**RockEU2**
**Robotics Coordination Action for Europe Two**

Grant Agreement Number: 688441

01.02.2016 – 31.01.2018

Instrument: Coordination and Support Action

# RockEU2 Cognitive Architecture Schema

David Vernon, University of Skövde, Sweden

Markus Vincze, Technische Universität Wien, Austria

Deliverable D3.3

| | |
|---|---|
| Lead contractor for this deliverable: | University of Skövde |
| Due date of deliverable: | January 31, 2018 |
| Actual submission date: | January 31, 2018 |
| Dissemination level: | Public |
| Revision: | 1.0 |

## History of Changes

| Name | Status | Version | Date | Summary of actions made |
|---|---|---|---|---|
| | | | | |

## Executive summary

As specified in the work plan, Deliverable D3.3 was to present a cognitive architecture *schema*, i.e. a cognitive architecture blueprint, setting out the component functionality, control flows, and mechanisms for specifying behaviour, from which a specific cognitive architecture could be derived, but specified at a level of abstraction that is independent of a specific application niche that the full architecture targets.

The specification of the schema was to be based on the catalogue of cognitive systems capabilities that were to be identified in deliverable D3.2.

However, arising from the change in strategy set out in Deliverables D3.2 and D3.4, Deliverable D3.3 now takes the form of reflections on architectural requirements based on Deliverables D3.1, D3.2, and D3.4, including lessons learned from experience with CRAM when preparing deliverable D3.2 and afterwards.

This change in strategy was motivated by a recognition in D3.2 that designing a cognitive system is not simply a question of assembling off-the-shelf algorithms but of designing the entire system, as a whole, in the context of the target application domain and the run-time implementation infrastructure. This resulted in a focus in D3.2 on one of the very few open source cognitive architectures – CRAM (Cognitive Robot Abstract Machine) [1-6] – that has the demonstrated potential to be deployed as a complete cognitive robotic framework.

The change was also motivated by the related realization that that it makes more sense to base the design of a practical cognitive architecture for robotics on specific use-cases (so-called *design by use-case*) rather than on a long list of desirable features (so-called *design by desiderata*) [8].

We concluded that a cognitive architecture schema is not as useful for cognitive robotics, at the current level of maturity of the discipline, as it seemed to be when the work plan was originally written, hence the new revised focus of this deliverable on offering something that will be of practical value in advancing the deployment of cognitive robotics in industry.

Despite this change in approach, Deliverable D3.3 still forms an integral part of Work Package 3, as explained in Appendix I. It draws on D3.1 Industrial Priorities for Cognitive Robotics and D3.4 Cognition-Autonomy Framework through the list of cognitive functional requirements derived from a meta use-case that encapsulates the industrial requirements and are specified using the terminology of autonomous systems. It draws on Deliverable D3.2 by using the CRAM case study to provide insights on the practical issues of specifying, designing, and implementing a practical robotics cognitive system. These insights and the related architectural requirements are taken up in Deliverable 3.5 on the implications for software engineering in cognitive robotics.

## Content

# 1. Introduction

The goal of Task 3.3 was (i) to create a cognitive architecture *schema* that can be used both to provide a context for constituent cognitive system algorithms and the functionality they encapsulate and (ii) to identify those aspects that require the greatest attention in the short-term to develop effective cognitive robots.

According to the work plan, A *cognitive architecture schema* is not a cognitive architecture – it is a blueprint for the design of a cognitive architecture, setting out the component functionality, control flows, and mechanisms for specifying behaviour. It describes a cognitive architecture at a level of abstraction that is independent of the specific application niche that the architecture targets. Its purpose is to define the necessary and sufficient components and the organization required to create a complete cognitive robot system.

A cognitive architecture schema was viewed as an effective way of organizing the information in Deliverable D3.2. In planning this, we anticipated that we would be addressing difficult questions. For example: is it feasible to design a general reference schema, what is required to develop a re-configurable cognitive architecture, what are the software engineering implications, and can any existing models be used as a starting point?

The one question we didn't ask when writing the task description was: does a cognitive architecture schema make sense, given the embryonic state of development of the discipline. Two and a half years on, it is now evident that it doesn't. This deliverable explains why this is so. Nevertheless, the deliverable still fulfils its original intention to consolidate what was learned from Deliverables D3.1, D3.2, and D3.4, present something useful to aid the deployment of cognitive systems in industry and set the stage for Deliverable D3.5 on the implications for software engineering with cognitive robotics.

# 2. The story so far

Let us recap on the situation at this point. Work package 3 has five deliverables:

D3.1 Industrial Priorities for Cognitive Robotics (M12).
D3.2 Catalogue of Cognitive Systems Capabilities (M18).
D3.3 RockEU2 Cognitive Architecture Schema (M24).
D3.4 Cognition-Autonomy Framework (M18).
D3.5 Software Engineering Factors in Cognitive Robotics (M24).

Deliverable D3.1 identified on eleven industrial priorities for cognitive robotics (see Appendix II).

1. Safe, reliable, transparent operation.
2. High-level instruction and context-aware task execution.
3. Knowledge acquisition and generalization.
4. Adaptive planning.
5. Personalized interaction.
6. Self-assessment.
7. Learning from demonstration.
8. Evaluating the safety of actions.
9. Development and self-optimization.
10. Knowledge transfer.
11. Communicating intentions and collaborative action.

Deliverable D3.4 provided a list of thirteen cognitive functional requirements derived from a meta use-case that encapsulates the industrial requirements, specified using the terminology of autonomous systems, reflecting all eleven industrial priorities, and cross-checked against the results of previous similar exercises [9].

1. Perception.
2. Declarative knowledge & memory.
3. Procedural knowledge & memory.
4. Planner & plan executive.
5. Reasoning & inference mechanisms.
6. Meta-cognition.

7. Environment model.
8. Internal simulator.
9. Goal representations.
10. Declarative learning.
11. Procedural learning.
12. Attention mechanism.
13. Real-time action controller.

Deliverable D3.2 was to provide a definitive catalogue of algorithms and data structures for the implementation of cognitive systems: a set of essential algorithms and software implementations from which cognitive robots can be built. However, it eventually took a different form to the one originally intended. Instead of a catalogue of independent software algorithms, we focused on one of the very few open source cognitive architectures that has the demonstrated potential to be deployed as a complete cognitive robotic framework: CRAM (Cognitive Robot Abstract Machine) [1-6].

This change in strategy was motivated by a recognition in D3.2 that designing a cognitive system is not simply a question of assembling off-the-shelf algorithms but of designing the entire system, as a whole, in the context of the target application domain and the run-time implementation infrastructure. In other words, contemporary cognitive robots are constructed as custom systems that are tightly linked to their underlying development framework and middleware. Even assuming a common middleware such as ROS or YARP, there still remains the need to match knowledge ontologies, knowledge representations, reasoning, inference, and planning mechanisms.

This deliverable, Deliverable D3.3, was to bring all this together by encapsulating the list of algorithms and data structures in a cognitive architecture schema. However, arising from the change in strategy, Deliverable D3.3 now takes the form of reflections on architectural requirements based on Deliverables D3.1, D3.2, and D3.4, including lessons learned from experience with CRAM when preparing Deliverable D3.2 and afterwards.[1]

There is a second reason for this change in the nature of D3.3. This was the related realization that that it makes more sense to base the design of a practical cognitive architecture for robotics on specific use-cases (so-called *design by use-case*) rather than on a long list of desirable features (so-called *design by desiderata*) [8]. This is significant because cognitive architecture schemas are generally created from such a list. Furthermore, the process of designing a specific cognitive architecture from a cognitive architecture schema is a major undertaking because many design decisions have to be made that derive from the application requirements and the development and run-time environment. This undermines the cognitive architecture schema exercise, at least in the context of building practical cognitive robot systems.

We concluded that a cognitive architecture schema is not as useful for cognitive robotics, at the current level of maturity of the discipline, as it seemed to be when the work plan was originally written, and hence the new revised focus of this deliverable on offering something that will be of practical value in advancing the deployment of cognitive robotics in industry.

Let us now proceed to look at what we can learn from the exercise, first addressing insights from the realization of CRAM derived from Deliverable D3.2 and subsequent experience with CRAM, and second addressing the role of cognitive architecture schemas in the big picture of cognitive architectures.

1 We don't suggest all cognitive robots should be based on CRAM but it does prove an excellent exemplar of a maturing system of interoperable AI tools and techniques that can be freely used by the community, especially as a vehicle for industrial deployment.

## 3. Lessons learned from Deliverable D3.2

Useful cognitive architectures are *system* architectures: they encapsulate design choices that are constrained by their implementation infrastructure and application domain. When rendered in a much more abstract form as a schema, too much of the important low-level detail that makes cognitive architecture feasible is lost. These details matter. For example, there are many inter-dependencies between the components. You can try to abstract them by adopting a powerful middleware for robots, e.g. ROS or YARP, both of which are instances of the component-based software engineering paradigm, e.g. just as CRAM has done in adopting ROS, but the complexity remains and the gap in abstraction between a cognitive architecture schema and a real cognitive system architecture, specified at a level of abstraction that is suitable for implementation, is too great to be bridged without a great deal of difficulty.

This becomes alarmingly evident when you consider that even cognitive architecture are themselves already specified at a very high level of abstraction and there is a significant gap between them and a system architecture that has necessary and sufficient level of detail to facilitate implementation.

Consider either of the versions of the CRAM cognitive architecture we presented in Deliverable D3.2 in Figures 1 and 2 and the schematic in Figure 3 showing an intentionally-simplified depiction of the CRAM executive. Even the information required to fully specify the CRAM executive was deemed to be too complex to be usefully included in the on-line CRAM documentation. It seems then that there is little to be gained by compounding the problem and adding an additional schema-level layer of abstraction on top of the cognitive architecture and mapping functional capabilities to that level. *It doesn't offer any value when it comes to the extremely difficult task of realizing the architecture and building an operational cognitive system.*

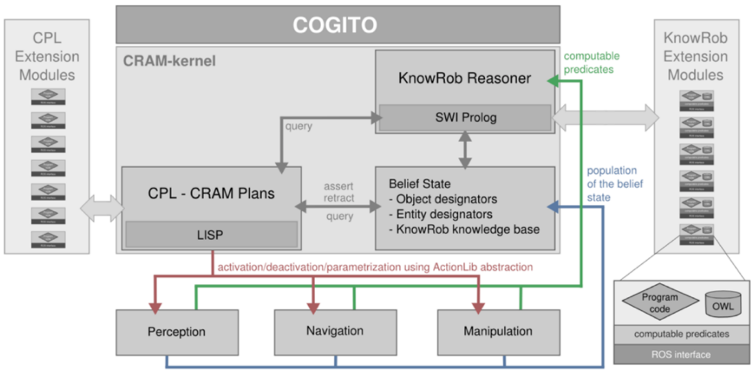

Figure 1: The Cognitive Robot Abstract Machine - CRAM - cognitive architecture as depicted in [1].

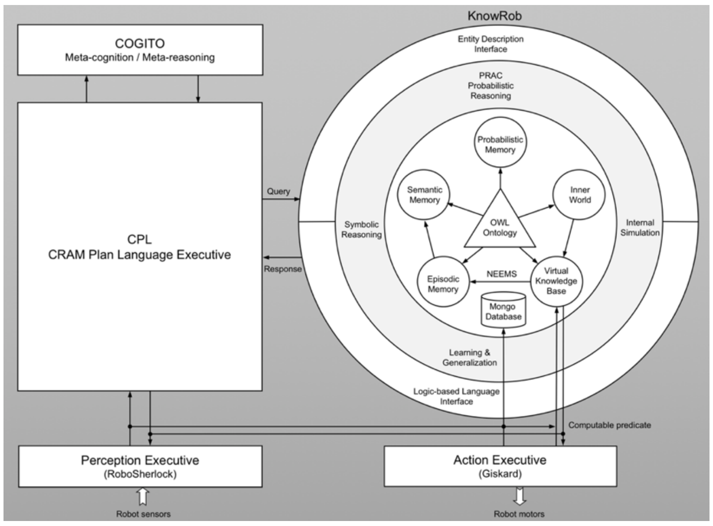

Figure 2: The Cognitive Robot Abstract Machine - CRAM - cognitive architecture as drawn by the authors of this deliverable based on personal communications with the CRAM designers and developers. This figure has not appeared in published documents.

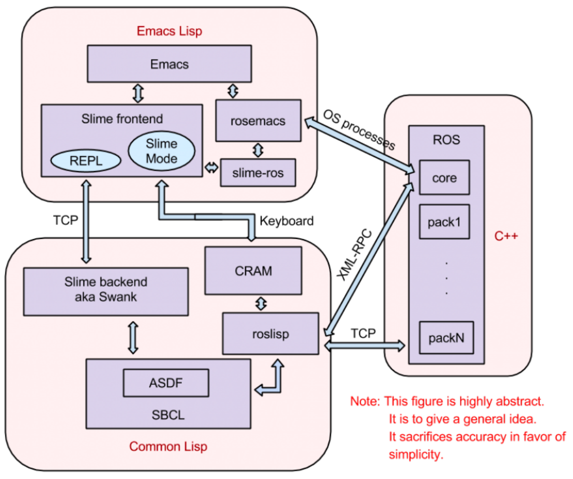

Figure 3: The main software components involved in the CRAM development process from the programming languages point of view. The figure has many inaccuracies (e.g. CRAM is not entirely written in Common Lisp, or ASDF is not actually a component of SBCL, it is, in fact, developed independently from SBCL but is automatically included when installing the latter, etc). On the other hand, it tries to keep things simple. [taken from CRAM setup and IDE instructions at http://cram-system.org/doc/ide].

The D3.2 exercise overlooked the simple truth that cognitive systems are software *systems* and the behaviour required of cognitive robots depends on the coherent operation of a network of non-trivial components. This coherence entails that all components are mutually compatible and share implemented knowledge representation and reasoning formalisms. Consequently, designing a cognitive system is not simply a question of assembling off-the-shelf algorithms: the entire system is *designed as a whole* and the design choices depend on the application context and practical considerations of available technical resources.

Arguably, the best you can do in the current circumstances where cognitive robotics is still an embryonic discipline is to follow some pragmatic guidelines. We present here a short list based on our experience in Work Package 3 and other projects.[2] First, let us remark that there are three different paradigms of cognitive science on which cognitive architectures are based: cognitivist, emergent, and hybrid. There are many differences between them but, broadly speaking, cognitivist systems are based on classical symbolic models of cognition, emergent on sub-symbolic developmental models, and hybrid models try to combine the best of both, normally complementing symbolic information processing with sub-symbolic associative networks (e.g. artificial neural network).

Here are some guidelines.

1. Choose either cognitivist or hybrid approaches. While some researchers[3] believe that the emergent paradigm is based on principles that are more consistent with natural cognition, the cognitivist and hybrid paradigms are far more mature technologically and are more suitable for deployment in industry today.

2. Adopt a suitable software engineering paradigm, e.g. component-based software engineering (CBSE); see Appendix III.

3. Because of the inevitably large size of a cognitive system which will probably have to be constructed as a multi-component system, recognize that you need to adopt one or other of a suitable publish-and-subscribe component-port-connector middleware that provides sufficient flexibility for inter-component (i.e. inter-node in ROS and inter-module in YARP) message communication between components using topics (ROS) or ports (YARP).

[2] For example, DREAM: Development of robot-enhanced therapy for children with autism spectrum disorder (ASD); www.dream2020.eu

[3] Including at least one of the authors of this deliverable.

4. Identify the key focus of your application domain, i.e. what problem is your cognitive system meant to solve, and build from there. Use these to create use-cases.

5. Identify your priority capabilities, e.g. by selecting from one of eleven industrial priorities in Deliverable D3.1, and focus on building a cognitive system with just one or two capabilities at the outset. For example, Predictive Sequence Learning (PSL) [36-38] focusses only on priority 7, Learning from Demonstration, in mobile robotics and DREAM [39] focusses on priorities 5 and 11 (personalized interaction, communicating intentions and collaborative action) in robot-enhanced therapy for children with autism spectrum disorder (ASD). CRAM [1-6] is more ambitious and focusses mainly on priorities 2, 3, 4, 6 and 10 (high-level instruction and task-aware execution, knowledge acquisition and generalization, adaptive planning, self-assessment, and knowledge transfer) in targeting abstract specification of robot actions which are semantically-underdetermined, i.e. vaguely- or roughly-stated, among other things.

6. As we learned from our attempts to map from the industrial priorities for cognitive behaviours, as described in Deliverable D3.1, to a list of AI tools and techniques, the difference in the level of abstraction between them is too great. We were more successful in creating a list of cognitive functional requirements at a lower level of abstraction in Deliverable D3.4 by using the terminology of autonomous systems and (meta) use-case analysis, linked to the industrial priorities. Follow that example.

7. Decide what functionality you want based on use-cases, e.g. in a similar manner to that in Deliverable D3.4 with the meta use-case analysis. This will require you to map meta-capabilities – e.g., adapt, self-monitor, anticipate – onto specific systems that realize operational capabilities, e.g. plan formulation and execution.

8. Create your system architecture based on this functionality but factoring in the constraints imposed by the availability of appropriate techniques, tools, and appropriately skilled developers.

9. Try to partner with R&D people in research labs – industry or university – to help develop and realize the cognitive system, targeting especially labs that have experience in these techniques and tools.

10. If possible, adapt or extend an existing operational cognitive architecture; don't opt to build from scratch if a reasonable amount of your required functionality is available in an existing system. Even if you do opt for an existing cognitive architecture, double check to check that the claimed functionality is operational and to what extent is has been tested in real-word scenarios. Be prepared for a steep learning curve.

## 4. Two complementary ways to design a cognitive architecture[4]

In this section, we address the role that cognitive architecture schemas play in the big picture of cognitive architectures and we highlight that this role is inconsistent with the goals of Work Package 3 (or, if it is not inconsistent, it is certainly not well aligned with them).

There are two conflicting agendas at play in the design of cognitive architectures [8]. One is principled: to create a model of cognition and gain an understanding of cognitive processes. The other is practical: to build useful systems that have a cognitive ability and thereby provide robust adaptive behaviour that can anticipate events and the need for action. The first is concerned with advancing cognitive science and psychology, the second is concerned with effective engineering. Success with one agenda may not necessarily lead, in the short term at least, to useful insights that lead to success with the other agenda.

These two views are patently different and they are not necessarily complementary or mutually-supportive. There is no guarantee that success in designing a practical cognitive architecture for an application-oriented cognitive robot will shed any light on the more general issues of cognitive science and neither is it evident that efforts in the design general cognitive architectures have been tremendously successful for practical applications.

[4] This section is an abstracted version of [8].

The origins of cognitive architectures have their roots in pioneering research in cognitive science by Allen Newell and his colleagues in their work on unified theories of cognition [10]. As such, a cognitive architecture represents any attempt to create a theory that addresses a broad range of cognitive issues, such as attention, memory, problem solving, decision making, and learning, covering these issues from several aspects including psychology, neuroscience, and computer science, among others. A cognitive architecture is, therefore, from this perspective at least, an over-arching theory (or model) of human cognition and, as such, focusses on generality and completeness (e.g. see [11]). We draw from many sources in shaping these architectures. They are often encapsulated in lists of desirable features (sometimes referred to as desiderata) or design principles [12-15]. This, then, is the first approach to designing a cognitive architecture (or a cognitive architecture schema): *design by desiderata* [8].

The alternative agenda focusses on the practical necessities of the cognitive architecture and designing on the basis of user requirements. This is referred to as *design by use case*. Here, the goal is to create an architecture that addresses the needs of an application without being concerned whether or not it is a faithful model of cognition. In this sense, it is effectively a conventional system architecture, rather than a cognitive architecture *per se*, but one where the system exhibits the required attributes and functionality, typically the ability to autonomously perceive, to anticipate the need for actions and the outcome of those actions, and to act, learn, and adapt. In this case, the design principles, or desiderata, do not drive the cognitive architecture — the requirements do that — but it helps to be aware of them so that you know what capabilities are potentially available and might be deployed to good effect. *Significantly, design by use case implies that it is not feasible to proceed by developing a cognitive architecture schema and then instantiating it as a specific cognitive architecture because routing the design through the meta-level schema tacitly abstracts away many of the particularities of the application that makes this approach useful.*

As we stated in the introduction, according to the work plan, a *cognitive architecture schema* describes a cognitive architecture at a level of abstraction that is independent of the specific application niche that the architecture targets. Consequently, the cognitive architecture schema is incompatible with the design by use-case approach advocated in [8].

Since we are intent here on building practical cognitive robotic systems and not on advancing cognitive science, we again conclude that the creation of a RockEU2 cognitive architecture schema is not a useful exercise.

## 5. Summary and Conclusions

As specified in the work plan, Deliverable D3.3 was to present a cognitive architecture schema, i.e. a cognitive architecture blueprint, setting out the component functionality, control flows, and mechanisms for specifying behavior, from which a specific cognitive architecture could be derived but specified at a level of abstraction that is independent of a specific application niche that the architecture targets.

The specification of the schema was to be based on the catalogue of cognitive systems capabilities that were to be identified in deliverable D3.2.

However, arising from the change in strategy set out in Deliverables D3.2 and D3.4, Deliverable D3.3 now takes the form of reflections on architectural requirements based on Deliverables D3.1, D3.2, and D3.4, including lessons learned from experience with CRAM when preparing deliverable D3.2 and afterwards.

This change in strategy was motivated by a recognition in D3.2 that designing a cognitive system is not simply a question of assembling off-the-shelf algorithms but of designing the entire system, as a whole, in the context of the target application domain and the run-time implementation infrastructure.

The change was also motivated by the related realization that that it makes more sense to base the design of a practical cognitive architecture for robotics on specific use-cases (so-called design by use-case) rather than on a long list of desirable features (so-called design by desiderata) [8].

Taken together, we conclude that a cognitive architecture schema is not as useful for cognitive robotics, at the current level of maturity of the discipline, as it seemed to be when the work plan was originally written, and hence the new revised focus of this deliverable with the goal of offering something that will be of practical value in advancing the deployment of cognitive robotics in industry.

## 6. References

[1] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In IEEE/RSJ In- ternational Conference on Intelligent Robots and Systems, pages 1012–1017, Taipei, Taiwan, October 2010.

[2] M. Beetz, D. Jain, L. Mösenlechner, and M. Tenorth. Towards Performing Everyday Manipulation Activities. Robotics and Autonomous Systems, 58(9):1085–1095, 2010.

[3] J. Winkler, G. Bartels, L. Mösenlechner, and M. Beetz. Knowledge enabled high-level task abstraction and execution. Advances in Cognitive Systems, 1:1–6, 2012.

[4] M. Tenorth and M. Beetz. KnowRob: A knowledge processing infrastructure for cognition- enabled robots. The International Journal of Robotics Research, 32(5):566—590, 2013.

[5] G. Bartels, M. Beetz, D. Beßler, M. Tenorth, and J. Winkler. How to use OpenEASE: An online knowledge processing system for robots and robotics researchers. In Bordini, Elkind, Weiss, and Yolum, editors, Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Istanbul, Turkey, May 4–8 2015.

[6] L. Kunze and M. Beetz. Envisioning the qualitative effects of robot manipulation actions using simulation-based projections. Artificial Intelligence, 247:352—380, 2017.

[7] D. Vernon and M. Vincze. Industrial priorities for cognitive robotics. In R. Chrisley, V. C. Müller, Y. Sandamirskaya, and M. Vincze, editors, Proceedings of EUCognition 2016, Cognitive Robot Architectures, volume CEUR-WS Vol-1855, pages 6–9, Vienna, December 2017. European Society for Cognitive Systems.

[8] D. Vernon. Two ways (not) to design a cognitive architecture. In R. Chrisley, V. C. Müller, Y. Sandamirskaya, and M. Vincze, editors, Proceedings of EUCognition 2016, Cognitive Robot Architectures, Volume CEUR-WS Vol-1855, pages 42–43, Vienna, December 2017. European Society for Cognitive Systems.

[9] J. L. Krichmar, "Design principles for biologically inspired cognitive architectures," *Biologically Inspired Cognitive Architectures*, vol. 1, pp. 73–81, 2012.

[10] A. Newell, *Unified Theories of Cognition*. Cambridge MA: Harvard University Press, 1990.

[11] J. E. Laird, C. Lebiere, and P. S. Rosenbloom, "A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics." *AI Magazine*, vol. In Press, 2017.

[12] R. Sun, "Desiderata for cognitive architectures," *Philosophical Psychology*, vol. 17, no. 3, pp. 341–373, 2004.

[13] J. L. Krichmar and G. M. Edelman, "Principles underlying the construction of brain-based devices," in *Proceedings of AISB '06 - Adaptation in Artificial and Biological Systems*, ser. Symposium on Grand Challenge 5: Architecture of Brain and Mind, T. Kovacs and J. A. R. Marshall, Eds., vol. 2. Bristol: University of Bristol, 2006, pp. 37–42.

[14] J. E. Laird, "Towards cognitive robotics," in *Proceedings of the SPIE — Unmanned Systems Technology XI*, G. R. Gerhart, D. W. Gage, and C. M. Shoemaker, Eds., vol. 7332, 2009, pp. 73 320Z–73 320Z–11.

[15] D. Vernon, C. von Hofsten, and L. Fadiga, "Desiderata for develop- mental cognitive architectures," *Biologically Inspired Cognitive Architectures*, vol. 18, pp. 116–127, 2016.

[16] M. Y. Jung, M. Balicki, A. Deguet, R. H. Taylor, and R. Kazanzides, "Lessons learned from development of component-based medical robot systems," *Journal of Software Engineering for Robotics*, vol. 5, no. 2, pp. 25–41, 2014.

[17] D. Alonso, C. Vincente-Chicote, F. Ortiz, J. Pastor, and B. Alvarez, "V3CMM: a 3-view component meta-model for model-driven robotic software development," *Journal of Software Engineering for Robotics*, vol. 1, no. 1, pp. 3–17, 2010.

[18] C. A. R. Hoare, "Communicating sequential processes," *Communica- tions of the ACM*, vol. 21, no. 8, pp. 666—677, 1978.

[19] H. Bruyninckx, M. Klotzbu¨cher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, “The BRICS component model: A model- based development paradigm for complex robotics software systems,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC ‘13. New York, NY, USA: ACM, 2013, pp. 1758–1764.

[20] H. Bruyninckx, “Robotics software framework harmonization by means of component composability benchmarks,” BRICS Deliverable D8.1, 2010, http://www.best-of-robotics.org.

[21] D. Brugali and P. Scandurra, “Component-Based Robotic Engineering (Part I),” *IEEE Robotics and Automation Magazine*, pp. 84–96, December 2009.

[22] D. Brugali and A. Shakhimardanov, “Component-Based Robotic Engineering (Part II),” *IEEE Robotics and Automation Magazine*, pp. 100– 112, March 2010.

[23] P. Soetens, H. Garcia, M. Klotzbu¨cher, and H. Bruyninckx, “First established CAE tool integration,” BRICS Deliverable D4.1, 2010, http://www.best-of-robotics.org.

[24] A. Shakhimardanov, J. Paulus, N. Hochgeschwender, M. Reckhaus, and G. K. Kraetzschmar, “Best practice assessment of software technolo- gies for robotics,” BRICS Deliverable D2.1, 2010, http://www.best-of-robotics.org.

[25] M. Radestock and S. Eisenbach, “Coordination in evolving systems,” in *Trends in Distributed Systems, CORBA and Beyond*. Springer, 1996, pp. 162—176.

[26] C.Schlegel,A.Steck,andA.Lotz,“Model-drivensoftwaredevelopment in robotics: Communication patterns as key for a robotics component model,” in *Introduction to Modern Robotics*. iConcept Press, 2011.

[27] D. Vanthienen, M. Klotzbücher, and H. Bruyninckx, “The 5C-based architectural Composition Pattern: lessons learned from re-developing the iTaSC framework for constraint-based robot programming,” *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 17–35, 2014.

[28] “http://www.omg.org.”

[29] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[30] G. Metta, P. Fitzpatrick, and L. Natale, “YARP: yet another robot platform,” *International Journal on Advanced Robotics Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[31] J. Baillie, “URBI: towards a universal robotic low-level programming language,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005*, 2005, pp. 3219—3224.

[32] P. Soetens, “A software framework for real-time and distributed robot and machine control,” Ph.D. dissertation, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium, 2006.

[33] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics. Springer, 2007, ch. Orca: a component model and repository, pp. 231—251.

[34] A. Makarenko, A. Brooks, and T. Kaupp, “On the benefits of making robotic software frameworks thin,” in *IEEE/RSJ International Confer- ence on Intelligent Robots and Systems, IROS 2007*, 2007.

[35] E. I. Barakova, J. C. C. Gillesen, B. E. B. M. Huskens, and T. Lourens, “End-user programming architecture facilitates the uptake of robots in social therapies,” *Robotics and Autonomous Systems*, vol. 61, pp. 704–713, 2013.

[36] E. A. Billing, T. Hellstr¨om, and L.-E. Janlert. Predictive learning from demonstration. In J. Filipe, A. Fred, and B. Sharp, editors, Proc. Second International Conference on Agents and Artificial Intelligence ICAART 2010, volume CCIS 129, pages 186–200, Berlin Heidelberg, 2011. Springer-Verlag.

[37] E. A. Billing, T. Hellstro¨m, and L.-E. Janlert. Robot learning from demonstration using predictive sequence learning. In A. Dutta, editor, Robotic Systems – Applications, Control and Programming, pages 235– 250. Intech, 2012.

[38] E. A. Billing, H. Svensson, R. Lowe, and T. Ziemke. Finding your way from the bed to the kitchen: reenacting and recombining sensorimotor episodes learned from human demonstration. Frontiers in Robotics and AI, 3(9), 2016.

[39] P. G. Esteban, P. Baxter, T. Belpaeme, E. Billing, H. Cai, H.-L. Cao, M. Coeckelbergh, C. Costescu, D. David, A. De Beir, Y. Fang, Z. Ju, J. Kennedy, H. Liu, A. Mazel, A. Pandey, K. Richardson, E. Senft, S. Thill, G. Van de Perre, B. Vanderborght, D. Vernon, H. Yu, and T. Ziemke, "How to Build a Supervised Autonomous System for Robot-Enhanced Therapy for Children with Autism Spectrum Disorder", Paladyn, J. Behav. Robot., Vol. 8, pp. 18-38, 2017.

.

# Appendix I – Work Package 3 Tasks and Outcomes

The key goal of Work Package 3 was to accelerate the deployment of cognitive systems in industrial applications and to identify the technologies that most urgently need further development. It comprises five tasks:

Task 3.1: Industrial priorities (months 1-18)
Task 3.2: Catalogue of Cognitive Systems Capabilities (months 6-18)
Task 3.3: RockEU2 Cognitive Architecture Schema (months 1-24)
Task 3.4: Cognition-Autonomy Framework (months 1-18)
Task 3.5: Software Engineering Factors in Cognitive Robotics (months 9-21)

The original proposal identified some links between these tasks, stating that Task 3.1 would provide input for Task 3.3, 3. 4, and 3.5, and that Task 2 would provide input for Task 3.1, 3.3, 3.4, and 3.5. While indicative of their mutual relevance, these links lack the specificity necessary to render them operationally useful. Here, we aim to make explicit the flow of information between these five tasks and to highlight what each task will endeavour to do with that information.

Figure A1 depicts a flow-graph showing the five tasks and their outputs.[5] Some of these outputs are deliverables while others encapsulate information that will be derived from these deliverables.

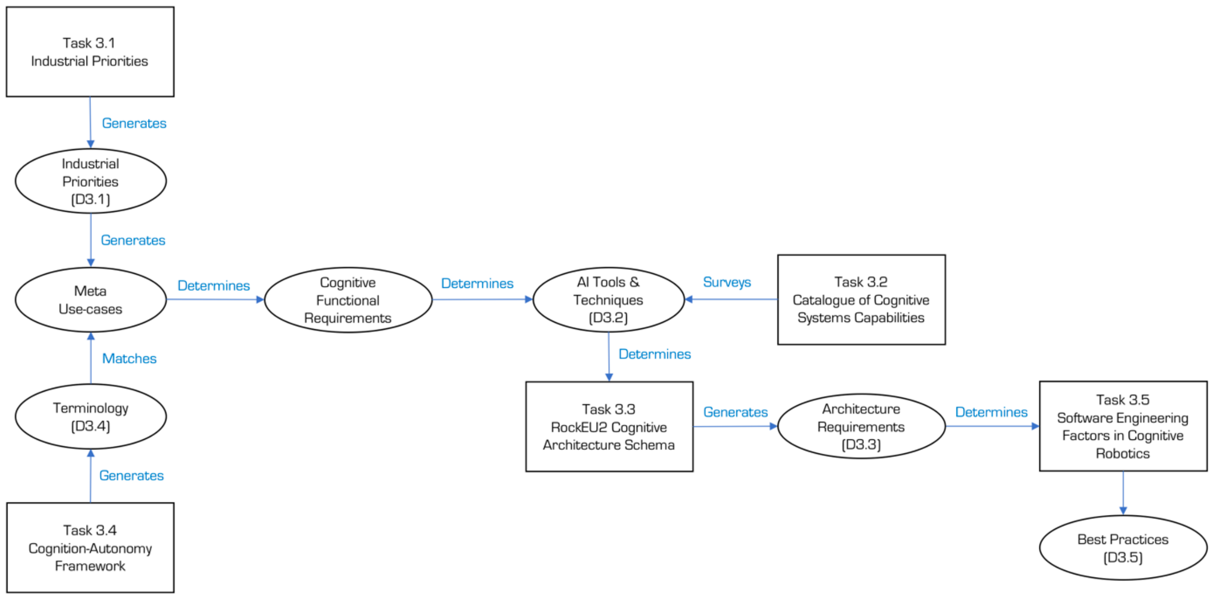
Task 3.1
Industrial Priorities
Generates
Industrial Priorities [D3.1]
Generates
Meta Use-cases
Determines
Cognitive Functional Requirements
Determines
AI Tools & Techniques [D3.2]
Surveys
Task 3.2
Catalogue of Cognitive Systems Capabilities
Matches
Determines
Terminology [D3.4]
Task 3.3
RockEU2 Cognitive Architecture Schema
Generates
Architecture Requirements [D3.3]
Determines
Task 3.5
Software Engineering Factors in Cognitive Robotics
Generates
Task 3.4
Cognition-Autonomy Framework
Best Practices [D3.5]

Figure A1: Information flow among the five task in Work Package 3 – Cognitive System Coordination.

Let's look at each of these outputs. The industrial priorities document, which is the product of Task 3.1, is complete. It comprises, essentially, eleven desiderata for cognitive robotics. These desiderata represent a distillation of the views of many industrial developers and, by extension, their customers. However, the distillation has rendered them somewhat abstract in formulation and they are difficult to map directly to the AI tools and techniques that are required to make them a reality, or even indirectly through an intermediate list of

[5] Although Figure 1 shows the information flow among the work package 3 tasks, it should not be interpreted as a schedule of tasks (e.g. as a form of PERT chart). All tasks are operational concurrently.

requirements for cognitive functionality. Therefore, we exposed the essence of all eleven desiderata in a meta[6] use-case. This was used to provide the list of requirements for cognitive functionality.

Notably, this use-case uses the terminology that resulted from an exercise (Task 3.4) in viewing cognitive robots as autonomous systems and highlighting the language required to specify autonomous behaviour as opposed to the language used to describe cognitive behaviour.

A parallel task, Task 3.2, to create a catalogue of implemented and accessible cognitive systems capabilities had the goal of identifying the AI tools and techniques that can be deployed today to satisfy the cognitive functional requirements. As a result of this change of strategy, Task 3.2 did not produce such a list (see Deliverable D3.2 for details) and instead it used a case-study of an open cognitive architecture for deployment of cognitive robots in everyday tasks to identify typical AI tools and techniques.

It was originally envisaged that Task 3.3 would map the catalogue of AI tools and techniques to cognitive architecture requirements, also known as a cognitive architecture schema. For the reasons set out in the main text of this deliverable, this proved to be futile goal and instead we listed our reflections on architectural requirements based on Deliverables D3.1, D3.2, and D3.4, including lessons learned from experience with CRAM when preparing deliverable D3.2 and afterwards, focussing on the pragmatic issues of specifying, designing, and implementing a practical robotics cognitive system.

In turn, these architecture requirements must be converted to a fully functional system. That requires sophisticated software engineering. While there is more than one popular software engineering paradigm in robotics – component-based software engineering and model-based software engineering to name two of the most prominent – the deployment of that paradigm should factor in the type of software architecture that is required to integrate the AI tools and techniques into an effective cognitive robot control system. These insights and the related architectural requirements are taken up in Deliverable 3.5 on the implications for software engineering in cognitive robotics. Sone of the issues are anticipated in Appendix III.

[6] They are meta use-cases because they are more generic in their description than would be a use-case tied to a real target application.

# Appendix II – Industrial Priorities for Cognitive Robotics[7]

While cognitive robotics is still an evolving discipline and much research remains to be done, we nevertheless need to have a clear idea of what cognitive robots will be able to do if they are to be useful to industrial developers and end users. The RockEU2 project canvassed the views of thirteen developers to find out what they and their customers want. The results of this survey follow, cast as a series of eleven functional abilities.

*1. Safe, reliable, transparent operation*

Cognitive robots will be able to operate reliably and safely around humans and they will be able to explain the decisions they make, the actions they have taken, and the actions they are about to take. A cognitive robot will help people and prioritize their safety. Only reliable behaviour will build trust. It will explain decisions, i.e. why it acted the way it did. This is essential if the human is to develop a sense of trust in the robot.

A cognitive robot will have limited autonomy to set intermediate goals to when carrying out tasks set by users. However, in all cases it defers to the user's preferences, apart from some exceptional circumstances, e.g. people with dementia can interact in unpredictable ways and the robot will be able to recognize these situations and adapt in some appropriate manner.

The freedom to act autonomously will have formal boundaries and the rules of engagement will be set on the basis of three parameters: safety for people, safety for equipment, and safety of the robot system. The rules may change depending on the environment and a cognitive robot will not exceed the limits of safe operation. The limits may be application specific, e.g., the robot should not deviate further than a given specification/distance/etc. A cognitive robot will use this type of knowledge to act responsibly and will ask for assistance when necessary (e.g. before it encounters difficulties). In particular, in emergency situations, the robot will stop all tasks to follow some emergency procedure. Ideally, if the user is deliberately trying to misuse the robot, e.g. programming it to assist with some unethical task, a cognitive robot will cease operation.

*2. High-level instruction and context-aware task execution*

Cognitive robots will be given tasks using high-level instructions and they will factor in contextual constraints that are specific to the application scenario when carrying out these tasks, determining for themselves the priority of possible actions in case of competing or conflicting requirements.

Goals and tasks will be expressed using high-level instructions that will exploit the robot's contextual knowledge of the task. This will allow the robot to pre-select the information that is important to effectively carry out the task. The goals will reflect the user's perspective. This means that all skills which implicitly define the goals are tightly linked to real- world needs and to the solution of specific problems, e.g., "get me a hammer". The following guidelines will apply.

1. Instructions will use natural language and gestures to specify the goals.
2. Natural language will be relatively abstract but will be grounded in the codified organizational rules, regulations, and behavioural guidelines that apply to a given application environment. This grounding means that each abstract instruction is heavily loaded with constraints which should make it easier for the robot to understand and perform the task effectively.
3. The goals should be specified in a formalized and structured way, where the designer defines them well and can verify them. For example, teach the robot the environment it is working in, follow a described route to reach each of the target locations and reach these positions to carry out the task. These clearly-specified tasks are tightly coupled with risks and costs, e.g. of incorrect execution.
4. It should be possible for the robot to be given goals in non-specific terms (e.g. assist in alleviating the symptoms of dementia), guidelines on acceptable behaviour (or action policies), and relevant constraints, leaving it to the robot to identify the sub-goals that are needed to achieve these ultimate goals.

[7] The following are abstracted from Deliverable D3.1 and [8].

5. A cognitive robot will learn ways of measuring the success of outcomes for the objectives that have been set, e.g., creating a metric such as the owner's satisfaction related not only to the directly specified objective but also the manner in which the job was done). It should learn from these metrics.

A cognitive robot will consider the contextual constraints that are specific to the application scenario. It will determine the priority of potential actions, e.g., in case of competing or conflicting needs. For example, the robot might know the procedure to be followed but the locations to be visited or the objects to be manipulated need to be specified (or vice versa). For example, when an automated harvester encounters a bale of straw, it can deal with it as an obstacle or something to be harvested, depending on the current task. For example, the robot might engage in spoken interaction with older adults until the goal is communicated unambiguously, using context to disambiguate the message and allow for the difficulties in dealing with different accents, imprecise speech, and poor articulation. A cognitive robot will know what is normal, i.e. expected, behaviour (possibly based on documented rules or practices) and it will be able to detect anomalous behaviour and then take appropriate action. The following guidelines will apply.

1. It will be possible to pre-load knowledge about the robot's purpose and its operating environment, including any rules or constraints that apply to behaviour in that environment.
2. It will be possible to utilize domain-specific skill pools (e.g. from shared databases) so that the robot is pre- configured to accomplish basic tasks without having to resort to learning or development.
3. The robot will continually improve its skills (within limits of the goals and safety, see above) and share these with other robots.
4. The robot might assist the user by proposing goals from what it understood and the user makes the final selection.

The level of detail in the description required by a cognitive robot will decrease over time as the robot gains experience, in the same way as someone new on the job is given very explicit instructions at first and less explicit instructions later on. One should need to demonstrate only the novel parts of the task, e.g., pouring liquid in a container, but not the entire process. It will be possible to instruct the robot off-line if there is no access to the physical site; e.g., using a simulation tool, with the robot then being deployed in the real scenario.

### *3. Knowledge acquisition and generalization*

Cognitive robots will continuously acquire new knowledge and generalize that knowledge so that they can undertake new tasks by generating novel action policies based on their history of decisions. This will allow the rigor and level of detail with which a human expresses the task specification to be relaxed on future occasions.

A cognitive robot will build and exploit experience so that its decisions incorporate current and long term data. For example, route planning in a factory, hospital, or hotel should take into account the history of rooms and previous paths taken, or it might take another look to overcome high uncertainty. In general, the robot will overcome uncertainty in a principled manner.

A cognitive robot will generalize knowledge to new task by understanding the context of a novel task and extrapolating from previous experience. For example, a care-giving robot will reuse knowledge of a rehabilitation exercise, customizing it to another person. A welding robot will weld a new instance of a family of parts. In general, a cognitive robot will extract useful meaning from an interaction for a future and more general use, with the same or another user. This may extend to learn cultural preferences and social norms.

For example, in a domestic environment, a cognitive robot will learn how to do simple household tasks, e.g. how to grasp different objects and them bring to a person that wants them. This will be continuously extended, allowing the robot to do more complex things, including cooking.

### *4. Adaptive planning*

Cognitive robots will be able to anticipate events and prepare for them in advance. They will be able to cope with unforeseen situations, recognizing and handling errors, gracefully and effectively. This will also allow them to handle flexible objects or living creatures.

A cognitive robot will be able to recognize that circumstances have changed to avoid situations where progress is impossible. It will also be able to recognize errors and recover. This may include retrying with a slightly different strategy. The learning process will be fast, ideally learning from each error.

A cognitive robot will be able to learn how to handle errors, how to react to situations where, e.g., a human is doing something unexpected or parts are located in an unexpected place.

A cognitive robot will be able to anticipate events and compensate for future conditions. For example, an automated combine harvester will be able to apply a pre-emptive increase of power to compensate for the demands caused when an area of high yield is encountered.

A cognitive robot will be able to learn about the environment it is in and modify the its current information accordingly. That is, it will adapt to changes in the environment, verifying that the environment matches with what is known, or there is a change and updates. This may require an update of the task but only after asking the user.

A cognitive robot will be able to manipulate flexible or live objects, e.g. living creatures such as laboratory mice. To do so means that the robot must be able to construct a model of their behaviour and adapt its actions as required, continually refining the model.

*5. Personalized interaction*

Cognitive robots will personalize their interactions with humans, adapting their behaviour and interaction policy to the user's preferences, needs, and emotional or psychological state. This personalization will include an understanding of the person's preferences for the degree of force used when interacting with the robot. A cognitive robot will be able to adapt its behaviour and interaction policy to accommodate the user's preferences, needs, and emotional state. It will learn the personal preferences of the person with whom it is interacting. For example, an autonomous car will learn the preferred driving style of the owner and adopt that style to engender trust.

A cognitive robot will understand nuances in tone to learn a person's voice, detecting signs of stress so that it can react to it and review what it is doing. In the particular case of interaction with older adults, the robot will be able to understand gestures to help disambiguate words.

A cognitive robot will able to extrapolate what has been taught to other situations. For example, it might remember that the user has certain preferences (e.g. to be served tea in the morning) and the robot will remember that preference. However, the robot will not allow these learned preferences to over-ride critical actions policies.

In cases where showing the robot what to do involves physical contact between the user and the robot, the robot will be able to learn the dynamics of the user, i.e. his or her personal preferred use of forces when interacting with objects in the environment.

A cognitive robot will be able to the psychological state of a user, e.g. based on the facial expressions, gestures, actions, movements. Based on this, it will be able to determine what they need by cross-referencing that with knowledge of the person's history.

A cognitive robot will be able to make decisions from a large body of observed data, thereby assisting people who typically make decisions based on learned heuristic knowledge but without a quantitative basis for this decision-making. For example, there is a need to provide farmers with a fact- based quantitative decision-making framework. A cognitive robot or machine would observe the physical environment and the farmer and provide a sound bases for making improved decisions.

*6. Self-assessment*

Cognitive robots will be able to reason about their own capa- bilities, being able to determine whether they can accomplish a given task. If they detect something is not working, they will be able to ask for help. They will be able to assess the quality of their decisions.

If a cognitive robot is asked to perform a certain task, it will be able to say whether it can do it or not. It will detect when something is not working and will be able to ask for help.

A cognitive robot will assess the quality of its decisions and apply some level of discrimination in the task at hand, e.g. being selective in its choice of fruit to harvest.

7. *Learning from demonstration*

Cognitive robots will be able to learn new actions from demonstration by humans and they will be able to link this learned knowledge to previously acquired knowledge of related tasks and entities.

Instructions will be communicated by demonstration, through examples, including showing the robot the final results, with the robot being able to merge prior know-how and knowledge with learning by demonstration. Some of this prior knowledge should be extracted from codified organizational rules, regulations, and behavioural guidelines.

The situation is analogous to training an intern or an apprentice: a trainer might ask "Has someone shown you how to do this? No? Okay, I'll show you how to do three, then you do 100 to practice (and to throw away afterwards). If you get stuck on one, call me, and I'll show you how to solve that problem".

A cognitive robot will learn and adapt the parameters to achieve the task. Today in the assembly of components, often robot assembly is not robotized because it requires too much engineering and it is too difficult for robots because it is based on traditional programming, tuning and frequent re-tuning of parameters.

Teaching will exploit natural language, gaze and pointing gestures, and by showing the robot what to do and helping it when necessary.

Actions will be expressed in high-level abstract terms, like a recipe, ideally by talking to it. For example, "go to hall 5 from hall 2 and pick up the hammer" or "open the valve".

When being taught, the robot should be anticipating what you are trying to teach it so that it predicts what you want it to do and then tries to do it effectively.

It will be possible to provide direct support for the robot, switching fluidly between full autonomy, partial autonomy, or manual control.

*8. Evaluating the safety of actions*

When they learn a new action, cognitive robots will take steps to verify the safety of carrying out this action. If a robot learns new action, it will be difficult to certify the new action. The process of generating a new action will involve interaction with the world and that may already be harmful. So, when learning a new action, there needs to be a step to verify the safety of carrying out this action. For example, showing a new action plus defining safety and success such that the robot can check if it achieved success.

*9. Development and self-optimization*

Cognitive robots will develop and self-optimize, learning in an open-ended manner from their own actions and those of others (humans or other robots), continually improving their abilities.

A cognitive robot will be able to use what it has learned to determine possible ways to improve its performance, e.g. through internal simulation at times when the robot is not working on a given task. It will also be able to learn from its mistakes, e.g., breaking china but learning from the effect of the action. A cognitive robot will learn to optimize the actions it performs (e.g. doing something faster) within the certified limits of safety and without increasing the risk of failure and associated costs.

*10. Knowledge transfer*

Cognitive robots will be able to transfer knowledge to other robots, even those having a different physical, kinematic, and dynamic configurations and they will be able to operate seamlessly in an environment that is configured as an internet of things (IoT).

A cognitive robot will be a crucial component of cyber- physical systems where the robot can be used, for example, as a way of collecting data from large experiments.

### *11. Communicating intentions and collaborative action*

Cognitive robots will be able to communicate their intentions to people around them and, vice versa, they will be able to infer the intention of others, i.e. understanding what someone is doing and anticipating what they are about to do. Ultimately, Cognitive robots will be able to collaborate with people on some joint task with a minimal amount of instruction.

The need for people around a cognitive robot to be able to anticipate the robot's actions is important because, if cognitive robots are to be deployed successfully, people need to believe the robot is trustworthy. A cognitive robot will be able to interact with people, collaborating with them on some joint task. This implies that the robot has an ability to understand what the person is doing and infer their intentions.

# Appendix III – Component-Based Software Engineering (CBSE)[8]

CBSE targets the development of reusable software [16] and has been widely adopted in the robotics community [17]. It complements traditional object-oriented programming by focussing on run-time composition of software rather than link-time composition. Consequently, it allows different programming languages, operating systems, and possibly communication middleware to be used in a given application. Related to the classic concept of communicating sequential processes (CSP) [18], components are individually-instantiated processes that communicate with each other by message- passing. Typically, component models assume asynchronous message-passing where the communicating component is non- blocking, whereas CSP assumed synchronous communication. The key idea is that components can act as reusable building blocks and that applications and system architectures can be designed by *composing* components. This gives rise to the two key concerns of component-based models: *composability* (the property of a component to be easily integrated into a larger system, i.e. to be reused under composition) and *compositionality* (the property of a system to exhibit predictable of the components are known) [19].

In complex robotics systems, as in other large software systems, integration is difficult. It is made easier by adopting practices that support composability and compositionality. As Bruyninckx notes, composability and compositionality are not independent and "achieving full compositionality and composability is an ideal that is probably impossible to achieve" [20] because the information hiding characteristic inherent in good component design conflicts with the transparency required by system builders to optimize system robustness by selecting components whose internal design is such that it can cope with as many system variations as possible.

Although they share some common principles, CBSE and object-oriented approaches are not identical. Typically, the granularity of components in CBSE is larger than that of objects in object-oriented approaches. Thus, the functionality encapsulated in a component is usually greater than that of an object. Also, components are explicitly intended to be stand- alone independently-executable reusable pieces of software with well-defined public interfaces.

On the other hand, in CBSE, as in object-oriented programming, the component specification is separated from the component implementation. A component exposes its functionality through abstract interfaces that hide the underlying implementation. Thus, the implementation can be altered without affecting any of the systems or other components that use (i.e. interface with) that component. Components exchange data through their interface.

In robotics, a component implements a well-encapsulated element of robot functionality and robot software systems are constructed by connecting components [21, 22]. The connections are often, but not necessarily made using ports in the components. As we will see below, this echoes the key elements of the CPC model.

## CPC AND THE BRICS COMPONENT MODEL

Focussing on the issues that are particularly important in developing robot software, the BRICS Component Model (BCM) [19], [23], [24] is a best-practice CBSE model that provides a set of principles, guidelines, meta-models, and tools for structuring the development of individual components and component-based systems. It does so in a framework-nonspecific manner, i.e. at an abstract (meta) level that does not refer explicitly to the implementation of the required functionality on a specific robot platform.

BCM brings together two strands of software engineering: the separation of the so-called 4Cs of communication, computation, configuration and coordination [25] and robotics- oriented MDE [26]. BCM extends the 4Cs to 5Cs [27] by splitting configuration into finer-grained concepts of configuration and

[8] The material in this appendix is abstracted from [10]. Note: it is not intended to pre-empt the findings to be documented in deliverable D3.5. On the contrary, it is intended merely to provide some technical background to the arguments put forward in this deliverable on the nature to two of the most prevalent robotics middleware systems, ROS and YARP.

composition. The core principle is that good component-based engineering practice strives to decouple the design and implementation of these five concerns.

*Computation* refers to the system's main functionality, i.e. the information processing the component has been designed to perform. *Communication* provides the data required by the computation and takes care of the quality of service of data communication, e.g. timing, bandwith, loss (accuracy), priority, and latency. *Coordination* refers to the functionality that determines how all the components in a system work together and, thus, how the component or system behaves. *Configuration* refers to the functional aspects concerned with influencing the behaviour of the computation and communication elements either at start-up or at run-time. Finally, *composition*, the fifth C introduced in [19], provides the glue that binds together the other four Cs, each of which is focussed on decoupling functionality. In contrast, composition is very much concerned with coupling: how the other four Cs interact. Strictly, this C does not reflect software functionality but is rather a design characteristic that seeks to find a good trade- off between composability and compositionality [27].

Complementing CBSE, MDE aims to improve the process of code generation from abstract models that describe a domain. The OMG [28] defines four levels of model abstraction, going from higher to lower levels of domain specificity (i.e. from platform-independent to platform-specific) by adding platform knowledge. These four levels can be characterized as follows [19].

M3 Domain-nonspecific: the highest level of abstraction using a meta-meta-model.

M2 Platform-independent representation of a domain, using, e.g., a *Component-Port-Connector* (CPC) meta- model.

M1 Platform-specific model: a concrete model of a specific robotic system but without using a specific programming language.

M0 The implementation level of a specific robotic sys- tem, with defined software frameworks, libraries, and programming languages.

Level M2 is of particular interest here because the abstract model, i.e. the CPC meta-model, maps directly to the principles of component-based software engineering. The essence of this model is as follows [19]. An application system has zero or more components and zero or more connectors. A component has zero or more ports, and a port belongs to one and only one component. A connector is always between two ports. Finally, and of particular importance in the context of the current discussion, components expose their data and service interfaces on their ports and exchange data over the attached connectors.

The principles of CBSE have another important role in that they also constrain the selection of a software environment that will be used to provide the abstraction layer between the application code, on the one hand, and the middleware and operating system services (including support of distributed systems processing and device input/output), on the other. This abstraction layer is typically provided by the robot programming framework. It provides an abstract interface between the robot software system (implemented as a network of components) and the underlying operating system and middleware.

The CBSE CPC approach offers several advantages in configuring robotics applications. The robot application can be implemented simply by identifying the components to be instantiated and specifying the connections between their port interfaces. Each component can encapsulate some coarse- grained robot functionality, i.e. the component can perform some substantial function while maintaining its cohesion. It is possible to have multiple instances of the same component by having a mechanism for uniquely naming each instance of the component and propagating that name to the component's port names. A key characteristic is the support for external configuration of the component allowing the behaviour of individual instances of a component to be customized by the application developer without recourse to the component developer. This goes hand in hand with a facility to allow components to run in different contexts, meaning that the robot programming framework allows the component user to specify the location of the components configuration files and other resources. Runtime configuration of the behaviour of the component can be provided to allow users and other components to control the component's processing, if required. In many robot programming frameworks, e.g. YARP, components run asynchronously on a distributed system with the inter-component communication over ports being handled transparently by the robot programming framework. This means that the logical definition of a component-based application is independent from its run-time configuration and the ports provide abstract component interfaces. This communication infrastructure may provide multiple transport layer protocols, e.g. udp, tcp, and mcast.

What is significant about this is that adopting the CBSE CPC approach imposes a minimal number of required configuration practices on the component developer. Specifically, these are to support:

1) Unique naming of each instance of a component and propagation of that name to the port names;

2) Renaming any of the ports through which the interfaces are exposed;

3) External configuration of the component behaviour;

4) External specification of the context (i.e. path) to search for the associated configuration files;

5) Run-time configuration of the component behaviour reading parameter values from a port.

Clearly, the underlying robot programming framework is pivotal in supporting these practices and making them straightforward for the component developer to implement. There are many robot programming frameworks. These include ROS [29], YARP [30], URBI [31], and Orca [32, 33]. All have their respective advantages and disadvantages, as explained in various surveys and comparisons [34, 35]. ROS, a popular choice in many instances, has been criticized in the context of CBSE as lacking an abstract component model [26]. On the other hand, while YARP does not have an abstract component model either, it provides a framework that explicitly supports all the CBSE needs identified above.