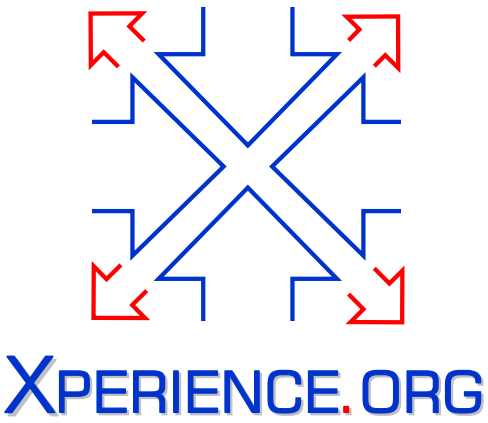
Project Acronym:	Xperience
Project Type:	IP
Project Title:	Robots Bootstrapped through Learning from Experience
Contract Number:	270273
Starting Date:	01-01-2011
Ending Date:	31-12-2015



Deliverable Number:	D1.1.1
Deliverable Title:	Initial definition of Xperience benchmarks
Type (Internal, Restricted, Public):	PU
Authors:	G. Metta, T. Asfour, A. Ude, N. Krüger, Ch. Geib, J. Piater, F. Wörgötter, M. Steedman, G. Sandini, R. Dillmann
Contributing Partners:	ALL

Contractual Date of Delivery to the EC:	31-07-2011
Actual Date of Delivery to the EC:	06-02-2012

Contents

Chapter 1

Executive Summary

This deliverable has several goals in the context of Xperience. It first tries to provide the foundations of a model of cognition that incorporates development, sensori-motor coordination, affordances and high-level prediction, interaction and communication in terms of "structural bootstrapping". Secondly, it proposes a set of benchmarks to formalize the question of how to measure cognitive systems performance in the context of the definition of cognition set above. Finally, it tries to provide a "paper" cognitive and software architecture to implement cognitive skills based on development, affordances and "structural bootstrapping".

In the following, we will analyze first our stance on cognition, followed by a sketch of a cognitive architecture which tries to merge different views on the development of affordances and their use to reason about using structural bootstrapping. The rationale of this approach to cognition stems from the following consideration [from the Xperience Annex I]:

> *Current research in enactive, embodied cognition is built on two central ideas: 1) Physical interaction with and exploration of the world allows an agent to acquire and extend intrinsically grounded, cognitive representations and, 2) representations built from such interactions are much better adapted to guiding behavior than human crafted rules or control logic. Exploration and discriminative learning, however are relatively slow processes. Humans, on the other hand, are able to rapidly create new concepts and react to unanticipated situations using their experience. "Imagining" and "internal simulation", hence generative mechanisms which rely on prior knowledge are employed to predict the immediate future and are key in increasing bandwidth and speed of cognitive development. Current artificial cognitive systems are limited in this respect as they do not yet make efficient use of such generative mechanisms for the extension of their cognitive properties.* **[page 5 - Summary]**

As mentioned in the Annex I, this is only version 1.0 of the cognitive architecture design, which will be refined and augmented with technical details throughout the continuation of the Xperience project. Especially, results from actual implementation and consequently experiments will be used to improve the initial design. The evolved cognitive architecture will be described finally in D1.2.1 (due at month 31). We plan to re-run the exercise of improving the architecture description approximately every 12 months in order to keep the structure up to date with the development of the corresponding software modules.

In summary, this deliverable (and our modus operandi) plans for a set of operational (read practical) steps as follows:

- definition of a developmental cognitive architecture;
- use the cognitive architecture to constrain the software architecture (also described here);
- definition of a set of benchmark to validate the implementation;
- detailed definition of the implementation (in software) and of the scenario (e.g. kitchen scenario, etc.).

These steps are described in the following and shown diagrammatically in figure 1.1. We start by defining a generic cognitive architecture and proceed to the implementation of some of its salient features which are then validated through the benchmarks. The result of the testing and validation phase can both signal revisions in the implementation and in the structure of the cognitive architecture itself. This deliverable describes the *Cognitive Architecture* and the *Benchmark* blocks of the diagram. The *Software Architecture* consists of the work developed in several WPs and will be the result of the integration effort of WP5.

In this work, we are particularly indebted with the view on cognition provided and developed by the RobotCub and PACO+ projects from which we have absorbed ideas considerably and founded our recent work. We would like also to thank David Vernon, much of this work has been taken (inspired in parts and sometimes verbatim) from his book on the development of cognition which is in turn derived from RobotCub [VvHF10].

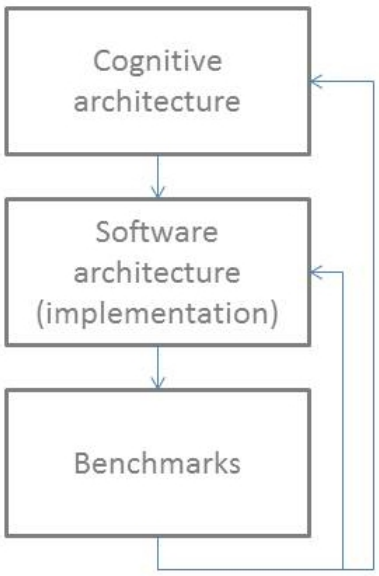


Figure 1.1: A block diagram showing our modus operandi in Xperience. The cognitive architecture and benchmarks are described in this deliverable, while the implementation is the result of the work developed in the other WPs.

Chapter 2

Cognitive architecture

2.1 Definition

Our work is founded on the premise that (a) cognition is the process by which an autonomous self-governing agent acts effectively in the world in which it is embedded, that (b) the dual purpose of cognition is to increase the agents repertoire of effective actions and its power to anticipate the need for and outcome of future actions, and that (c) development plays an essential role in the realization of these cognitive capabilities.

Cognitive agents act in their world, typically with incomplete, uncertain, and time-varying sensory data. The chief purpose of cognition is to enable the selection of actions that are appropriate to the circumstances. However, the latencies inherent in the neural processing of sense data are often too great to allow effective action.

Consequently, a cognitive agent must anticipate future events so that it can prepare the actions it may need to take. Furthermore, the world in which the agent is embedded is unconstrained so that it is not possible to predict all the circumstances it will experience and, hence, it is not possible to encapsulate a priori all the knowledge required to deal successfully with them. A cognitive agent then must not only be able to anticipate but it must also be able to learn and adapt, progressively increasing its space of possible actions as well as the time horizon of its prospective capabilities. In other words, a cognitive agent must develop.

There are many implications of this stance. First, there must be some starting point for development – some phylogeny – both in terms of the initial capabilities and in terms of the mechanisms which support the developmental process. Second, there must be a developmental path – an ontogeny – which the agent follows in its attempts to develop an increased degree of prospection and a larger space of action. This involves several stages, from coordination of eye-gaze, head attitude, and hand placement when reaching, through to more complex exploratory use of action. This is typically achieved by dexterous manipulation of the environment to learn the affordances of objects in the context of ones own developing capabilities. Third, since cognitive agents rarely operate in isolation and since the world with which they interact typically includes other cognitive agents, there is the question of how a cognitive agent can share with other agents the knowledge it has learned. Since what an agent knows is based on its history of experiences in the world, the meaning of any shared knowledge depends on a common mode of experiencing the world. In turn, this implies that the shared knowledge is predicated upon the agents having a

common morphology and, in the case of human-robot interaction, a common humanoid form.

While this view is fairly broad in scope, we argue that it is not enough to build artificial cognitive systems that show fast acquisition of new skills from a small number of examples (possibly one). That is, the definition is complete but in practice, we need mechanisms that enable fast learning. To this end we would like to augment the generic enactive view of cognition sketched above with the so-called "structural bootstrapping". Structural bootstrapping is an idea taken from child language acquisition research. Structural bootstrapping is a method of building generative models, leveraging existing experience to predict unexplored action effects and to focus the hypothesis space for learning novel concepts. This approach enables rapid generalization and acquisition of new knowledge and skills from little additional training data. Moreover, thanks to shared concepts, structural bootstrapping enables enactive agents to communicate effectively with each other and with humans. Structural bootstrapping can be employed at all levels of cognitive development (e.g. sensorimotor, planning, communication). In a sense, together with the standard view of discriminative learning methods whose inference is typically supported by generic basis functions, here we advocate the use of a new inference model which supports the use of complex patterns that are simultaneously more specific and more powerful in providing generalization from small training sets and transport of concepts across domains. Therefore, even if both ways of making inference from data eventually amount to what basis function is employed, in practice we need to consider different methods because the implementation can vary considerably. It can be the case that structural bootstrapping patterns are engineered in rather than learned. Effectively, joining different methods in a layered architecture, enables the type of powerful learning advocated recently by the "deep learning" community.

By equipping embodied artificial agents with the means to exploit prior experience via generative inner models, the methods to be developed in Xperience are expected to enable efficient learning through exploration, predictive reasoning and external guidance.

The concept of structural bootstrapping is derived from early language acquisition processes described for example by Trueswell and Gleitman [TG07], who point out that a crucial component of children's post two word stage language acquisition is what Gleitman [Gle90] called syntactic bootstrapping, whereby, once basic syntax is learned, the syntactic type of a novel world can be worked out generatively (often on one trial), from the type that would complete a parse, while the meaning can be established from the context that supports it. For example, an unknown word may be assigned the role of the acted on entity, if being recognized from its position in the sentence as the object. If subject and verb are known to the agent, the actual effect of the execution of the action encoded by the verb (the effect of the action) can be inferred, too, leading to an inference about the changes (of the attributes/adjectives of) the object. Thus, the recognition of the underlying grammatical structure together with partial knowledge of some words leads to a powerful predictive process by which in a single shot the correct meaning of the missing parts can be inferred.

In Xperience we generalize this notion to structural bootstrapping and employ it at all levels of the cognitive domain. Solutions found for one problem, can be transferred to similar domains. For example, different grippers, pliers, pincers, and tongs all share structural and action similarities that allow an agent that knows how to use one of them to plan for the use of the others (possibly with some fine tuning of motor skills).

Effectively, the Xperience cognitive architecture can be characterized as hybrid: i.e. using simultaneously features of cognitivism and of the so-called emergent view of cognition [VvHF10].

2.2 Requirements

The proposed cognitive architecture aims at integrating the concept of "structural bootstrapping" with the autonomous development of affordances. Several keywords that distinguish the two components are shown in the table below:

	Xperience	previous projects
cognitive model	cognitivist	emergent
learning type	generative	discriminative
data flow	inside-out	outside-in
representation	punctuated	distributed
	symbols	sub-symbolic
	discrete	continuous
training set size	≈ 1	large

The generic a priori requirements of this architecture are summarized below in part derived from the Annex I of Xperience and in part from our previous work in the RobotCub and PACO+ projects:

- the proposed architecture has to incorporate the autonomous learning (development) of the basic symbolic representations of the cognitive architecture, which are in the following called OACs (object-action-complexes [KGP+11]): a precise definition is reported later;
- by virtue of the previous remark, the architecture need to define a phylogenetic configuration of the system and its developmental rules;
- the proposed architecture has to define mechanisms to learn the "structures" needed to apply generative methods;
- as a consequence, motivations play a fundamental role both in promoting the development of representations in outside-in processes and in the learning of the generative models;
- the proposed architecture has to define the interaction of the outside-in and inside-out processes;
- the architecture can (and should) reuse its components (development and structural bootstrapping) also in the interaction with other cognitive agents;
- the previous remark enables a definition of cooperation, interaction and communication in terms of the generative methods, that is, in terms of the possibility of reusing knowledge across domains thanks to improved generalization;

The cognitive architecture has to address the technical and scientific goals of the project and it is in fact the foundation of our work in the following periods. In particular, it has to:

1. employ a stimulus driven developmental approach (outside-in) to generate semantic sensorimotor categories constituting the basic experience of the system. These categories can be seen as the elemental token of exchange with higher level cognition [Cla01]. Fundamentally, these developmental processes rely on explorative learning and adaptation;
2. augment the above mechanism for the acquisition of a shared epistemology with structural bootstrapping for the generative extension of knowledge (inside-out). These generative models are subsequently validated and further extended through experience;
3. combine these two processes into a dynamic loop for behavior-based model updating;
4. use the developed sensorimotor categories as a foundation for natural language understanding and communication between humans and robots. In this respect, we will prove the transferability of sensorimotor categories between cognitive agents using multiple robotic platforms and human experimenters in order to build complete systems of interacting agents;

A fifth goal of the project, that is to

> *...transfer the above aspects to technology and create a complete system by fundamentally focusing this effort on merging outside-in techniques with inside-out model-building, which represents the scientific core concept of Xperience. This will be done by implementing and validating the theory on robots of humanoid shapes and letting them interact with each other or with humans. Finally, we will develop validation tools and procedures to assess progress.*

is clearly an overarching goal of our work (providing a set of constraints) that forces us to make plans for a cognitive architecture that can be implemented into physical instantiations of cognitive systems and, in particular, instantiations shaped as humanoid robots.

At the general level, the Annex I of Xperience is considerably precise at identifying some elements of the architecture. More in details:

- **exploration based knowledge acquisition** – Xperience will continue to use a stimulus driven developmental approach to provide the system with a mechanism for creating experiences and for the formation of semantic sensorimotor categories by grouping (generalization). This outside-in process is explorative and data-driven and relies on development and sensorimotor coordination. At its highest level, through exploration based learning the outside-in process delivers affordances creating a grounded symbolic layer. Following the developmental roadmap defined in [VvHF10], we require the architecture to include the following skills:

- posture and locomotion
- gaze
- reaching
- manipulation
- social abilities

complemented by vision (and in general perception) of space and objects. We will see that each mode of action needs to be supported by specialized visual primitives (and core systems [VvHF10]). In order to develop these skills autonomously (data-driven, outside-in), we require the architecture to define the system's phylogeny and ontogeny. That is, we have to define the system's initial state and its developmental rules. Furthermore, such architecture has to enable the following possibilities (at least at the level of sensorimotor control):

- **perception and action**: clearly, the architecture has to provide means for controlling action via perception, detect mistakes and correct them through learning or local adaptation (i.e. in closed-loop);
- **anticipation**: since feedback control (closed-loop) is not always well adaptive because of delays, a form of prospection is required, this is implemented by learning internal models and relying on them for open loop control;
- **adaptation**: internal models are not perfect and modification to the controller might be needed continuously to an ever changing system's or interaction dynamics. This is true both for low-level motor control as well as to high-level interaction dynamics and therefore, learning is required at all levels;
- **motivations**: exploratory as well as social motives keep the system in a continuous state of "excitation" and acquisition of new knowledge. They are required to guarantee that the cognitive system can develop in autonomy;
- **autonomy**: this is the ability of the system of maintaining its structural consistency (homeostasis) while interacting with the environment and simply stated it is a requirement to forbid fiddling with its structure and internal state once it's been deployed. Clearly, this is a very high level property of a cognitive system which we might only see realized if we are completely successful.

To summarize, we need to be able to deploy a set of functionalities on our platforms and for each of them define both the phylogeny and ontogeny (that is, the initial state and the learning rules). For each motor control skill, we will implement both closed-loop control (servoing) and anticipation (open loop model based control) together with machine learning methods that starting from data improve the internal models of the system's interaction with the environment. Examples of the machine learning methods are supervised function approximation techniques (e.g. Gaussian processes) and reinforcement learning. The architecture will include motivations and also in this case, we will consider both an initial state (core motives) and development (intrinsic system's goals).

A further element of development has to do with the OACs, the Object-Action Complexes, which are the link between continuous sub-symbolic representations of the motor acts and the punctuated representations needed for structural bootstrapping

(and for applying structural bootstrapping in prediction, interaction and communication). OACs have been defined in the PACO+ project and will be considered in a later section.

- **building of generative models** must rest on a solid methodological basis. Here Xperience transfers methods developed from research on human language acquisition, by which humans generalize knowledge across domains. We use structural bootstrapping for the generative extension of knowledge. Driven by motivation, using the current system's goal, the agent will internally simulate and reuse experience to generate potential (novel) solutions to satisfy the goal. This way the system is endowed with the ability to imagine new solutions via generalization beyond the observed examples, to predict future events from past experiences, and to explain observed events (imagining a causal chain leading to that event).

  Cognitive agents which have a memory and some model of their world – hence, which have already stored some sensorimotor categories – can perform one-trial identification, hence cognition in its original sense. Learning of this kind requires a generative theory that covers more instances than have hitherto been encountered, so that novel actions and objects can be recognized either as instances of known concepts or by augmenting the theory with a new concept of a known type. The latter process crucially involves inference from the state of the world under the current internal knowledge representation, to a compound concept of the known type. In the limit, this process becomes the full scientific hypothetico-deductive method including the collection of further knowledge that is not currently internally represented at all. We assume that the intermediate process of structural bootstrapping is a crucial component of cognitive development. For example, an agent who knows how to peel potatoes with a knife, and who encounters a potato peeler, may well not understand its function on the basis of appearance alone. However, a single demonstration of its use will be understandable in terms of an existing theory of peeling potatoes, so that its syntactic type and role in existing plans is immediately available. The main remaining problems are those of motor learning, and possibly some hypothetico-deductive exploration of other predicted affordances (such as peeling other things).

  These methods will be applied in the Xperience cognitive architecture not only to language but in the wider context of the syntax of actions (represented as OACs in the general case).

  In summary, we see structural bootstrapping as the following process:

  - The robot learns patterns of recurring combinations of symbols (OACs);
  - These patterns are general enough to be transferred across domains (they act as templates for combinations of OACs);
  - Therefore, the same patters can be used to "try" inference in domains which are different from those where the patterns were initially acquired (the process of structural bootstrapping).

- **integration of outside-in and inside-out processes** – a generative dynamic loop for model updating is created in a natural way. This is done by the interaction between outside-in and inside-out, which will be employed in Xperience as one major drive for the generation of knowledge. Note, multiple processes and inner

loops are not excluded in this dynamic architecture, by which internal processing can take place. This question is essentially concerned with the stable interaction of the outside-in processes with the inside-out processes. Stimulus driven processes are responsible for learning through sensorimotor exploration while the generative processes are responsible for the instantiation and validation of the internally constructed models. Their interaction must be modulated to ensure stable and improving operation of the system (autonomy preservation). Like in natural systems, this is accomplished by a motivational system (sometimes referred to as the affective system) which is ultimately concerned with improvement of the predictive capacity of the cognitive system and the expansion of its possible modes of interaction with the world around it.

This is a property of the architecture as a whole rather than anything specific to a module or subsystem. That is, we need to modulate the interaction of outside-in vs. inside-out processes and make sure they give rise to meaningful and stable overall dynamics which in practice enables the acquisition of new "symbols" (new OACs) and their use for new "patterns" (syntax) in the overall aim of managing the system's behavior.

- **cooperation, interaction and communication** – we expose the same developmental processes (the same cognitive architecture) of exploration based and generative learning not just to a world of objects but also to a world with other cognitive agents. These cognitive agents present much higher number of degrees of freedom in their behaviors and possible reactions to the robot. Other cognitive agents can be seen as more complicated "objects" in a sense possessing a complex dynamical behavior.

  However, the same generative principles can be employed. If structural bootstrapping discovers a model which is shared by two agents, e.g. let's imagine the robot and a person, then effectively for the robot the number of degrees of freedom of the responses of the other agent (the person) will be effectively lower since his/her behavior becomes more predictable. The robot and the person share a common experience of the world. In this way, the consensual understanding of the world that the robot inhabits will be consolidated and expanded in its updated generative model.

  The core elements of cooperation, interaction and communication and their development are described in the Annex I and they are reported here for completeness.

  The earliest symbolic communications of a child, including its earliest use of its first language, are necessarily attached to a previously established conceptual system for representing the world, including other individuals. Much of the apparently explosive increase in the child's cognitive abilities that follows the onset of language can be argued to stem from the ability to recognize the semantic type of a concept that is in established use in adult utterance, and whose content could in principle be inferred from the context under the child's existing theory, but which, unless aided by the communication, the child might never discover.

  Gleitman [Gle90] called this process "syntactic" bootstrapping of language, and showed that it is central to acquisition of verbs like "think" and "believe", and the corresponding concepts. It is the linguistic manifestation of the more general process of generative structural bootstrapping outlined earlier. It is crucial to this process that the child already has a knowledge representation that will allow it to entertain

the new concept. We claim that the same process of structural bootstrapping can be applied in artificial cognitive systems, with the same prerequisite of a generative symbolic theory capable of expressing the concept to be learned. The same process underlies even later cognitive processes such as learning from explanation and ostentation, to formalize much abused notions like "imitation" and "imagination".

Structural bootstrapping using generative models allow us to successfully approach the problem of advanced interaction and communication between agents because it enables one-trial identification by inner simulation (imagining). Similar to inference from self-observation, through structural bootstrapping agents will be able to (partially) infer intentions of other agents.

This approach enables the efficient transfer of information and of cognitive concepts between different platforms including the interaction and communication between the human and the robot.

2.3 Object-Action Complexes - OACs

The definition of OACs is taken verbatim from the work of Wörgötter et al. [WAK+09] which is in turn part of the work of the PACO+ project. We will also briefly address the alternative view of a similar concept derived from the RobotCub project.

OACs had first been discussed by the European PACO+ Consortium as a possible way to better formalize the requirements for a machine to approach some level of cognitive complexity. OACs are related to state-action transitions e.g. known from machine learning [GMP+06][SB02].

They rest on the suggestion that objects and actions are inseparably intertwined. Starting with Gibson's notion of affordances: A hollow thing with liquid may suggest drinking. For this we define an OAC formally by $\left[O \rightarrow^A O'\right]$, which says object O suggests action A and transforms under this action into object O' (cup-full to cup-empty) as the final outcome of this action. Note, rigorously one should define the OAC with respect to the Attributes (full, empty) of an object that get altered by an action. This should be kept in mind when using the abbreviated notation $\left[O \rightarrow^A O'\right]$. The notion of OACs, however, goes beyond Gibson and the intertwining of Objects and Actions becomes more evident when considering the role of Actions more closely. While objects may suggest actions, it is often the action(-plan) that defines the object-ness of a physical thing. This become clear by following example: It is the action of drinking that makes this $\text{thing}_{hollow,full}$ "a cup" ("a container", etc.). The decisive influence of the action becomes immediately obvious if you plan to turn the (same!) $\text{thing}_{solid,bottom}$ upside down to use it as "a pedestal" for some figurine for your mantelpiece decoration. Hence, the planned and executed action turns a thing with some (required) properties into a meaningful object. Depending on the planned actions, different properties of the same thing (hollow, full vs. solid bottom) become important. Clearly it is a very difficult (cognitive) problem for an agent to find out which properties are important and which are not.

Therefore, according to this definition an OAC consists of:

- the object O;
- the action A;

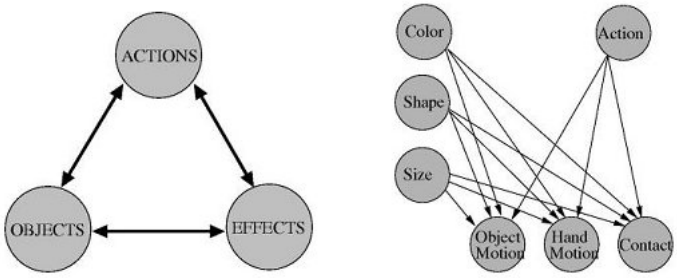


Figure 2.1: (a) General affordance scheme relating actions, objects (through their characteristics) and the resulting effects. (b) A particular BN encoding affordances.

Inputs	Outputs	Function
(O, A)	E	Predict effect
(O, E)	A	Recognize action & planning
(A, E)	O	Object recognition & selections

Table 2.1: Using affordances for prediction, recognition, and planning.

- the effects on the object that make it into O'.

Similarly, the RobotCub project arrived at a definition of affordances that is strikingly compatible to that of OACs. The route that led to the RobotCub's definition was slightly different owing to the collaboration with brain scientists. In this case we decided to consider explicitly the effect E on the object rather than the new object's state O'. For any practical implementation this difference is irrelevant.

For learning affordances, RobotCub's model used Bayesian Networks (BN) to model the dependencies between robot actions, object characteristics, and the resulting effects [MNN$^+$10]. Briefly, a BN is described by a set of nodes that represent random variables, a set of directed arcs that encode conditional dependencies and a set of conditional probability distributions. A BN encodes causality since an arc from a node X to a node Y can be interpreted as X causes Y. We assumed that the iCub has developed certain skills prior to be able to learn affordance: a motor repertoire (A), perhaps derived from experience, an object feature repertoire (F) also potentially acquired via object manipulation and the effects (E) resulting from manipulating the environment.

The interaction of the iCub with the environment is therefore formalized in using one action a from A on certain objects with features F (or a subset of them) to obtain effects e from E. This information can be used to estimate the BN structure and parameters using different learning algorithms. These parameters can be updated online as the robot performs more experiments. Also, they can be updated by observation of other agents. Examples are shown in Figure 2.1.

This model has some nice properties; for example, affordances can be learned autonomously by experience and by self-observation, restricting the update of the probability distributions. Features can be either selected or ignored, depending on their salience, and the model can be used to perform prediction, recognition, and planning, depending on how the affordance network is traversed. This traversal is based on probabilistic

queries. These queries may take as input any combination of actions, objects and features and compute conditional distributions of one or more of the other variables. Table 2.1 summarizes some of the basic operations that can be performed with the network.

From these compatible description of OACs, it is clear that they represent the link between the sub-symbolic world – having to consider continuous actions – and the symbolic nature of objects as specific "chunks" of the world in the agent's own epistemology. In the following, we will need to be able to consider, represent and store at least the three components of the OACs specified above, i.e. objects, actions and transformed objects (or effects) with the caveat that what changes is a specific attribute (or set of attributes) of the object (empty, full, etc.). Advanced details of the representation of OACs is also shared by the two models, e.g. both have a concept of reliability of the model (probabilities).

The latest definition of OACs combines elements of these two views into a single representation which has been completely detailed in the paper by Krüger and colleagues [KGP+11] as part of the PACO+ project. This text is repeated in D2.3.1. We think that it is important to clarify it here too since it makes the discourse more complete.

OACs formalize sensorimotor schema [Pia76, CA97] and by that generate the required data for generalization. An OAC is a dynamic entity that unifies the perceptions and associated actions involved in the performance of a habitual behavior which is described in detail in the paper mentioned earlier [KGP+11]. The schema represents knowledge generalized from all the experiences of the behavior generated. It also includes knowledge about the context in which the behavior was performed as well as expectations about its effects. During cognitive development OACs are refined and combined. An OAC definition is split into three parts, (1) a symbolic description consisting of a prediction function defined over an attribute space, together with a measure of the reliability of the OAC, and (2) an execution specification that defines how the OAC is executed by the embodied system and generates experience in terms of 'experiments' and (3) a specification of how the learning associated with the OAC is realized based on the 'experiments' generated by the executed OACs. More formally (see also figure 2.2):

Definition 2.3.1. An Object-Action Complex (OAC) is a *triple*:

$$(id, T, M) \tag{2.1}$$

- id is a unique identifier for an execution specification,
- $T : S \rightarrow S$ is a prediction function defined on an attribute space S encoding the system's beliefs as to how the world (and the robot) will change if the control is executed, and
- M is a statistical measure representing the success of the OAC within a window over the past.

An execution function execute (see below) can map an OAC id to an 'experiment' which is defined in the following way:

Definition 2.3.2. Given an attribute space S and an OAC with identifier id defined on S, an experiment is a triple:

$$(s_0, s_p, s_a) \tag{2.2}$$

where:

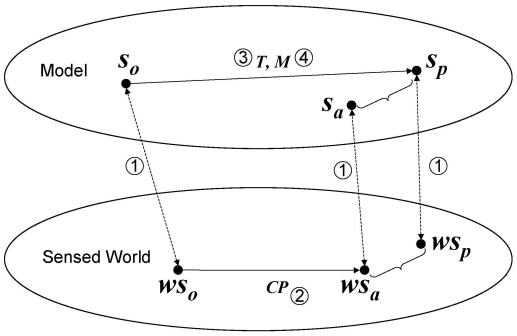


Figure 2.2: Graphical representation of an OAC and the OAC learning problems. This shows how the actual state ws_a (corresponding to s_a in the model) resulting from the execution of the control program CP diverges from the state s_p predicted by the OAC's prediction function T. This divergence drives the learning (i.e. refinement) of the OAC. For further explanation see text.

- $s_o \in S$, captures the state of world before execution
- $s_p \in S$ such that OAC id predicts that state s_p will result from its execution in s_o, i.e., $s_p = T_{id}(s_o)$, and
- $s_a \in S$ such that s_a is observed as a result of actually executing OAC id in state s_o.

Thus, an experiment is an *empirical event* by which OACs will be grounded into sensory experience. As empirical grounded events, such experiments can be used to update OACs in cycles of execution and learning based on evaluations of their success. Note that sometimes an experiment is actually not used directly for learning but stored in some short term memory (see, e.g., [Bad99]) until resources for learning are available (e.g., during 'sleeping phases').

The definition of OACs as capturing both symbolic and control knowledge about actions highlights a number of learning problems (addressing different aspects of the OAC as indicated in figure 2.2) that must be addressed for OACs to be effective (for details, see [KGP+11]). We note that while each of these learning problems can be addressed by recognizing the differences between predicted states and those states actually achieved, they may still require different learning algorithms (e.g., Bayesian, neural network-like, parametric, non-parametric, etc.). It is up to the OAC designer to choose an appropriate learning mechanism.

2.4 Structural bootstrapping

Besides OACs, the other major component of the Xperience architecture is *Structural Bootstrapping*. The following text taken verbatim from D2.3.1 enters somewhat into the elaboration of the idea for Xperience.

Cognitive development proceeds at two different levels of abstraction. Figure 2.3 shows two such parallel tracks of development. On the bottom is the sensorimotor track which shows the development of sensorimotor schema or OACs, which are observable in infant

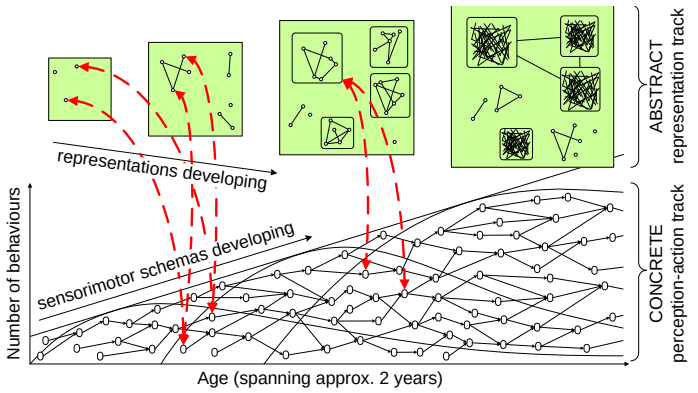


Figure 2.3: Conceptual diagram, an overview of infant developments on both a low level sensorimotor track and a higher level representational track; for explanation see text.

behavior. Each node in the lower track corresponds to one OAC. A directed edge travels from each ancestor node to its descendants; for example the OAC for pulling a string descends from a basic grasping OAC (a number of examples are outlined in [GKKed]). Some OACs have more than one ancestor. The top of Figure 2.3 is the abstract track which shows the parallel development of the underlying world knowledge. Nodes in the upper track correspond to (for example) fragments of object knowledge which are common to a number of OACs, and fragments of spatial relationships; these are gradually linked up as development progresses (to the right), to eventually form a more comprehensive world knowledge.

Early fragments of object and spatial knowledge are likely to be very context specific, and are strongly associated with the OACs they have been abstracted from. It is only after a long developmental process (moving to the right in Fig. 2.3) that these fragments become more objective, and this developmental process must involve some sort of "representational re-description" [CKS93].

For the lower track we see a developmental process in which a small set of innately defined OACs lead to a large variety of OACs through branching and specialization. During this developmental process, the effects of the OAC become increasingly predictable and can then be used more and more purposefully by the cognitive agent for the planning of behavior. In parallel to (and triggered by) the development of individual OACs, more generic world knowledge is built up; as illustrated in the upper track of Fig. 2.3. This is done through the abstraction of empirical data gained during the execution of the OACs.

There is a parallel development and interaction of observable behaviors and the increasing abstract world knowledge which is based on the experiments generated by the OACs. *Structural bootstrapping* addresses the process indicated by the red arrows (see figure 2.3) in which generated abstracted world knowledge effects the behavioral track by means of establishing new OACs or modulating existing OACs. More specifically, the dashed red arrows in figure 2.3 illustrate bidirectional links between the abstract and sensorimotor tracks. To avoid clutter only six links are shown, but in reality all representational fragments will be linked to the OACs. In one direction (outside-in) representations are linked to the OAC they have been generalized from (and hence can immediately link to actions which can manipulate the represented object or spatial relation).

In the other direction (inside-out), more advanced OACs make use of the newly formed representations, for example in their description of the context in which a behavior may

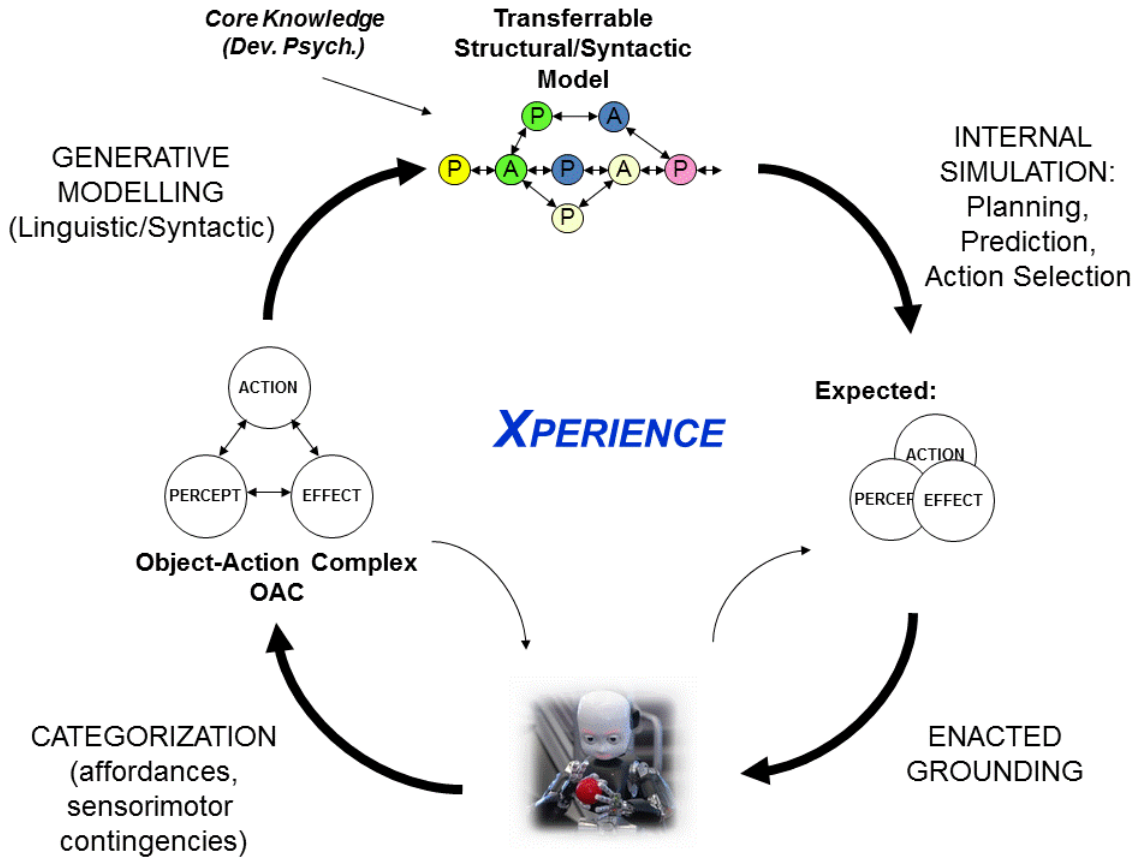


Figure 2.4: Initial sketch of the Xperience cognitive architecture from the Annex I.

be performed, or its effects, or the control policy followed during execution of the schema. *These feedback processes are at the core of structural bootstrapping since they allow to apply abstracted concepts generated in the outside-in process as an inside-out process facilitating cognitive behavior based on predictions derived from internal concepts.*

2.5 Components

This section contains additional details about the proposed components (modules) of the Xperience cognitive architecture. A very first diagram showing the architecture is shown in figure 2.4 as derived from Annex I. In particular, given the central role played by the OACs, we will concentrate first on the algorithms needed for learning them from data (outside-in component of the architecture).

The following list represents the major components of the planned architecture and will be detailed below. Clearly at this stage, while some research has been started, the modules are not fully integrated or made to fit into a coherent implementation of the Xperience architecture. On the other hand, this is exactly the task of WP5, namely, to transform the Xperience research into a complete and coherent software implementation to run on various robotic platforms (at least the iCub and ARMAR-III at the moment). These are:

- action generation, i.e. kinematics, dynamics, inverses (closed and open loop);
- action coding, i.e. dynamical motion primitives (discrete and periodic);

- body models, i.e. simulation of the robot's body;
- visual representation, i.e. Early Cognitive Vision system (ECV);
- object visual representation, i.e. Markov random field hierarchy;
- planning with OACs;

We have also a number of examples of special OACs as follows:

- grasp densities, merging the concept of "effects" (success vs. failure) with the visual appearance of the object represented as ECV features;
- iCub learning simple affordances using Bayes Net (BN);

and we clearly have to define structural bootstrapping as a set of modules that "use" OACs to obtain better generalization, faster learning and complex behaviors as those needed for interacting with other cognitive agents.

In what follows, we describe a software architecture which follows a significant subset of the guidelines identified in section 2.2 and 2.3 to a greater or lesser extent. The goal of this preliminary architecture is to integrate at least conceptually some of the motor skills (e.g. reaching, manipulation) with learning (adaptation of motor skills) and the construction of OACs (affordances) in simple cases. In other words, the current goal is to build a minimal but faithful functioning system as a proof of principle. Figure 2.5 shows a first sketch of the software modules with approximate connectivity cast into the three-tier architecture shown earlier in figure 2.4.

Gaze control, reaching, and locomotion constitute the initial set of simple goal-directed actions. Memory systems are included to effect a simplified version of internal simulation in order to provide prediction (prospective abilities for action) as well as the construction of affordances from the combination of sensorimotor inputs. Importantly, everything is filtered by a "diffused" attention system which selects inputs and outputs by focusing information processing to the salient elements of the scene and, conversely on the output side, toward the most appropriate action or subset of skills. Attention can be both exogenous as well as endogenous.

At the top tier, we find the goal/motivational system which provide initially with the motivations to explore the sensorimotor space or to interact with others (social motives) as characterized by Von Hofsten in [VvHF10] chapter 2 and 3. Structural bootstrapping is considered at the top of the hierarchy by enabling one-shot learning and affordance-based (or OACs based) internal simulation and planning skills.

In terms of the available software, with varying degrees of completeness, we find:

- action generation and coding:

  gaze, reach and grasp (platform dependent).

- body models:

  simulation of the robot's body and planning (e.g. grasp).

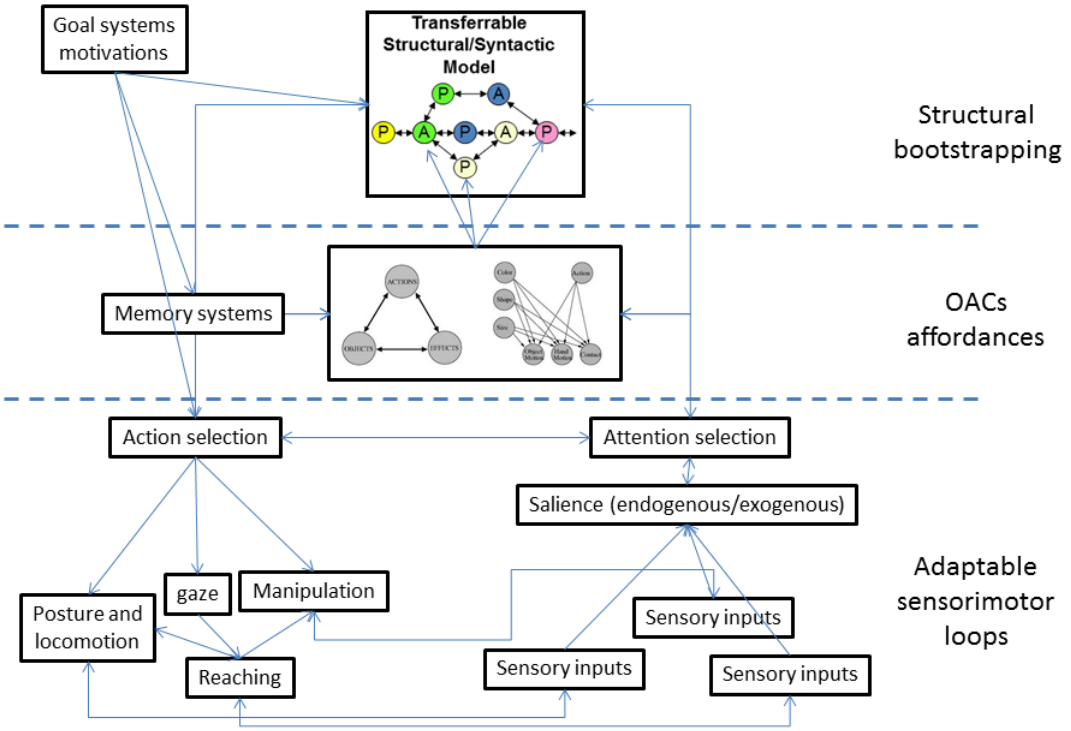


Figure 2.5: Initial sketch of the Xperience software architecture derived from the cognitive architecture.

- visual models:

  early vision and visual representation of objects and actions.

and clearly it remains yet to achieve the first OACs/affordances using an integrated architecture. Detailed reporting of the implementation of these skills can be found in the relevant deliverables, namely D2.1.1, D2.3.1, D3.2.1 and D4.1.1. It is fair to say that a great deal of integration is still to be done especially to smooth the use of the architecture on various hardware platform (e.g. iCub, ARMAR-III, etc.). This activity is planned as part of WP5 and will be delivered (as planned) at the end of the second year.

With respect to the three major groups of modules described above, we report additional details and the link with the other Xperience deliverables in the following table. In the following, level 1 corresponds to the "Adaptable Sensorimotor Loop" of figure 2.5, level 2 to the "OACs", and level 3 to the "Structural Bootstrapping". There are modules that conceptually span multiple levels which for simplicity have been added to one level only.

Deliverable	Module description	Position in the architecture, figure 2.5
D2.1.1	Scene and graph tracking (Oculus system)	level 1
D2.1.1	View-point invariant features (ECV system)	level 1
D2.1.1, section 2.3	Visual modeling of objects, pose estimation, part based models, etc.	level 1
D2.1.1, chapter 3	Visuo-haptic object representation (including segmentation)	level 1
D2.1.1	Inverse kinematics	level 1
D2.1.1	Dynamics and force/impedance control	level 1
D2.3.1	OAC learning (from pushing)	level 2
D2.3.1	Probabilistic manipulation functions (using ECV system from D2.1.1	level 2
D2.3.1	Extended PKS (informed search)	level 3
D4.1.1	Movement primitives	level 1

Chapter 3

Benchmarking

3.1 General considerations

The definition of benchmarks in Xperience responds to the task 1.2.1 (WP1.2) of the Annex I. In particular we will define benchmarks to test the following five aspects of the cognitive architecture:

B1 We will define benchmarks to test the ability of the agent to acquire and learn knowledge from stimuli;

B2 We will define benchmarks which demonstrate how efficiently structural bootstrapping works at different levels. Generative model building will be assessed in the sensorimotor, the planning and the communication/interaction domain;

B3 We will define a benchmark by which the interplay between stimulus driven and internal, generative processes can be assessed. As this is a dynamic process this benchmark will have to take the shape of a procedural assessment;

B4 We will define benchmarks for interaction and communication. These will rely on scenario 2 in WP5 and involve the comparison of human-human interaction with respect to the case where one person is replaced by the robot;

B5 The complete systems aspect is the central overarching research goal of Xperience. Success will be measured and benchmarked by reporting on how the individual components work together and how they will be able to fulfill the tasks in scenarios 1 and 2 (see WP5).

In short the five definitions of benchmark represent the five components of the Xperience cognitive model as described earlier in 2.2. In short, B1 is meant to test the outside-in component of the architecture, B2 similarly tests the inside-out component, B3 is designed specifically to test and validate the interplay between B1 and B2, B4 the interaction and communication with other agents and finally, B5 is designed to test the complete system.

3.2 Benchmarks for learning

Xperience project focuses on the performance of the learning process. In this context we need to benchmark by how much and how quickly the task knowledge and robot skills improve as the amount of data available to the learning agent increases. Since interactive tasks normally involve other agents, learning of such tasks often take place in collaboration with these agents. In Xperience this will normally mean that a robot learns from a human through imitation or its extension coaching. While it is possible that a robot guides the learning process of another robot, such learning is only possible with already highly developed cognitive agents. It is therefore uncertain if this type of learning can be achieved already within the Xperience project. Finally, especially in the case of tightly coupled interaction, the acquired motor knowledge needs to be suitably adapted to take into account the performance of the collaborating agent. Methods such as reinforcement learning can be used for this purpose. In the following we focus on key performance indicators (KPIs) related to the learning of tightly and loosely coupled interactive tasks in Xperience project.

3.2.1 Key performance indicators

In imitation learning, the learning agent observes how another agent performs the task and uses the acquired information to execute the task by itself. In this context we define the following KPIs:

1 Quality of learning: How good is the acquired task knowledge? Note that this indicator is always task-specific and cannot be defined in general;

2 Efficiency of learning: How many training examples are needed to achieve satisfactory performance?

3 Stability of learning: Standard deviation of the number of training examples for learning of a specific task.

In Xperience we study the following types of coaching 1) kinesthetic guiding, where the teacher physically guides the robot arm to execute the task, 2) imitation with the teacher in the loop, where the teacher actively observes the robot performance and possibly adapts its own performance to improve the robot's performance, and 3) coaching that involves verbal communication, where the teacher specifies which elements of the task are important and guides the adaptation of the acquired sensorimotor knowledge. The above three KPIs still apply to coaching scenarios, but the following additional KPIs specifically for coaching will be defined:

4 Quality of learning II: How much better is the task knowledge after coaching compared to standard imitation learning? Like before, this is a task-specific indicator that cannot be defined in general;

5 Efficiency of learning II: Reduction of the number of training examples needed to learn the task compared to standard imitation learning;

6 Amount of involvement of a teacher: How many times does the teacher need to intervene before the task is learned?

Interactive tasks cannot be learned by an agent alone, but need to be tested and refined during the actual execution of the task. We are going to use the following KPIs to benchmark this adaptation process:

7 Speed of refinement: How long does it take to refine the available motor knowledge to successfully execute the task in collaboration with another agent?

8 Efficiency of refinement: Number of trials before an interactive task can be successfully executed;

9 Speed of adaptation: How long does it take to successfully execute a task in collaboration with a different agent?

10 Efficiency of adaptation: Number of trials before motor knowledge is successfully transferred for collaboration with another agent.

Tasks often involve the manipulation of objects. It is therefore important to consider the quality of learning with respect to library of objects that can be manipulated by the robot. We can define the following KPIs, which are important for fast learning from experience:

11 Speed of acquisition: How many trials do we need to add information about new objects to the library of known objects?

12 Efficiency of generalization: How many specific objects belonging to a particular class do we need before we can apply the acquired knowledge to all objects of this class?

13 Efficiency of transfer: How many examples do we need before we can transfer the knowledge from one class of objects to another or from one task to another?

Item 13 is at the core of Xperience project.

3.2.2 Scenarios

Here we propose a list of scenarios that is not yet complete and will be extended and refined in the course of the project. For example, in the realm of the affordance discovery and learning we consider several types of possible affordances:

- Grasping;
- Pushing;
- Fill-able;
- Stackable;
- Switchable;

- Cut-able;
- Open-able.

with the consideration that some of them are reusable to experience other affordances, e.g. in the case of the "stackable", the agent needs to be able to "grasp" an object in order to experience the possibility of stacking it on top of another.

We then further characterize the scenarios into tightly coupled physical interaction scenarios, as for example:

- Collaborative pushing: learn how to push an object in collaboration with another agent;
- Collaborative carrying: learn how to carry an object in collaboration with another agent.

and, loosely coupled interaction, such as:

- Bi-manual learning about objects in collaboration with a human teacher: learn object identities (for recognition), affordances, and categories by applying different actions (pushing, relocating, etc.) to them. Here the human teacher can help the robot when it is stuck due to its physical limitations or due to inability to select a proper combination of actions.

3.3 Benchmarks

We consider the five set of requirements (B1..B5) in turn starting from B1 while considering the limits of the scenarios defined above. We also constrain this discussion to the benchmark of the first year of the project, i.e. effectively B1 and B2. We briefly cover some of the other definitions (B3..B5) although they will be finalized in D1.2.1 (M31).

In B1 we suggest benchmarks to evaluate and compare different aspects of sensorimotor exploration-based learning. Some of the benchmarks come in multiple versions as for example as a set of real world experiments and as problems being solved by a simulator. The advantage of the latter is that (1) the benchmark can be used by laboratories not having extensive robot equipment and (2) that the results are independent of the actual embodiment. However, for this a suitable simulation is required which is sufficiently realistic to produce meaningful results. This is possible for actions such as grasping, pushing and inserting (see [PKJ+11]) but remains still unrealistic for others like cutting or sawing. It also remains completely unrealistic to simulate aspects of cognitive systems such as interaction with peers and communication.

In the following list, we indicate with letter "a" the benchmarks designed to work on real world data in real time and "b" the simulated benchmarks. We further distinguish multiple benchmarks of the same type by a numeral (e.g. B2b-1). We have:

B1a Learning to grasp unknown objects. We will provide a set of rigid objects of a variety of shapes based on the KIT object data base [KIT]. The first task is to

produce grasps from stereo images without using analytic information about the 3D shape of new objects. Success is measured by the overall performance and by the incremental increase of performance through learning on the successes and failures on the objects which have been explored. Later in Xperience we will show how structural bootstrapping will help to discover a grammar of feature relation/grasp associations which can then be used to hypothesize grasps for novel objects. Grasp can be substituted in other experiments by various other forms of actions such as pushing.

B1b Benchmark B1a will (similarly to [PKJ+11]) come with a simulation environment which allows testing methods independently of the robot hardware (see also the Simox package described later).

B2a Affordance learning (sensorimotor). In correspondence to WP5.2 we will learn five categories of objects with different affordances (for example, fill-able, stackable, switchable, cut-able, open-able). We will measure how successful the associated actions are performed before and after learning on five real objects on which these affordances are meaningful. For each object we will define three contexts in which these operations need to be slightly adapted to (e.g., "filling with small and slightly larger items", "stacking while being filled or empty", etc.). We will measure how well the affordance applies in a new context and how fast learning improves performance. At least three categories will be also benchmarked with a simulator.

B2b-1 Semantics of high level symbolic actions. Learning the semantics of high level symbolic actions is one of the critical subproblems necessary for structural bootstrapping. In previous work on the PACO+ project, first research results on this problem were achieved at UEDIN based on data produced using the SDU robot. In the Xperience project, one of our objectives will be to perform similar learning research in the far more complex kitchen demonstration domain. To this end, one of our benchmarks will be to define a corpus of real (and possibly simulated) robot action executions (including action failures) performed by the Xperience robots within for example the KIT kitchen domain.

We anticipate the actions contained in this dataset will include all of those actions needed for both of the large scale demonstrations of the Xperience project. This corpus may also contain actions that while not directly related to the demonstrations are easily available on the robot platform. On the basis of this data set we will demonstrate the learning of state transition functions similar to those normally encoded in STRIPS type action operators [MPS10, MPS09].

In our previous work, a ten-fold cross-validation procedure was used to test the performance of the learning model, and was repeated across different numbers of training examples to assess how many examples would be needed to learn an adequate model. The performance was measured by considering the fluents which the model predicted would change versus the fluents which did change, and calculating the balanced F-measure, the harmonic mean of precision and recall (true positives/predicted changes and true positives/actual changes, respectively). We anticipate using a similar set of metrics to demonstrate this benchmark.

B2b-2 Testing mirroring systems for action and recognition. One of the tasks of Xperience is the development of a system for planning and plan recognition called ELEXIR. This system has two significant research contributions relevant to Xperience. First,

it defines a formal separation between the semantics of actions and a syntax for action use. This distinction is critical in order to understand how structural bootstrapping can be accomplished in the domains of reasoning about actions. Thus, defining the formal underpinnings of and demonstrating such a system is a critical benchmark for Xperience.

ELEXIR's second research contribution for Xperience is that the same action representation is used for both planning and plan recognition (mirroring). This means that successfully learning knowledge about action from plan recognition (*structural bootstrapping through demonstration*) can then immediately be used to planning. Therefore, to demonstrate the effectiveness of the ELEXIR system requires demonstrating the system performing both tasks. While there no generally agreed upon test sets in the plan recognition community, the planning community has run the International Planning Competition (IPC) for over ten years and has made the problem domains available. Such domains provide a rich source of data for demonstrating planning. In addition the domains (in conjunction with a planner) can be used to generate test data for plan recognition systems as well. Therefore, as one of our benchmarks for the project we will demonstrate the ELEXIR system achieving state of the art performance on both planning and plan recognition on a selected subset of the IPC domains. The metric for evaluating this benchmark, will be to achieve state of the art performance on these tasks with respect to both speed and and accuracy.

B2b-3 Semantic parsers map sentences onto compositional formal representations of their meaning. The natural evaluation for a semantic parser is then to test whether it can return the correct meaning representations for new unseen sentences. This evaluation requires a corpus of natural language sentences labeled with formal representations of their meanings. In order to be meaningful these formal representations must be defined within a closed model of meaning. They must also be easily expressible in natural language.

Database queries to databases containing information about common concepts meet these requirements and are easy to collect from untrained users. For these reasons, semantic parsers are commonly evaluated on their ability to parse natural language database queries to formal representations of their meaning. The parsers are scored according to accuracy over a test set – the proportion of the test sentences for which they return the correct meaning representation. In light of these requirements, one of the benchmarks for the Xperience project will be to demonstrate state of the art performance on grammar induction on the standard datasets such as GeoQueries, Atis, etc.

The GeoQuery dataset [ZM96] contains English natural language queries to a database about US geography. This dataset was collected from a number of users and has been translated into Spanish, Turkish and Japanese by [WM06]. It has served for the last fifteen years as the benchmark dataset for the semantic parsing task e.g. [ZM96, KWM05, KM06, WM06, WM07, ZC05, ZC07, LNLZ08, CGCR10, LJK11]. As such, success in parsing GeoQuery is widely regarded as a reliable indicator of semantic parser quality.

The Atis dataset contains natural language dialog commands to a flight booking dataset. Unlike the GeoQuery dataset, the Atis dataset contains raw unedited utterances. These utterances contain much more syntactic variation and noise than

those in the GeoQuery dataset. For this reason, Atis provides a harder (and more realistic) test for semantic parsers. The Atis dataset seems poised to become a benchmark dataset for the next tranche of work on semantic parsing [ZC07].

For the benchmarks related to B3 to B5 we have:

B3a Affordance learning (transfer through structural bootstrapping at the sensorimotor level). For each category from Benchmark B2a, we will introduce at least two new ("unknown") objects that exhibit, by construction but without robot exploration, the same affordances as the existing ("known") objects within their respective categories. We will measure:

1. the extent to which each affordance (e.g., fill, stack, switch, cut, and open) can be transferred from known to unknown objects. Possible quantification can, for example, be achieved by counting the proportion of attempted manipulations that are successful,
2. how fast learning on these unknown objects leads to a refinement of the OACs (or sensory motor schemas) associated to the applicable affordances. Quantification can, for example, be achieved by comparing the proportion of successful manipulations as a function of the number of learning trials (a) starting from transferred OACs and (b) starting from scratch.

B3b Affordance learning (transfer through structural bootstrapping on sensory motor level and the language level): We allow the system to use the Internet or other large data-bases to find out about the affordances associated to new objects [TMKW11]. We also measure how much this additional information helps the system to apply the corresponding sensory motor schema to the new objects introduced in benchmark B3a.

B4a Testing the ELEXIR on robotic systems. Having demonstrated the ability of the ELEXIR system to perform both planning and plan recognition in the IPC domains (see B2b-2), this benchmark will demonstrate the same capabilities within the robotic domains. ELEXIR will do this based on hand defined grammars for the motion primitives for one or more of the KIT, SDU, JSI, or IIT robotic platforms. As in B2b-2, the metrics we will use to measure this benchmark will be those generally accepted in the community of speed in plan construction and accuracy of plan recognition.

Chapter 4

Software poll

The results of the software module poll has ideally to coincide with the required modules of the architecture listed above at least up to the level of the learning of OACs. In the following we report the current status of the poll.

This is the pool of components that we will use (and initial reports are contained in D2.1.1, D2.3.1, D3.2.1, D4.1.1, D5.2.1, and D5.3.1.

General information	
Package name	Learning of dynamic movement primitives
Functionality	The package implements discrete and periodic dynamic movement primitives and the built-in mechanisms to react to external signals, one-shot learning of dynamic movement primitives from a single demonstration, online learning of periodic movements, and learning from multiple demonstrations using statistical methods.
Licensing and authorship	
Authors	Andrej Gams, Denis Forte, Ales Ude
Copyright holder	Jozef Stefan Institute
License	Not yet officially released
Programming information	
Dependencies	Depends on Gaussian Processes for Machine Learning package gpml (`http://www.gaussianprocess.org/gpml/code/matlab/doc/index.html`)
User interface mechanism	Input through Matlab interface
Distribution	Sources in a tarball
Compiler	Matlab
Build system	Not applicable
OS on which the module was tested	Mac/Windows/Linux
Files	
Format of input data files	Text files and communication with a robot via UDP
Format of output data files	Text files and communication with a robot via UDP
Format of configuration files	Not applicable
Other dependencies	The software can be run independently of the robot. Robot interface is provided via UDP interface, but the user must write the appropriate software for the robot side of communication.

General information	
Package name	Oculus
Functionality	Real time vision Software for video segmentation and segment tracking, providing also scene graphs. It integrates simple cameras, stereo cameras, Kinect device and video files as inputs. Allows embedding of own software modules.
Licensing and authorship	
Authors	Jeremie Papon, Alexey Abramov, Eren Aksoy, Florentin Wörgötter
Copyright holder	The authors, see above
License	Currently there is no license, but we are working on it. The package is "free to be used by project partners". Most probably the source code will be released under one of open source licenses very soon.
Programming information	
Dependencies	Installation process and the list of all required libraries can be found under `https://oculus.unfuddle.com/`.
User interface mechanism	It is a GUI application requiring user's input for the system configuration. Modules can also write to other "devices" for daisy-chaining the Oculus output with any other application.
Distribution	`https://oculus.unfuddle.com/`
Compiler	g++
Build system	See `https://oculus.unfuddle.com/`
OS on which the module was tested	Linux
Files	
Format of input data files	Camera streams or raw image data
Format of output data files	Image data (PNG)
Format of configuration files	XML
Other dependencies	Specific hardware requirements exist. The system needs camera input and for processing GPUs are required, please see documentation attached.

General information	
Package name	PKS - Planning with Knowledge and Sensing, pemPKS - Plan execution monitor for PKS
Functionality	PKS is a goal-directed symbolic planner that produces action plans for controlling the robot's high-level behaviour. pemPKS is a plan execution monitor controls the invocation of the planner and determines whether the next action in a plan should be executed or whether the plan should be reconstructed as a result of changes in the world state. The goal on XPERIENCE is to replace the existing plan execution monitor with a general module that allows other planning backends (not just PKS) and to experiment with planners other than PKS.

Licensing and authorship	
Authors	Ron Petrick
Copyright holder	Ron Petrick
License	The plan execution monitor is open source (no formal licence yet), however, the currently distributed planner backend is currently closed source (binary library). The plan for XPERIENCE is to release an open source version of the planner that works with the plan execution monitor.

Programming information	
Dependencies	Standard C/C++ libraries for building the planner and the basic plan execution monitor. ICE to build the ICE interface for the plan execution monitor.
User interface mechanism	Console / ICE interface.
Distribution	Tarball with binaries and source files.
Compiler	gcc
Build system	cmake
OS on which the module was tested	Linux

Files	
Format of input data files	XML / plain text / ICE interface
Format of output data files	XML / plain text / ICE interface
Format of configuration files	plain text / ICE interface
Other dependencies	None.

General information	
Package name	Nuklei
Functionality	Nuklei is a C++ library that implements kernel functions and kernel density estimation for $SE(3)$ data. Nuklei also provides tools for manipulating $SE(3)$ transformations, and for manipulating point clouds. Nuklei includes implementations of efficient algorithms for, among others: • Evaluating and simulating Gaussian distributions and von Mises-Fisher distributions; • Evaluating and simulating arbitrary $SE(3)$ distributions with kernel density estimation; • Kernel logistic regression; • Reading, writing, transforming clouds of $SE(3)$ points. Potential applications of $SE(3)$ density models: robot position and orientation (navigation), object pose (pose estimation), gripper pose (grasping).

Licensing and authorship	
Authors	Renaud Detry
Copyright holder	Renaud Detry
License	GNU GPL v3

Programming information	
Dependencies	Required nonstandard libraries: • Boost >= 1.38 • GSL >= 1.8 • BLAS and LAPACK Optional nonstandard libraries: • OpenCV >= 1 • CGAL >= 3.3
User interface mechanism	Nuklei is a library. It comes with a few example console applications that require data input.
Distribution	Git (http://nuklei.sourceforge.net/)

Compiler	GCC 4 (also LLVM/clang 2.8+, with some caveats)
Build system	SCons
OS on which the module was tested	Linux, MacOS X

Files	
Format of input data files	custom XML, plain ASCII
Format of output data files	custom XML, plain ASCII
Format of configuration files	none (AFAIK)
Other dependencies	Nuklei is currently not thread safe.

General information	
Package name	Cognitive Vision Software – CoViS (http://www.covig.org/)
Functionality	The package provides a core library and a set of applications. Core functionality is the extraction of a generic scene representation. This representation is modelled as a functional abstraction of the human visual system. Further functionality includes the use of this representation for for pose estimation, grasp generation etc.

Licensing and authorship	
Authors	Marcus Ackermann, Leon Bodenhagen, Emre Baseski, Lars B. Christensen, Mogens Christiansen, Christian Gebken, Dibyendusekhar Goswami, Oliver Granert, Daniel Grest, Marco Hahn, Jesper Juul Henriksen, Thomas Jaeger, Lars B. W. Jensen, Ushanthan Jeyabalan, Jeppe Barsoe Jessen, Sinan Kalkan, Thomas Kiesche, Dirk Kraft, Anders Kjaer-Nielsen, Norbert Krueger (co-ordinator), Mads Thorsted Nielsen, Florian Pilz, Mila Popovic, Martin Poerksen, Nicolas Pugeault, Torge Rabsch, Volker Roelke, Bodo Rosenhahn, Kasper Broegaard Simonsen, Daniel Wendorff, Brian Wettegren, Jan Woetzel, Shi Yan
Copyright holder	University of Southern Denmark
License	BSD

Programming information	
Dependencies	openCV, openGL, libxml, cairo, qt4, boost, tclap, blas, lapack, glut, nurbs++*, gl2ps*, libraw1394*, libdc1394*, xmlrpc*, CUDA* dependencies marked with an asterix are optional (some functionality will be missing)
User interface mechanism	Most components are console applications, our visualization component is a GUI application
Distribution	tarball (+svn acces for selected users)
Compiler	gcc
Build system	cmake
OS on which the module was tested	Linux

Files

Format of input data files	Depends on the specific application. E.g., images, custom xml files.
Format of output data files	Depends on the specific application. E.g. custom xml files, ascii.
Format of configuration files	Custom xmlfiles.
Other dependencies	The system is able to directly talk to firewire cameras.

General information	
Package name	EarlyVision
Functionality	Build on the Integrating Vision Toolkit (IVT), the EarlyVision library implements methods for active visual perception. These methods support the execution of head eye movements as well as the continuous perception based on the generated visual input. The provided methods include: • kinematic calibration of active camera systems; • execution of saccadic eye movement and smooth pursuit; • visual attention mechanisms; • continuous accumulation of visual information; • visual short term memory.

Licensing and authorship	
Authors	Kai Welke
Copyright holder	Kai Welke, Karlsruhe Institute of Technology (KIT)
License	GNU General Public License v3

Programming information	
Dependencies	IVT, Qt3, liblevmar
User interface mechanism	User interface via IVT (Qt3)
Distribution	Source code downloadable at SourceForge.net as tarball (starting from September 2011): earlyvision.sourceforge.net
Compiler	gcc 4.1+
Build system	Makefiles
OS on which the module was tested	Linux

Files	
Format of input data files	
Format of output data files	
Format of configuration files	
Other dependencies	Active camera system

General information	
Package name	VisionX
Functionality	Framework for the integration of vision processing components and camera capturers. Low-level communication based on ICE and shared memory, high level communication based on ICE only.

Licensing and authorship	
Authors	Kai Welke, David Gonzalez, Jan Isaac
Copyright holder	Kai Welke, David Gonzalez, Jan Issac, Karlsruhe Institute of Technology (KIT)
License	GNU General Public License v3

Programming information	
Dependencies	ICE, IVT
User interface mechanism	User interface via IVT (Qt3)
Distribution	SVN repository at KIT
Compiler	gcc 4.1+
Build system	CMake
OS on which the module was tested	Linux

Files	
Format of input data files	
Format of output data files	
Format of configuration files	
Other dependencies	

General information	
Package name	Integrating Vision Toolkit (IVT)
Functionality	The Integrating Vision Toolkit (IVT) is a stand-alone easy-to-use, platform- independent open source C++ computer vision library with an object-oriented architecture. It offers a clean camera interface and a general camera model, as well as many fast implementations of image processing routines and mathematic data structures and functions. The IVT offers its own multiplatform GUI toolkit.

Licensing and authorship	
Authors	Pedram Azad
Copyright holder	Pedram Azad, Karlsruhe Institute of Technology (KIT)
License	Modified BSD license (3-clause BSD)

Programming information	
Dependencies	For IEEE1394 camera support if required: libdc1394v2
User interface mechanism	Examples, tools with GUI using either Qt3, Qt4 or internal GUI components (Linux).
Distribution	Source code downloadable at SourceForge.net as tarball: ivt.sourforge.net
Compiler	Linux, MacOSX: gcc 4.1+, Windows: Microsoft Visual Studio
Build system	Makefiles or Visual Studio porjects
OS on which the module was tested	Linux, Windows, MacOSX

Files	
Format of input data files	bitmap (bmp)
Format of output data files	bitmap (bmp)
Format of configuration files	OpenCV camera calibration file format
Other dependencies	Active camera system

General information	
Package name	OpenMMM
Functionality	The library OpenMMM handles a variety of types of motions and (biomechanical) models. This library is an implementation of the conceptual framework MMM as proposed in various publications. The essential part of the MMM framework is based on a three-dimensional whole-body, kinematic model enriched with proper body segment properties (BSP), such as mass distribution, segment length, moment of inertia, etc., in order to compute gross body dynamics.

Licensing and authorship	
Authors	Stefan Gaertner, Martin Do
Copyright holder	Stefan Gaertner, Martin Do, Karlsruhe Institute of Technology (KIT)
License	GNU General Public License v3

Programming information	
Dependencies	GSL 1.12, Xerces-C 2.8, GAUL (http://gaul.souceforge.net/), kmlocal 1.7.3 (http://www.cs.umd.edu/ mount/), SNOPT 7.0, (http://www.scicomp.ucsd.edu/ peg/), Qt4, SoQT 1.4, Coin3D 2.5
User interface mechanism	
Distribution	Source code downloadable at SourceForge.net as tarball
Compiler	gcc 4.1.3 or higher
Build system	Makefiles
OS on which the module was tested	Linux

Files	
Format of input data files	MMM motion files in xml, c3d motion files
Format of output data files	MMM motion files in xml
Format of configuration files	Model files in xml
Other dependencies	

General information	
Package name	SIMOX
Functionality	Simox is a leightweight platform indepedent C++ toolbox containing three libraries for 3D simulation of robot systems, sampling based motion planning and grasp planning. The Virtual Robot library is used to define complex robot systems which may cover multiple robots with many degrees of freedom. The robot structure and its visualisation can be easily defined via XML files and environments with obstacles and objects to manipulate are supported. The libraries Grasp Studio and Saba use these definitions for planning grasps or collision-free motions. State-of-the-art implementations of sampling-based motion planning algorithms (e.g. Rapidly-exploring Random Trees) are served by the Saba library which was designed for efficient planning in high-dimensional configuration spaces. The library Grasp Studio offers possibilities to compute grasp quality scores for generic end-effector definitions (e.g. a humanoid hand). The implemented 6D wrench-space computations, offer the possibility to easily (and quickly) measure the quality of an applied grasp to an object. Furthermore, the implemented planners are able to generate grasp maps for given objects automatically.

Licensing and authorship	
Authors	Nikolaus Vahrenkamp
Copyright holder	Nikolaus Vahrenkamp, Karlsruhe Institute of Technology (KIT)
License	GNU General Public License v3

Programming information	
Dependencies	Qt4, Coin3d
User interface mechanism	3D visualization via Coin3d and Qt
Distribution	Source code downloadable at SourceForge.net as tarball: simox.sourceforge.net
Compiler	gcc 4.1+, Microsoft Visual Studio
Build system	CMake
OS on which the module was tested	Linux, Windows 7, XP, MacOSX

Files

Format of input data files	XML based
Format of output data files	XML based
Format of configuration files	XML based
Other dependencies	

General information	
Package name	RobWork (RobWork,RobWorkStudio,RobWorkSim)

Functionality	RobWork is a collection of C++ libraries for simulation and control of robot systems. RobWork is used for research and education as well as for industrial robot applications. Features of the library include: • Math for robotics (transformations, Jacobians and such); • Extensible stateless design for handling multi-threading issues and shared state issues; • Kinematic modeling of various types of industrial manipulators; • Support for serial robots, parallel robots and tree-structured (e.g. hands or multiple serial robots). Beta support for Conveyor and mobile devices; • Path-planning and inverse kinematics algorithms; • Simulation of sensors (range, camera, tactile); • Task and trajectory generation and representations; • Tools for geometry manipulation and collision detection; • Tools for *opengl* visualization of geometry, textures, lighting and other data types; • Tools for grasp planning and analysis, including several measures for evaluating grasp quality; • *Lua* script interface; • Plug-in structure. Besides the core part RobWork has a number of add-ons including: • RobWorkStudio provides a plugin-able graphical user interface. Plug ins for path planning, inverse kinematics, scripting, virtual sensors, logging and other simple manipulations of a robotics scene is included; • RobWorkSim is a simulator suited for dynamic grasp simulation, with support for simulating various tactile sensors, cameras and range scanners. Plugins for dynamic simulation of large grasp databases are available as well as for simple rigid body simulation. The goal of RobWork is not to include or become a component/communication framework such as e.g. YARP or Orocos. In fact for large projects we (RobWork developers) use and also encourage others to use these frameworks in conjunction with RobWork.

Licensing and authorship	
Authors	RobWork is developed at the robotics department of the Maersk McKinney Moller Institute (MMMI) at the University of Southern Denmark. The focus of the department is on industrial robots and their applications. Main developers include: • Lars-Peter Ellekilde (MMMI) • Jimmy Alison Jorgensen (MMMI) Past developers: • Anders Lau Olsen (past MMMI) • Lars Jessen (past MMMI) Contributors: • Preben Holm (MMMI) • Anders Glent Buch (MMMI)
Copyright holder	The Robotics Group, The Maersk Mc-Kinney Moller Institute, Faculty of Engineering, University of Southern Denmark
License	(Apache License, Version 2.0) RobWork is distributed under the Apache License, Version 2.0. For convenience, a number of open-source libraries are distributed together with RobWork; the RobWork license does not apply to these libraries.

Programming information

Dependencies	The different packages of RobWork has different dependencies which is separated in three groups: Mandatory (M), Optional (O), Optional but Distributed with RobWork (OD) • RobWork - the core package with only two mandatory dependencies: Boost (M), Xerces (M), Yaobi (OD), Lua (OD), toLua (OD), PQP (OD), qhull (OD) • RobWorkStudio - the visualisation package adds one additional dependency on Qt (M) • RobWorkSim - the dynamic simulation package only adds additional optional dependencies such as ODE (O), Bullet (O) and Moby (O)
User interface mechanism	The core RobWork is a c++ toolbox, with several console applications as examples of use: • With RobWorkStudio a complete GUI is available which provide pluginable GUI functionality; • With RobWorkSim dynamic simulation can be performed in both console and GUI application (through RobWorkStudio). Plugins for generating large grasp databases as well as ordinary rigid body simulations are provided.
Distribution	The package is distributed as: binary installer (win and ubuntu/debian), binaries/source and svn.
Compiler	RobWork is multi-platform and any gcc compatible compiler will do. It is tested on: gcc (Linux), mingw-gcc (win), vc++
Build system	CMake and Visual Studio projects
OS on which the module was tested	Linux, Windows and earlier versions tested on Mac

Files	
Format of input data files	Most input files are xml based but several non-xml based image formats (PPM, PGM, RGB) with qt (png, jpeg tiff and several more) and 3d object/scene description formats are also supported (AC3D, 3DS, STL, IVG, OBJ). The main robot scene description is a custom format in xml, but an alpha version for Collada format is coming up.

Format of output data files	Most output file-formats are xml based. RobWork defines formats for loading/saving: paths, trajectories, property-maps, robotic scenes, states.
Format of configuration files	ini and xml
Other dependencies	

Bibliography

[Bad99] Alan D. Baddeley. *Essentials of Human Memory*. Psychology Press, Taylor and Francis, 1999.

[CA97] Fernando J. Corbacho and Michael A. Arbib. Schema-based learning: Towards a theory of organization for biologically-inspired autonomous agents. In *Agents*, pages 520–521, 1997.

[CGCR10] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL-2010)*, pages 18–27, Uppsala, Sweden, 2010.

[CKS93] Andy Clark and Annette Karmiloff-Smith. The cognizer's innards: A psychological and philosophical perspective on the development of thought. *Mind & Language*, 8(4):487–519, 1993.

[Cla01] A. Clark. *Mindware: an introduction to the philosophy of cognitive science*. Oxford University Press, Oxford, UK, 2001.

[GKKed] F. Guerin, N. Krüger, and D. Kraft. A survey of the ontogeny of tool use: from sensorimotor experience to planning. *IEEE TAMD*, submitted.

[Gle90] L. Gleitman. The structural source of verb meanings. *Language acquisition*, 1:3 – 55, 1990.

[GMP+06] C. Geib, K. Mourao, R. Petrick, M. Pugeault, M. Steedman, N. Krger, and F. Wörgötter. Object action complexes as an interface for planning and robot control. In *IEEE RAS International Conference on Humanoid Robots*, Genova, 2006.

[KGP+11] Norbert Krüger, Christopher Geib, Justus Piater, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, Alejandro Agostini, and Rüdiger Dillmann. Object-action complexes: Grounded abstractions of sensorimotor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011. doi:10.1016/j.robot.2011.05.009.

[KIT] KIT. The kit object database. http://i61p109.ira.uka.de/ObjectModelsWebUI.

[KM06] Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

[KWM05] Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*, 2005.

[LJK11] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, 2011.

[LNLZ08] Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. A generative model for parsing natural language to meaning representations. In *Proceedings of The Conference on Empirical Methods in Natural Language Processing*, 2008.

[MNN+10] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. von Hofsten, K. Rosander, J. Santos-Victor, A. Bernardino, and L. Montesano. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23:1125 – 1134, 2010.

[MPS09] Kira Mourão, Ronald P. A. Petrick, and Mark Steedman. Learning action effects in partially observable domains. In *Proceedings of the ICAPS 2009 Workshop on Planning and Learning*, pages 15–22, Thessaloniki, Greece, September 2009.

[MPS10] Kira Mourão, Ronald P. A. Petrick, and Mark Steedman. Learning action effects in partially observable domains. In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2010)*, pages 973–974, August 2010.

[Pia76] J. Piaget. *The psychology of intelligence*. 1976.

[PKJ+11] M. Popovic, G. Kootstra, J.A. Jorgensen, D. Kragic, and N. Krüger. Grasping unknown objects using an early cognitive vision system for general scene understanding. In *International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, 2011.

[SB02] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction, 2002 edition*. Bradford Books, MIT press, Cambridge, MA, 2002.

[TG07] J.C. Trueswell and L.R. Gleitman. *Oxford Handbook of Psycholinguistics*, chapter Learning to parse and its implications for language acquisition. Oxford University Press, Oxford, 2007.

[TMKW11] M. Tamosiunaite, I. Markelic, T. Kulvicius, and F. Wörgötter. Generalizing objects by analyzing language. In *IEEE/RAS International Conference on Humanoid Robots*, Bled, Slovenia, 2011.

[VvHF10] D. Vernon, C. von Hofsten, and L. Fadiga. *A Roadmap for Cognitive Development in Humanoid Robots*. Springer, 2010.

[WAK+09] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Borr. Cognitive agents a procedural perspective relying on the predictability of object-action-complexes (oacs). *Robotics and Autonomous Systems*, 57:420 – 432, 2009.

[WM06] Yuk Wah Wong and Raymond Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*, 2006.

[WM07] Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Association for Computational Linguistics*, 2007.

[ZC05] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[ZC07] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proc. of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.

[ZM96] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, 1996.