# Transferring Object Grasping Knowledge and Skill Across Different Robotic Platforms


Ali Paikan
iCub Facility
Italian Institute of Technology (IIT)
Genova, Italy
Email: ali.paikan@iit.it

David Schiebener
H$^2$T Lab
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: schiebener@kit.edu

Mirko Wächter
H$^2$T Lab
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: waechter@kit.edu

Tamim Asfour
H$^2$T Lab
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: asfour@kit.edu

Giorgio Metta
iCub Facility
Italian Institute of Technology (IIT)
Genvoa, Italy
Email: giorgio.metta@iit.it

Lorenzo Natale
iCub Facility
Italian Institute of Technology (IIT)
Genova, Italy
Email: lorenzo.natale@iit.it



***Abstract*—This study describes the transfer of object grasping skills between two different humanoid robots with different software frameworks. We realize such a knowledge and skill transfer between the humanoid robots iCub and ARMAR-III. These two robots have different kinematics and are programmed using different middlewares, YARP and ArmarX. We developed a bridge system to allow for the execution of grasping skills of ARMAR-III on iCub. As the embodiment differs, the known feasible grasps for the one robot are not always feasible for the other robot. We propose a reactive correction behavior to detect failure of a grasp during its execution, to correct it until it is successful, and thus adapt the known grasp definition to the new embodiment.**


## I. INTRODUCTION

Sharing knowledge and skills across different robotic platforms is a challenge for collaborating research institutes. For future service robots, sharing real world experiences and acquired knowledge offers a huge opportunity to accelerate and achieve proper handling of novel situations. The first step in this direction is to achieve software compatibility. Best practices in software architecture for robotics promote modularity and distributed component–based [1] software development. Following these paradigms, several attempts have been made to develop middlewares such as ROS [2], OROCOS [3] to facilitate robot programing in distributed environments, some of which are based on customization of some standard communication libraries (e.g., CORBA [4], ICE [5]). Diversity of robotic middlewares makes knowledge transfer between robotic platform more difficult due to lack of interoperability between software architectures.

Although there is no silver bullet to solve this problem, the robotics literature describe some attempts to support interoperability between software architectures. Fitzpatrik et al. [6] described design choices that proved crucial to achieve compatibility between middleware, namely: support for interchangeable protocols, IDLs and multiple name servers. Wienke et al. [7] proposed a meta–model for data type mapping and a code generation toolchain to improve interoperability of robotic frameworks. In the real–world, middleware users usually rely on custom bridges [8] which perform the transformations and operations required to interconnect heterogeneous systems. Bridges are a suboptimal solution because they introduce maintenance overheads and communication latencies. In some applications the reduction of performance may not even be acceptable.


978-1-4673-7509-2/15/$31.00 ©2015 IEEE

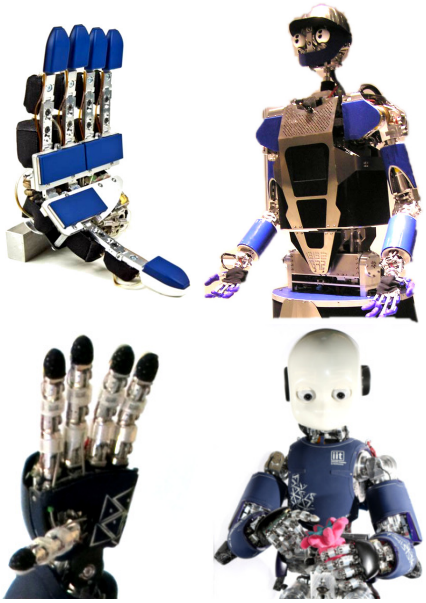



Fig. 1: The ARMAR-III (top) and iCub (bottom) robots and their respective hands.

A more fundamental stumbling block to skill transferring and research collaboration in robotic is diversity in the embodiment and physical characteristic of robots. For abstract skills these differences are not the major issue since the robot control is mostly done in the task–related space which is independent

from how the primitive actions are performed by the robot. For example, adopting a pick–and–place skill from another robot is respectively easier if both robots know how to grasp the object. On the other hand, skills can be more robot dependent (such as object grasping). That means that their feasible execution heavily depends on specific characteristics of the robot such as the size of the hand or number of the fingers. Therefore the skill and knowledge for a robot can not be easily and feasibly transfered to the other one without proper adaptation.

This paper illustrates an experiment of knowledge and skill transfer between the humanoid robots iCub [9] and ARMAR-III [10] (See Fig. 1) which differ significantly in their physical characteristics and software framework. A conceptual representation of grasp skill is demonstrated in Fig 2. As shown in the figure, the upper layer contains grasp knowledge, the middle layer represents robot software framework and the lower layer shows the relevant robot platform. The grasp knowledge is, in fact, a database of feasible grasp descriptions defined by the position and orientation of the robot hand relative to each known object. The grasp descriptions can be given to robot as prior knowledge or it can be automatically reasoned and learned by robot from interaction with the object. Regardless how the grasp knowledge is provided, it is highly robot dependent. For example, due to the different robot embodiments (in particular size of the hands) grasps that are feasible on ARMAR–III cannot be successfully realized on the iCub and they need to be adapted to a new embodiment. Moreover, the software framework which is used to implement the grasp skill on the ARMAR–III is entirely different from the one is used for programing the iCub robot.

Therefore, two problems arise: *i)* how to interconnect software components developed for the two robots and *ii)* how to adapt a grasping skill to robots with different embodiments. We address the first problem by proposing a solution to interconnect the two middlewares (i.e., ArmarX [11] and YARP [12]) using plug-in system. The approach is based on our previous works on software reusability using port plug–ins [13] and extends it for the interoperability of different frameworks. Moreover, to be able to use the grasp knowledge of Armar-III directly on the iCub, we adopt a run–time learning and adaptation approach [14] to detect the causes of failure of a grasping skill during its execution and reactively correct it until it is successful. In the rest of paper we describe these approaches and demonstrate the experimental evaluation of our grasp-skill-transfer scenario from ARMAR-III to the iCub.

## II. BRIDGING THE MIDDLEWARES

YARP and ArmarX are two robotic middlewares which are independently developed based on component–based and distributed software architecture techniques. Both middlewares support data streaming based on the publish–subscribe paradigm and Remote Procedure Calls (RPC) for inter–component communication but still differ in the way they implement these functionalities. While ArmarX relies on the ZeroC Internet Communication Engine (ICE) [5], YARP aims at abstracting communication from the underlying protocol (known as YARP carriers) which implements data transfer among the end points. However, the available carriers in YARP do not support communication in an ICE–based network. YARP and ArmarX also differ in the RPC (services) implementations. YARP uses the Apache Thrift IDL [15] format

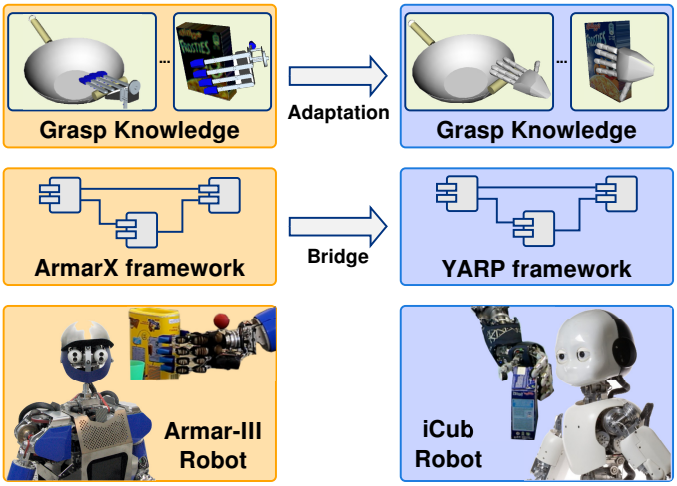


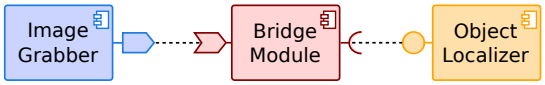

Fig. 2: A conceptual representation of transferring grasp knowledge and skill from ARMAR–III to the iCub robot.

and syntax with its own native serialization, while ArmarX employs the Slice IDL to define interfaces and classes in a network transparent manner. For these reasons the software components from one middleware cannot communicate with those implemented in the other framework and the two middlewares need to be bridged in some way.

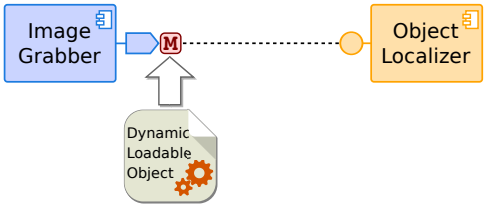
There have been many research efforts and studies on the interoperability of the middlewares which mostly concentrate on the bridging of communication protocols or data type mapping and conversion. For example, a set of carriers has been implemented in YARP to support YARP-ROS [6] interoperability via XML/RPC and TCPROS protocol. However, interconnecting two robotic platforms requires more than simply solving the interoperability problem on the communication level. Performance constraints and incompatibility at the component interfaces are other issues that need to be solved. Each middleware may provide different ways for accessing the robot sensory data or commanding the actuators. For example, separate interfaces are implemented in YARP to control the iCub joint sets individually (e.g., left arm, right arm, torso) while ArmarX sees them as a unique set and has a single interface to access all the joints. Further, images from the iCub cameras are streamed out using data–flow ports while, in contrast, image processing components (e.g. an object localizer) in ArmarX require an image provider service (ICE proxy) to access the image data using remote procedure calls.

### A. Port Plug-ins approach

Consider the example from Fig. 3a. The Image Grabber is a YARP component which captures camera images and streams them out using a data–flow port. The Object Localizer is a component already implemented in the ArmarX framework which requires a specific ICE proxy object to access a stereo image pair and localize a known object in it. To allow the image data from the YARP network to be accessible in the ArmarX framework, a simple solution is to implement a dedicated bridge (called Bridge Module in Fig. 3a). It has a YARP input port to receive the image data, converts it into the ArmarX image format and provides the




(a) Using a separate bridging component




(b) Using a port plug–in. A port monitor (indicated by 'M') loads plug–ins implemented as dynamic loadable objects.

Fig. 3: Different ways to bridge components from different middlewares.

required ICE service to make the image data available in the ArmarX framework for the Object Localizer. The drawback of using a separate bridging component is that it introduces communication and execution overhead to the system since data has to be transfered to the Bridge Module, received and then processed. Moreover, when many of these bridging components are required, their maintenance and deployment put extra overhead to the application development cycle.

In our previous works, we have introduced the port plug–ins [13] approach and its application in software reusability [16] and coordination of the components [17]. The basic idea is to extend the port's functionalities in order to dynamically load a run–time script and plug it into the port of an existing component without changing the code or recompiling it. In our approach a port extension is called Port Monitor: in brief it allows accessing data passing though a connection from/to the port for monitoring, filtering and transforming it. A port monitor has a set of callbacks which can have their corresponding implementations in the user's script. Using these callbacks, users have full control over the port's data and can access and modify it.

Until recently port plug–ins have been implemented using a scripting language (Lua) and mostly used for data filtering and conversion within components which are implemented in the YARP framework. But in fact port plug–ins can be used as hooks that convert data and make it accessible for the components from other frameworks too. Towards this and to achieve better performance, the port monitor object is extended to be able to load plug–ins implemented using compiled languages and linked as dynamic loadable object.

The idea is also demonstrated in Fig. 3b. A port monitor (shown as a red box with an M) is attached to the output port of the Image Grabber which loads a plug–in implemented as a dynamic loadable object. Using this plug–in, user's code directly accesses the image frame data from the port, converts it and provides the required ICE proxy object to make it accessible for the Object Localizer component. In other words, the required functionalities for bridging which are implemented in the Bridge Module (from the architecture shown in Fig. 3a), can be transparently moved to the plug–in attached to the output port of Image Grabber (Fig. 3b).

Using the plug–in architecture for bridging components has the decisive advantage of reducing the communication latency between the components. In our reactive grasping application, it is crucial to minimize the communication latency between the Image Grabber and the components which perform perception to achieve real–time visual collision detection of the robot hand and the object. Data–flow ports are usually used to stream data whenever higher rate or communication bandwidth is required. Thus, in bridging YARP to ArmarX, we have implemented several plug–ins for the data–flow ports of YARP components to make the streamed data (such as image frames, motor encoders status) accessible for the ArmarX components.

## III. ADAPTATION OF THE GRASP SKILLS

Due to the different robot embodiments (in particular size of the hands) grasps that are feasible on ARMAR–III cannot be successfully realized on the iCub. Thus, when the iCub robot tries to execute the grasp, there is a substantial risk that it will fail, as the different hand geometry may cause collisions with the object before the intended grasp pose has been reached. To deal with this problem, we apply our method for visual collision detection presented in [14]. We observe the hand during the approach towards the object, and when it prematurely collides, causing the object to move, this is detected. In such a case, the hand pose is corrected reactively until the grasp can be executed without collisions. The approach is explained in more details in the following subsections.

### A. *Transfer of the Grasp Skill*

Grasping as implemented in ArmarX consists of three basic blocks: The first is the memory which contains - for each known object - a visual descriptor for recognition and descriptors of feasible grasps for the robot ARMAR-III. The second is a component for visual object recognition and localization based on the descriptors in the memory, using the stereo cameras of the robot. The third component executes the grasp using the object pose from the localizer and the grasp definition from the memory.

Using the bridge system developed between ArmarX and YARP, these components can also be run on iCub. The object recognition and localization pose no difficulties as both robots have a calibrated stereo camera system. The desired hand pose relative to the object for grasping it is taken from the memory. In ArmarX, a velocity based inverse kinematic is implemented that can be used to move the hand towards a desired goal. Given the kinematic model of the iCub robot and using the interfaces provided by the bridge system, this component can directly control the iCub joints to reach a given point in cartesian space.

Thus, the grasp skill designed for ARMAR–III can be executed on the iCub. However, the desired hand pose for grasping is appropriate for the hands of ARMAR-III which are significantly bigger than those of iCub, although both have 5-fingered hands, but with an only roughly comparable shape (see fig. 1). Therefore it is necessary to monitor the grasp execution closely and check for premature collisions

between hand and object, and, if such collisions occur, react appropriately.

### B. Visual Collision Detection

The visual collision detection is based on the fact that when the robot hand collides with the object, the latter starts to move. While collisions can in theory be detected by haptic or force sensors, those are usually not sensitive enough for our task. However, our visual collision detection approach is sensitive enough to detect any soft collision but only can be applied to objects that move when they are pushed.

We use optical flow to recognize that the hand caused the object moved due to a collision. To this end, the hand is localized in the camera image. When the hand touches the object it generates in its proximity an optical flow pattern. The detection of such a motion is complicated by the fact that the robot itself moves and thus the whole camera image contains significant optical flow. Therefore, we need to determine whether the optical flow in front of the hand is different from the optical flow in the rest of the image. To this end, we cluster all pixels based on the components of their optical flow vector to obtain regions of uniform optical flow. Given these sets of pixels with similar optical flow, we check an area next to the hand in the direction of its motion that has roughly the size of the object. If we find a set whose pixels lie mostly inside that area rather than outside of it, then this set is apparently caused by a motion that exists solely in that area. This strongly indicates that the hand caused an object to move with relation to the environment.

### C. Corrective Reaction

When a collision has been detected, the grasp attempt is most probably failing. Therefore, the robot needs to react to this event in an appropriate manner so the grasp can still be completed successfully. When a collision occurs, the robot retracts its hand from the object, and a corrective transformation (i.e. a slight change of hand position and/or orientation) is determined and applied to the intended grasp pose. The hand performs this corrective motion at a safe distance from the object and then again approaches the corrected grasp pose. If another collision occurs, this is repeated, until the grasp pose is reached without premature collisions.

The interesting question here is which corrective transformation yields the highest chance of a collision-free approach. We tested different possible strategies [14] and proposed that the robot should try to determine which finger caused the collision, which can be done based on the hand tracking and knowledge of the object shape (or at least its position). Then the hand position and orientation are modified so that the finger has touched the object is moved away from it. Empirically, we determined that a rotation by 20° or a translation of 25 mm or an averaged combination is effective. The optimal choice of those values probably depends on the size and shape of the hand, but we left them unchanged for the execution on iCub and had similar results on ARMAR-III.

## IV. EXPERIMENTAL EVALUATION

To evaluate our skill transfer and the bridging architecture we tested the adaptive grasp scenario on Armar-III and the iCub robot. The scenario involves different software components developed in the ArmarX framework to recognize and localize the object and the robot hand, to detect the object–hand collision from the optical flow and to control the robot to position the hand at an appropriate pose relative to the object for grasping it. The adaptive grasp application requires continuous and real–time processing of the image data from the robot cameras. We implemented a set of port plug–ins to allow the image processing components (e.g, visual contact detection) to directly access the image data from the iCub cameras with the minimum possible latency. To be able to control the iCub robot within the ArmarX framework, a Simox [18] kinematic model of the iCub robot was created. The model is internally used by the robot API of the ArmarX which allows the software components to transparently access the robot joints. A bridge component is also implemented which uses the YARP motor interfaces to access the iCub joints and map these interfaces to the proper ICE interface in the ArmarX framework. Using this bridge, the components in the ArmarX framework can access the iCub joints and control the motors in position, velocity or torque mode in the same way they control the ARMAR–III robot.

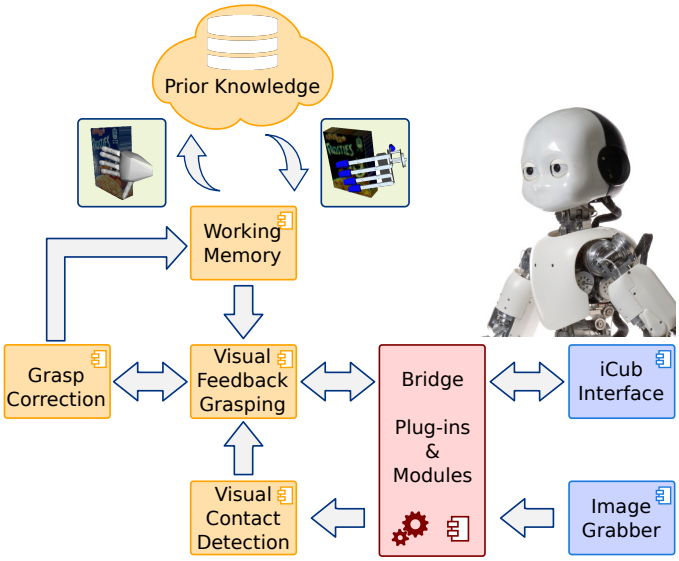
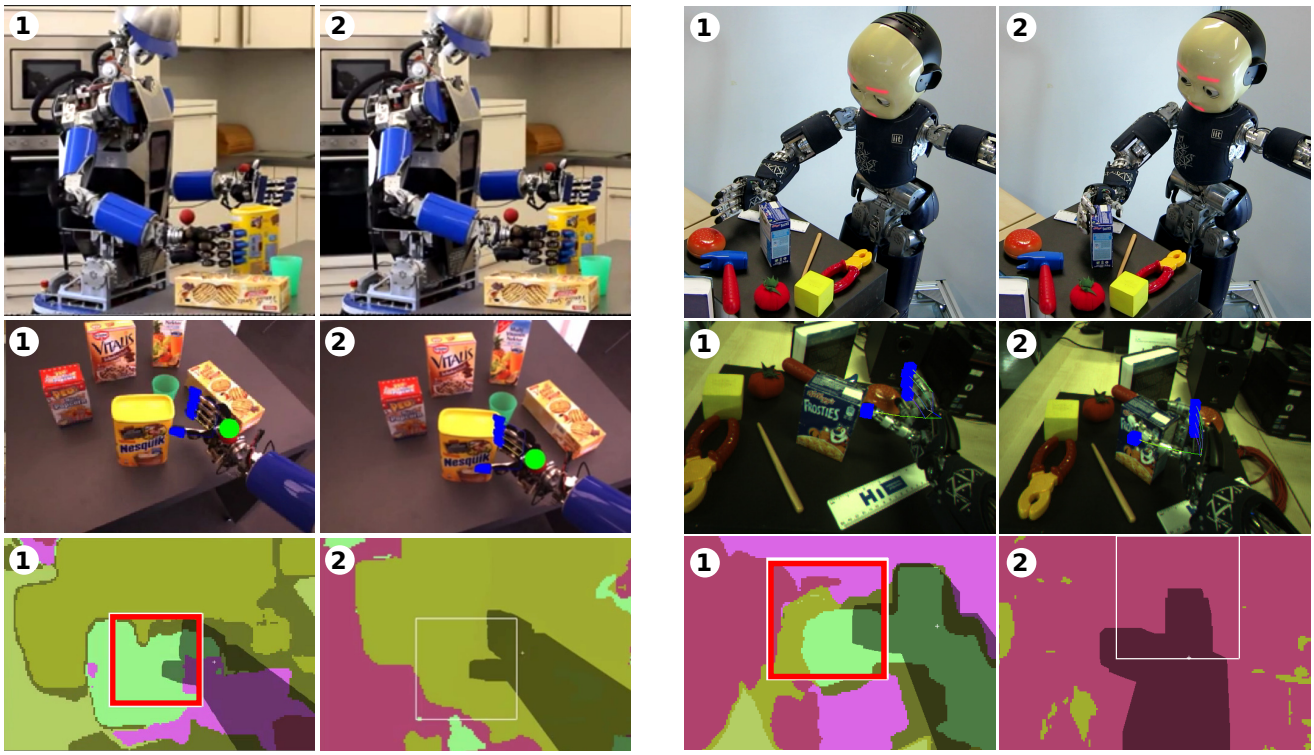


Fig. 4: The architecture of adaptive grasp scenario.

### A. Using grasp knowledge from ARMAR-III directly on iCub

In the first experiment we directly use the grasp knowledge from the ARMAR-III robot for iCub to grasp a known object. To evaluate if the grasp knowledge known by the ARMAR-III robot can be directly used by iCub, we performed a simple object grasping scenario. The required software components for the grasp scenario were executed in the ArmarX framework and, using our bridging system, they could localize the object and control the iCub robot. As we expected, due to different robot hand characteristics, the known grasp knowledge from the ARMAR–III robot was not valid for the iCub and the robot failed to grasp the object. We repeated the experiment with different grasp knowledge for different objects, and in most cases the robot failed to grasp the object due to a




(a) Reactive grasp experiment using the ARMAR–III robot

(b) Reactive grasp experiment using the iCub robot

Fig. 5: (a) ARMAR-III, (b) iCub robots grasping an object. The middle figures represent robot camera views with the estimated hand positions. The bottom figures show the result of the hand–object collision detector using the optical flow clustering. For both sets of figures, (1) shows collision detection and (2) shows reaching for a successful grasp.

mispositioning of its hand causing premature collisions of one or more fingers with the object during the final approach phase.

### *B. Adapting grasp knowledge to the iCub robot*

In the second experiment, we applied the reactive grasping method to let the iCub robot learn the correct grasp definition. We use a known grasp definition for ARMAR-III as the initial grasp suggestions for the iCub. The robot then tries to grasp the object, detects the failure(s) and corrects its hand's pose with respect to the object until it finds a position and orientation of the hand that allows for a successful grasp.

A simplified architectural view of the adaptive–grasp scenario is shown in Fig. 4. The prior grasp knowledge of ARMAR–III is fed to the Visual Feedback Grasping using the Working Memory unit. The former uses this knowledge and attempts to perform a reactive grasp on the iCub robot using the bridge. The Visual Contact detection monitors any collision with the object and informs the Visual Feedback Grasping unit. If the grasp is not successful, a new configuration of the hand is generated by the Grasp Correction unit and it is used for another attempt to grasp. The process is repeated until the robot successfully grasps the object. The successful configuration of the iCub hand with respect to the object is considered as valid grasp knowledge and it is written to the grasp–knowledge database for future use.

Fig. 5 demonstrates the reactive grasp experiment similarly performed on the ARMAR–III (a) and iCub (b) robots. The figures on the top show the robots in action. The middle figures represent corresponding robot camera views with the estimated hand positions. The bottom figures represent the result of the hand–object collision detector using the optical flow segmentation. For both sets of figures, in (1) the robot detects a collision of the hand with the object and in (2) the robot finds the correct hand position for a successful grasp.

As shown in Fig. 5b, using the knowledge from the ARMAR–III knowledge base, the iCub detects the box and attempts to reach and grasp it. As the suggested grasp knowledge from ARMAR–III is not precisely correct for the iCub hand, one of its fingers collides with the object. As the robot attempts to reach the desired grasp position, it slightly pushes the box. That generates a region in which the optical flow is different from the rest of the environment (the region marked by the red box in Fig.5b is checked for such a unique optical flow cluster). Thus, the collision is detected which causes the robot to immediately retract its hand from the box. As we have explained in Section III-C, the robot then tries to correct its hand orientation and repeatedly attempts to grasp the object until it can safely (i.e. without a collision) reach a feasible grasp pose. This is shown in Fig. 5b (2) where the iCub closes its hand in a collision–free position next to the object. The corresponding hand position and orientation with respect to the object are thus known as a correct grasp definition and are stored in the iCub's grasp knowledge base for future use.

A similar experiment which has been also done with the

ARMAR–III robot, is shown in Fig. 5a. In the experiment with the ARMAR–III, the robot is provided with an approximately correct grasp definition for a known object by the human. As this knowledge (i.e. a description of a feasible grasp) is not accurate, the robot fails to grasp the object during its first attempt (Fig. 5a (1)). The robot then tries to find the correct hand pose (Fig. 5a (2)) and store it as a valid knowledge for grasping the object.

## V. DISCUSSION

The knowledge that is used as the initial grasp suggestion for the iCub robot, is taken from the ARMAR–III knowledgbase. It is then automatically adapted to the iCub embodiment in the course of the interaction with the object. Alternatively, the robot could start without any prior knowledge of grasping and develop the grasp description by reactively interacting with the object. Another solution is geometrically reasoning about the feasible grasps. Although all these hypothesis are valid, using grasp knowledge from other robots can accelerate an unknown object learning process.

Moreover, developing a new skill on a robot requires considerable amount of work and time to design and implement a new set of software components. Thus, having a proper set of bridging plug-ins to interconnect middlewares, allows sharing skills between different robots without any reimplementation of required software components. The only drawback of the port–plugins approach is that it needs the required software libraries for the bridge to be available on the same machine where the plug–ing is deployed into the port of the component. However, this is a common concern in system maintenance which can be compromised for the performance of the middleware interoperability.

It is fair to say that in the current implementation, the port plug–ins have been used only for data–flow ports. Interconnections between RPC–style services have been implemented using dedicated bridges. In terms of performance this proved to be acceptable because RPC connections are never used in time critical loops. This is because the intrinsic bidirectional nature of the communication introduces timing dependencies between processes and imposes communication latency.

## VI. CONCLUSIONS

This paper has described the knowledge and skill transfer between two different robots with different embodiment. More specifically, we have shown how a grasping skill from ARMAR-III can be used on the iCub robot which has a significantly different hand and software framework used for its programming. We have discussed the major issues in bridging the middlewares and proposed the port–plugin approach for bridging components from different frameworks to reduce the communication latency, especially, when higher bandwidth for streaming data is required.

To transfer the grasping skill between two robots, we took the ARMAR–III's knowledge of grasping and used it as the initial grasp suggestions for the iCub. The robot then tried to adapt the skill to its own embodiment by interacting with the object and finding a valid grasp definition using our approach for visual collision detection and correction. In this way, the skill from one robot is transfered and automatically adapted to a new embodiment. In our future work, we will investigate transferring more complex skills and knowledge such as tool use and affordances between different robots.

## ACKNOWLEDGMENT

This research was supported by the European FP7 ICT project No. 270273 (Xperience), project No. 611832 (WALK-MAN) and project No. 611909 (KoroiBot).

## REFERENCES

[1] D. Brugali and P. Scandurra, "Component-based Robotic Engineering Part I : Reusable building blocks," vol. XX, no. 4, pp. 1–12, 2009.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[3] H. Bruyninckx, "Open robot control software: the OROCOS project," in *IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 2001, pp. 2523–2528.

[4] OMG, "Common Object Request Broker Architecture (CORBA/IIOP).v3.1," OMG, Tech. Rep., Jan. 2008.

[5] Z. Inc., "Internet communications engine," http://zeroc.com/ice.html.

[6] P. Fitzpatrick, E. Ceseracciu, D. E. Domenichelli, A. Paikan, G. Metta, and L. Natale, "A middle way for robotics middleware," *Software Engineering for Robotics*, (accepted).

[7] J. Wienke, A. Nordmann, and S. Wrede, "A meta-model and toolchain for improved interoperability of robotic frameworks," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2012, pp. 323–334.

[8] J. Matai, Y.-H. Suh, H. Kim, K.-W. Lee, and H. Kim, "Integration framework for interoperability of distributed and heterogeneous robot middlewares," in *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*. IEEE, 2008, pp. 2337–2343.

[9] G. Metta, G. Sandini, and D. Vernon, "The iCub humanoid robot: an open platform for research in embodied cognition," *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pp. 50–56, 2008.

[10] T. Asfour, K. Regenstein, P. Azad, J. Schröder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann, "ARMAR-III: An integrated humanoid platform for sensory-motor control," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2006, pp. 169–175.

[11] K. Welke, N. Vahrenkamp, M. Wächter, M. Kröhnert, and T. Asfour, "The armarx framework-supporting high level robot programming through state disclosure." in *GI-Jahrestagung*, 2013, pp. 2823–2837.

[12] G. Metta, P. Fitzpatrick, and L. Natale, "Towards Long-Lived Robot Genes," *Elsevier*, 2007.

[13] A. Paikan, P. Fitzpatrick, G. Metta, and L. Natale, "Data Flow Port Monitoring and Arbitration," *Software Engineering for Robotics*, vol. 5, no. 1, pp. 80–88, 2014.

[14] D. Schiebener, N. Vahrenkamp, and T. Asfour, "Visual collision detection for corrective reactions during grasp execution on a humanoid robot," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2014.

[15] Apache, "Thrift interface description language," http://thrift.apache.org/docs/idl.

[16] A. Paikan, V. Tikhanoff, G. Metta, and L. Natale, "Enhancing software module reusability using port plug–ins: an experiment with the iCub robot," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[17] A. Paikan, G. Metta, and L. Natale, "A port–arbitrated mechanism for behavior selection in humanoid robotics," in *The 16th International Conference on Advanced Robotics*, 2013, pp. 1–7.

[18] N. Vahrenkamp, M. Kröhnert, S. Ulbrich, T. Asfour, G. Metta, R. Dillmann, and G. Sandini, "Simox: A robotics toolbox for simulation, motion and grasp planning," in *Intelligent Autonomous Systems 12*. Springer, 2013, pp. 585–594.