A Model Based Engineering Tool for ROS Component Compositioning, Configuration and Generation of Deployment Information

Monika Wenger, Waldemar Eisenmenger, Georg Neugschwandtner
Ben Schneider, Alois Zoitl
fortiss GmbH
Deutschland, München
Email: {wenger, eisenmenger, neugschwandtner, schneider, zoitl}@fortiss.org

***Abstract*—Programming industrial robots requires experts – not only to create the robot applications, but also for changing them later due to different product requirements. Part of the reason for this is that all robot vendors provide their own robot programming language. Due to the language differences, robot applications cannot be reused for different robot types. Often, additional experts are required since one expert is trained only for a special robot type. This makes the use of robots uneconomical for small and medium sized enterprises. The ReApp project addresses this problem by providing a workbench based on ROS (Robot Operating System). A central part of this workbench is the skill and solution modeling tool, which allows the model-based design of robot applications composed of reusable components, and is described in this paper.**

I. Introduction

The shift from mass production to individualized production leads to smaller lot sizes and more product variants. This requires more flexible production processes, with production equipment providing better support for adaptability and reconfiguration. In mass production, robots are used due to their availablility, reliability, and accuracy, but also to minimize occupational hazards for workers. Currently, almost every manufacturer of industrial robots provides their own robot programming language, as illustrated in Table I. This requires system integrators to be trained for each robot vendor and makes reuse of robot applications between manufacturers almost impossible [1]. The effort involved in modifying the robot application program and the need for robotics experts to make the modification [2] make the use of robots uneconomical for lot size one production and, especially, for small and medium sized enterprises (SMEs).

To make robot applications affordable for SMEs, the *Reusable Robot Applications for Flexible Robot Plants Based on Industrial ROS (ReApp)* project aims to provide a development environment ("workbench") for creating, applying and managing reusable robotics software components. The ReApp workbench supports different user roles during the stages of component development, e.g. component developers or system integrators.

TABLE I
Robot vendors and their programming languages[1]

Robot Vendor	Programming Language
ABB	RAPID
Kuka	KRL
Comau	PDL2
Yaskawa	Inform
Kawasaki	AS
Fanuc	KAREL
Stäubli	VAL3
Universal Robots	URScript

Robot Operating System (ROS) [3] is a robotics middleware providing a convenient infrastructure for communication between software components. It also provides libraries for recurring problems such as image recognition, control or motion planning. A ROS system consists of nodes that specify interfaces to exchange typed messages according to the publish/subscribe and/or client/server communication paradigm [4]. For each node, parameters can be defined which can be used by the component implementation.

The ROS component model [4] provides the base for the ReApp workbench. ReApp extends the component model with an ontological classification system for hardware, software, and capabilities [1]. The capability based concept records expert knowledge regarding the usage and deployment of a specific component. The classification is supposed to simplify the discovery of suitable components and applying them correctly within any application. The ReApp workbench consists of five [1] parts: A cloud-based semantic repository [5], the Component Modeling Tool (CMT), the Skill and Solution Modeling Tool (SSMT), the Component Code Generation Tool (CCGT) and the Integration Platform & Development Environment (IPDE), with the first four using a semantic description of the ReApp components as their exchange format. ReApp uses a model-driven design methodology with automatic code generation and semantic technologies to allow the development and reuse of ROS based components at higher abstraction levels. Within this work the SSMT, a model based

[1]http://fabryka-robotow.pl/category/english/

978-1-5090-1314-2/16$ 31.00

engineering tool for ROS component compositioning, configuration and generation of deployment information, is presented. The SSMT is a central part of the ReApp workbench.

In the following, existing development tools for ROS components are discussed. Afterwards, the structure and functionality of the SSMT are introduced, and it is outlined how it provides real-time support by integrating International Electrotechnical Commission (IEC) 61499 components. The SSMT's functionality is illustrated by showing its functionality in the context of one of the three scenario demonstrators within the ReApp project. At the end, a conclusion and an outlook are given.

II. Related Work

Several tools have been designed which are supposed to simplify the development of robot applications with ROS. rxDeveloper[2] is a platform independent visual programming tool to design parameterized node graphs and automate the creation of ROS launch and parameter files [6] from them. Each connection between two nodes represents a communication channel. In combination with the other ROS tools, these node graphs can be started and monitored from within rxDeveloper.

The open source framework ROSMOD[3] provides a model-driven development workflow and a run-time management environment for the creation, deployment, and management of component-based software with ROS. ROSMOD defines a component model where each component contains executable code that implements its functionality and manipulates state variables [7]. Each component is triggered by periodic and sporadic timers to realize real-time behavior for ROS. Components interact with other components through interface elements which are represented by publishers, subscribers, clients, and servers. Non-blocking messages publish to a specific message topic and subscribe to a specific message topic. Services are offered by servers and required by clients. The code generators create a ROS workspace skeleton which contains C++ classes for each ROSMOD component package, specific messages and services, logger and XML parser specific files and build system files as well as deployment specific XML files [7].

The Robot Application Development and Operating Environment (RADOE) framework aims to support easy simulation and execution of robot applications [8] supporting as many robot manufacturers as possible. Narayanamoorthy et al. [8] focus on a part of this framework which supports the creation and visualization of ROS launch files. The ROS launch files are specified by graphical representations of launch file elements. This graphical represenation is then used to generate the desired ROS launch files.

The BRICS Integrated Development Environment (BRIDE) is part of the Best Practice In Robotics (BRICS) project and allows model-driven engineering of robotic applications based on ROS. Its Integrated Development Environment (IDE) is based on Eclipse and supports the graphical modeling of ROS nodes, the generation of code skeletons in C++ or Python and the generation of the necessary tool chain files, like the manifest XML, the dynamic reconfigure files, and the CMakeLists.txt file. It provides ready-to-compile C++ or Python ROS components. For BRIDE, a Domain Specific Language (DSL) has been developed which formalizes the general programming abstractions and concepts of ROS [9]. This DSL is the base for the graphical editor and the code generators.

Besides the ROS based tools, several model-based approaches exist. Nordmann et al. [10] analyzes different DSLs for the robotics domain according to accessibility, documentation, use case (visualization, execution, simulation), evaluation method (simulation, real hardware, executability on different platforms) and the DSL's development process.

Alonso et al. [11] describes V^3CMM, a modelling language consisting of three views (structural, coordination and algorithmic) which are based on UML class models, UML state machines and UML activity diagrams respectively.

Schlegel et al. [12] and Brugali [13] describe model driven engineering toolchains within the robotics domain. SmartSoft and the Proteus project are both based on UML profiles for the specification of a platform independent model and the generation of platform specific code, whereas HyperFlex supports the configuration of component-based systems based on ROS with variation points (e.g., the navigation strategy).

Yoong et al. [14] uses IEC 61499 function blocks (which provide interface and internal behavior) to generate reliable and deterministic C/C++ code for robotics technology components (which only provide an interface) and to design safety-critical robotics applications according to a model-driven approach.

III. Skill and Solution Modeling Tool

The ReApp *Skill and Solution Modeling Tool (SSMT)* supports system integrators in assembling applications by drawing from a repository of already programmed applications. The SSMT hides the complexity of ROS and makes its powerful functionalities more easily accessible for users not familiar with ROS. It allows the graphical composition and configuration of ROS nodes, creating so-called ReApp Skills. For each composed Skill, the SSMT generates a corresponding ROS launch file as well as ROS package information.

A Skill is one of the six types of Apps in ReApp. Each ReApp App is a special type or composition of ROS node(s), as introduced in Table II. Besides its immediate ROS related functionality, the SSMT also allows sharing work via a cloud based repository, the ReApp Store. From within the SSMT, all kinds of ReApp Apps can be uploaded to the store and downloaded again to be re-used in a new composition. To prevent faulty Apps from being uploaded, the SSMT interfaces with suitable verification routines.

The SSMT follows a model-based design, which is a natural match with its task of manipulating and transforming structural elements. Like the entire ReApp engineering workbench, the

[2]http://code.google.com/p/rxdeveloper-ros-pkg

[3]https://github.com/rosmod

TABLE II
TYPES OF REAPP APPS

App	Description	Role
Software-Component	Individual node programmed for a particular function	Component developer
Hardware-Access-Component	Individual node that provides an interface to real hardware	Component developer, hardware vendor
Coordinator	Coordinates the subcomponents of a Skill	System integrator
Dummy	Draft nodes, to be replaced with nodes containing an implementation	System integrator
Skill	Composition of nodes and their relations, can contain dummies	System integrator
Solution	Skill parameterized for one target platform, without dummies	System integrator

SSMT is based on the popular Eclipse platform which covers a variety of development tasks. In particular, the Eclipse ecosystem provides comprehensive support for model-based approaches. The Eclipse Modeling Framework (EMF), Graphical Editing Framework (GEF), Draw2d, Xpand, and Xtend projects in particular are essential in the domain of model driven development and graphical representation.

The SSMT is based on the IDE from the Framework for Distributed Industrial Automation and Control (4DIAC) project,[4] an IEC 61499 implementation. This framework provides graphical editors for modelling distributed control applications and deploying them to various platforms. Since 4DIAC is also based on Eclipse, its functionality can be accessed by the SSMT. The SSMT extends 4DIAC's editors. This has two significant advantages:

- Large parts of the functionality could be reused and adapted, decreasing the implementation effort.
- The SSMT could be conveniently extended to support IEC 61499, allowing to not only compose, but also program SW-/HW-Access-Components with graphical tools.

Since ROS does not provide deterministic realtime support natively, IEC 61499 support in ReApp has another important benefit: It provides developers with the possibility to implement deterministic realtime behaviour according to the IEC 61499 standard within a ReApp application.

A. Integration within the Workbench

The SSMT has a central role within the ReApp workbench. Figure 1 shows the relations between SSMT and the other tools of the ReApp workbench. SSMT uses Web Ontology Language (OWL) files as the exchange format to instantiate the different ReApp App types introduced in Table II, with Software- and Hardware-Access-Components being created by the CMT. It also uses OWL as the exchange format to upload applications to and download them from the ReApp store [15].

SSMT and CMT are integrated tools which communicate inside one workstation (PC/machine), whereas the communication between the ReApp store and the IPDE is over workstation borders. Launch and configuration information of a specific Skill or Solution is generated by the SSMT and transmitted to the IPDE, which contains the executables needed to run the ReApp App.

[4]http://www.eclipse.org/4diac/

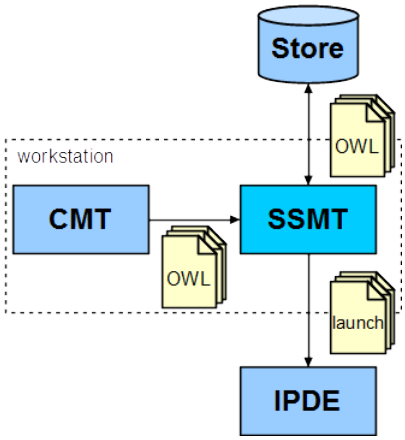

Fig. 1. The Skill and Solution Modeling Tool (SSMT) and its connections to the Component Modeling Tool (CMT), the ReApp Store and the Integration Platform & Development Environment (IPDE)

B. Meta-Model for SSMT

The SSMT uses an internal Ecore model for the graphical visualization of a Skill. This model is also used to generate the content for the IPDE and the OWL based exchange format. With the SSMT, system integrators and end users create M1 level application models. The model hierarchy follows the Meta Object Facility (MOF) [16]. MOF provides Classes and Objects, as well as the ability to navigate from a Class to an Object. Such a layered architecture is able to handle any number of layers or meta-levels. The different ReApp layers are listed in Table III.

TABLE III
REAPP MODEL LAYERS BASED ON META OBJECT FACILITY (MOF)

Layer	Description	Role in ReApp
M3	Ecore meta model: EClass, EReference, EAttribute, EDataType	Defined by EMF
M2	ReApp Skill model: Apps and their connections in terms of M3 level concepts	Ecore meta-model for Skills
M1	A specific Skill, consisting of Apps and their connections	Created with the SSMT
M0	"Program" code which is ready for execution: ROS launch and configuration files, IEC 61499 files	Generated by the SSMT

Each ReApp component in SSMT is expressed by an Ecore meta-model which represents the M2 layer of Table III. Any OWL file loaded by the Skill editor creates an instance of this Ecore meta-model and fills it with the data provided by the OWL file. Each ReApp project stores a Skill file based on this meta-model which contains the information for the graphical representation of a Skill. Since EMF uses an XML Metadata

Interchange (XMI) persistence format each Skill file is an XMI file. Several elements of the Skill meta-model inherit from 4DIAC elements, which means they can use the infrastructure provided by the 4DIAC code base.

Figure 2 illustrates the Ecore meta-model used within the SSMT. The top level element within the meta-model of the SSMT is the `ReAppSkillProject`, which references a `Skill`. Each Skill is a node network and builds the entire functionality of an App. The node network is represented by the `AppNetwork` element, which contains a list of nodes and a list of connections for each interface type. Each Node has a `ReAppInterfaceList` which can contain `TopicDeclarations`, `ActionDeclarations` and `ServiceDeclarations` as well as a list of `Parameter` and `Capability` elements.

There is one sub type of the `Node` element for each node type. `SWComponents` and `HWAccessComponents` are `TypedNodes` since their interfaces can inherit from other components which are collected within a type list. A Skill can also contain instances of other Skills, which are represented by the `SkillNode` node type. To coordinate the different node instances a coordination node can be added, which is represented by the `CoordinatorApp` node type. During the design phase, some node types might be unknown or not possible to clearly identify. For such cases, a "placeholder node" has been defined which can be configured to fill the spot and later be replaced by a fitting node – a local one or from the store. These node types are represented by the `DummyNode` type. Additionally, each node can hold a list of search and verification results. Search results are node types which fulfil the description of the node that the search operation has been applied to. This allows to list any node type which could be used instead. Verification results are produced for Skills which have to satisfy the requirements of their child nodes as well as the connected message types.

C. Skill and Solution Editor

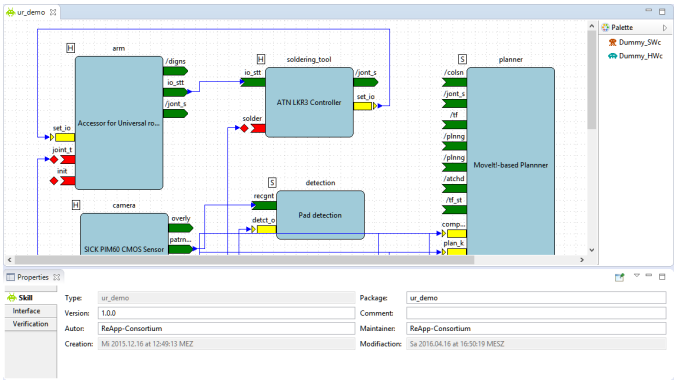

Fig. 3. Graphical editor of the Skill and Solution Modeling Tool (SSMT), with flyout palette (right) and tabbed property sheets (bottom)

The general style of user interaction as well as the programming paradigm can roughly be likened to Simulink[5] or similar tools, where users can place blocks representing certain functions and create logical connections between them. On startup, the tool offers the user the choice between a blank canvas, continuing previous work on a locally stored Skill or opening any Skill from the Store for editing. The main user interface elements are the drawing canvas and the view where the Apps can be chosen and placed on the canvas. This view is populated with search results from the ReApp store and the local workspace. Parameters of placed App instances can be edited within tabbed property sheets. Placeholder nodes can be dragged from the flyout palette, dropped within the canvas of the editor and configured via the tabbed property sheets. Topic/service/action connections between Apps can also be set up within the editor. Feedback is provided if the types of the connected endpoints do not match. Apps and connections can be rearranged by dragging and deleting as needed. If an App instance is deleted, its parameter settings and connections are lost. Figure 3 illustrates how an App could look like.

Since any App can be included in a Skill (and Skills are Apps as well), Skills can be nested. At the level of the enclosing Skill, only the outside interface of a nested Skill is visible. If the developer has not provided graphical layout information (equivalent to "source code"), the nested Skill remains entirely opaque. However, if "source code" is available, the goal is for the SSMT to support easy movement of Apps together with their parameter settings and, connections between the nested and its enclosing Skill in order to better manage complex skills. The created ReApp Skill stores its model information in an XML file that follows the structure within the Ecore model. This file can store all information from the internal model. Vice versa, the entire model can be restored from the XML file. When this XML file is saved, the launch and configuration files are generated automatically in parallel. The launch file is an XML file which contains important information for the deployment of the nodes in the ROS environment. This is achieved by a model-to-text transformation. For this purpose, the Eclipse Xpand framework is used, which allows developers to create template files with rules to generate text based files from internal models.

D. Project Management

The different views which support the Skill and Solution editor in terms of project management are based on the Common Navigator Framework (CNF). The framework provides functionality to visualize the ReApp project structure in a user friendly way. This includes methods like drag-and-drop and interfaces for property views to visualize the App information. The ReApp perspective provides three types of project management views. All of them represent the ReApp Apps that are available for use in the Skill/Solution editor. Each view focuses on a different type of information. The three views are:

- Application Chooser: This project manager view represents all available ReApp Apps on the local machine. Each App is represented as one single object in the view. The underlying structure is much more complex,

[5]www.mathworks.com/products/simulink

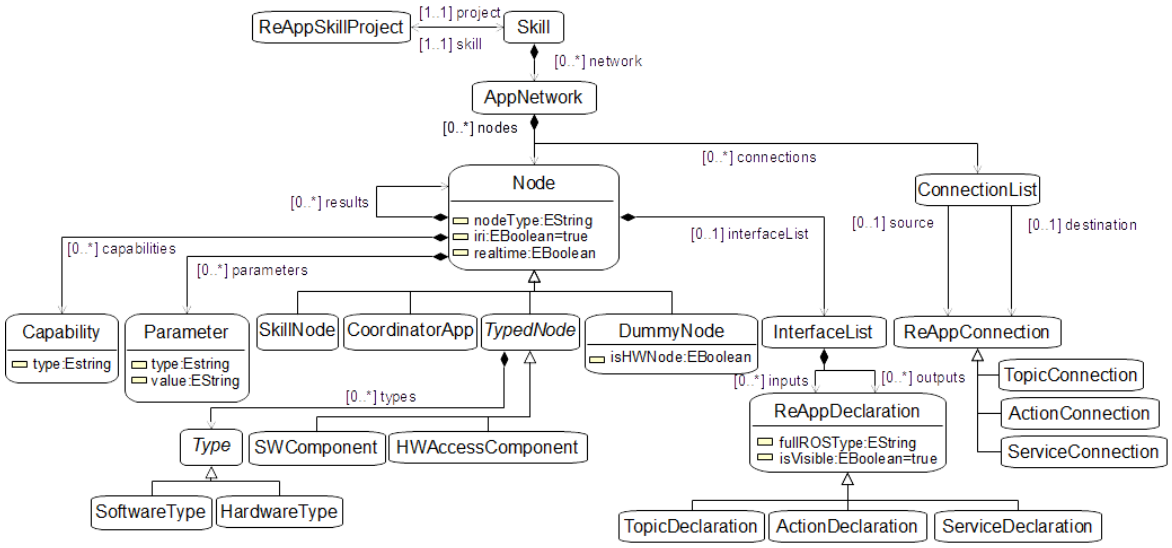

Fig. 2. Ecore meta-model of the Skill and Solution Modeling Tool (SSMT)

because it contains all model, deployment and runtime information. Since this level of detail is not relevant to the Skill Engineer, it is not displayed.

- Store Navigator: As the name suggests, this view contains all ReApp Apps which can be downloaded from the online ReApp Store. The information about the available Apps is obtained via Internationalized Resource Identifier (IRI) queries to the ReApp store. The list of Apps returned by these queries is used for generating a visual representation in the view.
- Navigator: This project manager view lists the model, deployment and runtime information of all available ReApp Apps on the local machine. Like the other two views, this view also supports using the model file of a Skill as well as OWL files of Software- or Hardware-Access-Components within the Skill and Solution editor. All files listed within this view can also be opened and edited with a textual editor.

IV. IEC 61499 Implementation in ReApp

In ReApp, components are programmed with the CCGT. When the component developer has defined the modeling information and semantic annotations for a component, the CCGT generates a code skeleton, where the developer just has to add the logic. The CCGT creates C++ code skeletons. Generally, ROS nodes could be also programmed with Python, Java and some other text based programming languages [17]. No matter which programming language is used, ROS lacks in terms of realtime behaviour which is important for tasks in industrial automation that require strict timing. ReApp addresses this topic by providing the possibility to program applications according to the IEC 61499 [18] standard and by executing them on a corresponding realtime capable runtime environment.

The realtime environment used is the 4DIAC runtime (FORTE). FORTE is capable of being ported to various operating systems and can be also installed on the ReApp integration platform, which runs Ubuntu[6] Linux with the realtime preemption patch [19]. Figure 4 illustrates how ROS based ReApp components can be programmed following the IEC 61499 standard. This is done within the graphical editors of the 4DIAC development environment, which have been integrated into the SSMT. On request, the SSMT automatically creates an the IEC 61499 application (function block network). This application contains interface function blocks that match the ROS component interface and are connected to a basic function block (FB). The behaviour of the component can then be specified by editing the Execution Control Chart (ECC) of this basic FB (Figure 4, bottom) as defined by the IEC 61499 standard. This is especially appropriate for coordination components.

Both ROS and FORTE can be used in parallel on the integration platform, with their interaction supported by a convenient communication infrastructure. ROS communication can generally be divided in three categories [17], which partly use standard communication patterns also provided by FORTE. However, ROS also offers a more complex type of communication which does not fit a common standard pattern. Table IV lists the three communication models in ROS and their counterparts in IEC 61499. The corresponding counterparts are used for connecting the ROS environment and the IEC 61499 environment.

TABLE IV
Comparison between ROS and IEC 61499 communication

ROS	IEC 61499
Topic	Publish/Subscribe FB
Service	Server/Client FB
Action	Custom FB

[6]http://www.ubuntu.com/

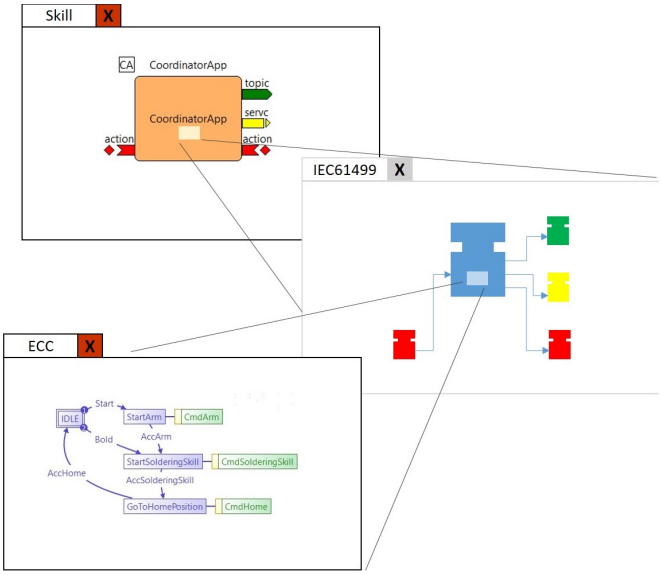

Fig. 4. Editors for the programming of nodes (blue) according to the IEC 61499 standard

A. ROS Topics

ROS Topics represent an unidirectional transaction model to transmit data. It can be imagined as an communication channel where nodes can publish data and other nodes can listen for data. This is one of the simplest communication models which is also implemented in FORTE by the Publish and Subscribe FBs. To make ROS data available for a IEC 61499 FB, the ROS data types have to be mapped to the internal data types of FORTE. This is achieved by extending the layered communication architecture of FORTE [20]. Currently, only the five data types listed in Table V can be mapped between IEC 61499 and ROS.

TABLE V
IEC 61499 VS. ROS DATA TYPES

ROS	IEC 61499
std_msgs/Empty	—
std_msgs/Float64	LREAL
std_msgs/Bool	BOOL
std_msgs/Int32	DINT
std_msgs/String	STRING

In ROS, many data types are available which are not provided natively by IEC 61499. In most cases, these complex data types are compositions of standard datatypes. For example, `geometry_msgs/Vector3.msg`, a data type for vectors, is a composition of three `std_msgs/float64.msg` datatypes. To meet this challenge, the Publish and Subscribe FBs provide 0...n interface elements for data.

B. ROS Services

Client/server communication is implemented by Services in ROS. This messaging paradigm represents a bidirectional communication pattern, where a client node sends a request message to one server node and gets one response from this server. The Service data types are definied as `rossrv` files where data types of request and response messages are stored. FORTE also provides function blocks to achieve this kind of communication. These are the Client and Server FBs. Like the Publish/Subscribe FB, the Client/Server FB also provides 0...n data interfaces for sending and recieving data. As with the Publish/Subscribe FB, this is used for representing complex data types.

C. ROS Actions

With Actions, ROS provides a special communication pattern especially suited for tasks with a long processing time where it is helpful to monitor the current status. Communication between an Action client and an Action server is established by five topics as shown in Figure 5. The client uses two publishers to trigger and stop execution on the server, and the server uses three publishers for sending status messages and results.

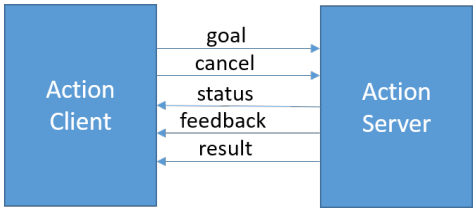

Fig. 5. Schematic representation of ROS Action communication

The data type definitions are stored in `action` files, where types for a goal, feedback and result are set. Since Actions do not implement a basic standard communication model, they could not be implemented with native communication FBs of the IEC 61499 standard. Therefore, creating new custom FBs is inevitable. The current implementation requires creating a new FB for every new Action type. As of now, this also means that the FORTE IEC 61499 runtime must be recompiled before the new FB can be used.

V. DEMONSTRATION EXAMPLE

In the ReApp project, three pilot demonstrators serve to evaluate the ReApp tools. Each of them adresses different tasks in industrial automation. In the following, the use of the SSMT will be exemplified using one of these pilot demonstrators. Figure 6 illustrates the structure of its robot cell.

It consists of a Universal Robots[7] UR10 robot arm with an automatic soldering iron as its end effector. At the end of its arm, the robot also carries a SICK[8] PIM60 vision system. The robot is mounted on top of a workbench which integrates the robot controller as well as the PC running the ReApp integration platform. A protected environment is guaranteed by acrylic glass on three sides. From the front of the workbench, a worker can change the workpiece. A safety system allows robot and human to work collaboratively.

[7]http://www.universal-robots.com/
[8]http://www.sick.com/

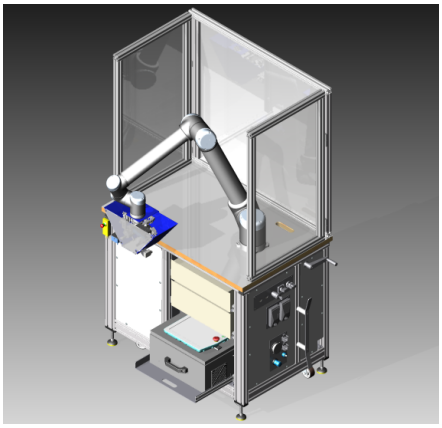

Fig. 6. Structure of the ReApp soldering process demonstrator

In this case, the workpiece is a printed circuit board with LEDs on it, where a solder joint has to be applied at a predefined position. The vision system detects and locates the workpiece and the soldering pads. The robot arm then brings the soldering tool into the right position and the soldering process starts. The procedure comprises the following tasks:

- Moving the multi-axis robot arm
- Controlling the soldering tool (via RS-232)
- Path planning
- Communicating with the vision system
- Receiving information from the safety system

Figure 8 shows a simplified overview of the entire demonstrator skill. The skill is composed of a coordinator and four subskills:

- HMI Skill: The skill contains a SW-Component and a HW-Access-Component for interacting with the user. It provides the user with the necessary interface to start the solution and observe its current state.
- Object Recognition Skill: This skill provides interaction with the vision system. To do its work, this skill requires additional apps for obtaining camera images and for processing these images.
- Presoldering Skill: Checks all preconditions, in particular the status of the safety components. Also, the apps for path planning are triggered when this skill is executed to generate a trajectory for the robot.
- Soldering Skill: This skill performs the actual soldering process by triggering the end effector. It is executed when the robot arm has reached the right position.

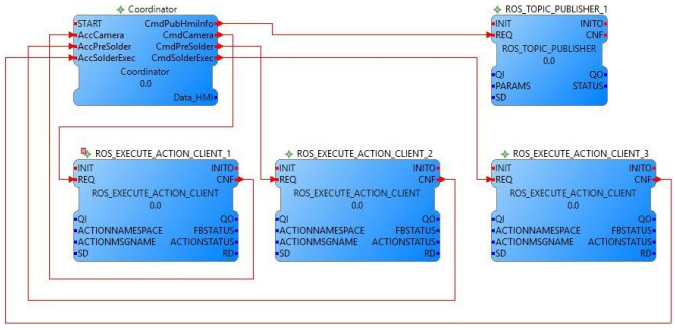

Fig. 7. IEC 61499 Application (programmed with the ECC Editor) within the Coordinator App

The coordinator node is a software component controlling the sequential execution of the individual subskills. This coordinator node can be programmed in various ways, in particular using the ROS SMACH (state machine) Python libraries or with the ECC editor as illustrated in Figure 4. Figure 7 shows an example how an IEC 61499 application network running on FORTE could look like. To deploy the IEC 61499 application on FORTE in the ReApp environment, a FORTE boot file is generated for the coordinator node.

VI. Conclusion

Today, robot programming requires experts specialized in a specific manufacturer's technology. The reuse of robot programs, even partial, is usually impossible due to different robot languages being employed by robot vendors. The ReApp project aims to provides reusable robot applications. As a central part of the ReApp workbench, the Skill and Solution Modeling Tool (SSMT) offers the possibility to create and reuse robot applications based on ROS. The SSMT provides a graphical editor and supporting views to compose a robot application out of different types of nodes. The nodes being composed can be provided locally or can be obtained from the ReApp store. Each node can be configured and connected with other nodes.

This robot application model is used to generate deployment information which can be transmitted to the desired deployment platform. Besides creating the composition and the deployment information, the SSMT is also designed to verify the node network, exchange nodes with the ReApp store and search for nodes with specific capabilities or interfaces. Furthermore, the SSMT supports specifiying a node's behaviour according to the IEC 61499 standard, providing realtime capabilities for ROS based robot applications.

The SSMT has been used to program the ReApp pilot demonstrators. Within this work, its application to an automated soldering process has been shown. For future versions of the SSMT, improvements and extensions for the support of the verification of Skills, node search and IEC 61499 integration are considered. In particular, the Action communication FBs should be extended to provide a generic Action type, supporting the deployment of FBs with new Action types during runtime.

Acknowledgment

The work leading to the presented engineering tool has received funding from the *Bundesministerium für Wirtschaft und Energie (BMWi)* under the grant agreement 01MA13001A. It has been developed within the ReApp project, which is part of the technology program *AUTONOMIK für Industrie 4.0*. Further information about the ReApp project is available at: http://www.reapp-projekt.de/

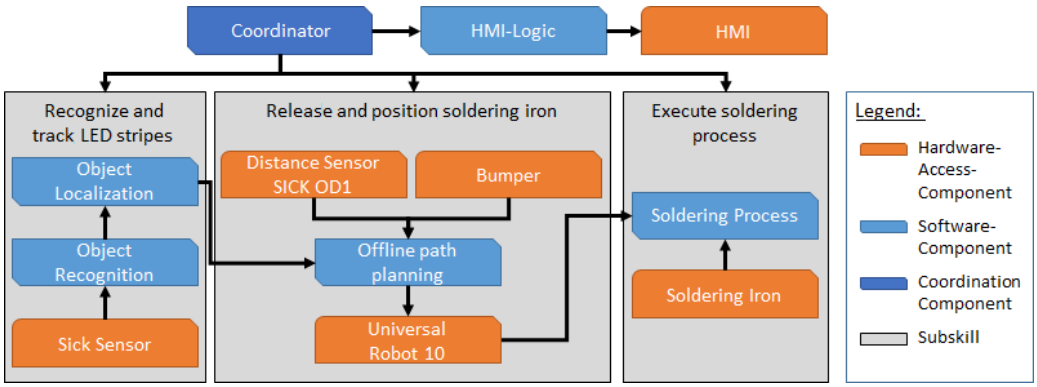

Fig. 8. Skill design for the ReApp demonstrator: Recognising and soldering LED stripes

References

[1] S. Zander, G. Heppner, G. Neugschwandtner, R. Awad, M.Essinger, and N. Ahmed. "A Model-Driven Engineering Approach for ROS using Ontological Semantics". In: *International Workshop on Domain-Specific Languages and models for Robotic systems (DSLRob)*. 2015.

[2] T. Dietz, U. Schneider, M. Barho, S. Oberer-Treitz, M. Drust, R. Hollmann, and M. Hägele. "Programming System for Efficient Use of Industrial Robots for Deburring in SME Environments". In: *German Conference on Robotics*. 2012.

[3] M. Quigley, K. Conley, B. P. 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*. 2009.

[4] A. Koubaa, ed. *Robot Operating System (ROS) – The Complete Reference*. Springer International Publishing, 2016. ISBN: 978-3-319-26052-5.

[5] R. Awad, U. Reiser, N. Ahmed, and S. Zander. "Flexiblere Automatisierung: Wiederverwendbare Roboter-Applikation". In: *Digital Engineering Magazin* 18.2 (2015), pp. 19–21.

[6] F. Müllers, D. Holz, and S. Behnke. "rxDeveloper: GUI-Aided Software Development in ROS". In: *Intl. Conference on Robotics and Automation (ICRA)*. 2012.

[7] P. S. Kumar, W. Emfinger, A. Kulkarni, G. Karsai, D. Watkins, B. Gasser, C. Ridgewell, and A. Anilkumar. "ROSMOD: A Toolsuite for Modeling, Generating, Deploying, and Managing Distributed Real-time Component-based Software using ROS". In: *Intl. Symposium on Rapid System Prototyping (RSP)*. 2015.

[8] A. Narayanamoorthy, R. Li, and Z. Huang. "Creating ROS Launch Files Using a Visual Programming Interface". In: *International Conference on Cybernetics and Intelligent Systems (CIS) and on Robotics, Automation and Mechatronics (RAM)*. 2015.

[9] A. Bubeck, F. Weisshardt, and A. Verl. "BRIDE – A Toolchain for Framework-Independent Development of Industrial Service Robot Applications". In: *International Symposium on Robotics (ISR)*. 2014.

[10] A. Nordmann, N. Hochgeschwender, and S. Wrede. "Simulation, Modeling, and Programming for Autonomous Robots". In: vol. 8810. Lecture Notes in Computer Science. Springer Link, 2014. Chap. A Survey on Domain-Specific Languages in Robotics.

[11] D. Alonso, C. Vicente-Chicote, F. Ortiz, J. Pastor, and B. Álvarez. "V3CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development". In: *Journal of Software Engineering for Robotics (JOSER)* 1.1 (2010).

[12] C. Schlegel, T. Hassler, A. Lotz, and A. Steck. "Robotic software systems: From code-driven to model-driven designs". In: *International Conference on Advanced Robotics (ICAR)*. 2009.

[13] D. Brugali. "Model-Driven Software Engineering in Robotics: Models Are Designed to Use the Relevant Things, Thereby Reducing the Complexity and Cost in the Field of Robotics". In: *IEEE Robotics and Automation Magazine* 22.3 (2015).

[14] L. H. Yoong, Z. E. Bhatti, and P. S. Roop. "Simulation, Modeling, and Programming for Autonomous Robots". In: vol. 7628. Lecture Notes in Computer Science. Springer Link, 2012. Chap. Combining IEC 61499 Model-Based Design with Component-Based Architecture for Robotics.

[15] A. Bastinos, P. Haase, G. Heppner, S. Zander, and N. Ahmed. "ReApp Store – A Semantic AppStore for Applications in the Robotics Domain". In: *International Semantic Web Conference – Industry Track*. 2014.

[16] Object Management Group. *OMG Meta Object Facility (MOF) Core Specification*. 2015.

[17] J. M. O'Kane. *A Gentle Introduction to ROS*. Independently published, 2013. ISBN: 978-14-92143-23-9.

[18] A. Zoitl and R. Lewis, eds. *Modelling Control Systems Using IEC61499*. IET, 2014. ISBN: 978-1-84919-760-1.

[19] S.-T. Dietrich and D. Walker. "The Evolution of Real-Time Linux". In: *7th RTL Workshop*. 2005.

[20] 4Diac Consortium. *Communication Architecture*. 2014. URL: http://www.eclipse.org/4diac/documentation/html/development/forte_communicationArchitecture.html.