FP6-004381-MACS

MACS

Multi-sensory Autonomous Cognitive Systems Interacting with Dynamic Environments for Perceiving and Using Affordances

Instrument: Specifically Targeted Research Project (STReP)

Thematic Priority: 2.3.2.4 Cognitive Systems

D5.3.3 Robot prototype learning affordances through self-experience (V2.0)

Due date of deliverable: May 31, 2007
Actual submission date v2: July 13, 2007

Start date of project: September 1, 2004 Duration: 39 months

Österreichische Studiengesellschaft für Kybernetik (OFAI)

Revision: Version 2

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

EU Project MACS

Deliverable 5.3.3

Robot prototype learning affordances through self-experience V2.0

Georg Dorffner, Jörg Irran, Florian Kintzler

Number:	**MACS/5/3/3 V2**
WP:	5.3
Status:	Version 1.0
Created at:	July, 2007
Revised at:	July 13, 2007
Internal rev:	

FhG/AIS	Fraunhofer Institut für Intelligente Analyse- und Informationssysteme, Sankt Augustin, D
JR_DIB	Joanneum Research, Graz, A
LiU-IDA	Linköpings Universitet, Linköping, S
METU-KOVAN	Middle East Technical University, Ankara, T
OFAI	Österreichische Studiengesellschaft für Kybernetik, Vienna, A
UOS	Universität Osnabrück, Osnabrück, D

This research was partly funded by the European Commission's 6th Framework Programme IST Project MACS under contract/grant number FP6-004381. The Commission's support is gratefully acknowledged.

© OFAI 2007

Corresponding author's address:

Georg Dorffner, Jörg Irran, Florian Kintzler
Österreichische Studiengesellschaft für Kybernetik (ÖSGK)
Freyung 6
A-1010 Vienna, Austria

Fraunhofer Institut für Intelligente Analyse- und Informationssysteme
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany

Tel.: +49 (0) 2241 14-2683
(Co-ordinator)

Contact:
Dr.-Ing. Erich Rome

Joanneum Research
Institute of Digital Image Processing
Computational Perception (CAPE)
Wastiangasse 6
A-8010 Graz
Austria

Tel.: +43 (0) 316 876-1769

Contact:
Dr. Lucas Paletta

Linköpings Universitet
Dept. of Computer and Info. Science
Linköping 581 83
Sweden

Tel.: +46 13 24 26 28

Contact:
Prof. Dr. Patrick Doherty

Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara
Turkey

Tel.: +90 312 210 5539

Contact:
Asst. Prof. Dr. Erol Şahin

Österreichische Studiengesellschaft
für Kybernetik (ÖSGK)
Freyung 6
A-1010 Vienna
Austria

Tel.: +43 1 5336112 0

Contact:
Prof. Dr. Georg Dorffner

Universität Osnabrück
Institut für Informatik
Albrechtstr. 28
D-49076 Osnabrück
Germany

Tel.: +49 541 969 2622

Contact:
Prof. Dr. Joachim Hertzberg

Contents

1 Introduction

This deliverable is an update of deliverable D5.3.3V1. It describes the implementation of the affordance learning architecture which was specified in deliverable D5.3.2. This architecture realises the learning process described in deliverable D5.3.1.

A major part of this deliverable consists of software, that is provided to the MACS partners via the MACS CVS server gibson. This software also contains detailed documentation and instructions (in HTML format) how to start and use the implementation of the learning architecture and/or its components, as well as the implemented comparison algorithms and the processes necessary to extend the architecture, e.g. by adding new data types, comparison algorithms, or partitioners etc. This documentation is continously kept up to date. A demo application space is provided together with the software as well as example configuration files to be used with the new control modules, which allow a modular contol of the whole learning process.

The learning architecture (see figure 1) is implemented as a distributed system that can operate on different computers in a computer network. The functionalities of the different modules can be used detached from each other. Section 2 of this document describes the software environment used to implement and run the learning architecture software and introduces the used data structures and data types and the procedures necessary to extend the architecture by new data types.

Since the learning architecture is embedded into an affordance based control architecture (the learning architecture realises the *Learning Module* of the architecture described in [Rom06]), which provides sensor data to the architecture and to which the derived data is to be provided to be used for realising an affordance based control, section 4 discusses the data flow to, from, and within the learning architecture with special respect to the data processing middelware *dyknow*, respectively *esgm* (see Deliverable D4.1.1).

The functionality of the major parts of the learning architecture is described in Deliverable D5.3.2 [DIK06]. The implementation of the learning architecture exploits a clustering algorithm to integrate the learning steps 1 (partitioning of the application spaces) and the learning steps 2a and 3a (extraction of relevant sensor channels), which reduces computing time. Nevertheless the architecture is designed modularl by using interfaces for the central processing modules like `Partitioner`s etc., to enable users to efficiently test alternative methods for the different processing steps without the need to change the whole system. This document thus includes an introduction to the implemented data structures and the algorithm used for partitioning of the application spaces and describes the used clustering algorithm in section 5.

In this document words in `typewriter` format refer to classes and interfaces of the implementation of the learning architecture (therefore some plurals of words ending with a y are grammatically incorrect spelled as ys not ies to get the correct reference to the concerning class or interface). Please see the HTML documentation of the implementation for further information on the implemented interfaces and classes.

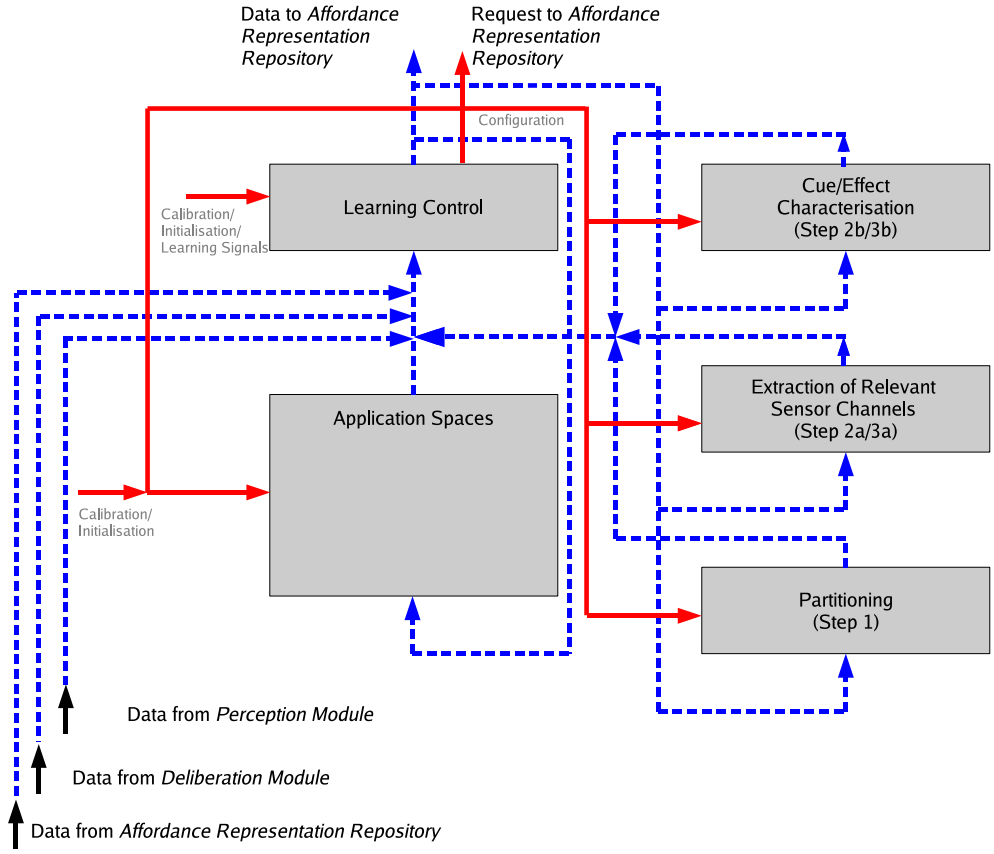

Figure 1: Implemented architecture for learning basic affordances by self experience, as described in Deliverable D5.3.2

*

2 Software

2.1 Software-Environment

The learning architecture software is completely written in Java using the SDK Version 5.0 and is thus platform independent. To ensure type safety the distribution of the modules is realised using Java RMI. The learning architecture is contained in the *Learning Module* of the overall affordance based robot control architecture (see Deliverable [Rom06] and figure 2). Therefore it is connected to the rest of that control architecture using the communication middelware CORBA.

The application spaces which in the learning architecture store the data trajectories recorded during the application of an action, are populated with data received from the sensor system of the robot. This sensor system has to include proprioception and the sensed state of the behaviour system to be able to extract the desired data. This data acquisition was demonstrated at the MACS second year review on November the 20th 2006. See sections 4.1 and 4.2 for a description of the IO of the learning module with regard to the overall affordance based control architecture. Since the specification of the ESGM idl file `value.idl` (see software on the MACS-project CVS server gibson and deliverable D4.1.1 [DMRW04]) is not conform to the IDL standard[1], in addition to the Sun Java Software Development Kit (Java SDK >= 5.0), the compiler and classes of the JacORB[2] project are needed (see section 4.1) for the acquisition of data, but not for the learning process itself.

The interfaces to the Kurt2 robotic platform used by partner OFAI are based on Java Native Interfaces (JNI). The JNI interface is implemented for the Linux-CAN-Bus system and tested on a SuSE Linux 9.3 according to the MACS specification of the software environment[3]. For the connection to the servos of the pan-tilt-cameras and the laser scanner as well as the used robotic arm, USB-COM adapters are used, which are accessed using the Java-COMM framework which makes the system platform independent. It was tested on SuSE Linux 9.3 in accordance with MACS specification of the software environment. Partner OFAI uses the Java-COMM provided by IBM[4], which is more intuitive and faster to install than the Java-COMM by Sun and expects the same COM-ids on Linux and Windows platforms. To access the cameras of the robot, partner OFAI uses the Java Media Framework (JMF). In accordance with the MACS software environment specification the access to the robot is realised using CORBA interfaces specified within the MACS project by partner FhG/iAIS.

For visualisation of the resulting dependency graphs of the partitioning process, the Graphviz-tools[5] can be used. These tools were also used to create the graph images shown in this document.

[1] http://www.omg.org/docs/formal/02-06-39.pdf
[2] http://www.jacorb.org/
[3] The interface also works fine with Fedora Core 5 and Core 6.
[4] www.ibm.com/developerworks/java/jdk
[5] www.graphviz.org

2.2 Data Structures within the Learning Architecture

2.2.1 Basic Data Structures

As basic data structures within the learning architecture all objects can be used that fulfil the following two requirements:

- implement method `String toString()` to be able to store the data (for later offline learning and for data persistence within the `ApplicationSpaces`, see below)
- implement a constructor with a *String* as parameter for re-creation of stored data

Thus all basic data structures like `Double`, `Integer`, `Boolean`, `Float`, `Long`, `Sting` etc. can be used within the learning architecture. In addition the following complex data types are implemented, for storing data received from the perception module:

- `macs.ofai.learning.data.types.ColorBlob`
- `macs.ofai.learning.data.types.Point2dFeature`

All data types, that are provided by the perception module are to be reimplemented by the learning architecture, because the ESGM (see Deliverable D4.1.1 [DMRW04]) does not transport complex data structures that consist of the basic data structures implemented in the DYKNOW system, but attribute-value-pairs that are to be re-transformed into data structures on the receiver side (see section 4.1). In addition to these data structures, comparison algorithms are to be implemented for the new data structures, as well as for already existing data structures, when they are used to store data of a new semantic (see section 2.4)

2.2.2 Generic Trajectories

A `GenericTrajectory` stores time series consisting of time-stamped data of a generic data type $< T >$. The processing of the data is held type independent as far as possible. (see section 2.2.1 for the demands on objects used as basic data types within the learning architecture).

At those points in the data processing, where type dependent processing is mandatory factory classes are used to create the processing objects, e.g. `Comparator`s to be able to compare trajectories (`ComparatorFactory`). In addition a `TrajectoryWriterFactory` and a `TrajectoryReaderFactory` are used to be able to store the data to different storage locations like file systems, or data bases (and to read them from these location, see section 2.3).

This enables the user to add data types, different ways to store the data (hibernation, data base, etc.), and different comparison method by simply adding the concerning classes without any need to change the learning procedure itself.

Each `GenericTrajectory` has to contain an ID that specifies the source from which it was created, e.g. “IR sensor 6”, “Sift Filter X”, or “Behaviour A”.

Since all recorded `GenericTrajectory`s result from the application of a behaviour (see section 2.2.4) a `GenericTrajectory` contains time stamps for the begin and the end of the application of the causing behaviour. See software documentation of the following methods of class `GenericTrajectory`:

- `long getAbsoluteBegin()`
- `long getAbsoluteEnd()`
- `void setBeginOfApplication(long beginOfApplication)`
- `void setEndOfApplication(long endOfApplication)`
- `void setRelativeBeginOfApplication(double relativeBeginOfApplication)`
- `void setRelativeEndOfApplication(double relativeEndOfApplication)`
- `long getBeginOfApplication()`
- `long getEndOfApplication()`
- `double getRelativeBeginOfApplication()`
- `double getRelativeEndOfApplication()`

Related to these time stamps, there are methods, to extract a certain period of the trajectory:

- all data of the pre-action-application phase:
  `public GenericTrajectory<T> getPreApplicationTrajectory()`
- all data of the post-action-application phase, whereby this phase starts with the begin of the action application and ends with the end of the `GenericTrajectory`:
  `public GenericTrajectory<T> getPostApplicationTrajectory()`
- all data of the action-application phase, whereby the application phase starts with the begin of the action application and ends with the end of the action application:
  `public GenericTrajectory<T> getApplicationTrajectory()`

2.2.3 Trajectory Sets

A `TrajectorySet` stores and manages a set of `GenericTrajectory`s. These trajectories are recorded during monitoring an action application.

Each `TrajectorySet` has to contain an *individual* ID, e.g. specifying the time at which it was recorded or the number of the action application trail the set was recorded from.

All TrajectorySets used within the learning architecture result from the application of an action/behaviour. The begin and the end of the application of this action is recorded within the GenericTrajectorys stored within the set. the bin and end of the action application are thus to be equal for all the GenericTrajectorys stored within the set. See software documentation for the following functions methods of class TrajectorySet:

- setRelativeBeginOfApplication(double relativeBeginOfApplication)
- setRelativeEndOfApplication(double relativeEndOfApplication)

Since the LearningControlModule is responsible for managing the input and output of the learning architecture, the TrajectorySets used within the learning architecture are either created within the LearningControlModule (e.g. by receiving data from the ESGM) or read from a storage location (see section 4.1).

2.2.4 Application Spaces

As described in Deliverable 5.3.2 [DIK06], an ApplicationSpace is a behaviour-related data base, to store TrajectorySets and all assigned data structures (like Partitioners etc.).

The stored TrajectorySets have to result from the application of the behaviour the ApplicationSpace is assigned to (see section 4.1). Thereby the action can either be a basic action or a complex action consisting of a sequence of actions.

In addition an ApplicationSpace stores instances of Partitioners, ChannelExtractors and EventCharacterisers that are derived from the trajectories that are stored within the ApplicationSpace. See Deliverable 5.3.2 [DIK06] for the description of the purpose of an ApplicationSpace and the online HTML software documentation for more detailed information on the implementation.

For test purposes the current implementation of the learning architecture includes TestApplicationSpaces, which are used for the examples in this deliverable.

2.3 Writing and Reading Data Structures

The implementation of the learning architecture provides interfaces for storing and reading the data structures ApplicationSpaces, TrajectorySets, GenericTrajectorys, and instances of Partitioners. These interfaces can be found in the corresponding sub-packages under package macs.ofai.learning.io. In the current implementation these interfaces are implemented for storing the data to the file system.

When storing an *ApplicationSpace* using ApplicationSpaceFileWriter, a directory is created in which the contained TrajectorySets, Partitioners, ChannelExtractors and EventCharacterisers are stored. To store a TrajectorySet, sub-directories are created for each GenericTrajectory in the set and the GenericTrajectorys are stored in these directories.

The mechanisms to store GenericTrajectorys are changed and simplified significantly, compared to the software delivered with the first version of this document. However, the structure of the data files does not change, so that old data records can still be processed by the learning architecture.

Since all data types handled by the learning architecture have to implement method String toString() and a constructor, accepting a *String* as parameter (see section 2.2.1), type specific TrajectoryWriters and TrajectoryCreator are no longer needed. Instead the data structures itself handle the conversion of the data to and from strings. However the classes managing reading and writing of data, remain storage location specific (e.g. for file system storage or data bases). To be able to add readers and writers for different types of storage locations without the need to change any of the processing classes, the factories TrajectoryWriterFactory and TrajectoryReaderFactory are used, that scan for classes that implement the interfaces TrajectoryWriter and TrajectoryReader. The implementation for storing data to and reading data from the file system are called GenericTrajectoryXMLFileWriter and GenericTrajectoryXMLFileReader.

Since a GenericTrajectory does not only contain the time stamped data, but also markers for the begin and the end of the application of the action that caused the recording of the trajectory, these markers are also stored in the XML file.

Since the learning architecture specifies interfaces for Partitioners, ChannelExtractors and EventCharacterisers which are to be implemented by classes that realise the function of the concerning object, the interfaces PartitionerReader, PartitionerWriter, ChannelExtractorReader, ChannelExtractorWriter, EventCharacteriserReader and EventCharacteriserWriter are to be implemented for the corresponding classes, as done by the classes GraphPartitionerReader and GraphPartitionerWriter.

Please see the online HTML documentation of the software for a more detailed description of the factories and the implemented reader, writers, and creators.

2.4 Comparison of Trajectories

To make the learning architecture as flexible and extendible as possible the learning architecture is for the major part implemented data-type independent. The only exception is the comparison of trajectories. Comparison of GenericTrajectorys is used within the implemented partitioning processes and the channel extraction mechanisms to cluster trajectories (see section 5).

For each type of data that is to be processed by the learning architecture at least one comparison method is to be provided to the system by an implementation of interface macs.ofai.learning.comparison.Comparator. The Comparator compares complete GenericTrajectorys (not a single datum).

Since data within similar data structures can have a different semantic, it can be necessary to implement a new comparator for a new data stream, even if there are already comparators for GenericTrajectorys containing the same data structures. For example GenericTrajectorys of vectors of three double values could contain Cartesian coordinates or RGB colour values. Both be compared by using the Cartesian distance, however there

are better ways for comparing RGB values.

Additionally even data from the same data source can have multiple meanings. For example double value trajectories resulting from an infrared sensor could be compared using the difference between the values or by measuring if they both contain a peak that exceeds a certain threshold anywhere in a specified time interval. The system is therefore designed to be able to use different `Comparator`s (i.e. different distance measures) for one and the same data structure and thus realises the clustering of action application related data with respect to different aspects.

These are the reasons, why the semantic of each data stream provided by the *Perception Module* must be documented in addition to its syntax. On a bilateral meeting partner OFAI and JR_DIB therefore started to document the implemented data structures. This list is available via the projects dito server (in section *MACS-ext* → *Discussions* → *Data Structures*) and must be kept up to date by the project partners.

The following `Comparator`s are implemented within the learning architecture (in package `macs.ofai.learning.comparison`, see software documentation for more details on the used algorithms and their implementation):

- `ComparatorDoubleArea` (data type: `java.lang.Double`)
  relates the area between the two trajectories to the discrete integral of the maximum function of the two trajectories
- `ComparatorDoubleCorrelation` (data type: `java.lang.Double`)
  calculates the cross-correlation between the two trajectories
- `ComparatorIndex` (data type: `macs.ofai.learning.data.types.Index`)
  calculates the ratio between the number of differing indices and the total number of indices.
- `ComparatorAction` (data type: `macs.ofai.learning.data.types.Action`)
  calculates the ration between the number of differing action entries and the total number of action entries

Additional `Comparators` can be added to the learning architecture by simply implementing interface `macs.ofai.learning.comparison.Comparator` and, if it is not in the standard package for comparators (`macs.ofai.learning.comparison`) adding the location to the config file for the concerning control modules (see section 3).

3 Control

3.1 Interfaces of the Learning Architecture Modules

As described in section 2.2 the learning architecture is for the most part implemented data type independent to be able to easily add and change data types to be processed by the affordance based control architecture. Additionally the modules that are necessary for implementing an affordance learning architecture that follows the structure described in deliverable D5.3.2 [DIK06] (depicted in figure 1) are defined as Java RMI `Interface`s (extending interface `java.rmi.Remote`) so that different partitioning-, extraction-, characterisation-, and control-methods can be implemented without the need to change the architecture itself.

The interfaces defining the necessary functionality of the central learning modules are defined in package `macs.ofai.learning.modules.rmi`:

- `ApplicationSpacesModuleInterface`
  to be used to implement a module that stores `ApplicationSpaces`, which contain the data that is to be used for learning (see section 2.2.4).

- `PartitioningModuleInterface`
  to be used to implement a module that creates one or more `Partitioner` objects for `ApplicationSpace`s. These objects are instances of classes that implement interface `macs.ofai.learning.partitioning.Partitioner`. Partitions are related to one `ApplicationSpace` and contain the learning algorithm for partitioning itself as well as a method that decides, based on the learned data, to which partition a `TrajectorySet` from the `ApplicationSpace` belongs.

- `RelevantSensorChannelExtractionModuleInterface`
  to be used to implement a module that creates `ChannelExtractor`s for the pre- and post-application phase of the trajectories belonging the same partitions of the associated `ApplicationSpace` (for cue-event and outcome-event detection). Classes which implement the interface `ChannelExtractor` are expected to find sensor channels (id of the `GenericTrajectory`s) in the `TrajectorySet`s of each partition of an `ApplicationSpace` that are representative for the concerning partition and thus for the concerning action application result. Therefore the `ApplicationSpace` must of course be partitioned first (see @link PartitioningModuleInteface).

- `EventCharacterisationModuleInterface`
  to be used to implement a module that creates `EventCharacteriser`s for the relevant sensor channels, extracted by the `ChannelExtractor`s (cue- and outcome-related characteristics).

Since the module *Learning Control* within the learning architecture (see figure 1) is itself designed modular, there are several interfaces to be implemented for controlling the dataflow within the learning architecture. Nevertheless there can be one master control class, that manages these sub-modules. The interfaces to be implemented for learning control are also defined within package `macs.ofai.learning.modules.rmi`:

- `BehaviourMonitoringModule`
  to be implemented to realise a module that monitors the agent's perception system with respect to the agent's own actions. It records sensor-data trajectories while the agent applies one of its behaviours. These trajectories have to additionally contain data from the pre-application phase and the post-application phase as described in Deliverable D5.3.1 and D5.3.2. How this is done is up to the implementing class. The class could use fixed time intervals or use filters to detect changes within the environment.

- `PartitioningControlModule`
  to be implemented to realise a module that observes a module that implements the interface `ApplicationSpacesModuleInterface` and invokes method `partition(...)` of an associated module that implements interface `PartitioningModuleInterface` when the state of one of the `ApplicationSpace`s that are stored in the module (that implements interface `ApplicationSpacesModuleInterface`) has changed significantly compared to the state at the time of the last invocation of the partition process. To decide what is significant is up to the the implementing class and could be based simply on the number of (novel) `TrajectorySet`s within an `ApplicationSpace` or based on more complex algorithms, e.g. involving time etc.

- `RelevantSensorChannelExtractionControlModule`
  to be implemented to realise a module that observes the states of the `Partitioner`(s) associated to the `ApplicationSpace`s within a module that implements interface `ApplicationSpacesModuleInterface`, and invokes a module that implements interface `RelevantSensorChannelExtractionModuleInterface`, by calling its method `extractRelevantSensorChannels(...)`, whenever the state of the `Partitioner` changes significantly. This method could be called whenever the state changes, or depending on how much the state of the `Partitioner` change.

- `EventCharacteriserControlModule`
  to be used to implement a module that observes the states of `ChannelExtractor`(s) that are associated to the `ApplicationSpace`(s) within a module that implements interface `ApplicationSpacesModuleInterface`, and invokes a module that implements interface `EventCharacteriser`, when there are either new channels to be described or the data within an already described channel differs from its description.

- `DataSupplyControlModule`
  to be implemented to realise a module that transfers the data, derived by the learning architecture into representations needed by components that store this information (e.g. *Affordance Representation Repository* or need the information to control the agent (*Affordance Based Planning* and *Affordance Based Execution*).

3.2 Management of Learning Architecture Modules

All modules within the learning architecture must be created, initialised (with a set of parameters), registered to an rmi registry and later de-registered and de-constructed. Thereby data has to be safed to or read from a storage location. To unify these processes,

class `macs.ofai.learning.modules.management.ModuleManager` provides methods to read the configuration of a module of the learning architecture from an xml config file, to create the module, and to registers and un-register it to an RMI registry (using the specifications provided by the config file).

The structure of the xml config file used to create and initialise a module within the learning architecture is the same for all modules. The document type definition (DTD)[6] to be used for these config files can be found in the dtd folder of the software project on the projects CVS server gibson in file *moduleConfig.dtd* (see also annex A.2).

To enable the `ModuleManager` to manage a module, it hast to fulfil the following requirements:

- implement interface `java.rmi.Remote`, so that it can be made available to other modules via a RMI registry
- implement a constructor that accepts a `macs.ofai.tools.ParameterProvider` as parameter, so that the module can be initialised using the parameters specified within the config file
- implement method `public void stopModule()`, so that the module is enabled to store relevant data when the module is stopped (this is optional, since not all modules need to save data).

The information that is to be provided within the config file to enable the `ModuleManager` to create and manage a module within the learning architecture is:

- the module's `Type`: The canonical name of the module to be loaded.
  The learning architecture provides the following modules (implementations of the interfaces explained in section 3.1) in package `macs.ofai.learning.modules`:
  - `ApplicationSpacesModule`
    implements `ApplicationSpacesModuleInterface`
  - `PartitioningModule`
    implements `PartitioningModuleInterface`
  - `RelevantSencorChannelExtractionModule`
    implements `RelevantSensorChannelExtractionModuleInterface`
  - `EventCharacterisationModule`
    implements `EventCharacterisationModuleInterface`

  The current implementations of the control module interfaces are located in package `macs.ofai.learning.modules.control`:
  - `BehaviourMonitoringModuleESGM`
    implements `BehaviourMonitoringModule`
  - `PartitioningControlModuleCounterBased`
    implements `PartitioningControlModule`

[6] http://www.w3.org/TR/REC-xml/#sec-prolog-dtd

  - `RelevantSencorChannelExtractionControlModuleBasic`
    implements `RelevantSencorChannelExtractionControlModule`
  - `EventCharacteriserControlModulePrototype`
    implements `EventCharacteriserControlModule`
  - `DataSupplyControl`
    Not implemented yet since the *Affordance Representation Repository* is not specified yet.

- the `RegistryHost`: Specification of the RMI registry host, that manages the data exchange between the modules, e.g. `127.0.0.1` or `server.domain.net`
- the modules `ID`: ID that should be used for registering the module to the RMI registry. For the major learning modules the ID has to be one of the following:
  - `ApplicationSpacesModule`
  - `RelevantSensorChannelExtractionModule`
  - `EventCharacteriserModule`
  - `PartitioningModule`
  - `LearningControlModule`

  For the control modules the ID has to be one of the following:
  - `BehaviourMonitoringModule`
  - `PartitioningControlModule`
  - `RelevantSensorExtractionControlModule`
  - `EventCharacteriserControlModule`

The parameters that are to be specified for the different modules can be found in the software documentation of the concerning modules in package `macs.ofai.learning.modules`. Example config files are located in folder *config* of the learning architecture project on CVS server gibson.

The `ModuleManager` itself is invoked on the command line and has two optional parameters:

-r Read-Test: The `ModuleManager` tries to read the config file and exits

-c Create-Test: The `ModuleManager` tries to read the config file, to create the module and exits

Without any option, the config file will be read, the module will be created and registered to the registry host specified in the config file and stopped when the user presses `ENTER`. This module could be extended by a CORBA interface, so that the creation, registration and de-construction process could be remotely controlled.

3.3 Implementations of the Learning Architecture Module Interfaces

In the following sections, the usage of the main learning modules is described. For a detailed description of the interfaces, methods and the used algorithms as well as the status of implementation please see the software documentation on the projects CVS server gibson, which is continously kept up to date together with the software itself.

3.3.1 ApplicationSpacesModule

One of the most central modules in the learning architecture is the *Application Spaces Module*. Each module that is meant to implement the functionality of an *Application Spaces Module* as it s describe in Deliverable D5.3.2 has to implement the Java interface `ApplicationSpacesModuleInterface`.

This is done by class `macs.ofai.learning.modules.ApplicationSpacesModule`. This module uses the `ApplicationSpacesModuleWriterFactory` to find writers for storing the `ApplicationSpace`s and the `ApplicationSpacesModuleReaderFactory` to find readers to re-load them. Therefor, the parameters needed to control the `ApplicationSpacesModule` are related to the storage location for reading and writing of data:

For information for creation of the `ApplicationSpacesModule` specify either

- `ID`: specifying the ID of the `ApplicationSpacesModule` that is newly created

or, if data should be read from a storage location, specify the following parameters:

- `ModuleLocationReader`: Location parameter for the used reader. In case of a file reader this is a directory name
- `MediatypeReader`: Type of the media from which the `ApplicationSpacesModule` is to be read. These strings are specified by class `MediaType`. Currently `filesystem` is supported.
- `ApplicationSpacesModuleReaderPackage`: Location where the reader factory class `ApplicationSpacesModuleReaderFactory` should search for a reader. Write DEFAULT for the default package.

Optional one can specify a set of parameters specifying how to save the data from the `ApplicationSpacesModule` in the end. If these parameters are not specified, the data will not be saved and all changes in between creation and destruction of the module are lost.

- `ModuleLocationWriter`: Location parameter for the used writer. In case of a file writer this would be a directory name
- `MediatypeWriter`: Type of the media to which the `ApplicationSpacesModule` is to be written. These strings are specified by class `MediaType`. Currently `filesystem` is supported.

- `ApplicationSpacesModuleWriterPackage`: Location where the writer factory class `ApplicationSpacesModuleWriterFactory` should search for a writer. Use the string `DEFAULT` for the default package.

Thus a configuration file used to start an `ApplicationSpacesModule` by reading the data from the provided demo application space, and later write the data back to the file system could look like the following:

```
<?xml version="1.0"?>
<LAModule>
   <Type>macs.ofai.learning.modules.ApplicationSpacesModule</Type>
   <RegistryHost>192.168.0.2</RegistryHost>
   <ID>ApplicationSpacesModule</ID>
   <Parameters>
      <!-- For Reader:
           (or just specify an ID to create a new,
            empty ApplicationSpacesModule) -->
      <Parameter name="ModuleLocationReader">
         ../applicationSpacesModules/demo
      </Parameter>
      <Parameter name="MediatypeReader">
         filesystem
      </Parameter>
      <Parameter name="ApplicationSpacesModuleReaderPackage">
         DEFAULT
      </Parameter>
      <!-- For writer: (optional) -->
      <Parameter name="ModuleLocationWriter">
         ../applicationSpacesModules/demo_new
      </Parameter>
      <Parameter name="MediatypeWriter">
         filesystem
      </Parameter>
      <Parameter name="ApplicationSpacesModuleWriterPackage">
         DEFAULT
      </Parameter>
   </Parameters>
</LAModule>
```

3.3.2 PartitioningModule

To realise the functionality of a *Partitioning Module*, class `PartitioningModule` in package `macs.ofai.learning.module` implements the module interface *PartitioningModuleInterface* (see section 3.1) and thus provides the partitioning method

```
partition(ApplicationSpacesModuleInterface appModule,
```

```
String applicationSpaceID)
```

This method requests the `ApplicationSpace` with the specified ID from the specified interface to the application spaces module `appModule` and creates a `Partitioner` for it or updates the `Partitioner`. The type of `Partitioner` (the class implementing interface `Partitioner`) that is used within this method must be specified by parameter *Partitioner* in the xml config file. This means, that the learning architecture can contain more than one `PartitioningModule`, one for each class that implements interface `Partitioner`. This should be regarded when specifying the ID of the module. Please regard that the ID of this module must be unique and be entered in the config file of the `PartitioningControlModule`.

The `Partitioner` provided with the learning architecture (`GraphPartitioner`) uses the algorithm described in section 5. The described partitioning method can on the one hand be used to find outcome related clusters (by using the post-application phase), and can on the other hand be re-used to find sub-clusters in the outcome-related clusters, by sub-clustering these clusters with respect to the pre-application phase. Therefore the config file of the `PartitioningModule` contains the obligatory parameter `Part` which has to be one of the following:

- `PRE`: The trajectories used for clustering are reduced to the data that was recorded, before the application of the related behaviour started.
- `POST`: The trajectories used for clustering are reduced to the data that was recorded, after the application of the related behaviour started, including the data theat was recorded, after the application of the behaviour stopped. This option is to be used to start a `PartitioningModule` that realised the functionality of a *Partitioning Module* as described in Deliverable D5.3.2.
- `DURING`: The trajectories used for clustering are reduced to the data that was recorded, while the the related behaviour was applied, thus excluding all data that was recorded before and afterwards.
- `COMPLETE`: The whole trajectory is used for clustering.

Thus an xml config file for a *Partitioning Module* could look like this:

```
<?xml version="1.0"?>
<LAModule>
   <Type>macs.ofai.learning.modules.PartitioningModule</Type>
   <ID>GraphPartitioner</ID>
   <RegistryHost>127.0.0.1</RegistryHost>
   <Parameters>
      <Parameter name="Part">
         POST
      </Parameter>
      <Partameter name="Partitioner">
```

```
            macs.ofai.learning.partitioning.GraphPartitioner
         </Parameter>
      </Parameters>
   </LAModule>
```

3.3.3 PartitioningControlModuleCounterBased

The task of a module that implements the abstract class `PartitioningControlModule` is to observe the `ApplicationSpace`s that are stored in a module that implements interface `ApplicationSpacesModuleInterface` and to invoke a module that implements `PartitioningModuleInterface`, when the internal methods detect a significant change within an `ApplicationSpace`.

The class `PartitioningControlModuleCounterBased` implements the abstract class `PartitioningControlModule` by using the number of stored `TrajectorySet`s within an `ApplicationSpace`. The parameters needed for this class are:

- `Threshold`: If the number of new `TrajectorySet`s (that were previously not included into the partitioning process) exceeds this threshold, the partitioning process is invoked.
- `UpdateInterval`: Time in between two checks, if partitioning is needed.
- `ApplicationSpacesModuleInterface`: Specification of the rmi registry and the ID of the module, implementing interface `ApplicationSpacesModuleInterface`, that should be observed by the control module.
- `PartitioningModuleInterface`: Specification of the rmi registry and the ID of the module, implementing interface `PartitioningModuleInterface`, that should be used to start the partitioning.

Thus a config file for a `PartitioningControlModuleCounterBased` could look like this (no line break in the `Type` declaration, this is only for layout reasons):

```
<?xml version="1.0"?>
<LAModule>
   <Type>
      macs.ofai.learning.modules.learningcontrol.
      PartitioningControlModuleCounterBased
   </Type>
   <RegistryHost>127.0.0.1</RegistryHost>
   <ID>PartitioningControlModule</ID>
   <Parameters>
      <Parameter name="Threshold">4</Parameter>
      <Parameter name="UpdateInterval">10</Parameter>
      <Parameter name="ApplicationSpacesModuleInterface">
         //127.0.0.1/ApplicationSpacesModule
```

```
      </Parameter>
      <Parameter name="PartitioningModuleInterface">
         //127.0.0.1/GraphPartitioner
      </Parameter>
   </Parameters>
</LAModule>
```

3.3.4 RelevantSensorChannelExtractionModule

The class `macs.ofai.learning.modules.RelevantSensorChannelExtractionModule` implements the module interface `RelevantSensorChannelExtrationModuleInterface`. The module provides the extraction method

```
public static void extractRelevantSensorChannels(
                     ApplicationSpacesModuleInterface appModule,
                     String applicationSpaceID, int part )
```

This method requests the `ApplicationSpace` with the specified ID from the specified interface to the application spaces module `appModule` and creates a `ChannelExtractor` for it or updates the `ChannelExtractor`. Before this method can extract relevant sensor channels, the involved application space (specified by the interface and the ID string) has to be partitioned (see section 3.3.2).

The type of `ChannelExtractor` (the class implementing interface `ChannelExtractor`) that is used within this method must be specified by parameter *ChannelExtractorType* in the xml config file. This means, that the learning architecture can contain more than one `ChannelExtractionModule`, one for each class that implements a `ChannelExtractor`. This should be regarded when specifying the ID of the module. Please regard that the ID of this module must be unique and be entered in the config file of the concerning control module.

The `ChannelExtractor` provided by the learning architecture (`GraphChannelExtractor`) re-uses the algorithm described in section 5 and as described above utilises the functionality of the `Partitioner` by invoking it for the pre-application part of the `TrajectorySet`s stored in a cluster in the `ApplicationSpace`. See also page 27, section 5.4, and the software documentation on CVS server gibson, for a more detailed description of the implementation of the channel extraction method.

Since a `ChannelExtractor` can be used for extracting relevant channels concerning the outcome and relevant channels concerning the cues (see Deliverable D5.3.2), the parameter `Type` can either be `OUTCOME` or CUE.

Thus a config file for a `RelevantSensorChannelExtractionModule` could look like this:

```
<?xml version="1.0"?>
<LAModule>
   <Type>
      macs.ofai.learning.modules.RelevantSensorChannelExtractionModule
```

```
   </Type>
   <ID>GraphPartitioner</ID>
   <RegistryHost>127.0.0.1</RegistryHost>
   <Parameters>
      <Parameter name="Type">
         OUTCOME
      </Parameter>
      <Partameter name="ChannelExtractor">
         macs.ofai.learning.channelExtraction.GraphChannelExtractor
      </Parameter>
   </Parameters>
</LAModule>
```

4 Interfaces to the Learning Architecture

The affordance learning architecture processes data received form an agent's perception system and provides the derived affordance related information to the control of the agent. This means, that the affordance learning architecture is embedded into an affordance based control architecture. In the MACS project the affordance based control architecture, described in deliverable D2.2.2 [Rom06] is used and the learning architecture is contained in its *Learning Module*.

The affordance based control architecture provides data (from the sensors, internal states, proprioception, state of behaviour system, preprocessed sensor data etc.) to the learning architecture via the *Perception Module* and provides a data base to store the extracted affordance information in (the *Affordance Representation Repository*) (see figure 2). As shown in figure 1 (and depicted in figure 2) the `LearningControlModule` is the central point in the learning architecture that manages the incoming and outgoing data. The `LearningControlModule` is on the one hand connected to the perceptional part of the control architecture and on the other hand to the repository in which the acquired knowledge is to be stored. In addition the *Learning Module* would benefit from a connection to the *Deliberation Module* in order to request the execution of action for which additional data is required to be able to (re)start the learning process.

The current implementation of the module *Learning Control* consists of several sub modules for managins the data acquisition, the partitioning, the channel extraction etc. (see software documentation of package *macs.ofai.learning.modules.learningcontrol* and the classes within that package)

4.1 Data Acquisition

For the connection to the *Perception Module* the `LearningControlModule` contains the sub-module `BehaviourMonitoringModule` that is responsible for the incoming data from the perceptual system. The `BehaviourMonitoringModule` monitors the state of the agents perceptual system and is responsible for storing the recorded `TrajectorySet`s with the appropriate length (including pre- and post-application phase, see Deliverable D5.3.2 and D5.3.1) into the `ApplicationSpace`s stored in the `ApplicationSpacesModule`.

The `BehaviourMonitoringModule` is to be adapted, when the underlying perceptional or behavioural system changes (e.g. new types of sensor data) or is replaced. The data is pulled from the *Perception Module*. Thus the interface to the perceptual system (including proprioception and perception of the behaviour systems state) must be known to adapt the `BehaviourMonitoringModule` to the used interfaces and data structures. When the perception system changes and new data types are introduced, new `Comparator`s, `TrajectoryWriter`s and `TrajectoryConstructor`s are to be implemented and the `BehaviourMonitoringModule` is to be adapted, but the learning architecture itself does not need to be changed.

The `BehaviourMonitoringModule` can be used separately to create `ApplicationSpace`s independent from the learning process. The created `ApplicationSpacesModule` can then be stored for later partitioning, channel extraction and characterisation.

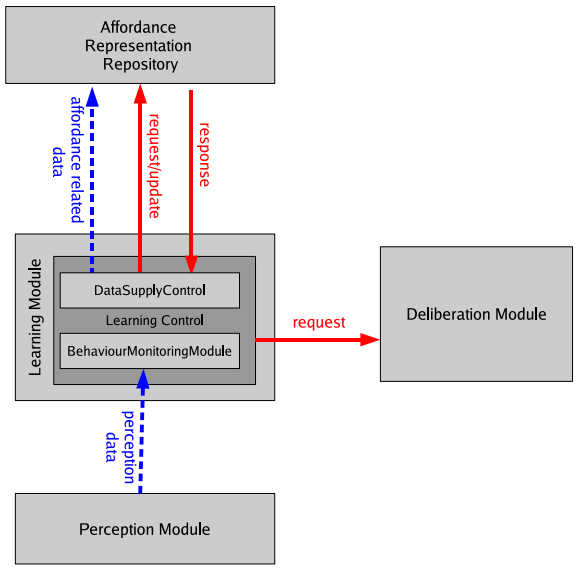

Figure 2: Embedding of the affordance learning architecture (inside the *Learning Module*) into the affordance based control architecture (described in deliverable D2.2.2 and the upcoming deliverable D2.3.1). The *Learning Module* reads the data provided by the *Perception Module*, stores it internally, starts the learning process when enough data is acquired and provides the derived data to the *Affordance Representation Module*.

4.1.1 Data Acquisition using Using ESGM: Access

Within the MACS project a data middelware named *DYKNOW* is uded to implement the *Entity Structure Generation Module (ESGM)* which is the inteface to the *Perception Module* and provides mechanisms for the synchroniced retrieval of data. The esgm is to be contaged using CORBA. Therefor the idl files distributed with the ESGM are to be compiled to java classes.

The IDL file `value.idl` specified by the ESGM system is not IDL standard conform. The file is in confilct with the rule described on page 42 of the IDL specification[7]. Neitherthe idl compiler provided with the Sun java SDK, nor the IBM idl compiler, nor the idl compiler of the OpenOrb project[8] can handle the following code:

```
1 struct AttributeValuePair; // forward declaration
2 typedef sequence<AttributeValuePair> AttributeValuePairSeq;
3 struct EntityFrame {
4    string name;
5    AttributeValuePairSeq attributes;
6 };
```

[7] http://www.omg.org/docs/formal/02-06-39.pdf
[8] http://openorb.sourceforge.net

However, the with the compiler from the JacORB project[9] is more fault-tolerant and compiles the whole set of idl files provided by ESGM. See appendix section A.2.1 for installation instructions. The classes and interfaces compiled by using these instructions can be downloaded from the MACS project's CVS server gibson (project `PMtoLMinterface`).

4.1.2 Data Acquisition using Using ESGM: Data Structures

The *Enity Structure Generation Module (ESGM)* does only transport data structures that are already known to the system. To enable the system to deal with all types of data structures composed outof these basic data structures, the DYKNOW system transports the data within `EntitiyStructure`s, which contain an array `AttributeValuePair`s, i.e. sets of labeled basic data structures. These arrays of `AttributeValuePair`s must be retransferred into data structures to be stored and processed by the learning architecture. This includes all data structures specified by the *Perception Module* like the `ColorBlob` or the *Point2dFeature* specified by partner JR_DIB and the behaviour state structures to be specified by partner IAIS.

In the `PMtoLMinterface` project partner OFAI will therefore provide class `DataStructureConverter`, which will be responsible for transfering the ESGM data types into Java classes, that afterwards can be processed by the learning architecture.

Each time a new data structure is introduced, the `DataStructureConverter` must be adapted in addition to the necessary implementation of adequate comparators. Therefor partner JR_DIB and OFAI started a specification document documenting the structure itself and the semanic of the data structures. This document is made available to the partner via the Dito server in section *MACS-ext* → *Discussions* → *Data Structures*.

4.2 Data Supply

The affordance related data acquired by the learning process must be stored to be used by the *Execution Module* and the *Deliberation Module* of the affordance based control architecture. Therefore the affordance based control architecture provides the *Affordance Representation Repository* (*ARR*). As described in section 2.3 in deliverable D5.3.2. the ARR stores $n : m : l$ relations between cue events, behaviours and outcome events. From this data, affordance triples can be derived either within the *ARR* or within the execution or deliberation part of the control architecture.

The *Learning Module* is designed to push the derived data into the *Affordance Representation Repository*. As shown in figure 2 the `LearningControlModule` is responsible for pushing the data into the repository. The data structures used in the *ARR* depend on what the deliberation and execution part of the overall affordance based control architecture require. The current state of implementation of the *Learning Module* provides trajectory prototypes and the method described in section 4.2.3 of deliverable D5.3.2 (characterisation of the trajectories using neural networks).

[9] http://www.jacorb.org/

5 Clustering

In this section the clustering algorithm is described that is used to partition the application spaces. This algorithm has the advantage that it incorporates the detection of relevant sensor channels and is thus used within the partitioning process and the process of extracting relevant sensor channels with respect to the outcome. To use the results of the algorithm for two of the processing steps saves processing time. Nevertheless, via using interfaces to define `Partitioner`s and `ChannelExtractor`s the software is designed so that different algorithms can be easily implemented and used within the learning architecture.

The classes and interfaces used for the clustering method described in this section are located in package `macs.ofai.learning.clustering`.

5.1 Algorithm

The goal of the clustering is to find groups in the application spaces that share features. The clustering is designed to be able to cluster the whole trajectories or parts of the trajectories, more precisely the pre-application and the post-application part. The pre-application part covers the part of the trajectory recorded before the application of the action began and the post-application part covers the part after the application began, including the part that was recorded after the application of the concerning action ended.

In case of clustering the post-application part of the trajectories, the resulting partitions are expected to correspond with the result of the application of the action that is assigned to the application space, e.g. a group that corresponds to all trials where the entity was liftable and a group that corresponds to all trials where the entity was non-liftable.

In this section the used algorithm is described. The next section (section 5.2) shows that this algorithm leads to a result that is in accordance with the goal of the clustering.

1. Cluster all trajectories using different comparators.
2. Create a graph that reflects the dependencies between the resulting clusters
   (a) Create a node for each (trajectoryID,comparator,cluster) combination
   (b) Create edges between a node a and a node b if they correlate
3. While the following process changes the dependency graph
   (a) Summarise those Nodes that represent the same `Clusterer` but different clusters and that are both sons of the same node (the old summarised nodes are removed from the channel).
   (b) Create edges between a node a, representing more than one cluster and a node b if they correlate and there is an edge from b to a.
4. Extract the independent components (Dependency Sub-Graphs (DSG)) of the created dependency graph.

These steps are explained in the following sub-section. For the examples in these sections, the `ApplicationSpace` of action *Action A* is used which is provided in the test-*Application Spaces Module demo* together with the software. This `ApplicationSpace` contains 4 `TrajectorySet`s each containing 4 trajectories, with the ids *Action A* (containing data of type `macs.ofai.learning.data.types.Action`), *Channel A* (containing double values), *Channel B* (containing double values), and *Index Channel* (containing indices of type `macs.ofai.learning.data.types.Index`, which corresponds to integers, but differs in the implemented comparison methods).

5.1.1 Clustering of Trajectories

The first step of the algorithm is to cluster the trajectories stored in the `TrajectorySet`s of the concerning `ApplicationSpace`.

To be able to cluster the trajectories, there must be the possibility to measure the distance respectively the equality of two trajectories. This comparison is done by `Comparator`s (see HTML software documentation) which are to be implemented for the used data types (e.g. double values or complex shape descriptors etc.). A `Comparator` realises a distance measure for a special data type. Thus new `Comparator`s are to be introduced, if new data types are introduced. There can be more than one `Comparator` for a data type, e.g. see HTML documentation of `ComparatorDoubleCorrelation` and `ComparatorDoubleArea` which both provide a distance measure for double valued time series. This enables the system to compare trajectories using different measures as claimed in deliverable D5.3.2 [DIK06].

`Comparator`s are used by `Clusterer`s, which group those trajectories, which are equal (close to each other) concerning the used `Comparator`. In the current implementation this is done by using a threshold. Since the system is held modular, replacing this threshold method by a self organising clustering technique is possible without the need to change any other component in the learning architecture.

Different `Comparator`s can be used simply by adding them to the concerning folder (`macs.ofai.learning.comparison`), respectively by adding their location to the system. The `ComparatorFactory` is designed to automatically find and integrate them into the learning process. Depending on the type of data received form the sensors and the behaviour system, the used set of `Comparator`s has to be adapted and extended. If there is no `Comparator` for a data type, the concerning trajectories are ignored and a warning is given.

`Clusterer`s are created by a `TrajectoryClusterer`. A `TrajectoryClusterer` extracts the `GenericTrajectory`s from the `TrajectorySet`s in the `ApplicationSpace` that is assigned to the `TrajectoryClusterer` and adds them to the concerning `Clusterer` or creates a new `Clusterer`, if there is none for the concerning trajectory-ID.

The first step of the algorithm thus results in a set of `Clusterer`s (for each existing trajectory-ID one) each resulting in a set of `Cluster`s, representing different types of data in the concerning trajectories.

Figure 3 shows the result of clustering the trajectories of the test-`ApplicationSpace` *Action A*.

	ts 00	ts 01	ts 02	ts 03
Action A (macs.ofai.learning.data.types.Action)	0	0	0	0
Channel A (java.lang.Double-Area)	2	1	0	0
Channel B (java.lang.Double-Area)	0	0	0	1
Channel A (java.lang.Double-Correlation)	0	0	0	0
Channel B (java.lang.Double-Correlation)	0	0	0	1
Index Channel (macs.ofai.learning.data.types.Index)	0	0	1	1

Figure 3: Results of the process of clustering the trajectories of the test-`ApplicationSpace` *Action A* which is provided together with the learning software.

5.1.2 Building a Dependency Graph: Creating Nodes

Using the clustering results of the first step, a graph is to be created, which represents the dependencies between the clusters (types of trajectories) of the different `Clusterer`s. Each of these `Clusterer`s, is assigned to a trajectory-ID, uses a `Comparator`, and results in a number of `Cluster`s. Therefore graph nodes are created representing all possible (TrajectoryID, Comparator, Cluster) combinations resulting from the clustering of the trajectories done in the first step of the algorithm (see previous section). For example if the trajectories with ID *Channel A* are clustered by a `Clusterer` using a `Comparator` named *ComparatorDouble* resulting in 3 different clusters, the following three nodes are created:

- node 0 = (“Channel A”,“ComparatorDouble”,0)
- node 1 = (“Channel A”,“ComparatorDouble”,1)
- node 2 = (“Channel A”,“ComparatorDouble”,2)

5.1.3 Building a Dependency Graph: Creating Edges

The graph that is to be created has to reflect the dependencies between the different clusters (kinds) of trajectories. Therefore an edge between a node a and a node b is inserted, if a `TrajectorySet` X exists which contains a trajectory of type $a.TrajectoyID$ containing data of type $a.Cluster$ (concerning a `Clusterer` using $a.Comparator$ for trajectories with id $a.TrajectoryID$) and simultaneously contains a trajectory of type $b.TrajectoryID$ containing data of type $b.Cluster$, while this type of data ($b.Cluster$) does not occur in a `TrajectorySet` if clustering $a.TrajectoryID$ using a `Clusterer` that uses $a.Comparator$ leads to another type than $a.Cluster$.

More formally: Insert an edge between a node a and a node b representing different TrajectoryIDs and Comparators, if and only if the following two conditions are met:

1. $\exists$ TrajectorySet X:
   Clusterer(a.TrajectoryID,a.Comparator).cluster(X.trajectory(a.TrajectoryID)) = a.Cluster $\wedge$
   Clusterer(b.TrajectoryID,b.Comparator).cluster(X.trajectory(b.TrajectoryID)) = b.Cluster.
2. $\forall$ TrajectorySet X: $\nexists$ node c with (a.TrajectoryID=c.TrajectoryID) $\wedge$ (a.Comparator=c.Clusterer) $\wedge$ (a.Cluster$\neq$ c.Cluster):
   Clusterer(c.TrajectoryID,c.Comparator).cluster(X.trajectory(c.TrajectoryID)) = c.Cluster $\wedge$
   Clusterer(b.TrajectoryID,b.Comparator).cluster(X.trajectory(b.TrajectoryID)) = b.Cluster.

The test-`ApplicationSpace` for *Action A* contains 4 `TrajectorySet`s, each containing `GenericTrajectory`s with the IDs "Action A", "Channel A", "Channel B", and "Index Channel". The system uses

- a `ComparatorAction` for the `GenericTrajectory`s containing data of type `Action`,
- two comparators (`ComparatorDoubleArea` and `ComparatorDoubleCorrelation`) for the double valued trajectories with id "Channel A" and "Channel B", and
- `ComparatorIndex` for the trajectories with id "Index Channel".

See also HTML software documentation for the implemented `Comparator`s. The clustering of the trajectories leads to the clusters depicted in figure 3. Since channels "Action A" and "Channel A" concerning the correlation comparison only contain one cluster, these clusters correlates with all other clusters and are therefore ignored. The rest of the dependencies are shown in figure 4.

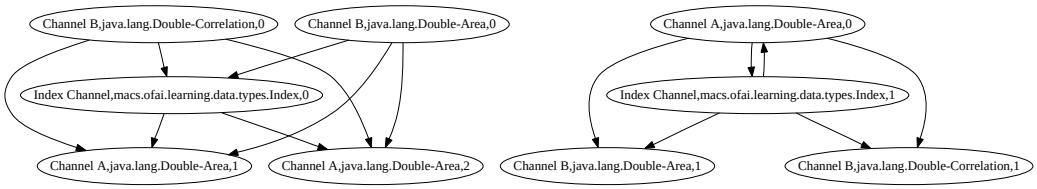

Figure 4: Dependency graph created on the basis of the results of the trajectory clustering process, depicted in figure 3.

As documented in the HTML software documentation, the resulting dependency graph can be stored in the dot-format and, e.g. transferred into a Postscript document or PNG image using the graphviz-tools[10].

[10] www.graphviz.org, Command: `dot -Tps *.dot -o dependencyGraph.ps`, respectively `dot -Tpng`

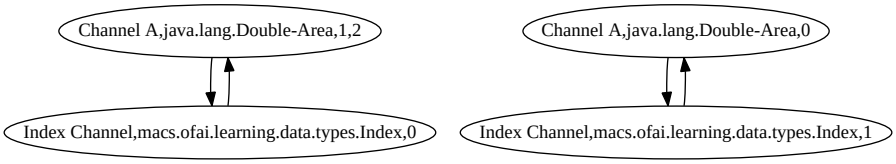

Figure 5: Result of the process summarising nodes that represent the same `Comparator` and trajectory-ID and are attached to the same parent node, performed on the dependency graph for the test-`ApplicationSpace` *Action A*, which is shown in figure4, and subsequent deletion of uni-directional edges.

5.1.4 Building a Dependency Graph: Summarising Nodes

In the dependency graph created so far, edges between nodes can exist, that are unidirectional or bidirectional. In case that a node a is connected to a set of nodes $B = \{b_1, b_2, \ldots\}$ with

- $b_i.Comparator$=$b_j.Comparator$ and
- $b_i.TrajectoryID$=$b_j.TrajectoryID$ for all $b_i, b_j \in B$

these nodes B are summarised to a node $\tilde{b}$ with

- $\tilde{b}.Comparator = b_1.Comparator$,
- $\tilde{b}.TrajectoryID = b_1.TrajectoryID$, and
- $\tilde{b}.Cluster = \{b_1.Cluster, b_2.Cluster, \ldots\}$ for all $b_i \in B$.

Then all nodes $b_i \in B$ are removed from the graph and the new node $\tilde{b}$ is inserted by adding an edge between a and $\tilde{b}$ and by replacing all edges $b_i \rightarrow c$ with $b_i \in B$ by $\tilde{b} \rightarrow c$.

After this summarising, the algorithm checks, if the sum of clusters represented by $\tilde{b}$ correlates with the cluster(s) represented by a. If this is the case, a new edge is introduced from $\tilde{b}$ to a, otherwise the new node is deleted.

Since the summarised node could recursively be connected to more than one node representing the same `Comparator` and trajectory-ID, the steps are to be repeated until there are no more nodes to be summarised.

5.1.5 Extracting Dependency Sub-Graphs

A Dependency Sub-Graph (DSG) is an independent component in the created dependency graph. As shown in section 5.2 there must be an independent component in the dependency

`*.dot -o dependencyGraph.png`

graph for each action application result if there is at least one `Clusterer` that results in a certain cluster (or set of clusters) that correspond to the result. Figure 5 already shows the two Dependency Sub-Graphs that can be derived from the dependency graph in the used example.

All graphs that do not contain any node that is either related to any action related trajectory (e.g. proprioception or behaviour system information etc.), can be erased, since an affordance related classification of the `TrajectorySet`s must per definition be action-related, e.g. if the clustering algorithm finds that there are clusters of locations (world positions the agent has visited), and this information is not coupled to any action related information, this DSG is meaningless concerning affordance learning (though this might be interresting for other purposes). Further approaches to reduce the number of resulting graphs can be applied, e.g. one could use a minimum number of `TrajectorySet`s, that must be described by a DSG.

Additionally the extracted sub-graphs have to prove their usefulness, in later stages of the learning process (finding relevant sensor channels and characterisation process) and in the application of these results by the deliberative and execution part of the over all affordance based control architecture (see *Deliberation Module* and *Execution Module* in Deliverable D2.2.2 [Rom06] and upcoming Deliverable D2.3.1). Thus there must be a feedback, when the deliberation-part uses the information from the *Affordance Representation Repository*, that can be requested by the learning process for elimination of useless data and correction of the learning process. This should be part of the second version of this deliverable together with the responsible partner for the deliberative part.

5.1.6 Query: Assigning an experience to a cluster

A `TrajectorySet` (an action related experience made by the agent) is assigned to a Dependency Sub-Graph (DSG), respectively a DSG is fulfilled by the `TrajectorySet`, if the trajectories clustered by the `Clusterer`s of the sub-graph do not violate the dependencies represented by the sub-graph.

In the used example all `TrajectorySet`s X would fulfil the DSG 0, if the following statements are true:

- `Clusterer`("Index Channel",`IndexComparator`).cluster(X.trajectory("Index Channel")) would result in cluster 0 and
- `Clusterer`("Channel A",`ComparatorDoubleArea`).cluster(X.trajectory("Channel A")) would result in cluster 1 or 2

Accordingly for `TrajectorySet`s X would have to fullfill the following statements, to belong to DSG 1:

- `Clusterer`("Index Channel",`IndexComparator`).cluster(X.trajectory("Index Channel")) would result in cluster 1 and
- `Clusterer`("Channel A",`ComparatorDoubleArea`).cluster(X.trajectory("Channel A")) would result in cluster 0

If the provided `TrajectorySet` does not fulfil the dependencies of any of the DSGs, then the situation is currently unknown to the agent and an empty list is returned. In this case the `TrajectorySet` represents a situation that is unknown to the agent and thus should be stored in the `ApplicationSpace` for later relearning.

If it turns out to be necessary, a qualitative statement could be used, i.e. for each DSG an index can be returned indicating, how many dependencies are fullfilled and how many are violated. Thus a DSG can be "nearly" fulfilled by the `TrajectorySet`, or be fullfilled by a certain percentage rate. This would enable the agent to classify a situation as probably being situation X or situation Y with a factor of uncertainty.

5.2 Correctness

The described algorithm assumes, that a set of `Comparators` exists, so that for each of the different action application results $i \in I$ (e.g. "liftable" and "not-liftable", see Deliverable D5.3.2 [DIK06]) one `Clusterer` C_i exists (using a Comparator $C_i.Comparator$ for trajectories with id $C_i.TrajectoryID$) that results in cluster c_i if and only if the clustered trajectory belongs to a `TrajectorySet` representing the application result i. (See section 5.3 for an extension of this assumption.)

To show that the described algorithm results in the desired set of graphs that correlate with the action application results, one has to show that:

> If the application space contains data of $n = |I|$ different results of the action application and for each application result $i \in I$ exists one `Clusterer` C_i that results in clusters $c_i = \{c_{i_1}, c_{i_2}, \ldots\}$ if and only if the trajectory belongs to a `TrajectorySet` representing the application result i, then the resulting set of Dependency Sub-Graphs contains n disjoint graphs that describe the `TrajectorySets` of the different results.

To show this assertion one has to show, that – using the described algorithm – the nodes that represent the ($C_i.TrajectoryID$, $C_i.Comparator$, Clusters c_i) combinations are not connected by a path in the graph and thus result in different DSGs.

An edge from one node x_1 to another node x_2 means that the cluster(s) represented by x_2 occurs if and only if the cluster(s) represented by x_1 occur. This relation is transitive. Thus from $x_1 \rightarrow x_2 \rightarrow x_3$ follows $x_1 \rightarrow x_3$.

If there is a directed path from one of the n nodes correlating with the application results (x_1) to another of these correlating nodes ($x_2 \neq x_1$), then the cluster represented by x_2 would occur if and only if the cluster represented by x_1 occurs. This would mean, that the result represented by x_2 only occurs if the result represented by x_1 occurs. This is a conflict with the assumption. Thus at least n different dependency sub-graphs are generated.

The fact that a perception of the outside world must be coupled to the internal perception of an action to be relevant for the affordance approach ensures that the algorithm creates a non-empty sub-graph that correlates with the action application result and thus leads to a correlating DSG.

The same statement holds, if one or more action application results correlate with more than one ($C_i.TrajectoryID$, $C_i.Comparator$, Clusters c_i) combination, i.e. if for each action application result $i \in I$ a set of $\mathcal{C}_i = \{(\texttt{Clusterer}\ C_j\ -\ \text{Cluster}\ c_j)\}$ exists so that one and only one of these (`Clusterer` C_j - Cluster c_j) combinations are fulfilled and all these sets are disjoint: $\mathcal{C}_i \cap \mathcal{C}_k = \emptyset$.

5.3 Extension

The described algorithm for the extraction of dependency sub-graph works on the basis of clustered trajectories. This basis can be extended by adding multi-trajectory comparison, i.e. clustering two or more trajectories at a time. This would require a change in the `TrajectoryClusterer`.

5.4 Using Clustering for Partitioning

The class `macs.ofai.learning.partitioning.GraphPartitioner` implements the described clustering method to cluster the `GenericTrajectory`s with respect to the post-application phase. This class implements the `Partitioner` interface and is used by the `PartitioningModule`. The partitioning process can thus be started by changing to the *bin*-directory of the software package and calling:

```
java macs.ofai.learning.modules.PartitioningModule\\
      [ApplicationSpacesModule location] [ApplicationSpaceID]
```

For the test-`ApplicationSpace` used as example in this section, the command would be :

```
java macs.ofai.learning.modules.PartitioningModule \\
     ../applicationSpacesModule/demo ''Action A''
```

The following set of files should result from this partitioning process:

- *SetPartitions.txt*, containing a list of `TrajectorySet`-IDs and an index indicating the partition to which the set belongs.
- *TrajectoryClusters.txt*, containing a LaTeX-table depicting the clusters of each trajectory from all `TrajectorySet` in the partitioned `ApplicationSpace` using the generated `Clusterer`s.
- A set of files ending with *.dot*, containing the graphs that were created during the partitioning process. File *graph_before_sum_0.dot* contains the graph before the summarising process, the file *graph.dot* contains the dependency graph after the summarising process and the deletion of all uni-directional edges and the other dot-files contain the dependency sub-graphs.

Additionally the `PartitioningModule` adds the created `Partitioner` (respectively the `GraphPartitioner`) to the `ApplicationSpace` and stores it in the ApplicationSpaces-Modules directory, using the old name extended with *_new*, i.e. in the example case a new ApplicationSpacesModule named *demo_new* is created containing the `ApplicationSpace` "Action A" containing the old data and a newly created `Partitioner`.

5.5 Using Clustering for Channel Extraction

Since the clustering process partitions the `ApplicationSpace` by extracting correlating trajectory-clusters, this process already extracts the relevant sensor channels for the post-application period and thus for the outcome-events. As depicted in figure 5 the relevant sensor channels extracted for the text-`ApplicationSpace` *Action A* are: *Channel A* and *Index Channel*.

To use the described clustering method to find relevant sensor channels for the detection of cue-events, the clustering is to be limited to the `TrajectorySet`s belonging to one partition and restricting the process to the pre-application part of the trajectories. This is implemented in the `GraphChannelExtractor`.

6 Outlook

The clustering method described in section 5 is implemented and utilised for the partitioning process and the extraction of relevant sensor channels. Those modules and methods which are currently under development are marked with TODO tags in the HTML software documentation of the implementation of the learning architecture. The implementation of the learning architecture is uploaded to the MACS CVS server gibson, where it is updated in short time intervals to provides the newest implementations to the project partners as often as possible. The HTML documentation which is part of this deliverable will thus also be kept up to date.

References

[DIK06] Georg Dorffner, Jörg Irran, and Florian Kintzler. Robotic learning architecture capable of autonomously segment action sequences into affordances. Deliverable MACS/5/3.2 v1, österreichische Studiengesellschaft für Kybernetik (öSGK), Vienna, Austria, 2006.

[DMRW04] Patrick Doherty, Torsten Merz, Piotr Rudol, and Mariusz Wzorek. A conference or journal article summarizing the results of task 4.1. Deliverable MACS/4/1.1 v2, Linköping Universitet Dept. of Computer and Info. Science, Linköping, Sweden, 2004.

[Rom06] Erich Rome et al. Development of an affordance-based control architecture. Deliverable MACS/2/2.2 v1, Fraunhofer Institut fur Intelligente, Analyse- und Informationssysteme, Schlo Birlinghoven, Sankt Augustin, Germany, 2006.

A Document Type Definitions

A.1 DTD - GenericTrajectory

```
<!ELEMENT GenericTrajectory (Id,Type,BeginOfActionApplication,
                             EndOfActionApplication,DataSets)>
<!ELEMENT Data (#PCDATA)>
<!ELEMENT Id (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT BeginOfActionApplication (#PCDATA)>
<!ELEMENT EndOfActionApplication (#PCDATA)>
<!ELEMENT DataSets (Datum*)>
<!ELEMENT Datum (Time,Data)>
<!ELEMENT Time (#PCDATA)>
```

A.2 DTD - Learning Architecture Module LAModule

```
<!ELEMENT LAModule (Type,RegistryHost,ID,Parameters)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT RegistryHost (#PCDATA)>
<!ELEMENT ID (#PCDATA)>
<!ELEMENT Parameters (Parameter*)>
<!ELEMENT Parameter (#PCDATA)>
 <!ATTLIST Parameter
    name CDATA #REQUIRED
>
```

A.2.1 Compiling ESGM idl files using JacORB on the SuSE 9.3 reference system

Install TAO and set environment variables TAO_ROOT and ACE_ROOT as documented in Deliverable D1.1.2 **CITE**, e.g. using the bash shell:

```
export TAO_ROOT=/home/macs/tao/ACE_wrappers/TAO
export ACE_ROOT=/home/macs/tao/ACE_wrappers
```

Install DYKNOW, respectively ESGM and set environment variable DYKNOWHOME and extend variables PATH and LD_LIBRARY_PATH as documented in DYKNOW istallation manual (see file README on the CVS gibson server), e.g. using the bash shell:

```
export DYKNOWHOME=/home/macs/dyknow
export PATH=${ACE_ROOT}/bin:${TAO_ROOT}/TAO_IDL:${DYKNOWHOME}/bin:${PATH}
export LD_LIBRARY_PATH=${ACE_ROOT}/lib:${LD_LIBRARY_PATH}:${DYKNOWHOME}/lib
```

Before one can install JacORB, one needs to install via yast the following pages (and all packes these packages depend on):

- avalon-framework
- avalon-logkit
- concurrent
- jpackage-utils >= 0:1.5

Then install JacORB and tanukiwrapper packages available at the following internet sites (linebreak is just used for readability reasons):

- http://rpmseek.com/rpm/jacorb-2.2.2-3jpp.noarch.html ?hl=com&cs=jacorb:PN:0:0:0:0:2307690
- http://rpmseek.com/rpm/tanukiwrapper-3.1.1-1jpp.i386.html ?hl=com&cs=tanukiwrapper:PR:0:0:0:0:2309730

Set the environment variable JACORB_ROOT to /usr/share/jacorb-2.2.2/, e.g. using the bash shell:

```
export JACORB_ROOT=/usr/share/jacorb-2.2.2/
```

File /usr/share/jacorb-2.2.2/bin/idl contains the wrong paths after installation, so correct the contens of that file to:

```
 #!/bin/sh
java  -classpath /usr/share/java/avalon-logkit.jar:\
/usr/share/java/avalon-logkit-1.2.jar:\
/usr/share/java/jacorb/idl.jar:/usr/share/java/jacorb/logkit-1.2.jar:\
${CLASSPATH} org.jacorb.idl.parser  "$@"
```

To compile all ESGM idl files to java use the following command:

```
${JACORB_ROOT}/bin/idl -all -I${DYKNOWHOME}/idl/dyknow/ \
-I${DYKNOWHOME}/idl/common -I${TAO_ROOT}/orbsvcs/orbsvcs/ \
-I${TAO_ROOT}/ -I${JACORB_ROOT}idl/jacorb/ \
-I/usr/share/jacorb-2.2.2/idl/omg/ esgm.idl
```

The interfaces created by using this procedure are available in the new project PMtoLMinterface project available on the projects CVS server gibson.