 438–444
32. Hutchinson S, Hager G, Corke P (1996) A tutorial on visual servo control. IEEE Trans Robot Autom 12(5):651–670
33. Jackson J (2007) Microsoft robotics studio: a technical introduction. Robot Autom Mag IEEE 14(4):82–87
34. Juarez A, Bartneck C, Feijs L (2011) Using semantic technologies to describe robotic embodiments. In: Proceedings of the 6th international conference on human–robot interaction. ACM, New York, NY, USA, HRI '11, pp 425–432
35. Kelly RF (2008) Asynchronous multi-agent reasoning in the situation calculus. Phd, The University of Melbourne
36. Kim JH, Jeong IB, Park IW, Lee KH (2009) Multi-layer architecture of ubiquitous robot system for integrated services. Int J Soc Robot 1(1):19–28
37. Kunze L, Roehm T, Beetz M (2011) Towards semantic robot description languages. IEEE international conference on robotics and automation (ICRA). Shanghai, China, pp 5589–5595
38. Lemaignan S, Ros R, Msenlechner L, Alami R, Beetz M (2010) Oro, a knowledge management platform for cognitive architectures in robotics. In: IROS, IEEE, pp 3548–3553
39. Levesque H, Reiter R, Lespérance Y, Lin F, Scherl R (1997) Golog: a logic programming language for dynamic domains. J Log Progr 31(1–3):59–83
40. Lim GH, Suh IH, Suh H (2011) Ontology-based unified robot knowledge for service robots in indoor environments. IEEE Transac Syst Man Cybern 41(3):492–509
41. Loutfi A, Coradeschi S, Saffiotti A (2005) Maintaining coherent perceptual information using anchoring. In: IJCAI-05, proceedings of the nineteenth international joint conference on artificial intelligence, Edinburgh, Scotland, UK, July 30–Aug 5, 2005, pp 1477–1482
42. Marco DD, Tenorth M, Hussermann K, Zweigle O, Levi P (2013) Roboearth action recipe execution. In: Lee S, Yoon KJ, Lee J (eds) Frontiers of intelligent autonomous systems, studies in computational intelligence, vol 466. Springer, Berlin, pp 117–126
43. Martin D, Paolucci M, McIlraith S, Burstein M, McDermott D, McGuinness D, Parsia B, Payne T, Sabou M, Solanki M, Srinivasan N, Sycara K (2005) Bringing semantics to web services: the owl-s approach. In: Semantic Web Services and Web Process Composition, Lecture Notes in Computer Science, vol 3387, Springer, Berlin, Heidelberg, pp 26–42
44. Mastrogiovanni F, Sgorbissa A, Zaccaria R (2009) Context assessment strategies for ubiquitous robots. In: 2009 IEEE international conference on robotics and automation, ICRA 2009, Kobe, Japan, May 12–17, 2009, pp 2717–2722

45. Mastrogiovanni F, Sgorbissa A, Zaccaria R (2010) From autonomous robots to artificial ecosystems. In: Handbook of ambient intelligence and smart environments. Springer, Berlin. pp 635–668
46. Mastrogiovanni F, Paikan A, Sgorbissa A (2013) Semantic-aware real-time scheduling in robotics. Robot IEEE Trans 29(1):118–135
47. McCarthy J (1983) Situations, actions, and causal laws. Tech Rep, Memo 2, Stanford Artificial Intelligence Project, Stanford University
48. Michel O (2004) Webots: professional mobile robot simulation. Int J Adv Robot Syst 1(1):39–42
49. Montemerlo M, Thrun S, Koller D, Wegbreit B (2002) Fastslam: a factored solution to the simultaneous localization and mapping problem. In: Proceedings of the AAAI national conference on artificial intelligence, AAAI, pp 593–598
50. OWL Working Group W (2009) OWL 2 Web Ontology Language: document overview. W3C Recommendation
51. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng A (2009) Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software
52. Radestock M, Eisenbach S (1996) Coordination in evolving systems. Trends in Distributed systems. Lecture notes in computer science, vol 1161. Springer, Berlin, pp 162–176
53. Rhodes WL (1980) Color separation techniques. Color Res Appl 5(2):123–123
54. Richardson L, Ruby S (2007) Restful web services. Web engineering
55. Rockel S, Neumann B, Zhang J, Dubba SKR, Cohn AG, Konecny S, Mansouri M, Pecora F, Saffiotti A, Günther M, Stock S, Hertzberg J, Tomé AM, Pinho AJ, Lopes LS, von Riegen S, Hotz L (2013) An ontology-based multi-level robot architecture for learning from experiences. In: Designing intelligent robots: reintegrating AI II, Papers from the 2013 AAAI Spring Symposium, Palo Alto, California, USA, March 25-27, 2013, AAAI, AAAI Technical Report, vol SS-13-04
56. Saffiotti A, Broxvall M, Gritti M, LeBlanc K, Lundh R, Rashid MJ, Seo B, Cho Y (2008) The peis-ecology project: vision and results. In: 2008 IEEE/RSJ international conference on intelligent robots and systems, Sept 22–26, 2008, Acropolis Convention Center, Nice, France, pp 2329–2335
57. Sanderson D (2010) Programming Google App Engine—build and run scalable web apps on Google's Infrastructure. O'Reilly
58. Sgorbissa A, Zaccaria R (2004) The artificial ecosystem: a distributed approach to service robotics. In: Proceedings of the 2004 IEEE iConference on robotics and automation, ICRA 2004, April 26– May 1, 2004, New Orleans, LA, USA, pp 3531–3536
59. Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) Htn planning for web service composition using shop2. Web Semant 1(4):377–396
60. Tenorth M, Perzylo AC, Lafrenz R, Beetz M (2013) Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth Framework. IEEE Trans Autom Sci Eng (T-ASE)
61. Waibel M, Beetz M, Civera J, D'Andrea R, Elfring J, Galvez-Lopez D, Haussermann K, Janssen R, Montiel J, Perzylo A, Schiessle B, Tenorth M, Zweigle O, van de Molengraft R (2011) Roboearth. Robot Autom Mag IEEE 18(2):69–82
62. Wang V, Salim F, Moskovits P (2013) The definitive guide to HTML5 WebSocket, 1st edn. Apress, Berkely
63. Wei YH, Leng Q, Han S, Mok AK, Zhang W, Tomizuka M, Li T, Malone D, Leith D (2013) Rt-wifi: real-time high speed communication protocol for wireless control systems. SIGBED Rev 10(2):28–28
64. White T (2009) Hadoop: the definitive guide, 1st edn. O'Reilly Media Inc, Sebastopol