Open | Models Laboratory

OMiLAB-Rob

# OMiLAB Physical Objects

Dimitris Karagiannis

Christian Muck

OMiLAB®

universität wien

## Main Authors

Dimitris Karagiannis
dk@dke.univie.ac.at

Christian Muck
christian.muck@univie.ac.at

**October 2017**

Description of the *OMiLAB Physical Objects (OMiPOB)* environment

# OMiLAB Physical Objects (OMiPOB)

**Abstract**

Technological advances in robotics and artificial intelligence lead to ever increasing capabilities and complexity of human-machine interaction in the scope of the fourth industrial revolution. Conceptual modelling is one approach to reduce the complexity of (information) systems. Therefore, the OMiLAB Physical Object (OMiPOB) laboratory is engaged to find a way to ease the human-machine interaction by using conceptual models and offering an environment and an approach for performing experiments in a straightforward way. This work will introduce the virtual and physical parts of the OMiPOB environment, with the available Cyber-Physical Systems (CPS) and introduces already performed experiments.

## Table of Contents

# 1. OMiLAB Physical Objects (OMiPOB) environment

OMiPOB stands for *OMiLAB Physical Objects* and is part of the *Open Models Initiative Laboratory (OMiLAB)*. The OMiLAB is dedicated to the conceptualization, development and deployment of modelling methods. It offers a physical and a virtual space, where the physical part contains equipment and software for the development of modelling methods in a physical room and the virtual part is an online platform, where descriptions, information and tools, if available, of corresponding modelling methods can be found and used for free.

OMiPOB is a part of this environment and offers also a physical and virtual space. The OMiPOB laboratory is the physical unit and is located within the OMiLAB. It offers infrastructure and equipment for different kind of experiments with *cyber-physical systems (CPS)*. The virtual platform allows an overview of already existing experiments and offers the web services/interfaces, which can be used for experiments. The different functionalities of the infrastructure are available over these web services. Some of the services are also reachable over the internet, so that experiments can be done also from outside the physically laboratory. The basic idea of OMiPOB is an interaction between models and CPSs. The CPS should act based on models and the feedback of the robots should influence the model itself. The models offer the users an intuitive and straightforward way to interact with the CPSs and therefore improve the usability.

Within the OMiPOB environment the OMiLAB Robots (OMiROB) experiments are available. These are experiments, where CPSs in combination with knowledge engineering approaches are used to fulfil certain use cases. Therefore, the problem and the use case will be defined, after that an experiment will be implemented and the result data will be collected and analysed. The knowledge engineering approaches can be, e.g., an ontology, a rule engine, speech recognition, conceptual models and many more. The OMiROB experiments are one way to use the OMiPOB environment for experiments.

If one speaks about robots, people often have a picture of humanoid artificial actors in mind. In the context of OMiPOB and especially this description, the term robot is used in a more border meaning. If we talk over robots in this document, we mean machines, which are controlled by a computer program and are able to fulfil a certain number of tasks autonomous. This can be a humanoid robot, but also a car or a robotic arm. If they are controlled by a computer, they will be called robots in the context of this document. In which it doesn’t matter if the computer itself is embedded or it controls the robot over an external interface. Furthermore, the term CPS is used in this document. A CPS in this document is a robot with a tight connected digital representation, which is also called cyber twin. The physical and the digital part are always in a coherent state, which means that changes in one part will influence the other one as well. Another goal of using CPSs is, that they support the creation of networks by facilitating communication over the digital part of the CPS. These networks should than be able to cooperate and execute autonomous tasks. But also, the communication to external entities like other CPSs or information systems is easier through the digital representation of the robotic actors. For example, they can send and receive messages over the internet, a Bluetooth connection or an USB port. In this document the terms robot, robotic actor, actor and CPS are used synonymously, in a way that the meaning of a machine, that can be controlled with a computer using different communication methods.

Finally, it should be stated, that this description is based on the as-it-is state of the OMiPOB environment during the creation of this description. Because of the continuing development of OMiPOB it is possible, that single aspects change.

## 1.1 OMiPOB architecture

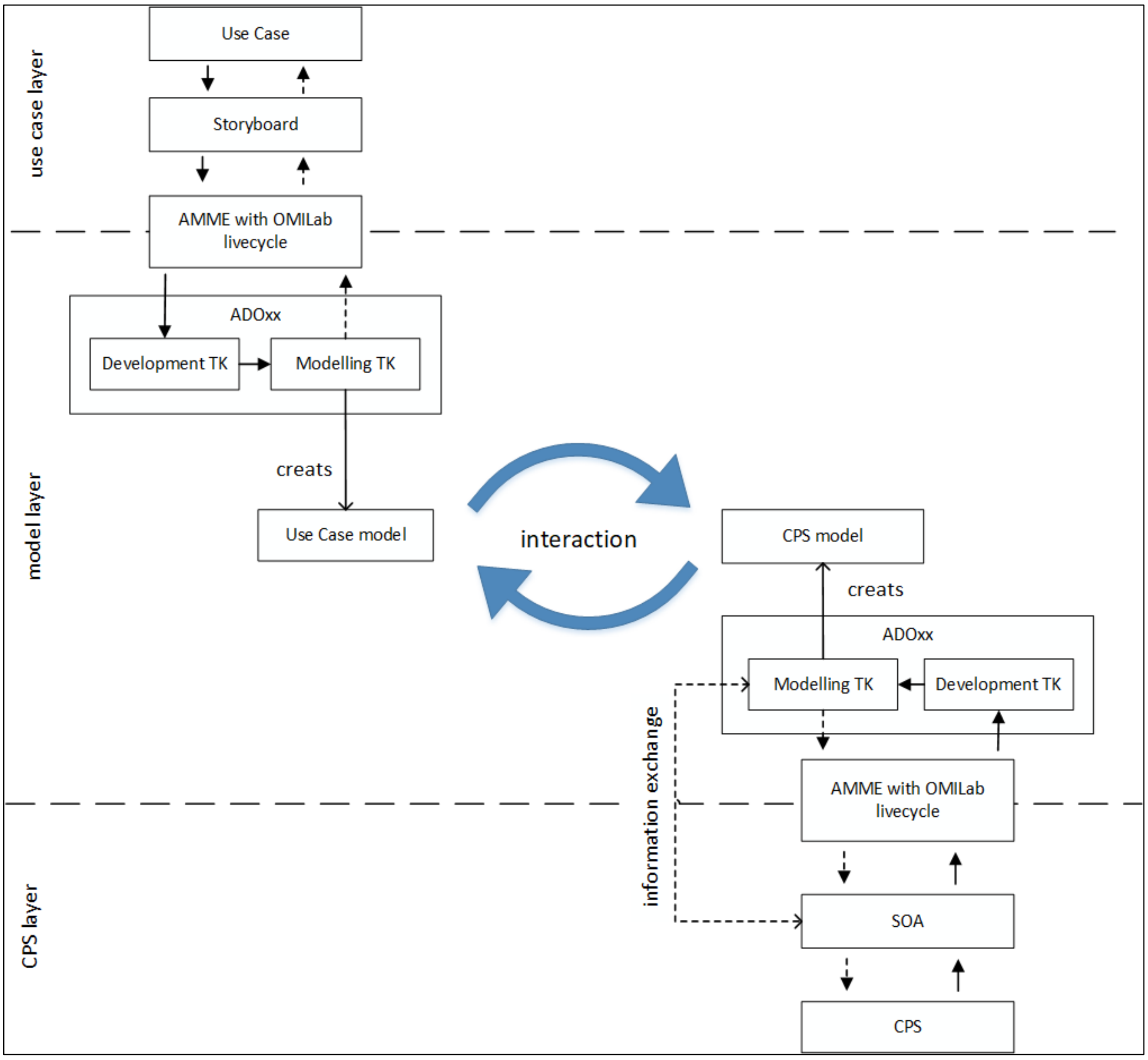
The OMiPOB environment is based on a three-layered architecture. The three layers are shown in picture 1. The top layer contains concepts of the real world, which belong to one use case. The use case will describe the excerpt of the real world, in which the experiment can be established. This layer is also divided in three abstraction levels, which contain different abstractions of the objects in the use case. The different abstraction levels symbolize the transformation from the real-world objects to the model. The

OMiPOB environment than offers an infrastructure, so that the experiment environment can be modelled and executed. The idea is, that the experiments contain scenarios from the domains of Industry 4.0 and/or *Factory of the Future (FoF)*, because so a scientific and economical relevance and mostly the need of CPSs is given. Further, this level should not be just a textual description, but a visual representation in form of a storyboard. This storyboard consists of different physical figures, that represent objects of the use case domain. The storyboards should facilitate the understanding of the domain and the chosen use case.

In the middle, there is the modelling layer, which contains models of the surrounding layers. So, on the one side there should be a modelling method for the use case, which allows a user to specify a scenario within the use case domain. On the other side, a modelling method to model different CPSs is given. This method will abstract from the implementation and the technical details of the CPSs and show what they can do. After both models are created in a modelling tool, they will interact to execute the given use case. In other words, the modelling layer contains models of the use case and the CPSs and connects these to level to execute the modelled scenario in the experimental environment.

The third layer is the bottom layer and contains the CPSs. The physical robotic actors are placed in the OMiPOB laboratory and the virtual representation can either be accessed over the internet or over a local connection to the CPSs. The type of access is depending on the CPSs and can be found in section 3. The different CPSs offer certain actions and these actions and their flow will be part of one of the models in the layer above. But the robots are also able to give feedback to the model and this information will be shown in the modelling tool itself.

To support the connection between the three layers, a well-established method is used to create the modelling methods. This method is used on both sides of modelling layer. So, once it is used to abstract the concepts of the use case and on the other hand to define the functionalities of the CPSs. This method is the *Agile Modeling Method Engineering (AMME)* with the iterative OMILab lifecycle (cf. [1]). The AMME framework defines important requirements towards the creation of a modelling method. This approach is requirement-depending and agile, so that the method can adapt to changing requirements quickly. The OMILab lifecycle defines five steps that should be taken during the creation of a modelling method. These are *create, design, formalize, develop and deploy/validate*. These phases depend on each other and should be used iterative to further improve the modelling method.



**picture 1: graphical representation of the layers and their connection**

Sources of OMiPOB architecture:

1. Karagiannis, D. (2015, October). Agile modeling method engineering. In Proceedings of the 19th Panhellenic Conference on Informatics (pp. 5-10). ACM.

## 1.2 Setting of a OMiROB use case

In this section, the tasks in the 3-layered architecture and how the different levels interact with each other, will be described. For this reason, a use case will be introduced, with a focus on the implementation in the OMiROB environment.

Through the digital transformation more and more tasks can be outsourced to information systems and CPSs. Through the increasing intelligence of the robotic actors, the given tasks are getting more and more complex. For example, they could bring children to their schools. Therefore, the CPS must know the route from home to school and when the school starts and ends. But to take care of the child, the CPS must be able to observe its surrounding and know how to react in certain situations. The goal is to give the robot this knowledge in a way, which is easy understandable for humans, so that no advanced knowledge in programming is needed. These robots and functionalities are developed from the OMiROB team, which try to implement and test new use cases in the OMiPOB environment, before an implementation in the real world. Therefore, they must define a new use case and then create a modelling method, which helps a CPS understand, how it can bring a child to school.

The OMiROB team wants to increase the attractiveness of their OMiROB environment by adding new use cases to it. The next one should allow parents to instruct robots to bring their children to school. Defining all the aspects for a new scenario in the robot supported childcare domain can be a very complex task. To enable an easy understandable way to communicate ideas within the team, story boards (e.g. SAP scenes[1]) over multiple abstractions levels are used. Story boards allow the users to visual their ideas through physical objects which they can rearrange in any order. The different abstraction levels allow the team to look at the use case at different granularity levels.

The implementation of the interfaces to the functionalities of the different CPSs and their interaction should be easy adaptable and expandable. Developers should be able to adapt the capabilities of existing CPSs or add new one to the OMiROB family. For the new use case, a CPS must be chosen, which can bring children to schools. Therefore, it must be able to move and recognise the environment. Then the interface must be analysed, if all the needed functionalities are available or if an adaptation of the interface is necessary. Therefore, a service-oriented architecture (SOA) is used to separate the huge system of many different CPS, into smaller and enclosed units, which are responsible for a specific task and communicate with each other. In the OMiROB environment separate services are implemented for the different CPSs. This not only allows to change the interface of one CPS independently from the others or easily add new CPSs with their own services, but also the combination of different CPSs, in a separate application.

The developers should be able to communicate with the robotic actors or let them communicate to each other in a straightforward way. If someone must adapt the service of a robot, the developer should not have to concern about how the robot is controlled in detail. Therefore, the different actors need a straightforward communication interface. To reach this goal, the concept of cyber-physical system (CPS) is used in the OMiROB environment. CPS consist of a digital and a physical part, which are tightly interconnected to each other. The virtual representation of the physical actor, improves the usability of the robot. Once this cyber representation is implemented, it can be reused in different application, like the services of the SOA.

Now we have on one hand the wanted use case, which should be executed and on the other hand the different CPSs and their services. The next step is to combine this two sides, in a way that allows users an easy and straight forward way to configure different instances of the use case. For example, to use different robots to guide the child between two points in the city. So, that the same method can be used by parents to tell a robot to bring their son to school and another to take their daughter to the kindergarten.  In the OMiROB environment these two sides are combined by using two groups of conceptual models. One which models the information of the use case and its environment and the other the CPSs and their capabilities. An application than let these two models interact with each other, so that the use case can be executed.

To support the creation of the modelling methods for both sides, which are the use case and the CPSs, an established approach should be used. This approach should support a fast evolving of the modelling methods, so that they can grow with their application domain. For our use case, the modelling method must contain concepts of the different tasks and their order, of the routing information, of the start- and end time of the school and many more. The models of the CPSs will be based on the offered microservices. So, they will represent the different actions the CPS is capable of and how the modelling tool can trigger the CPS to start with a given action. This action can be to move from one point to another, to warn the child about a dangerous situation or so on.  The method helps to translate this concepts into the modelling method. In the OMiROB environment the Agile Modeling Method Engineering (AMME) with the iterative

[1] https://experience.sap.com/designservices/approach/scenes

OMILab lifecycle is used (cf. [1]). This method introduces a requirement oriented and agile approach to create domain specific conceptual modelling methods.

The defined modelling methods should be readable by humans and machines. The theoretical aspect of the modelling method itself will be defined through the AMME framework with the iterative OMILab lifecycle, but after that a way must be found to improve the computer-readability. A meta-modelling platform allows a user to create a machine-readable definition of the modelling method, which can be easily changed or also shared with others. By the implementation of the new use case, this means that the important concepts of the domain, are defined in a meta-modelling platform and a modelling method is created. Therefore, a way to model the routes, the needed times and the capabilities of the robots is created. In the OMiROB environment the ADOxx platform is used. A further advantage of this application is, that a modelling tool can be generated automatically. This allows that the models can be created computer-based and therefore be adapted over time and shared with other team members.

For executing a use case, two groups of models are necessary. On the one side, the models of the domain of the use case and on the other side the models of the CPSs and their capabilities. The first group of models contains information of routes and which tasks must be done in which order. For example, to first leave the house, then go to the tram station and so on. The second group contains the functionalities, which the CPSs offer. For example, that it can move, hold the hand of the child, recognise the surrounding, react to speech commands and much more. These two models or group of models must interact with each other to get all the information needed, to execute the use case with the needed CPs. The models will be created with the ADOxx modelling toolkit, which allows the communication with external programs. These programs than handle the interaction of the models and the execution of the use case.

Sources of Setting of a OMiROB use case:

1. Karagiannis, D. (2015, October). Agile modeling method engineering. In Proceedings of the 19th Panhellenic Conference on Informatics (pp. 5-10). ACM.

# 2. Laboratory

In this chapter, the infrastructure of the OMiPOB environment will be described. The laboratory consists of a physical infrastructure and software that manages the virtual representation. The description is split into infrastructure and software. The infrastructure is the physical equipment, which is offered for experiments. The software defines the virtual space and offers interfaces to given functionalities of the OMiPOB environment.

## 2.1 Infrastructure

In this sub chapter, the hardware and physical environment of OMiPOB is described. Information from the room to the point of the different hardware, like router, server, … is given.

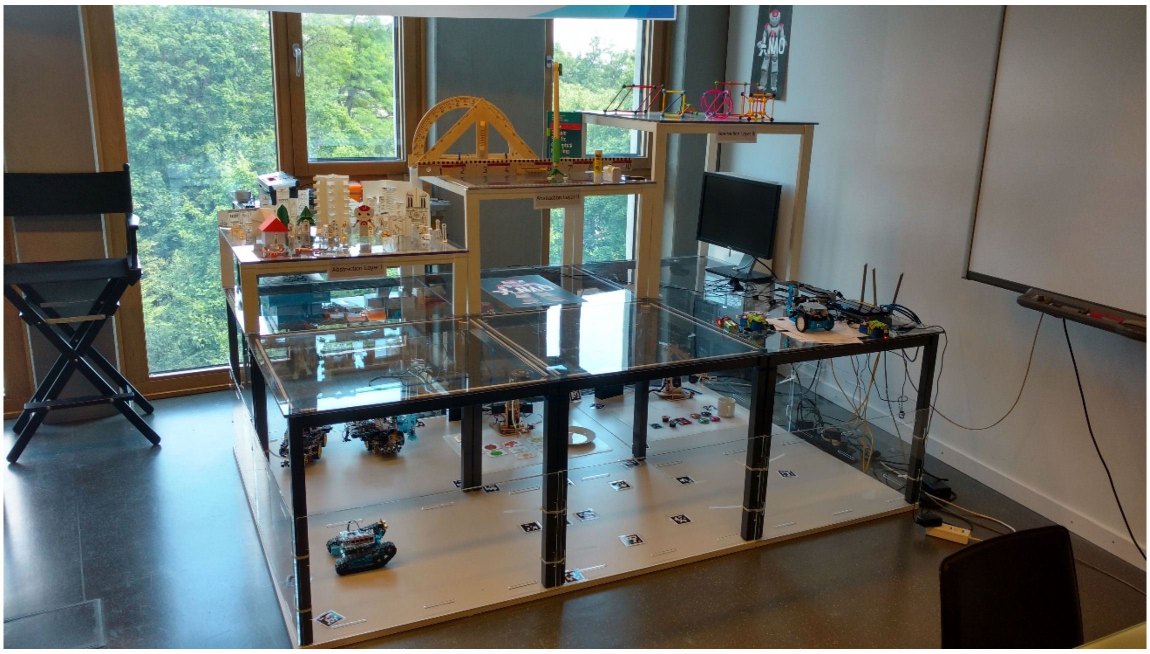
### 2.1.1 Room

For the OMiPOB laboratory a room is needed, so that the infrastructure, the CPSs and other hardware equipment can be placed and enough working space is available. One of the main parts of the laboratory is the experiment table, where most of the experiments can be performed. The experiment table can be seen in picture 2. The table itself has a surface of approximately 2x3 meters and represents the 3-layerd architecture of OMiPOB. So, the table itself consists of three levels, too. On the bottom level, which is on the floor, the experiment environment will be created. This mean that the CPSs and the experiment equipment will be placed on this level. The middle level, on normal desk height, will represent the model layer and will contain the model and the corresponding tool. The highest level represents the use case from the real world and the corresponding concepts. Further, the top level will consist of three table tops itself, which are placed on the top of the middle level. This allows the creators of an experiment to visualize their use case in different abstraction levels and symbolize the way from the real-world scenario to the model. On this abstraction levels representation of the involved objects should be placed, so that a storyboard is generated. These storyboards should consist of figures that are placed on the table tops, so that the use case can be visualized in an easy interpretable way.

The room itself offers approximately an area of 8x6 meters, for the creation of the experiments. Within this space the experiment table is placed. The additional space should be used for creating the experiment environment. For improving the group work, a conference table with eight seats is available. Near this table also two interactive whiteboards are available for improving group meetings. Within the laboratory also internet access is given.

Because on the working table experiments with several CPSs are possible, enough energy sockets and distributers should be available. For the CPSs of the OMiROB experiments, approximately ten sockets near the experiment table are needed.

Furthermore, space for the storage of additional and experiment equipment is also available.



**picture 2: experiment table**

### 2.1.2 Network and communication

Because some of the CPSs need a network connection and often internet, for the research relating to the experiment, is needed, communication networks are established in the room of the laboratory. Internet is available over the provided computers, over the eduroam-WLAN of the University of Vienna or over the OMiLAB WLAN router. The first two are mostly used for the research and other management tasks of the experiment. The last network will be used for the communication with and between the CPSs. This router is also connected to the internet, because some of the CPSs allow an interaction from outside the physical laboratory. The OMiLAB network is handled by a TP-Link TL-WR1043ND router.

For managing the connections and the different functionalities, server programs are needed. Therefore, server hardware is available within the infrastructure of OMiPOB. More details on the server applications will be given in section 2.2.2.

### 2.1.3 Cameras

Cameras are mounted around the bottom level of the experiment table, so that an overview of the experiment environment is given. Two cameras from the type Logitech HD PRO WEBCAM C920 and two from the type Microsoft LifeCam HD 3000 are mounted to the bottom layer of the experiment table. The cameras are connected to an Intel NUC NUC6i5SYH mini pc, which is also directly connected to the OMiLAB router. The Nuc controls the connection to the cameras and manages the live streams, which can be viewed over the internet.

The second use of these cameras is an automatic processing of the actions on the bottom layer of the experiment table and therefore allow a position recognition of the different CPSs. This should improve the coordination between the different robots.

### 2.1.4 omilab.tv

Omilab.tv is an online platform, within the OMiLAB environment. It is used to broadcast information of projects over the internet. For more detailed information on the application, please look at section 2.2.6.

Because this platform is based on a client-server architecture, server hardware is needed for the deployment. On the server, the *OM-Repository* and the *OM-TV* server application must be deployed and configured. Omilab.tv runs currently on an *Ubuntu 10.04.2 LTS server*. This server must also be reachable over the internet.

## 2.2 Software

This chapter addresses the different software applications and services which are used and offered within the OMiPOB environment and how their development process looks like.

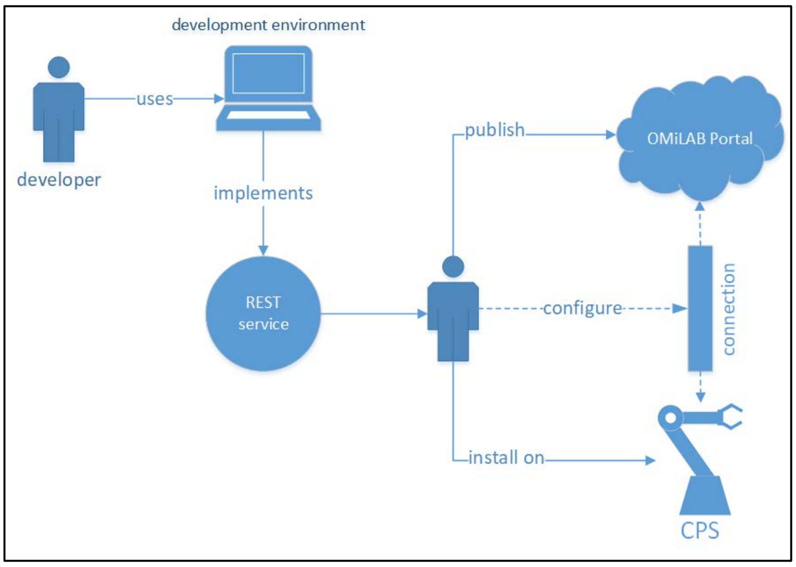
### 2.2.1 Development process

Within the OMiPOB environment, REST-services for different *Cyber-Physical Systems (CPS)* were and will be developed. A service implements an action or a group of actions, which a CPS (cf. section 3) is capable of.

The development process is triggered through the need of a new functionality for an CPSs. A developer then uses his development environment to implement the REST-service. After the service is finished, the developer takes the created code and installs it directly onto the CPS. In parallel, the service is published over the OMiLAB portal, so that users can see and use the new service over the internet. As last step, the developer configures the proxy, so that the published service is connected to the service, which is installed on the CPS. Then this new implemented function can be used also from other users, for their experiments. A graphical representation of the development process can be found in picture 3.

If there is no REST-interface implemented for an CPS, its standard communication capabilities are used. Then a developer uses his development environment to create a library, which can send commands to the standard interface of the CPS. This library then is published onto the OMiLAB GitLab server and a user, which wants to use such a CPS, can download and use the library in his own development environment. With this approach, the behaviour of the CPS must be tested in the physical OMiPOB laboratory.

For improving the collaboration of developers and enabling a version control of the software a GitLab server (cf. section 2.2.2) is used in the OMiPOB environment. Also, the standard programming language, that is used within OMiPOB is Java. Other languages are only used, if the usage of Java would be unnecessary complex.



**picture 3: development process of REST interface**

### 2.2.2 Server applications

The OMiPOB online platform is also managed by the OMiLAB server software, which is based on a service oriented architecture (SOA) and manages the internet appearance of the web page and functions that are needed within the OMiLAB. A sub part of the OMiLAB web page is the OMiPOB online platform, which is integrated in the OMiLAB online platform and uses the SOA architecture, too. This structure allows to integrate functionality and interfaces of the CPSs as a service of the platform. The web page of the OMiPOB environment can be found under the following link:

- http://www.omilab.org/psm/omipob

The link leads to the top-level site of OMiPOB, which is a sub part of the OMiLAB web page. This top-level site shows the different experiment groups, which are available. At the time of the creation of this description only the OMiLAB-Rob or OMiROB experiments were available. If one clicks on the tile, a list of all the published OMiROB experiments are shown. Each of these experiments can be clicked and a than a description and additional information will be shown to the user.

For the development and continuous improvement of the applications within the OMiPOB environment, a GitLab server is used. On this server, the code is stored and can be used and developed from different users. To facilitate the development of new experiments, the source code, for controlling the different CPSs, can be seen and downloaded from GitLab, without registration.

The GitLab group can be found under the following link:

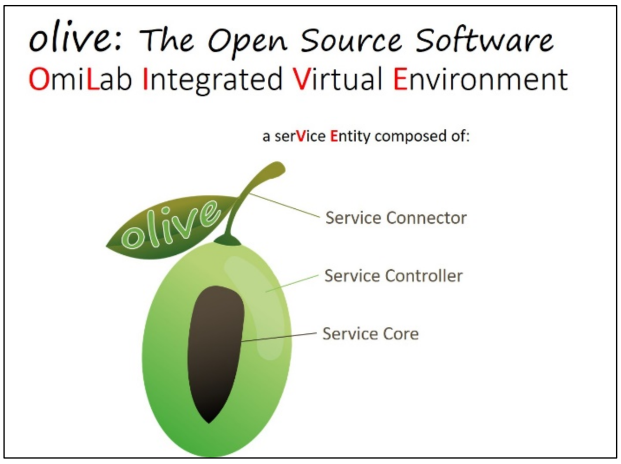
- https://gitlab.dke.univie.ac.at/OMiROB

### 2.2.3 Olive

With the basic OMiLAB portal micro service architecture it is hard to integrate or generate webpages, which functionalities are not mainly based on HTML. For example, to integrate a certain live stream, is a difficult task, where a separate connection must be opened. Other interactive application, especially with an interactive and individual presentation are tricky to create. One group of projects, which will use these capabilities, will be the OMiROB experiments. For those, often the integration of external services, like the interfaces of the CPSs or the webcams of the experimentation table, is needed. A HTML-form for triggering the robots is not a big problem, but to display the answers or implement other ways to interact with them, could be tricky. Further, with the current architecture only certain aspects of the graphical interface can be changed. The main template of the different project pages is the same for everyone.

Out of this reason the *Olive (**OMiLAB** **I**ntegrated **V**irtual **E**nvironment)* approach should advance the possibilities of the OMiLAB architecture. A service itself will thereby be separated into three parts, comparable to an olive:

- stem: service connector
- pulp: service controller/combinator
- core: service core


olive: The Open Source Software
OmiLab Integrated Virtual Environment
a serVice Entity composed of:
olive
Service Connector
Service Controller
Service Core

**picture 4: Olive approach concept**

The connector is the technology or approach, that the service uses for communication and offers to other applications as an interface. For example, a HTTP call or a WebSocket connection could be possible. So, the connector defines how the connection can be established from an external device and if this connection must be local or can be established over the internet.

The controller regulates the services, which are needed for one project and take care of the administration and management of the different services that are used for a project. The owner and/or admin can configure, what is displayed and which services are used. The user can use a connector of a controller, to use the given functionality of a project for his/her purpose. A project in the context of Olive is one bundle of technologies, approaches, services, … which are combined to reach a certain goal.

The core itself is the implementation of the actual logic of a service. This could be for example, to implement a OMiROB use case, which needs an interactive controlling tool for a CPS and/or a live stream to follow its movements. Different services can be integrated and adapted by the controller, to define one project. One service offers a specific functionality, like e.g. showing a livestream from a camera and could be ran on other computers than the controller. But the services will be used by the controller to generate a certain webpage and sent it to the user, over a given connector. But this page can contain dynamic content, which can communicate directly with a service.

For the users of the online platform, a service repository will be implemented, which will list the available services and offers information about, how they can be used and integrated in a project. Therefore, the repository will be the first place, to look, which functionalities are already available and which must be implemented, if needed. So, during the planning and designing this repository will help to estimate the effort of a project.

### 2.2.4 Live stream and OMiTag

The cameras, which are described in section 2.1.3, are used for offering live streams over the internet. These live streams show the bottom level of the experiment table with the CPSs. Therefore, the users can follow the moves of the robotic actors from outside the physical laboratory. For example, students can test their projects at home, even outside the picture 3opening hours of the university. The source code of the live streams can be found on the GitLab server under the name WebCam[2]. The streaming application runs on an Intel NUC NUC6i5SYH mini pc and this computer also forwards the streams to the users outside the local network. The nuke therefore uses a direct access reverse proxy to allow a connection from outside the university network.

But the cameras can also be used for processing the actions of the CPSs. At the time this description was created, the development team of OMiPOB was working on a locating system for of CPSs, which is using the cameras. The idea of the OMiTag[3] project is to use spared planar markers to identify the position of the CPSs and other fix points on the bottom level of the experiment table. The markers are therefore printed out and attached to the CPSs and fix points on the table. Then the application calculates the positions of the different markers, by using the cameras for identifying the spared planar markers. But not only the coordinates can be calculated, but also the rotation of the tags and therefore the rotation of the CPSs as well. For the recognition, the ArUco [4] library is used.

### 2.2.5 Modelling

Modelling is an important part of the OMiPOB idea and fills the middle layer of the three-layered architecture. The models of this layer, are often conceptual models, which represent the concepts of the given use case of an experiment and the used CPSs. These experiments will often need domain specific conceptual models and a corresponding modelling method, which needs to be tailored to the specific problem of the use case. Therefore, within OMiPOB the ADOxx[5] platform is used. This platform allows a user to define new modelling methods, consisting of a modelling language, a modelling produce and mechanisms and algorithms, in a straight forward way. An advantage of this platform is, that a modelling tool can be created instantly based on the definition of the modelling method. This also allows an easy incremental and iterative improvement of the method. The platform also allows to connect the created tool with external resource, which enables the communication with CPSs. The created tool also offers functionality for dynamically adopting created models. Therefore, data which are collected by CPSs or other external applications can be dynamically shown in the model, within the modelling tool.

### 2.2.6 omilab.tv

Omilab.tv is part of the virtual space of OMiLAB and is available for project teams within the OMiLAB-community. On this platform, project owners can open a channel for their project and publish further information on the modelling method and the corresponding tool. This channel will be broadcasted through the internet and allows an additional advertising of projects and its artefacts. Not only the modelling method information should be published, but also the team and the project itself should be introduced. Logos of institutions and research groups, group pictures and so on can be added to the channel as well. For displaying the information widgets are use. Widgets in the context of omilab.tv can be pictures or videos. In the screenshot (cf. picture 5) the widgets are the rectangles in the middle of the webpage. In the widget area of the webpage the active, the pervious and the next widgets are shown. The biggest widget is the active one and videos are only played, if they are active in that moment. On the left-hand side, the information of the project members can be displayed. The first panel contains a map with the location of the different partners of the project. Beneath that, information of the project members is displayed. The name, a picture of the person, their institution and a location map of the institution is shown. The member panels show one member at a time and switches after a few seconds. In the top left corner of the webpage the channels of the different project can be chosen, by clicking on their logo. Next to that the logo of the current chosen project is shown. The pictures of the map are automatically created by calling the Google Maps API.

[2] https://gitlab.dke.univie.ac.at/OMiROB/WebCam

[3] https://gitlab.dke.univie.ac.at/OMiROB/OMiTag

[4] http://www.uco.es/investiga/grupos/ava/node/26

[5] https://www.adoxx.org

The omilab.tv webpage also allows the creation of a news ticker for the project, which allows the displaying of additional information and news on the channel.

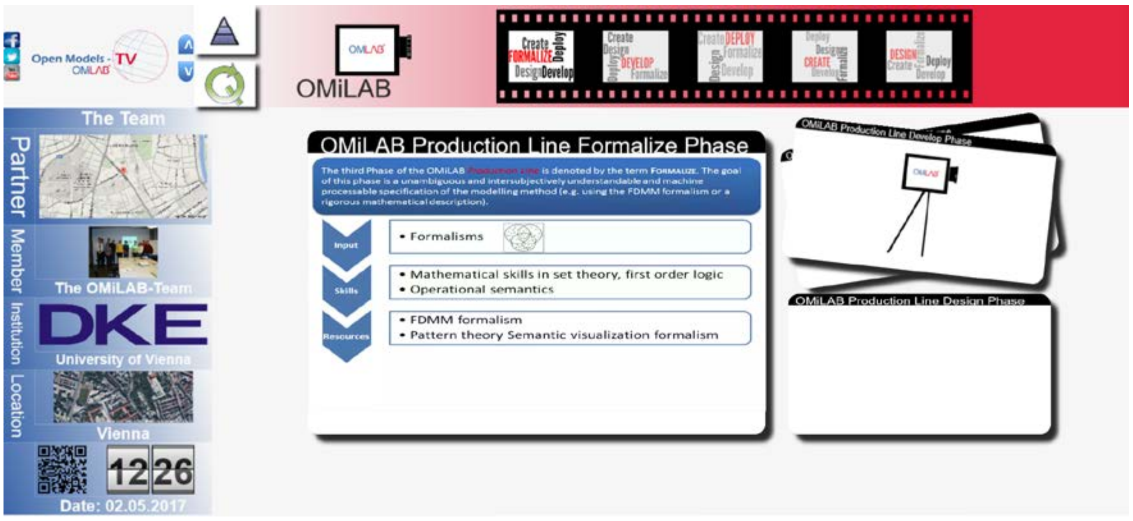
The application is based on a server-client architecture. The server backend is implemented in Java and can be divided into two servers. First the *OMi-TV* server which manages the generation of the webpages and the displaying of the information. It also handles the synchronisation of the channels, so that every user sees the same information at a time. The second is the *OM-Repository* server, which handles the saving of the data. Both server run a *Glassfish application server*[6]. For the functionality of the *OM-Repository* server additionally a databank is needed. Each server runs multiple software components, which interact with each other.

For the communication between the different components, RESTful services were implement for the different components, which are then called by other components. The exchange of the data is done in a JSON format. For example, the different information, which are shown on the user interface, are compounded after different REST calls.

On the client site HTML5 and JavaScript is used to establish the web interface.

Omilab.tv can be found under the following link:

- http://omilab.tv/



OMiLAB Production Line Formalize Phase

The third Phase of the OMiLAB Production Line is denoted by the term FORMALIZE. The goal of this phase is a unambiguous and intersubjectively understandable and machine processable specification of the modelling method (e.g. using the FDMM formalism or a rigorous mathematical description).

| | |
|---|---|
| Input | • Formalisms |
| Skills | • Mathematical skills in set theory, first order logic<br>• Operational semantics |
| Resources | • FDMM formalism<br>• Pattern theory Semantic visualization formalism |

Open Models - TV
OMiLAB
OMiLAB
The Team
Partner Member Institution Location
The OMiLAB-Team
DKE
University of Vienna
Vienna
12 26
Date: 02.05.2017
OMiLAB Production Line Develop Phase
OMiLAB Production Line Design Phase

**picture 5: screenshot of omilab.tv webpage**

## 3. Cyber-Physical Systems (CPS)

In the infrastructure section, the basic equipment and software of the OMiPOB laboratory was introduced. The focus of this section will be the classes of CPSs and the CPSs itself, which are available within the OMiPOB laboratory. These robotic actors can be used to execute different experiments in the environment of Industry 4.0, Internet of Things and so forth. To improve the identification of the capabilities and composition of an CPS, classes will be build, which contain similar CPSs. In this section, first the classification pattern will be described and subsequent CPSs, which are available in the OMiPOB environment, will be introduced.

### 3.1 Classification of CPSs

The CPSs within the OMiPOB environment, can be categorized in different classes. The classes describe different properties, which the robotic actors must fulfil or possess. For example, such properties could

[6] http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html

be, if the CPSs are able to move or not, or if they can communicate over the internet or not. The classes must no always be complementary, it is also possible that just an aspect of an CPSs is described and it possess more capabilities, than the class describes. Generally, the classes are not mutually exclusive to each other, even though for a sub set, this property can be defined. Therefore, it is possible, that one CPS is part of more than one class. An example for multiple exclusive properties could be moveable and stationary, because no CPS could be in both.

The classification can be done over different dimensions. The first distinction is, if the property refers to a capability or a component of the CPS. The capabilities can be further distinguished in actuate or sense. In the actuate category belong the capabilities of an CPS, which allows him to physically interact with its environment. In the sense category belong all those properties, which describe the perception capabilities of a CPS. The second category will also define which technologies or approaches are offered, so that the CPS can observe its surrounding, but not which sensors are used in particular.

The component dimension can be further split into software and hardware parts. In the first belong different application or services which were written for the CPSs. E.g. this could be, if there is an interface implemented, which allows a controlling over the internet or only over a local connection. In the second category belong those properties, which classify CPS by the hardware parts, which were used to build them. E.g. this can be, if a battery or a power pack is used for powering the CPS or which kind of sensors are available.

But not only the classes of CPSs will be described, but also the instances. For the different robots, properties of four categories will be captured: interaction parts, functional capabilities, software parts and hardware parts. This information will then be used to assign the CPS to one or more classes. For each instance, also a picture will be made, which show the CPS and a visualization of his capabilities and/or a use case scenario.

Classes are not limited to one aspect of the above-mentioned dimensions. The properties of different classes can be combined to define a new class, which than can help to define and find CPSs for a certain task. For example, one class could be a robot car which have a distance sensor. So, the combination of the properties moveable and sensing a distance are combined. This will be used to combine different CPSs, with similar capabilities into one group, so that they could be interchanged for each other.

The following classes of CPSs are currently available in the OMiPOB environment:

- Class1: RobotCar with basic functionality (OMiPoC2)
- Class2: RobotCar with enhanced functionality (OMiPoC1)
- Class3: RobotArm (OMiArm - DoBot)
- Class4: Car (OMiCar – mBot)
- Class5: DrawBot (OMiSpd – mDrawBot)
- Class6: Rover (OMiRov – mBot Ranger (Land Raider))
- Class7: Transporter (OMiMec – Makeblock Mecanum Wheel Robot Kit)
- Class8: Humanoid (OMiNAO)

These classes describe the capabilities of the CPSs. Because the properties came from different dimensions (actions, sensors, software, hardware), these classes are combined classes. For example, it could be defined that a class of CPSs need the ability to move over a platform and recognise the distance to different obstacles. Therefore, the class needs properties from the action and from the sense dimensions.

Also group of classes can be build, to combine different classes, which share a common aspect. For example, the groups moveable and stationary can be defined, which are also mutually exclusive and define the movement of a CPS. If the CPS can move freely on certain kinds of surfaces, it is called moveable. How it gets from A to B is not important in this context, just that it can. For example, if a car is able to drive on the ground it is moveable. But if a robot is limited to a given space, it is stationary, because he moves within fixed borders. Even if it can freely move within the space. But if a human is needed to enhance or change the location of the movement space, the CPS will be called stationary. For example, a robot is mounted to a white board, on which it can freely move, but if it needs a human to get to another white board, it is called stationary.

Two other multiple exclusive groups are live and offline, which belong to the software dimension. They define, if a CPS is controllable over the internet or not. So, if there is an interface for a CPS, which can be accessed from the internet, the CPS belongs to the live group. If the communication can only be done locally, e.g. over WLAN or a cable, it is called offline.

## 3.2 RobotCar with basic functionality

### 3.2.1 Requirements

This class of CPS belongs to the groups moveable and online. The car should be able to manoeuvre in different environments by using different sensors to recognise the world around them. The movement of the vehicle and the observation of the environment should thereby be managed from the CPS itself and different functionalities should be offered over defined interfaces.

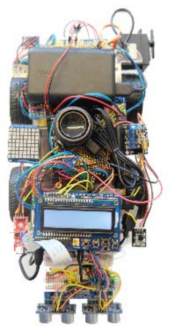
The CPSs of this class should be able to recognise the following aspects of their environment:

- measure the distance to obstacles in front of them
- measure the temperature
- recognise the ground beneath them
- differentiate between colours
- recognise sounds around them
- recognise the lighting conditions
- determine their location in space
- determine their orientation to the cardinal direction

For interacting with the environment, the cars should be able to display texts and simple patterns, play different sounds and point on objects.

Furthermore, the class belongs to the online group and therefore the offered actions and sensors should be callable over the internet. Also, the user should be able to follow the actions of the CPS from outside the laboratory.

### 3.2.2 Actor


**picture 6: PoC2**

In the OMiPOB environment, the OMiPoC2 car belongs to this class and fulfils the needed functionalities. OMiPoC2 is a self-constructed robot car, that can sense the environment, in which it is currently moving. The software of the car runs on a Raspberry Pi 2, which is mounted onto the vehicle and controls the different sensors and actuators directly. On this single-board computer also a REST-Service is implemented, which offers the functionality to third party devices. These services are also available over the internet and therefore can be used from outside the physical laboratory. For this purpose, the OMiPoC2 also implements a live stream, which shows the area in front of the robot car. This stream can also be watched over the internet.

The robot car possesses four wheels, each with a separated motor. This allows to control each of the tires individually. For displaying information, a LCD display and an 8x8 RGB LED matrix panel can be used. The LCD display allows the presentation of short texts and to change the display light colour. The LEDs of the panel and their colour can be set separately, so that simple patterns can be created. For an audible feedback, a speaker and a buzzer were mounted. The buzzer can only play a tone on different heights and the speaker can generate sounds. For pointing, the OMiCarPoC2 also possesses a forward-facing laser pointer. At the left rear end of the car, a servo motor is placed, on which small objects, like a flag, can be mounted. This motor than can rotate the mounted object in a vertical manner.

For observing the world around it, the robot car also possesses different sensors. One of the most important is the forward-facing camera. The camera is a Raspberry Pi Camera Board v1.3, which is used for offering a live stream but could also be used to process pictures from the environment of the car and identify objects. The remaining sensors are:

- four analogue and four digital line sensors, which are facing the ground at the front end of the car
- two colour sensors, one on each side
- a noise sensor, which recognises if there is a sound nearby
- two distance sensors, which sense the distance to obstacles in the front
- a temperature sensor
- an accelerometer and a gyroscope for measuring the position in space
- a compass
- a lux sensor for measuring the surrounding light

For coordinating the use of the robot car, an authentication service was implemented, which can be found on the project page of the OMiPoC2. Over this system, time slots for interacting with the CPS, can be reserved. The slots are half an hour long and a user can choose multiple slot at once. After the reservation of a slot, a token is shown to the user at the beginning of the authentication webpage. If more than one slot is chosen, the different tokens, separated by comma, are shown in the line. It is also possible to give back reserved slots, so that other users can use the CPS.

On the project web page, there is a web form, that can be used to control the robot car and read out the sensors. On a separated page of the project the live stream from the front camera can be viewed.

The project page of the OMiPoC2 can be found under the following link:

- http://austria.omilab.org/psm/content/omicarpoc2/info

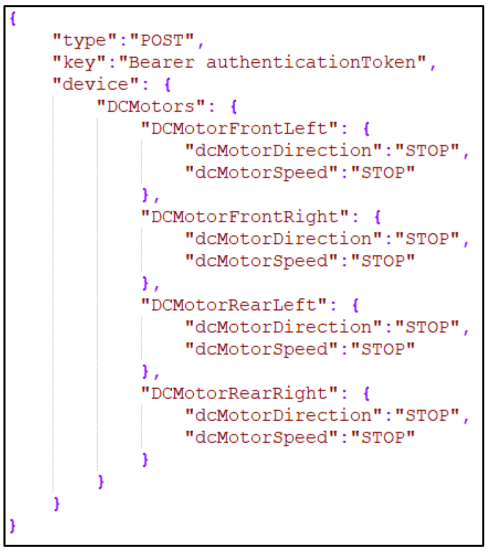
On the GitLab server the software of this CPS can be found under the name RobotCarPoC[7]. In this Java code the implementation of the sensors, actuators and the REST-interface can be found. The code is used for the OMiPoC1 (cf. section 3.3.2) and the OMiPoC2. In the following the REST-interface will be described. GET requests are used for obtaining information from the sensors and POST calls are defined for commanding the actuators. The available resources depend on the CPS, because there are differences in the composition of OMiPoc1 and OMiPoc2. The resource name for the REST call must be the name of the sensor or actuator, which should be used. For the OMiPoc2 the following sensor and actuator names are available:

- sensor:
  - IRSensors
  - TemperatureSensor
  - DistanceSensorLeft
  - DistanceSensorRight
  - LuxSensor
  - ColorSensorLeft
  - ColorSensorRight
  - NoiseSensor
  - RotGyrAccSensor
  - MagSensor
- actuator:
  - DCMotors
  - Laser
  - Display8x8
  - DisplayLCD

[7] https://gitlab.dke.univie.ac.at/OMiROB/RobotCarPoC

- Noise
- Sound
- ServoMotors

The sent requests also need parameters, which must be formatted as JSON. Every call must contain a JSON string with the JSON-keys device, key and type. Key contains the word “Bearer “ followed by a whitespace and the authentication token. The whitespace is necessary for the authentication. Type contains the request type of the call, which must be either GET or POST. For device, another JSON object must be given. This contains a key with the name of the sensor or actuator. For requests to the sensors, this is enough to get an answer but for the actuators, more parameters are needed. Therefore, the device JSON object contains additional fields or JSON objects, where the needed information is given. The values which can be added, are predefined by Java enum types[8], which contain the strings, that can be sent to the REST resource. This means that the information must be given as string to the REST service.



```
{
    "type":"POST",
    "key":"Bearer authenticationToken",
    "device": {
        "DCMotors": {
            "DCMotorFrontLeft": {
                "dcMotorDirection":"STOP",
                "dcMotorSpeed":"STOP"
            },
            "DCMotorFrontRight": {
                "dcMotorDirection":"STOP",
                "dcMotorSpeed":"STOP"
            },
            "DCMotorRearLeft": {
                "dcMotorDirection":"STOP",
                "dcMotorSpeed":"STOP"
            },
            "DCMotorRearRight": {
                "dcMotorDirection":"STOP",
                "dcMotorSpeed":"STOP"
            }
        }
    }
}
```

**picture 7: example JSON for controlling the four wheels of a robot car**

For the DCMotors actuator, which will control the four wheels, four JSON objects with the names: DCMotorFrontLeft, DCMotorFrontRight, DCMotorRearLeft and DCMotorRearRight are needed. Every one of these objects needs the two entries dcMotorDirection and dcMotorSpeed, with the string of the wanted action. An example of calling this actuator, can be seen in picture 7.

- The Laser actuator needs an entry named laserPattern.
- The Noise actuator needs an entry named noise.
- The Sound actuator needs an entry named sound.
- The Display8x8 actuator needs two entries named display8x8Shape and display8x8Color.
- The DisplayLCD actuator needs two entries named displayLCDText and displayLCDColor.
- The ServoMotors actuator needs the JSON objects ServoMotorARMBASE, ServoMotorARMREAR, ServoMotorARMFRONT, ServoMotorARMGRIPPER, and ServoMotorFLAG. Every one of these objects also needs an entry servoMotorAngle. But for the OMiPoC2 only the actuator ServoMotorFLAG is available.

The main address of the REST interface OMiPoC2 is:

- http://austria.omilab.org/omirob/omicarpoc2/rest

To this address the name of the different sensors or actuators must be added, with a slash at the beginning.

## 3.3 RobotCar with enhanced functionality

### 3.3.1 Requirements

This class of CPSs belongs to the groups moveable and online. The CPSs should be able to manoeuvre in different environments by using different sensors to recognise the world around them. Also, different actuators should be available to interact with the environment. The action of the vehicle and the observation of the environment should thereby be managed from the car itself and different functionalities should be offered over defined interfaces.

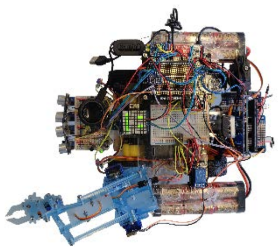
[8]https://gitlab.dke.univie.ac.at/OMiROB/RobotCarPoC/tree/master/src/main/java/org/omilab/omirob/device/carpoc/ifdevice

The CPSs of this class should be able to recognise the following aspects of their environment:

- measure the distance to obstacles in front of them
- measure the temperature
- recognise the ground beneath them
- differentiate between colours
- recognise and recording the sounds around them
- recognise the lighting conditions
- determine their position in space
- determine their orientation to the cardinal direction

For interacting with the environment, the cars should be able to display texts and simple patterns, play different sound and point on objects. Also, the possibility of lifting and holding objects should be given.

### 3.3.2 Actor


**picture 8: PoC1**

In the OMiPOB environment, the OMiPoC1 robot car belongs to this class and fulfils the needed functionalities. This CPS is a self-constructed robot car, that can observe its environment, by using different sensors. Through mounted actuators it is also able to directly interact with its surrounding. The software of the robot car runs on a Raspberry Pi 2, which is mounted onto the vehicle and controls the different sensors and actuators directly. On this single-board computer a REST-service is running, which offers the implemented functionalities over the internet. For improving the usage from outside the laboratory, also a live stream is given, which shows the environment in front of the car.

The robot car possesses four wheels, each with a separate motor. This allows to control each of the tires independent from each other. For displaying information, a LCD display and an 8x8 RGB LED matrix panel are available. The LCD display allows the presentation of short texts and to change the colour of the display light. The LEDs of the panel and their colours can be set separately, so that simple patterns can be created. For an audible feedback, a speaker and a buzzer are available. The buzzer can only play a tone on different heights and the speaker can generate sounds. For pointing at objects, the OMiCarPoC1 also possesses a laser pointer, which is mounted to the grappler base. At the right rear end of the car, a servo motor is placed, on which small objects, like a flag, can be mounted. This motor than can rotate the mounted object in a horizontal manner. On the left front end of the robot car, a robot arm with a grappler is mounted. This arm can turn its base, stretch and contract, lift and lower and open and close the grappler.

For observing the world around it, the robot car possesses different kind of sensors. One of the most important is the forward-facing camera. The camera is a Raspberry Pi Camera Board v1.3 and is used for offering a live stream over the internet, to improve the execution of the experiments from outside the laboratory. The remaining sensors are:

- four analogue and four digital line sensors, facing the ground at the front end of the car
- two colour sensors, one on each side of the car
- a noise sensor, which can recognise if there is a sound nearby
- two distance sensors, which sense the distance to obstacles in front of the car
- a temperature sensor
- an accelerometer and a gyroscope for measuring the position in space
- a compass
- a lux sensor for measuring the surrounding light

For coordinating the use of the robot car, an authentication service was implemented, which can be found on the project page of the OMiPoC1. Over this system time slots, for interacting with the CPS, can be reserved. The slots are half an hour long and a user can reserve multiple at once. After the reservation of a slot, a token is shown to the user at the beginning of the authentication webpage. If more than one slot is chosen, the different tokens, separated by comma, are shown. A reserved slot can be given back, so that another person can use it.

On the project homepage, there is a web form available, with which the robot car can be controlled and the sensor values are shown. On another site, a live stream of the camera can be watched.

The project page of the OMiPoC1 can be found under the following link:

- http://austria.omilab.org/psm/content/omicarpoc1/info

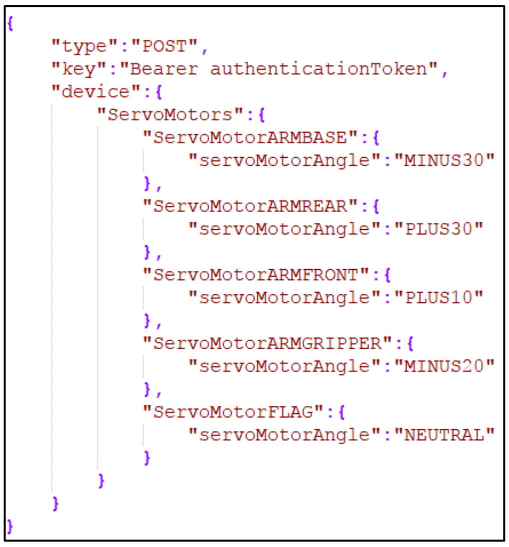
On the GitLab server the software of this CPS can be found under the name RobotCarPoC[9]. In this Java code the implementation of the sensors, actuators and the REST-interface can be found. The code is used for the OMiPoC1 and OMiPoC2. In the following the REST service will be described. GET requests are used for getting information from the sensors and POST calls are defined for invoking the actuators. The available resources are dependent on the CPS, because there are differences in the composition of OMiPoC1 and OMiPoC2. The resource name for the REST call must be the name of the sensor or actuator. For the OMiPoC2 the following sensor and actuator names are available:

- sensor:
    - IRSensors
    - TemperatureSensor
    - DistanceSensorLeft
    - DistanceSensorRight
    - LuxSensor
    - ColorSensorLeft
    - ColorSensorRight
    - NoiseSensor
    - RotGyrAccSensor
    - MagSensor

- actuator:
    - DCMotors
    - Laser
    - Noise
    - Sound
    - Display8x8
    - DisplayLCD
    - ServoMotors

[9] https://gitlab.dke.univie.ac.at/OMiROB/RobotCarPoC

The calls of the REST service also need parameters, which must be sent in JSON format. Every call must contain a JSON string with the JSON-keys device, key and type. Key contains the word "Bearer " followed by a whitespace and then the authentication token. The whitespace is necessary for the authentication. Type contains the request type of the call, which is either GET or POST. For device, another JSON object must be given. This contains a key with the name of the sensor or actuator. For the sensors, this is sufficient to get a positive answer. For the actuators, more parameters are needed. Therefore, the device JSON object contains additional fields or JSON objects, where needed information must be given. The parameter values for the different actuators, are predefined by Java enum types[10], which contain the strings, which can be sent to the REST resource. This means that the information must be given as string to the REST service.



```
{
    "type":"POST",
    "key":"Bearer authenticationToken",
    "device":{
        "ServoMotors":{
            "ServoMotorARMBASE":{
                "servoMotorAngle":"MINUS30"
            },
            "ServoMotorARMREAR":{
                "servoMotorAngle":"PLUS30"
            },
            "ServoMotorARMFRONT":{
                "servoMotorAngle":"PLUS10"
            },
            "ServoMotorARMGRIPPER":{
                "servoMotorAngle":"MINUS20"
            },
            "ServoMotorFLAG":{
                "servoMotorAngle":"NEUTRAL"
            }
        }
    }
}
```

**picture 9: example JSON for controlling the robot arm of the robot car**

For the DCMotors actuator, which will control the four wheels, four JSON objects with the names: DCMotorFrontLeft, DCMotorFrontRight, DCMotorRearLeft and DCMotorRearRight are needed. Every one of these objects needs the two entries dcMotorDirection and dcMotorSpeed.

- The Laser actuator needs an entry named laserPattern.
- The Noise actuator needs an entry named noise.
- The Sound actuator needs an entry named sound.
- The Display8x8 actuator needs two entries named display8x8Shape and display8x8Color.
- The DisplayLCD actuator needs two entries named displayLCDText and displayLCDColor.
- The ServoMotors actuator needs the JSON objects ServoMotorARMBASE, ServoMotorARMREAR, ServoMotorARMFRONT, ServoMotorARMGRIPPER, and ServoMotorFLAG. Every one of these objects also needs an entry servoMotorAngle. An example for the parameters of this kind of request, can be seen in picture 9.

The main address of the REST interface OMiPoC2 is:

http://austria.omilab.org/omirob/omicarpoc1/rest

To this address the name of the different sensors or actuators must be added, with a slash at the beginning.

## 3.4 RobotArm

### 3.4.1 Requirements

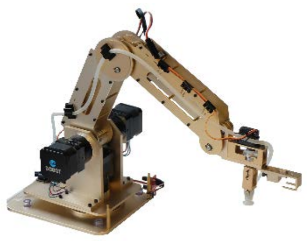
This class belongs to the groups stationary and online. The members of the class should be able to move objects within a restricted three-dimensional area with high precision and speed. Different task, like moving object or drawing something within this area should be possible, too.

### 3.4.2 Actor

In the OMiPOB environment, there exist two CPS that belong in this class. The robotic actors are Dobot Arm V1.0 robot arms and called OMiArm1 and OMiArm2. The arms are identical to each other and the usage and the configuration of them is very similar.

[10]https://gitlab.dke.univie.ac.at/OMiROB/RobotCarPoC/tree/master/src/main/java/org/omilab/omirob/device/carpoc/ifdevice

Usually a Dobot arm comes with its own controlling unit, to which a connection can be established over USB or Bluetooth. But because of problems with the acceleration sensors, this controlling unit was replaced through an Arduino board and a RAMPS board, which mange the controlling of the different motors of the arm. The Arduino board is connected to a Raspberry Pie 3, over a USB cable. On the Raspberry, a REST interface is implemented, which handles the functionalities, which are offered to the users and which can be called over the internet. The single-board computer receives the calls and forwards commands to the Arduino, which then controls the motors of the Dobot robot arm, over the RAMPS board.


**picture 10: OMiarm1 - Dobot arm v1.0 with vacuum suction cap and pen holder**

The arm itself offers movements on four axes and has a payload of 500g. Three axes are the movement of the whole arm, which means the rotation of the base, the stretching and the contracting and the lifting and the lowering of the arm. The fourth axis is the rotation of the servo motor, which can be mounted to the end of the arm and is useable with different extensions. The available extensions in the OMiPOB environment are a gripper, a vacuum suction cap and a pen or laser holder. The servo motor can be used to rotate the gripper and the vacuum suction cap.

The movements of the arms are based on a three-axes coordinate system. The x-axis is for stretching or contracting the arm, the y-axis is for moving the arm in a 90-degree angle to the x-axis and the z-axis is for lifting and lowering the arm. If coordinates are sent to the arm, the endpoint with the mounted extension, will be moved to these coordinates. Only one extension can be mounted at a time and the change must be done manually. The vacuum suction cap can be rotated through the servo motor and has a pump and a valve for grabbing an object. The gripper can also be rotated through the servo motor and the gripper itself can be opened and closed, through an own motor. It should also be mentioned, that coordinates not necessarily can be taken from one robot arm to the other. This means, that if someone has the coordinates of objects in front of one arm and places this object with the same distance in front of the other, the coordinates for the different robots to reach the objects can be different.

**picture 11: OMiArm2 - Dobot arm v1.0 with vacuum suction cap**

The movement of the arm in its coordinate system is calculated based on fixed positions of the arm. These positions are sensed through light barrier sensors, which are mounted onto the robot arm itself. If the CPS collides with an obstacle, it is possible that the coordinate system gets shifted and that found coordinates of objects doesn't fit anymore. For such cases, a reset function was implement, which uses the light barrier sensors to recalibrate the calculated position. One sensor is mounted onto the basic plate, and is the fixpoint for the rotation and the other two are fixpoints for the segments of the arms. During a reset, the arm moves until the sensor reacts and then the software knows, where the arm is in the moment.

The different actions, that the robot arms are capable of, are also offered by a REST interface, which can be called over the internet. For the two arms, the same source code is used and therefore they are offering the same functionalities. There are functions for moving the arm to a position, grabbing an item, getting the coordinates of the arm and rotate the extension of the arm.

The main addresses of the REST services of the different robot arms are:

- OMiArm1: http://austria.omilab.org/omirob/dobot1/rest
- OMiArm2: http://austria.omilab.org/omirob/dobot2/rest

The different functions for controlling a robot arm and how they can be used, will be described below. The names of the functions are also the resource names, which must be added to the URL of the basic address given above:

- **/positionXYZ:** A PUT request onto this resource, will move the robot arm to the attached coordinates. The coordinates must be formatted as JSON and contain three fields, with the labels x, y and z and the corresponding coordinates
- **/positionXYZ:** A GET request onto this resource will return the coordinates of the robot arm as JSON.
- **/positionXYZR:** A PUT request onto this resource, will move the robot arm to the given coordinates, as in /positionXYZ. But this function also requires the JSON file to contain a label r, which controls the rotation of the servo motor, which is mounted to the end of the robot arm.
- **/positionXYZR:** A GET request onto this resource will return the coordinates of the robot arm and the rotation value of the servo motor as JSON.
- **/positionXYZRG:** A PUT request onto this resource, will have the same effect as /positionXYZR. But the JSON file also needs to contain a label g, which controls the opening and closing of the gripper, if it is mounted.
- **/speed:** A PUT request onto this resource will set the speed and the acceleration value of the robot arm to the given values. This call must contain a JSON string, which contains the labels speed and accel, with the corresponding values.
- **/sequence:** A POST request onto this resource will start a sequence of actions, that will be executed by the robot arm. The call must contain a string, with the different instructions and the instructions must be separated by word-wraps. The following commands are possible: sleep, reset, move, moverg, pumpOn and valveOn. The instructions correspond to the other REST resources. The command move is like a PUT to /positionXYZ and moverg like a PUT to /positionXYZRG.
- **/grabOn:** A POST request onto this resource will activate or deactivate the pump and the valve of the arm and therefore controls, if something is grabbed or let go with the suction cap. This function is a combination of the functions /pumpOn and /valveOn. The call must contain a Boolean value, which must be true, if something should be grabbed and false otherwise.
- **/pumpOn:** A PUT request onto this resource will activate or deactivate the pump, corresponding to the given Boolean value, which must be attached to this call.
- **/valveOn:** A PUT request onto this resource will open or close the valve, corresponding to the given Boolean value, which must be attached to this call.
- **/reset:** A POST onto this resource will activate the rest function of the robot arm.
- **/emergencyStop:** A POST request onto this resource, will stop the ongoing action of the robot arm.

For using the robot arm, the user must get a token. This token reserve the arm for a given time slot, which will be half an hour long. It must be insert into the header of the http request, under the label "Authorization". The value of this header field, must be a string which starts with "Bearer " and then continues with the token. The whitespace before the token is mandatory. The timeslots can be reserved over the project pages of the robot arms, under the subpage Authentication. There a user, who is logged in, can reserve one or multiple timeslots and the tokens than will be shown in the top of the web page. The reserved time slots also can be given back, so that other users can use it again.

On the project pages, a live stream of the robot arms is offered over the internet. And a web form which sends commands to the robot arm can also be found. The project pages of OMiArm1 and OMiArm2 can be found under the following links:

- OMiArm1: http://austria.omilab.org/psm/content/omiarm1/info?view=desc
- OMiArm2: http://austria.omilab.org/psm/content/omiarm2/info?view=desc

The source code of the robot arms can be found under the following link:

- https://gitlab.dke.univie.ac.at/OMiROB/RobotArm/tree/master

Source for the chapter: RobotArm (accessed 5.4.2017):

- http://dobot.cc/dobot-armv1/specification.html

## 3.5 Car

### 3.5.1 Requirements

The CPS of this class should be small and moveable robots. The possibility to recognize changes in the ground and obstacles in front of them should be given in the standard form. Further should it be possible to easily extend the functionality of the robots by adding additional equipment. The CPSs should belong to the moveable and the online group.

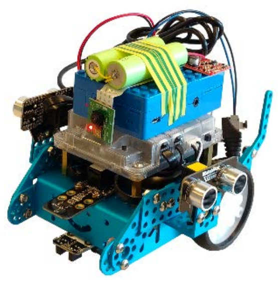
### 3.5.2 Actor

**picture 12: mBot**

In the OMiPOB environment there are three Makeblock© mBots available as CPS of this class. These are small cars with two bigger wheels in the back and one mini caster wheel in the front. For each of the big wheels an individual motor is available, so that the tires can be controlled separately. The standard version of the cars possesses a Me Ultrasonic Sensor and a Me Line Follower sensor. The first allows to measure the distance to obstacles in front of it and the second is available to sense if the car stands on a dark or a light ground. Because there are two sensor units on the Me Line Follower, the can recognise if it stands on the edge of different undergrounds.

**picture 13: mBot with display and accelerometer and gyro sensor**

The cars can be controlled by using an external device, which will be connected to the CPS over a USB-cable or via Bluetooth. In the OMiPOB environment there exists a Java library, which can be used to connect to one of the cars and allows the user to call different functions. The source code can be found on the GitLab server in the OMiROB group under the name MBotJava[11]. The given functions can control the motors of the wheels, generate a tone, activate the LED matrix and control a stepper motor. The prepared functions for the sensors can read an ultrasonic, a line follower, a light, a sound, a humiture and a gyro meter sensor. Further an input from a potentiometer or a joystick can be read, with the given Java library. In the standard version of the mBots, two motors for the big wheels, an ultrasonic and a line follower sensor are mounted. But additional equipment is available within the OMiPOB laboratory. A list can be found in section 4.2. For adding these, two ports are available on the standard version of the cars. As example, picture 13 shows car with additional hardware.


**picture 14: mBot with Raspberry Pi**

To further enhance the communication capabilities of the cars, a Raspberry Pi can be used. The single-board computes are stored in blue LEGO© boxes, which can be mounted onto the controller unit of the mBot. On the Raspberry Pi, a REST service is implemented, which is callable over the internet. The interface offers different functionalities of the car to a user, out- and inside the physically laboratory. A Raspberry Pi Camera Board v1.3 is also connected to the Raspberry Pi and faces the front of the car. This allows to get live stream, for controlling the movements of the car over the internet. The single-board computer is connected to the standard controller of the mBot over an USB cable.

Source for the chapter: Car (accessed 27.3.2017):

[11] https://gitlab.dke.univie.ac.at/OMiROB/MBotJava

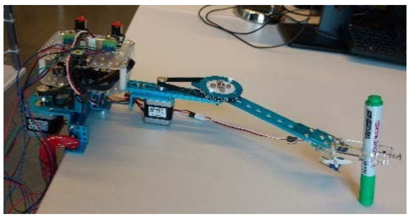
- http://www.makeblock.com/mbot-v1-1-stem-educational-robot-kit

## 3.6 DrawBot

### 3.6.1 Requirements

The CPSs of this class should be able to draw different and precise pictures or graphs onto a predefined area. The use of different pen types and surfaces should be supported. They should belong to the stationary and offline groups.

### 3.6.2 Actor


**picture 15: mDrawBot - mScara**

In the OMiPOB environment a mDrawBot kit from Makeblock© is available as CPS of this class. This kit contains two 42BYG Stepper Motor, two Me Stepper Motor Driver and a Me Orion controlling board. Further, different hardware parts like, beams, brackets and so on, are included, which enables the user to create one of four different versions of the mDrawBot. All the different types have the purpose to draw pictures on different surfaces with a pen or pen like item. The first build is the mScara, which is a robot arm, that is mounted to a flat surface, like a table. The arm can move in two dimensions to draw a picture. Also, the mCar can be build, which is a small car with a pen holder that can be used to draw pictures or graphs onto the ground. The third build is the mEggBot, which can draw on small cylinders, that can be hold and turned by the CPS itself. The last type is called mSpider and it can be mounted on a vertical surface, e.g. a white board. The pen holder can be moved by two strings, which are connected to the stepper motor. The motors are placed on the upper corners of a white board and can move the pen holder to draw a picture or a graph. Every build also can lift and lower the pen.

**picture 16: mDrawBot - mSpider**

In the OMiPOB environment the mScara was built to represent the DarwBot class. The CPS is mounted to a table, where different graphs and pictures can be drawn using the robot arm.

The DrawBot can be used by connecting an external device to it via an USB port. On this device, a program with the logic of the use case must run, which sends the different commands to the CPS. On the OMiLAB GitLab sever, functions for controlling the DrawBot are available in a Java library, in the Git project MBotJava[12]. In this code commands for moving the robot arm to a coordinate and lifting and lowering the pen are implemented. The moveto function needs x, y and f parameters. The values of x and y define the position, to which the pen should be moved and f is the value for the speed. The functions penUp and penDown don’t need any parameters because they lift and lower the pen to predefined levels. The port which is connected to the DrawBot must be specified in the code.

Source for the chapter: DrawBot (accessed 27.3.2017):

- http://www.makeblock.com/mdrawbot-kit

## 3.7 Rover

### 3.7.1 Requirements

CPSs of this class should be able to move in a more difficult terrain and the ability, to recognise the environment for improving the movements, should be given. Further the possibility to extend the

[12] https://gitlab.dke.univie.ac.at/OMiROB/MBotJava

functionality of the CPS should be supported. The CPSs should belong to the moveable and the online group.

### 3.7.2 Actor

In the OMiPOB environment the mBot Ranger kit, from Makeblock© is available to represent CPSs of this class.

The mBot Ranger kit contains a vehicle, which can be built into the three forms: Land Raider, Dashing Raptor and Nervous Bird. The first build is a vehicle with a caterpillar track system, which allows manoeuvre in more difficult terrains. The second is a car with two wheels in the front and a caster wheel in the back, which drives faster than the other forms. The last build is a two-wheeled vehicle which can balance itself during its movements. Compared to the CPSs from the Car class, the Ranger class CPS has a more powerful control unit and four ports for driven modules like servo or stepper motors, one hardware serial port for communication and five ports for sensors and other additional hardware. Therefore, it has in general more capabilities of extension, then the mBots. Also, more sensors and hardware are already available on the controlling unit itself. These are two light sensors, a sound sensor, a buzzer, a temperature sensor, a gyroscope sensor and different LEDs which can be controlled individually. Also, an ultrasonic and a line follower sensor are included in the standard kit, as mountable sensors. The additional hardware, which is available in the OMiPOB environment is listed and described in section 4.2 and can also be used with the CPSs of this class.

**picture 17: mBot Ranger**

For the Rover class of OMiPOB the Land Raider vehicle was build. In comparison to the CPSs of the Car class, the rover is bigger and can drive in more difficult terrain. Another improvement is the stronger controlling unit and the increased number of available port for additional equipment. On the OMiLAB GitLab server there are Java functions implemented, that can control the Rover and read its sensors. These functions are available in the MBotJava[13] git project. The library can be used to move its two caterpillar track systems separately, to set the built-in LEDs to a certain colour or to generate a tone. The MBotJava library also contains functions, for reading the built-in sensors. For the light sensor, the sound sensor, the temperature sensor and the gyro sensor are functions available. For the ultrasonic and the line-follower sensor the functions for the mBots can be used.

To further enhance the communication capabilities of the rover, a Raspberry Pi board can be used. The single-board computer is stored in blue LEGO© box, which can be mounted onto the controller of the CPS. On the Raspberry Pi, a REST service is implemented, which is callable over the internet. The REST service offers different functionalities of the car to a user, out- and inside the physically laboratory. A Raspberry Pi Camera Board v1.3 is also connected to the Raspberry Pi and faces towards the front of the rover. A livestream of the camera can be opened, to control the movements of the car over the internet. The single-board computer is connected to the standard controller of the mBot over an USB cable.

Source for the chapter: Rover (accessed 27.3.2017):

- http://www.makeblock.com/mbot-ranger

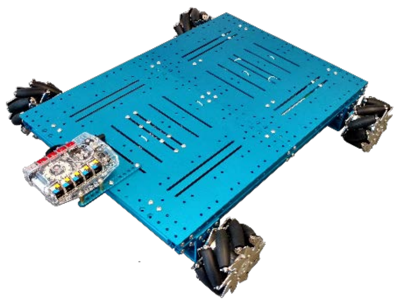
[13] https://gitlab.dke.univie.ac.at/OMiROB/MBotJava

## 3.8 Transporter

### 3.8.1 Requirements

CPSs of this class should be able to transport different kind of items and manoeuvre in a narrow space. Also, the ability to recognise different aspects of the environment should be given, so that the movements can be improved. The CPSs should belong to the moveable and the offline group.

### 3.8.2 Actor


**picture 18: Mecanum Wheel Robot Kit**

In the OMiPOB environment a Makeblock© Mecanum Wheel Robot Kit is available to represent the actors of this class. The kit contains a vehicle, that can be built in two sizes. Both are 100 mm high and the platform of one build is 360 x 386 mm and for the other one it is 496 x 512 mm. Both builds can drive 1 m/s and the smaller one has a rated load of 10 kg and the bigger one a rated load of 40 kg. Both consists of a platform with four wheels, where the motors are placed beneath the platform and the controller is mounted on a broad side of the CPS. Every wheel is actuated by a separate 36mm Encoder DC Motor and the four motors are controlled by two Me High-Power Encoder Motor Driver. The advantage of this vehicle is the usage of the Mecanum wheels. These wheels increase manoeuvrability of the vehicle, by allowing sidewise movements, without turning the vehicle itself.

In the OMiPOB environment a Me Auriga board is used as a controller for the vehicle. The board contains built-in capabilities and ten ports for extensions. The board, without any extensions, contains two light sensors, a sound sensor, a buzzer, a temperature sensor, a gyroscope sensor and different LEDs which can be controlled individually. For using extensions, four ports for driven modules like servo or stepper motors, one hardware serial port for communication and five ports for sensors and other additional hardware are available. The additional hardware, which is available in the OMiPOB environment is listed and described in section 4.2 and can be used with the CPSs of this class.

To control the CPS, which represents the actors of this class, a Java library is available on the OMiLAB GitLab server. The project of the library is called MBotJava[14]. The library isn't only able to control the motors, but also to use the built-in hardware of the Me Auriga board.

Source for the chapter: Transporter (accessed 31.10.2017):

- https://makeblockshop.eu/collections/robots/products/makeblock-mecanum-wheel-robot-kit

## 3.9 Humanoid

### 3.9.1 Requirements

CPSs of this class should be able to mimic humans and therefore enable a more natural interaction between the robotic actors and humans. They should be able to speak and move in a human way and belong to the moveable and the offline group.

[14] https://gitlab.dke.univie.ac.at/OMiROB/MBotJava

### 3.9.2 Actor

picture 19: blue NAO

In the OMiPOB environment two NAO robots are available as CPSs of this class. NAOs are 58 cm high humanoid robots, which can walk and move their different body parts like e.g. their head, arms, torso or legs. A NAO also possesses a diversity of sensors and different movements to interact with the world around it. The robot can sense, for example, if it is standing or lying and these sensors are also used for holding the balance. For a natural interaction with humans, the robot can recognise and generate sound and speech with his four directional microphones and loudspeakers. The visual recognition of the environment is based on two high resolution cameras and built-in recognition software. For example, a NAO can recognise the face of speaking person and looks at it during the interaction.

A NAO is also able to establish a connection with other devices over wireless LAN and Ethernet. The ability to access the internet over the given connection possibilities, is also given by the robot. For expanding the functionality and for the individualisation of a NAO the controlling through other devices or an installation of new software and apps directly on the robot, is enabled. Beginner programs and SDKs for languages like Python, C++, Java or JavaScript are available, for implementing own functions and services or to configure the robot.

picture 20: red NAO

A NAO itself can move his legs and arms at different points (elbow, knee, hands, …) and turn its head to look in different directions. To increase the feedback possibilities, a NAO possess LEDs on different positions, which can be controlled individually. Also, loudspeakers are available, which facilitate the interaction with humans. For recognising its environment, the robot has many different sensors like sonars on different locations, infrared sensor, position sensors, cameras, microphones, gyroscopes, accelerometers and touch sensors.

Source for the chapter: Humanoid (accessed 27.3.2017):

- https://www.ald.softbankrobotics.com/en/cool-robots/nao
- https://www.ald.softbankrobotics.com/en/cool-robots/nao/find-out-more-about-nao
- http://doc.aldebaran.com/2-4/dev/programming_index.html
- http://doc.aldebaran.com/2-4/family/nao_dcm/actuator_sensor_names.html

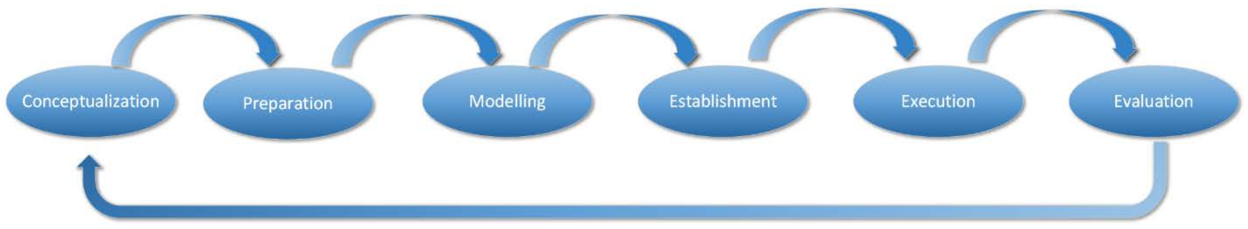
# 4. Experiments

In the sections before, the infrastructure, the software and the CPSs, which are available in the OMiPOB environment were described. This section now focuses on how the offered materials were used to establish and execute different experiments within the OMiPOB laboratory. First the general procedure, which is suggested, will be described. After that the additional experiment specific equipment will be introduced and finally already established experiments will be shown.

## 4.1 General procedure

The laboratory offers infrastructure, software and CPSs, which can be used for experiments. The available infrastructure and the corresponding software will offer a base functionality, one can use to implement a use case. The target oriented configuration and the creation of the experiment environment, will be done by using the different CPSs and individual created experiment equipment. Individual experiment equipment are those objects, which are needed for a specific use case. For example, obstacles that should be avoided or analysed by the CPSs. For some of the CPSs it is also possible, to adapt their abilities for a given use case, by adding sensors or actuators. For the execution of an experiment, the CPSs and other equipment will be placed within the bottom layer of the experimentation table. In this way, the use case environment will be created within the laboratory infrastructure and the experiment itself can be executed.

In the following a typical experiment process is shown and this should serve as a guideline for testing use cases in the OMiPOB environment.



**picture 21: typical OMiPOB experiment process**

The first step of the recommended procedure is the **Conceptualization**. Here the use case and the requirements will be defined. The description of the use case should contain the goal, the important entities and the important tasks. It can be done in only textual form or with the help of creative techniques like storyboards. The different objects of the storyboard can be put on the top layer of the experiment table. From the use case, the important entities, objects, … must be abstracted, and placed on the different abstraction levels within the use case layer of table. In the end, a definition of the experiment environment and its requirements should be available. The requirements define the aspects of the testing environment, which must be fulfilled to execute a test instance of the use case.

The next step is called **Preparation**. Here the gained requirements are compared to what is already available within the OMiPOB environment. Everything that is needed but not available should be created in this phase. For example, if an additional application or a special dataset is needed for the use case or if special equipment, like obstacles, must be built to create the experiment environment. All the needed information, objects, programs, … should be gathered and/or created in this step.

The next step is called **Modelling**. This means, if a new modelling method is needed, it will be created here. The overall goal of this step is, to create models and/or a new modelling method, if needed, upon which, the CPSs can act and execute the experiments. Therefore, computer understandable models must be generated, which contain all the gathered information from the prior steps. It is not necessarily needed that the model must be available as a file, because it would also be possible, that the modelling tool directly communicates with the CPSs to transfer the information.

Next comes the **Establishment** step, which has the goal to create the use case environment within the OMiPOB laboratory. Therefore, the needed CPSs must be adapted to meet the given requirements, if necessary. Subsequent the CPSs and the additional experiment equipment is placed in the bottom layer of the experimentation table and the initial state of the experiment will be established

Afterwards, the **Execution** step will be performed. The experiment itself will be executed and data will be collected. The information can be obtained by generating a computer readable datafile, by a visual controlling of the experiment through the user or through a direct interaction between the user and the experiment.

The last step is called **Evaluation** and analysis the gathered information from the step before. After all the data is evaluated, the user must decide if the set goal is reached or if changes in the experiment composition must be made. If the goal isn't reached, the user should go the first step and restart the procedure.

Because this is an iterative process, the experiment environment can be build incremental by identifying independent components, which are established, tested separately and combined before the execution of the whole experiment. So, the overall use case and experiment can be build part for part and then be combined in a last run of the process.

Furthermore, it should be mentioned, that some of the CPSs can be used over the internet and therefore, the different steps can be done from outside the laboratory, if no special experiment equipment is needed. Also, there are cameras, which can be used to watch the actions of the CPSs over the internet.

## 4.2 Experiment and additional equipment

For the configuration of an experiment environment, different additional equipment could be necessary. On the one hand additions to the functionality of CPSs could be needed and on the other hand the environment, in which the CPSs perform must be shaped to represent the use case.

The equipment for shaping the environment will be very diverse, so that it is important to have enough storage space and material that can be used for creating this equipment. For example, writing pads, cardboard boxes and so on could be necessary. For the adaptation of the CPSs, different sensors, actuators, … are available in the laboratory.

In the OMiPOB environment there exist different additional hardware for the Makeblock© kits, which can be added by hand, if this is necessary for the experiment. Also, the chassis of robots from Makeblock© are prepared with holes, where additional brackets can be mounted. Now a listing of the different hardware parts, and a short description is given:

- Me LED Matrix 8x16: which can be used to display simple symbols or constructs
- Me Potentiometer V1.1: which can create continuous values for regulating speed, brightness, …
- Me Rj25 Adapter V2.2: for mapping the standard Makeblock© ports to six pins, for integrating electronic modules from other manufactures
- ME RGB LED v1.1: consist out of four RGB LEDs, which can be controlled individually
- Me PIR Motion Sensor V1.1: detect motions from humans/animals from about 6 meters away
- Me Joystick v1.1: allow the controlling of robots in x-axis and y-axis direction
- Me 7-Segement Display v1.1: 4-digit display for displaying numbers and some characters
- Me Temperature Sensor-Waterproof(DS18B20): waterproofed temperature sensor with long wire and a 3-pinnconnector which is compatible with the Me Rj25 Adapter
- Me Light Sensor: detects the light intensity
- Me Sound Sensor v1.1: detects the intensity of sounds in the surrounding environment

- Me 3-Axix Accelerometer and Gyro Sensor v1.1: detects the angular rate and acceleration information

But not only additional sensors and other technical devices are available in the OMiPOB environment, but also chassis parts. For example, parts which enable to replace the wheels of a mBot with legs.

For the Dobot robot arms different extensions are available. These extensions are a gripper, a vacuum suction cap and a pen or laser holder. The gripper and the suction cap can be turned by a servo motor. The gripper also possesses a motor for opening and closing itself and for the grapping and releasing with the suction cap, a vacuum pump is available. These extensions can also be changed by hand, if this is necessary for an experiment.

Source for the chapter: Experiment and additional equipment (accessed 5.4.2017):

- http://www.makeblock.com/electronic-robot-kit-series-STEM
- http://dobot.cc/dobot-armv1/product-overview.html

## 4.3 Experiment examples

One main purpose of the OMiPOB laboratory is offering an environment, where users can define and perform experiments in the domain of Industry 4.0, FoF, and so on. The corresponding online platform also allows an online representation of the results and the composition of the experiments. The following link leads to the OMiROB online platform, where example experiments can be found:

- http://austria.omilab.org/psm/omirob

In the OMiROB environment an experiment should consists of a use case description, a knowledge engineering concept, a description of the experiment itself and its environment, the results and pictures. With knowledge engineering concepts, methods like ontologies, meta modelling, rule engines, artificial neuronal networks, machine learning, hidden Markov models and so forth, are meant. The pictures should show the experimental environment and graphs, which illustrates the use case, the knowledge engineering concept and/or the other aspects of the experiment. The use case describes a scenario from the real world and establishes the environment of the experiment. It should contain the open problem, solution approaches, which knowledge engineering concepts are or can be used and so on. This use case should not be limited to the OMiPOB laboratory, but rather give a starting point, from which the experiment, which will be established in the laboratory, is derived. This derivation then should be defined in the experiment description.

In the following, short summaries of experiments which were done by students or the OMiPOB team within the OMiPOB environment, will be given. The complete descriptions of examples are available on the OMiPOB webpage.

### 4.3.1 omiarm1 – OMiLab Robotic Arm 1

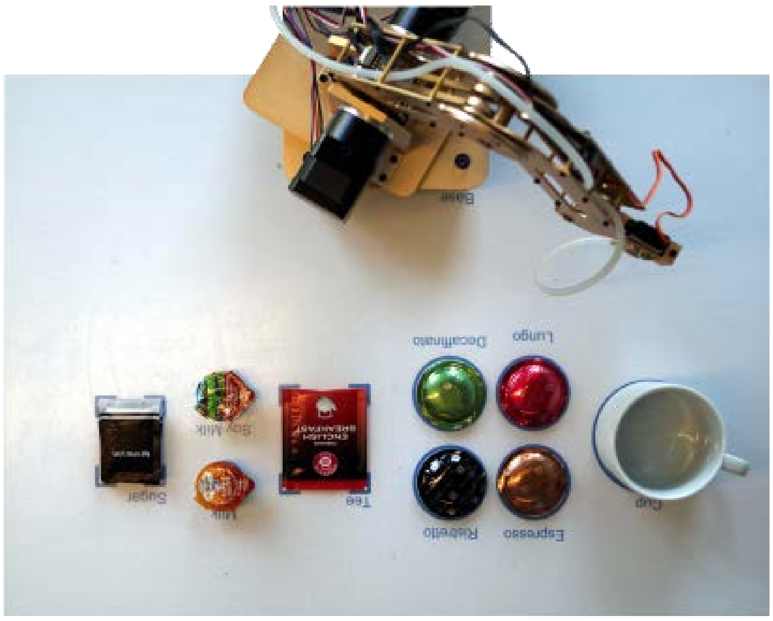
The use case of this experiment is the interaction between a CPS and real-world objects. The objects and their three-dimensional shapes should be described in a computer understandable way. Therefore, the knowledge engineering concepts fuzzy sets or ontologies should be used. The first method can be used to approximately model the shape of an object and the second offers a more exact way of describing an item.

The goal of this experiment is to use the REST interface of the OMiArm1 (cf. section 3.4) in combination with fuzzy rules or ontologies. A program should read the needed information out of the chosen concept and then call the needed REST resource with the needed parameters. As experiment environment, the creation of a coffee was chosen. Therefore, different components (cf. picture 22) of a coffee and a cup were placed in front of the OMiArm1. In the experiment, these components should be taken by the robot arm and put into the cup, by getting the position information from fuzzy rules or an ontology.

The home page contains a description of the REST interface and a web interface of the different functions, which are available. As result of this project, a video is available which shows how the robot arm prepares a coffee, where the information of the different components is read out of an ontology.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/omiarm1/info



**picture 22: experiment environment for omiarm1 - OMiLab Robotic Arm 1**

### 4.3.2 omiarm2 - OMiLab Robotic Arm 2

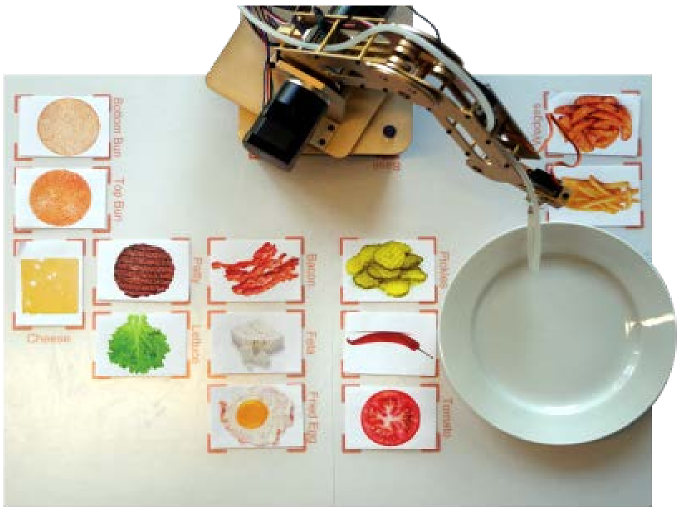
The use case of this experiment is the representation of knowledge for interacting components. This interaction often leads to an increase of the needed knowledge, then the single components would have. In this experiment, the information should be modelled as a production system and/or a process model. Production systems represent the needed knowledge in form of rules, which are fired if certain events happen within a given environment. Process models are often instances of a domain-specific modelling methods and represent a flow of actions. The focus of the models lies on prescriptive, descriptive and explanatory knowledge formalizations.

The experiment than was to use the OMiArm2 (cf. section 3.4) to build a burger. Therefore, cards, which symbolize the different ingredients, and a plate are placed in front of the robot arm and the chosen knowledge engineering concept is used to define the order in which these ingredients are put onto the plate. The program, which implements the concept, should call the REST-interface of the robot arm in the needed order. The experiment environment can be seen in picture 23.

This experiment is meant to be done by students, so there are no results available on the webpage. Thus, the home page contains a description of the REST interface and a web interface of the different functions, which are available.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/omiarm2/info



**picture 23: experiment environment for omiarm2 - OMiLab Robotic Arm 2**

### 4.3.3 omicar01 - OMiLab Robotic Car PoC 1

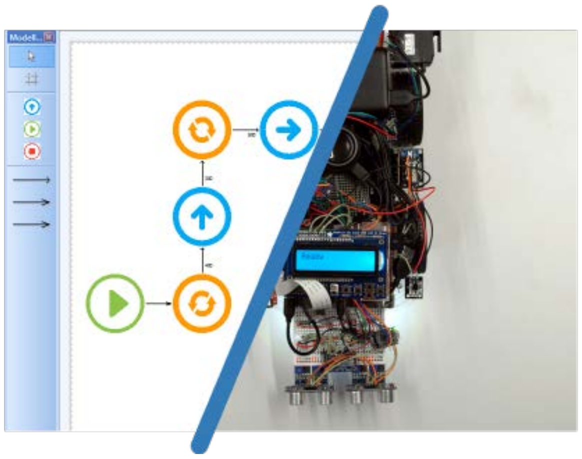
The use case of this experiment is to capture the characteristics of a system in the right abstraction level. For the modelling, the characteristics the concepts conceptual modelling and agile modelling method engineering (AMME) should be used. Conceptual modelling supports the creation of modelling methods and AMME allows to adapted modelling methods towards changing requirements through an iterative approach.

The goal of the experiment is to validate this modelling methods by using them in combination with the OMiPoC1 (cf. section 3.3) and its sensors and actuators. Therefore, a modelling method and/or a model should be generated, on which the OMiPoC1 can execute actions. After the experiment, the executed actions should be compared to the planed actions.

As a result of this experiment a video can be found on the experiment page. These videos show, how the car moves based on the model and the representation of the model is influenced by the actions of the CPS. Further, the web page contains information to the REST interface and a web interface of the different functions, which are available.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/omicarpoc1/info



**picture 24: experiment concept picture for omicar01 - OMiLab Robotic Car PoC 1**

### 4.3.4 omicar02 - OMiLab Robotic Car PoC 2

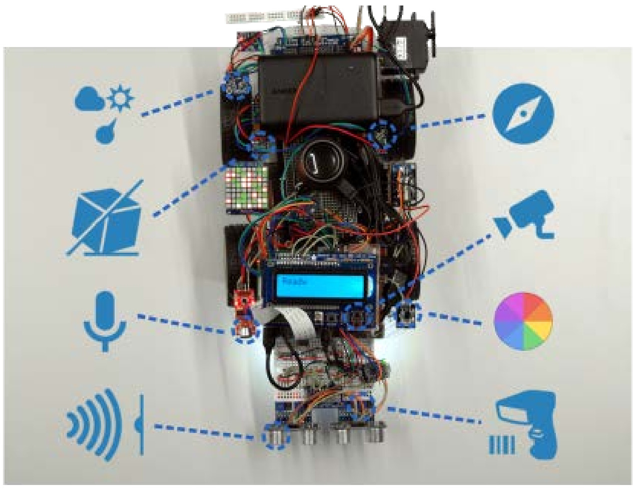
The use case of this experiment is to use different sensors to generate more information, which is more than just the sum of the sensed facts. This information gain should also be done automatically and therefore, the knowledge engineering concepts semantic networks and/or Bayesian networks should be used, to formally represent the additional knowledge, which can be gained through the combination of the sensors. Semantic networks allow to link the sensor information in a symbolic relation and Bayesian networks facilitate the generation of assumptions on posterior probabilities about hypotheses, given the sensor input.

The goal of this experiment is, to use the sensors of the OMiPoC2 robot car (cf. section 3.2) to implement sensor fusion by using semantic networks and/or Bayesian networks. The robot car offers different sensors, from which the information should be combined to produce additional knowledge about the environment of OMiPoC2.

This experiment is meant to be done by students, so there are no results available on the webpage. Thus, the home page contains information to the REST interface and a web interface of the different functions, which are available.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/omicarpoc2/info


**picture 25: experiment concept picture for omicar02 - OMiLab Robotic Car PoC 2**

### 4.3.5 XOMiAC1 - OMiLAB Robotic Arm & Car Experiment 1

The use case of this project was to simulate a delivery process from a warehouse to a production place. Therefore, an CPS must come to the warehouse, wait till the needed component is ready and then bring it to a given destination. The position of the different wares in the warehouse should be saved in an ontology.

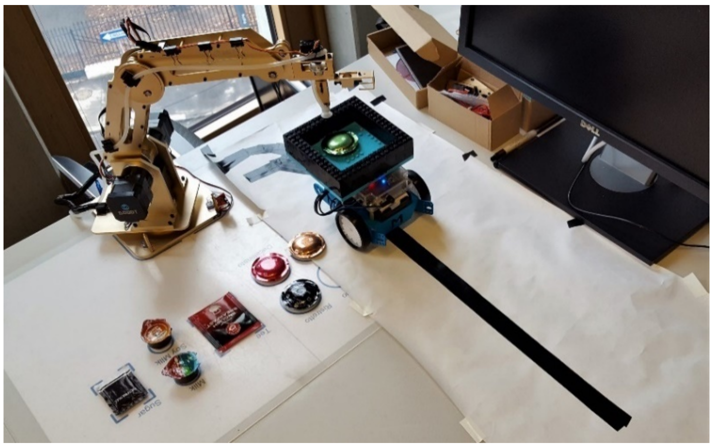
For the experiment CPSs of the classes RobotArm (cf. section 3.4) and Car (cf. section 3.5) are used. The car was equipped with a basket and delivers the components. The robot arm put the wares from the warehouse onto the car. In that way, all ingredients for a coffee should be delivered to a coffee cup. The car follows a line, which connects the load and the unloading places. The arm reads the coordinates of the ingredients from the ontology and put them onto the car. The experiment environment can be seen in picture 26. For defining the flow of the different actions, creating the ontology and to combine these two models the SeMFIS[15] tool was used.

[15] http://austria.omilab.org/psm/content/semfis/info

As results pictures and a video of the execution of the experiment is available on the experiment webpage.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomiac1/info



**picture 26: experiment environment for XOMiAC1 - OMiLAB Robotic Arm & Car Experiment 1**

### 4.3.6 XOMiArm2 - OMiLAB Robotic Arm Experiment 2

The use case of this experiment was to create a modelling method, which allows the definition of a burger composition, which a CPS cloud build, upon a model. The position of the different ingredients should be saved in an ontology. Therefore, the knowledge engineering concepts, which are used in this experiment are meta modelling and ontology.

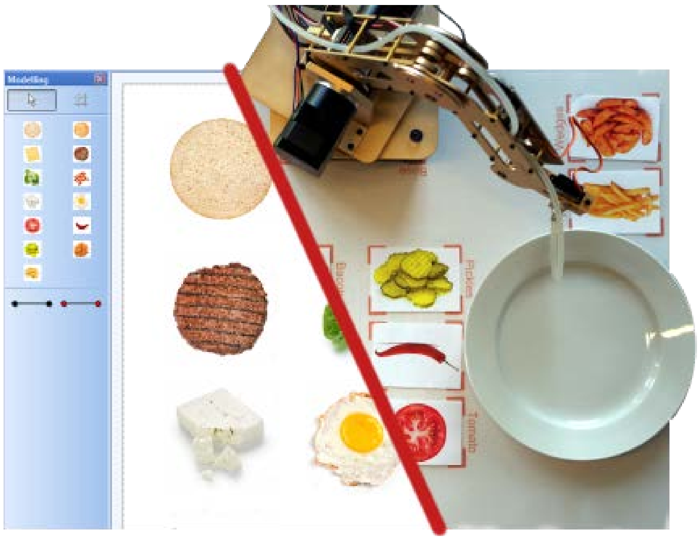
In the experiment, a CPS of the RobotArm class (cf. section 3.4) was used and the modelling method and the models were created with the ADOxx 1.5[16] platform. For defining the position of the ingredients of the burger, an owl-ontology was used. The execution of this experiment consisted of three parts. First an ontology was created. Second the Java REST client, which communicates with the robot arm was written. Third the modelling method was produced. In one model type the ingredients and the side dishes of a burger can be defined and in the other the quantity of the available ingredients can be modelled. In the modelling tool, also the token for the robot arm and the base address for the REST service can be entered. Furthermore, the tool contains two AdoScripts, which can control if enough ingredients are available and trigger the actions of the robot arm. In picture 27 the experiment environment and a part of the model can be seen.

As result screenshots and a video of the execution of the experiment can be found.

[16] https://www.adoxx.org/

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomiarm2/info



**picture 27: experiment environment for XOMiArm2 - OMiLAB Robotic Arm Experiment 2**

### 4.3.7 XOMiArm3 - OMiLAB Robotic Arm Experiment 3

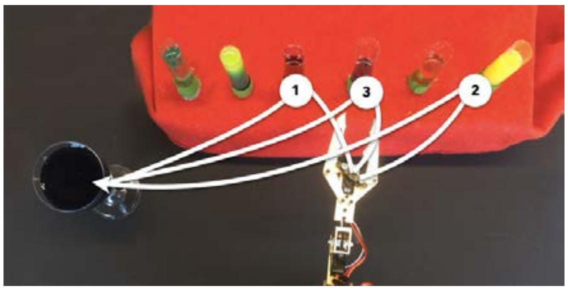
The use case of this experiment was to prepare a cocktail using a robot arm. For this also, the knowledge engineering concepts speech recognition and rule engine should be used. The composition of the cocktails should be saved in the rule engine and the user input should be possible over speech commands.

In the experiment, a CPS of the class RobotArm (cf. section 3.4) was used. This robot arm was equipped with a gripper, so that tubes, which contained the ingredients for the cocktails, can be taken. The program itself was based on a server-client architecture. The front end was a JavaScript based web interface and managed the user input inclusive the speech recognition and the backend implemented the rule engine and the communication with the REST interface of the robot arm. For the speech recognition, the W3C Web Speech API [17] was used. As experiment environment, the tubes and a cocktail glass were placed in front of the CPS. The robot arm than took the needed ingredients and filled them into the cocktail glass. The experiment environment can be seen in picture 28.

As result screen shots of the web interface, a video and pictures of the experiment execution are available on the webpage. Furthermore, the team of the experiment recognised that the gripper for the robot arm only kept close, if the arm was in motion.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomiarm3/info



**picture 28: experiment environment for XOMiArm3 - OMiLAB Robotic Arm Experiment 3**

[17] https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

### 4.3.8 XOMiCar2 - OMiLAB Robotic Car Experiment 2

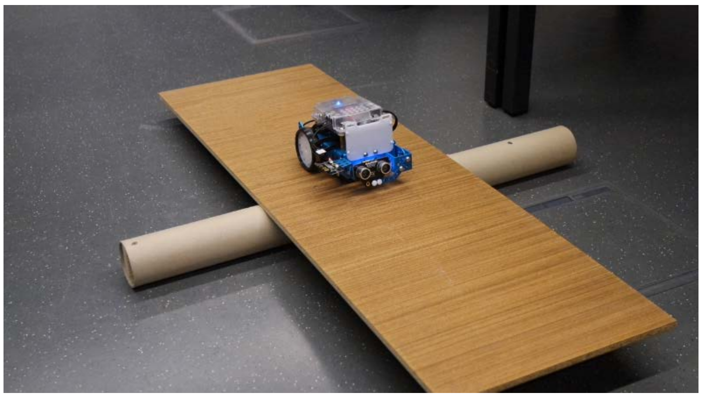
The use case of this experiment was to allow a CPS to react to fuzzy inputs of his environment. Therefore, fuzzy logic was chosen as knowledge engineering concept. Fuzzy logic allows to handle fuzzy inputs by assigning them degrees of membership to one or more sets. Based on the membership of the different sets for the input values, an output value is calculated.

In the experiment, a member of the class Car (cf. section 3.5) was used. This CPS should balance itself on a balancing board. This board was tiltable in one dimension and therefore the car had to drive forward and backward to balance itself. For sensing the angle of the balancing board, the car was equipped with a 3-axis accelerometer. The experiment environment can be seen in picture 29. For programming the fuzzy rules, the Java library jFuzzyLogic was used. With it also the sets for the input value of the accelerometer and for the output value of the speed were created. Both variables contain negative and positive values, for distinguishing on which side the balancing board is tilting or in which direction the car should move.

As result a video of the execution of the experiment can be found on the webpage. Also, the students, which conducted the experiment, mentioned as a conclusion, that it was hard to define the sets, because this required lots of trying and adapting.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomicar2/info



**picture 29: experiment environment for XOMiArm1 - OMiLAB Robotic Arm Experiment 1**

### 4.3.9 XOMiSpd1 - OMiLAB Robotic Spider Experiment 1

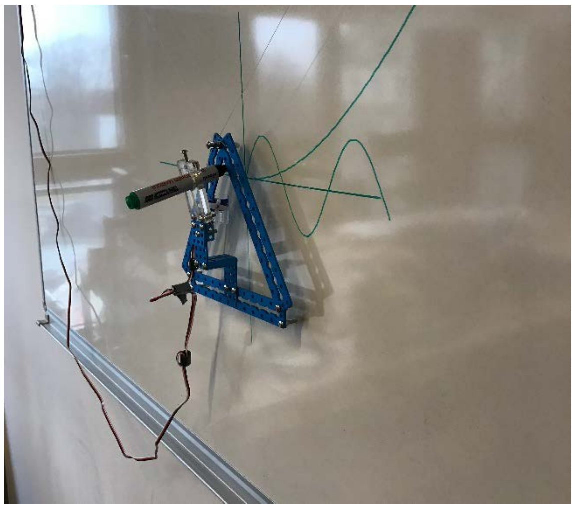
The use case of this project was to draw SVG images with the help of a CPS. As knowledge engineering concept, the concept of rule engine was chosen. The rule engine is used to process the data of the SVG images and then sends the CPS commands, depending on the given data from the picture.

For the experiment, a member of the DrawBot class (cf. section 3.6) was used and controlled from an external device. The connection was established via USB and the device ran the programme with the implemented rule engine. For finding the needed commands, the tags from the XML structure of the SVG file were parsed. After identifying a needed action, a command, using G-code standard, is sent to the robot. Instructions for moving the CPS and lifting and lowering the pen can be used to draw the picture. The robot was mounted to a white board, on which it also draws the pictures. The experiment environment can be seen in picture 30.

As result a video of the experiment execution is available on the web page. The students, who conducted the experiment, also mentioned that it was necessary to split the lines during the plotting, because otherwise the lines would get curved.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomispd1/info



**picture 30: experiment environment for XOMiSpd1 - OMiLAB Robotic Spider Experiment 1**

### 4.3.10 XOMiArm4 - OMiLAB Robotic Arm & Car Experiment 4

The use case of this experiment was, to let an CPS play a three disk puzzle towers of Hanoi game. As knowledge engineering concept constraint satisfaction was chosen. Therefore, the rules of the game were modelled by constraints and the CPS should find valid moves, which allows him to reach the goal state.

For the experiment, a CPS of the RobotArm class (cf. section 3.4) were used. The robot arm was used to move the disks, which were placed in front of it, by controlling it from an external device over its REST interface. This programme also contained the constraints, that were used for this game:

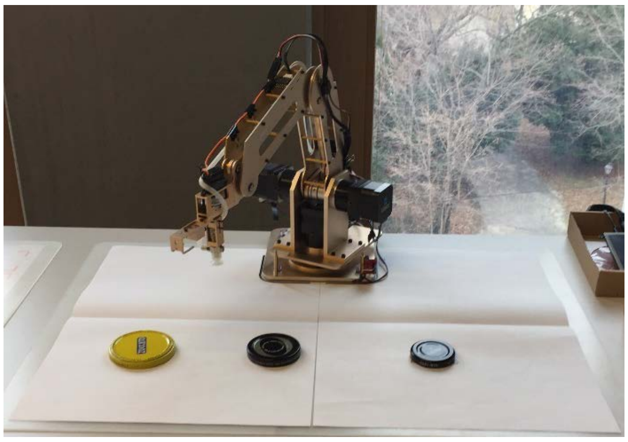
- one disk can be moved in one turn
- only the top disk of a stack can be moved
- only disks with a smaller value (e.g. size of disk) can be placed above another disk
- a disk which is moved from a place p, is not allowed to be moved back to p in the next move

The last constraint/rule is not part of the original rules, but was necessary for the experiment, so that no loops of the same actions can arise. For finding all the valid move sequences a breadth-first search in combination with constraint propagation were used. The experiment environment can be seen in picture 31.

As result a video of the of the experiment execution can be found on the webpage.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomiarm4/info



**picture 31: experiment environment of XOMiArm4 - OMiLAB Robotic Arm & Car Experiment 4**

### 4.3.11 XOMiCar1 - OMiLAB Robotic Car Experiment 1

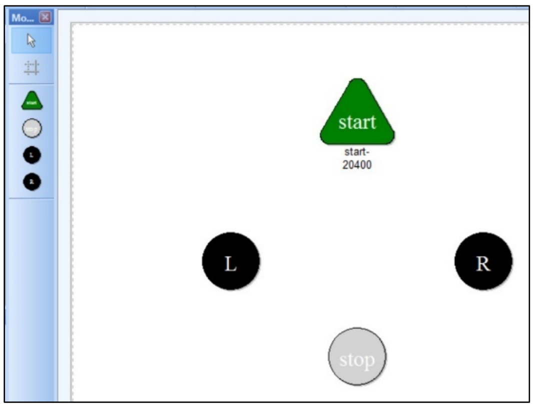
The use case of this experiment was to create a modelling method and a modelling tool, with which a CPS can be controlled. The knowledge engineering concept for this experiment was meta modelling. The modelling language should contain concepts for the actions and for the sensed information of a CPS. The actions and sensed information should be clickable and visible in a tool. The created modelling tool should be able to send commands to the robot and receive and display information from it.

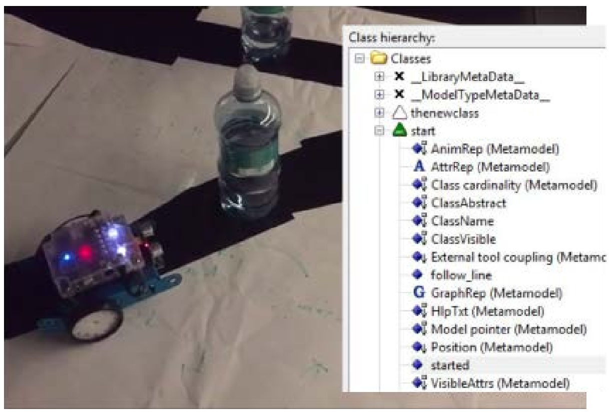
For the experiment, a member of the Car class (cf. section 3.5) was used. This car should follow a line until an obstacle occurs and subsequent stop in front of it. After the stop, this information should be shown in the modelling tool and the user can decide if the robot should drive around the obstacle on the left or the right side. After the bypass, the user can decide if the car starts again. Start, driving left and driving right are the modelled actions of the car and can be clicked in the tool for triggering the action. The stop object is not clickable but shows if the car is stopped. In picture 32 the different symbols are shown. The avoiding of the obstacle is an automatic process, after the action symbol was clicked. The communication with the car was done by Java code, which established a connection via Bluetooth and then sent the needed commands to the CPS. In picture 33 the experiment environment is shown.

As result a video of the experiment execution can be seen on the webpage.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomicar1/info





picture 32: screenshot of the modelling method

picture 33: experiment environment for XOMiCar1 - OMiLAB Robotic Car Experiment 1

### 4.3.12 XOMiArm1 - OMiLAB Robotic Arm Experiment 1

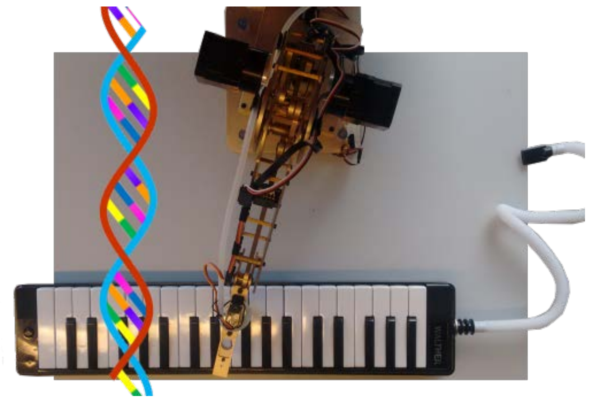
The use case of this experiment was to learn a robot, how to play simple songs on a piano. As knowledge engineering concept, the method of genetic algorithm was chosen. The genetic algorithm is used to learn a song, after the user fed the program a sequence of notes.

For the experiment, a member of the RobotArm class (cf. section 3.4) was used. In front of the robot arm a small keyboard was placed, so that the robot can play the needed tones. The experiment environment can be seen in picture 34. For the program, the different notes must be given as string, which is the starting point for learning the song. Through variables for the mutation ratio and the crossover ratio, the user can also influence the genetic algorithm. After the learning, the output was sent to the robot arm over its REST interface. Because the lack of equipment, for evaluating real produced sound, the input string was evaluated against a predefined target string.

As result a video of the experiment execution is available on the webpage.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomiarm1/info



picture 34: experiment environment for XOMiArm1 - OMiLAB Robotic Arm Experiment 1

### 4.3.13 XOMiCar3 - OMiLAB Robotic Car Experiment 3

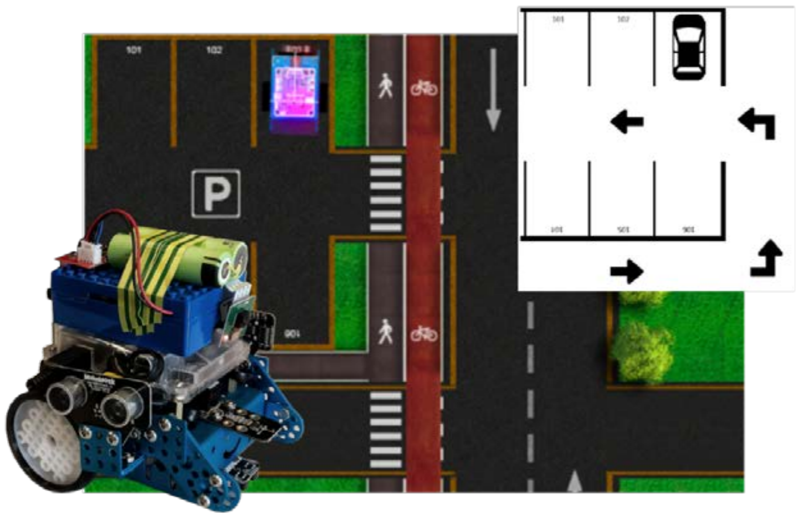
Some of the projects in the OMiPOB environment are also meant to increase the functionalities of CPS and therefore, not necessarily use a modelling method or a knowledge engineering approach. This student project was one of those and had the goal to implement a self-parking mechanism for CPS of the class Car. First the CPS must be placed on the beginning of the parking lot through other mechanisms or per hand. At the start of the parking lot, the parking function must be activated and then the CPS should find the next free parking space on its own or should drive to a user chosen parking spot. The CPS should be able to recognise if a parking space is free or not.

For the experiment one member of the Car class (cf. section 3.5) was used. The standard version of the CPS was adapted to meet the requirements of the use case. First a Raspberry Pi was used to allow a communication through a REST service. Further, two ultrasonic sensors were mounted, one on each side of the CPS. To increase the accuracy of the turns, a gyro sensor was mounted. The coordination of the CPS within the parking lot was done by marking the ground with lines. To recognise these lines, a line-follower sensor was used by the CPS. For testing and controlling the self-parking function, also a desktop application was written, which can be used to start the parking sequence, set a user wanted parking lot and receive feedback from the CPS. In the application, a model of the parking lot and a moving representation of the car can be seen. The movement of the car in the application and the CPS are depended from each other. In picture 35 one can see a concept graphic of the project. On the bottom-left corner a picture of the used CPS can be found and in the top-right corner, a model of parking lot from the desktop application is shown.

As result of the experiment, videos of the different tested versions of the experiment can be found on the project page.

The webpage of this project can be found under the following link:

- http://austria.omilab.org/psm/content/xomicar3/info

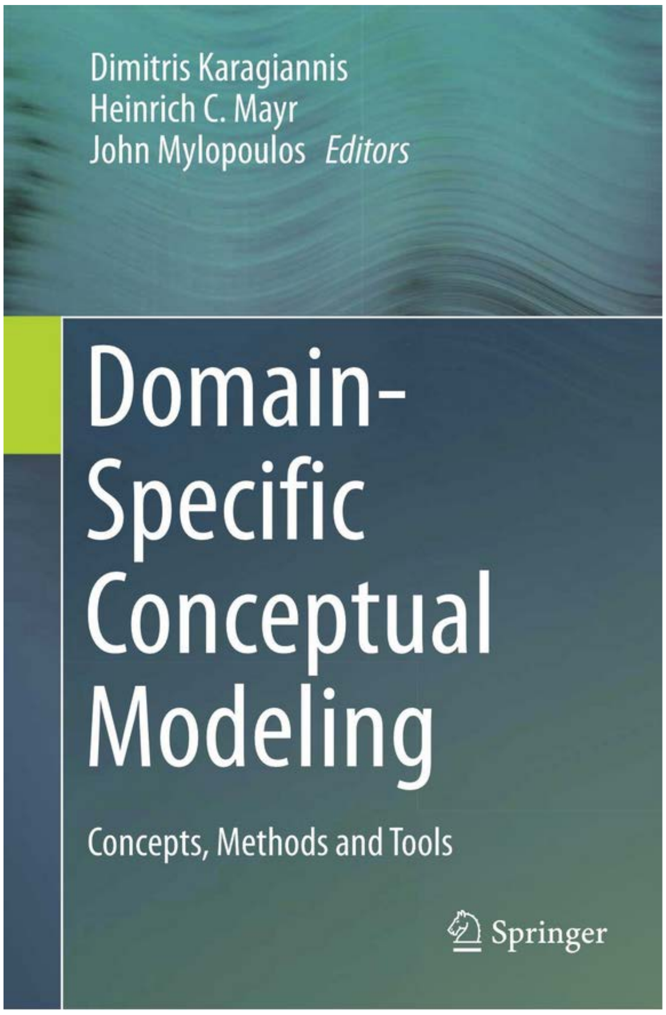


**picture 35: concept graphic for XOMiCar3 - OMiLAB Robotic Car Experiment 3**

## OMiLAB Publications:

|  |  |
|---|---|
| • **Feasibility Study** | – ISBN: 978-3-902826-00-8 |
| • **Conceptualization of i*** | – ISBN: 978-3-902826-01-5 |
| • **MM-DSL Spezifikation** | – ISBN: 978-3-902826-02-2 |
| • **OMI-LAB BOOKLET** | – ISBN: 978-3-902826-03-9 |
| • **The "IMKER" Case Study** | – ISBN: 978-3-902826-04-6 |

## New Publication:


Dimitris Karagiannis
Heinrich C. Mayr
John Mylopoulos *Editors*
Domain-Specific Conceptual Modeling
Concepts, Methods and Tools
Springer

Visit OMiLAB: www.omilab.org

Contact Person:

Elena-Teodora Miron

Email:

info@omilab.org

Address:

University of Vienna
Department of Knowledge Engineering
Währinger Straße 29
A-1090 Vienna

http://www.omilab.org/psm/omipob

ISBN 978-3-902826-06-0