A System Integration Approach for Service-Oriented Robotics

Gerhard Ebenhofer, Harald Bauer, Matthias Plasch, Sebastian Zambal, Sharath Chandra Akkaladevi, Andreas Pichler
PROFACTOR GmbH
Im Stadtgut A2
A-4407 Steyr-Gleink, Austria
{gerhard.ebenhofer, harald.bauer, matthias.plasch, sebastian.zambal, sharath.akkaladevi, andreas.pichler}@profactor.at

Abstract

The robotic system integrator’s dream of (re)using existing software components and benefiting from a common software framework for service orchestration becomes more and more evident. Current initiatives tend to promote the own framework or class library and do not put a lot of effort into illustrating how to integrate existing functionality from other frameworks.

Within joint projects between industry and scientific partners, we typically face the challenge of having several development teams using different techniques, class libraries and open source frameworks.

In this context, we want to present our development approach using 4DIAC development tools and explicitly highlight useful extensions added for a smooth integration of heterogeneous components of various frameworks which is shown by examples.

1. Introduction

Motivation for this paper emerged from the following two project examples:

LOCOBOT (“Toolkit for Building Low Cost Robot Co-Workers in Assembly Lines”) is a project funded by the European Commission. It incorporates a flexible robotic assistant platform to enhance manual production processes and to improve ergonomics by shifting problematic sub tasks from the human worker to the established robot co-worker. In this context, it favors using low cost components while providing a safe, tailor made robot assistant system inclusive the engineering tools required for its setup out of a modular robot tool kit [1][2].

DARWIN (Dexterous Assembler Robot Working with Embodied Intelligence), also funded by the European Commission [3], aims to develop an acting, learning and reasoning assembler robot that will ultimately be capable of assembling complex objects from its constituent parts. The Darwin system is designed as a compilation of individual modules covering different functionalities such as I/O, vision, motor control, reasoning system and others. With this modular architecture Darwin aims at supporting different robotic hardware platforms such as a service robot and an industrial robot.

We describe a development approach for such projects, based on the 4DIAC engineering tool [4], as well as the extensions developed to support the integration of different system components. By using illustrative examples, we explain the steps necessary for component integration.

Section 2 provides an overview of related development frameworks in the field of robotic system integration. Moreover, we will introduce the 4DIAC framework for distributed automation and control. Section 3 introduces our development approach for service-oriented robotics. Section 4 highlights the extensions developed to broaden the integration of different frameworks’ components. Section 5 gives insight for engineers how to get components ready for integration. How to put things together is briefly illustrated in Section 6 by an application example. Finally, Section 7 mentions our lessons learned and draws the relevant conclusions.

2. Related work

Various approaches targeting the integration of services based on different platforms into one system, build on Service-Oriented Architecture (SOA) concepts. SOA developed components are characterized by functional availability and are designed to push reuse and composition of existing functionality to enhanced services [5]. The major advantages of SOA are the treatment of services as independent resources and their accessibility without the need of platform specific knowledge [6]. Within the field of robotics, Kononchuk et. al. [7] developed an interconnection protocol, RoboCop, to combine robotic engineering toolchains and programs to a scalable control system structure. Frameworks building on SOA driven development, for integrating devices, including their sensors and actuators, into a robotic
system are proposed by [8] and [6]. In both cases devices are treated as ubiquitous services. Yang and Lee [6] additionally use a hierarchic approach to express complex services (components) as compositions of simple services. A comparison of implemented SOA approaches, which are adapted to robotic work-cell programming, is performed by Veiga et. al. [9]. Hereby, emphasis is put on methods for service description and service orchestration. The analyses proved that SOA driven development of service components, combined with graphical development environments, reduces the integration effort significantly [9].

In the domain of robotics, Orocos is one of the most mature and established open source software frameworks [10]. More recent work extends this data flow oriented framework with Java technology resulting in JOrocos - influenced by Service Component Architecture (SCA) concepts [11] and additionally bridging the gap between component-based approaches (e.g. using real time control functionality inside functional components) and modern information system technologies (e.g. using state-of-the-art Java toolkits for graphical user interfaces) [12].

YARP (Yet another robotic platform) [13] is an open source software library that enables module-based software development for robotic platforms. YARP modules run as separate executables and communicate over an extensible set of communication protocols such as TCP, UDP, MPI, etc. The communication patterns provided by YARP include publish-subscribe as well as client-server architectures. YARP was started in 2002 to support software development for humanoid robots such as the iCub [14].

To simplify large-scale software integration, ROS (Robot Operating System) [15] was developed in 2007, not designed for a specific target application (as a wide variety of frameworks before) but designed to support the philosophy of modular, tools-based software development [16]. A heavily growing ROS community proves the success of ROS' design principles, although it focuses mostly on Linux platforms and hardly provides information concerning the development approach.

In this context, the project BRICS (Best Practices in Robotics) proposed initial guidelines for component design resulting in better integrate-able components [17] according to the Model-Driven Engineering (MDE) approach. Recently, a reuse-oriented development process for component-based robotic systems [18] was published following the developed design philosophy using the BRICS Integrated Development Environment (BRIDE) [17], which is a prototypical software tool chain. It includes a graphical user interface to support their approach for robotic system engineering.

Investigating the domain of Industrial Automation, where in particular embedded devices have been pushing the technology more and more towards distributed control systems and also towards more service oriented architectures [19], one comes across the established 4DIAC initiative based on IEC 61499 [20], which proposes an open source framework for distributed industrial automation and control since 2007 [21]. The 4DIAC framework consists of an IEC 61499 compliant automation and control environment for distributed systems. Currently, a modular runtime environment, an integrated engineering environment, a library of function blocks (FBs) and a set of example projects are available.

The runtime environment FORTE supports the operating systems Windows, any posix compliant systems, NET+OS 7, eCos and VxWorks. Moreover, several evaluation boards (e.g. Digi Connect ME [22], Pandaboard [23]) and PLC systems (e.g. Bachmann M1 [24], Siemens EC31 with S7 IO Modules [25]) are supported.

The 4DIAC system has been applied in different domains. Examples on this are the control of an Inverted Pendulum where a simple PID controller was dynamically exchanged by a more complex state-space controller [26]. The Austrian Institute of Technology presented an example application of the usage of 4DIAC for Smart Grids Applications where IEC 61850 was integrated into IEC 61499 FBs [27].

Our development approach, described in the following section builds on the standard IEC 61499, rather than its predecessor IEC 61131. This is because the programming paradigm of IEC 61499 is more suitable for distributing control intelligence to autonomous devices, leading away from centralized control structures. Moreover, the event-driven execution structure of FBs, eases the development of applications to coordinate functional independent components, as required by our integration methodology.

3. Development approach

One of the main goals within the Locobot project, is the establishment of a robot tool kit consisting of independent hardware and software components usable on demand for arbitrary future robotic systems. Such systems require the integration of different functional areas like collision free platform navigation, object pose recognition for scene interpretation and manipulation planning for bin picking tasks. All these components are based on different technologies and frameworks. This usually leads to increased complexity during component integration. Therefore, we propose a service-oriented, modular approach that helps to reduce complexity during integration. This is
realized through a platform independent description of service component interaction interfaces.

Very similar as proposed in [18], we harmonized those interfaces (description in Section 4) to get integrate-able service components for 4DIAC thus investing in software development for reuse to come towards software development with reuse for the subsequent system integration stages.

Having defined stable interfaces, the integration of the components can be started in parallel to the development of the components itself. In this context, we used software mock objects [28] also enabling early bandwidth, coordination and communication tests. The results of early tests were used to enable an iterative refinement of component interfaces.

From a high-level point of view, a service component can be considered having a number of service functions which can be invoked. A service function is described by a number of inputs and outputs parameters as depicted to Figure 1.

![Figure 1: Abstraction of service components and services functions](image)

The functionality of a service component needs to be shifted into an IEC 61499 compliant FB, in order to be usable within the 4DIAC development environment. The FB acts as an interface to the component and enables its integration into a control application.

This abstraction of a component shown in Figure 1 is the basis for deriving a FB template, as depicted in Figure 2, for component integration.

![Figure 2: Function Block template for components](image)

Service function invocations are represented by event inputs and their associated parameters (data inputs). After a service function has been processed, the corresponding confirmation event is triggered and the output data is updated. Additional data inputs and outputs are used to set a maximum component response time, and to report status and error states.

In order to realize a desired system behavior a means of supervisory control is necessary to coordinate the system components. Having the interface descriptions of the components available, a control application can be modeled using a platform independent graphical workflow model. In a final step this workflow model can be transformed into a platform specific supervisory control application based on IEC 61499. This is accomplished through a code generation methodology described in [29]. Additionally, this approach supports the generation of control code which can be combined with a 3D simulation environment. This enables early validation of the workflow model in order to minimize design errors before running the application on the real target system.

### 4. Extensions for 4DIAC/FORTE to support different components

For integration and communication with external components the 4DIAC framework provides a set of extensions. These extensions can be grouped into communication extensions, a PLC IO access module and extensions of component functionality through integration of software libraries.

#### 4.1. Fieldbus protocols

In order to facilitate distributed control structures, devices need to communicate independently via fieldbus networks. Fieldbuses and their corresponding communication protocols need to be chosen based on their applicability for certain industrial applications. Although Ethernet and the TCP/IP protocol are more and more accepted in industry the same mistake as with fieldbus technologies has been made. Instead of developing one standard there are currently more than 10 different Industrial Ethernet protocols available.

To stay compliant with all the different protocols and fieldbuses the FORTE provides a flexible network interface [30] which can be extended with new protocols. In addition to the ASN.1 communication protocol, as described in the Compliance Profile for Feasibility Demonstration [31], the FORTE currently supports the fieldbus and communication protocols ModbusTCP [32], Ethernet Powerlink [33][34], CAN [35] and OPC-DA [36].

#### 4.2. ZeroMQ

Dealing with large amount of data (e.g. data from 3D sensors) which needs to be transferred in distributed systems requires efficient messaging
concepts. Therefore, we realized support for ZeroMQ [37] which is an asynchronous messaging library and is suitable to serve as scalable messaging protocol in distributed system architectures. The protocol is especially designed to support parallel computing without restricting the network topology, having very low communication overhead.

4.3. ROS communication layer

In order to reuse available ROS components we developed a ROS communication layer which enables the seamless integration of ROS Nodes into IEC 61499 applications. The basic representation of data in ROS is a message. It is used as communication entity in publish/subscribe connections as well as encapsulation of parameters and return value in a service request. A message is a structure of data types which can hold basic data types and other messages as members. This is the same principle as in IEC 61499 with structures, which are also able to hold sub-structures. The equivalence of both principles enables communication between ROS and IEC 61499.

The representation of services and topics in IEC 61499 can be handled by generic communication FBs. Services are represented using Server and Client FBs. Topic messaging is realized by using Publish and Subscribe FBs. During the initialization of the communication FB, the ROS Communication Layer establishes a connection to the topic or service, based on the specified ID parameter.

After successful connection establishment the algorithm can be invoked. Figure 3 shows the message flow for the topic oriented approach. The PUBLISH FB sends the required input data (SD_1 port) to the topic AlgorithmInput and is received by the ROS Node Algorithm.

4.4. YARP communication layer

In YARP special port objects deliver messages to any number of other ports. Such ports can be distributed across any number of machines and processes. For the communication between the ports a set of different communication protocols can be used. Similar to the ROS communication layer we integrated the YARP framework in 4DIAC. Based on the configuration of the ID parameter of the generic communication function blocks, the specified data values (at ports SD_n, RD_n) are encoded accordingly and sent to the given port. Additionally, optional configuration parameter can be added. For example the YARP library functions enable the configuration of a BufferedPort, which can keep all messages sent to it between two port read() commands. A common port configuration only stores the latest message.

4.5. Generic inputs and outputs

In many cases, functional components are connected to a PLC controller in order to provide the necessary infrastructure for time critical operation of sensors and actuators. Due to the large variety of PLC controller types, differently implemented FBs would be necessary, in order to provide access to I/O ports for every PLC type which should be supported.

To overcome this limitation and to provide great flexibility, a concept based on generic function blocks has been developed in [38] to provide a platform independent interface to PLC I/O. The main idea behind is to use a wrapper FB (Figure 4). Based on the parameterization and the current mapping to a hardware device, the platform specific PLC I/O FB is automatically chosen.
5. Get components ready for system integration

The implementation of a system component itself may have different origins concerning its base framework or software libraries. Section 3 gives an overview how component functionality is put into FBs to enable application modeling and code generation based on IEC 61499. This section shows how component FBs are implemented using the 4DIAC tool chain.

5.1. Design of a component FB

A component FB can be designed in the function block editor of the 4DIAC engineering environment. The interface of the FB should be designed based on the template described in Section 3. The FB type itself may be Basic, Composite or Service Interface.

Composite FBs have the advantage of enabling component integration using an internal FB network which additionally can include error handling functionalities. Moreover, the integration of the component functionality can be split up into a collection of FBs. The aggregation of FBs to a FB network enables the whole component functionality.

Components which depend on external software libraries can be implemented as Basic or Service Interface FBs. A detailed description how to integrate external libraries is given in section 5.3.

5.2. Generating FB source code

After the FB has been designed it needs to be integrated into the FORTE. Composite FBs can be directly loaded during runtime startup by parsing the FB type definition. Other FB types need to be exported into C++ code first, using the 4DIAC Type exporter. In cases of components which depend on external code or libraries, the environment only provides a skeleton code structure which can be extended to realize the desired functionality. Figure 5 depicts the generation of FB source code which can be enhanced using external library functions.

5.3. Integration of library based components

As the FB runtime environment FORTE is based on C++, external library elements providing component functionality can be integrated into a FBs source code. As the FORTE build system is based on CMake it is necessary to specify the required libraries and include files. This section gives an overview which steps are necessary to include an external library into the 4DIAC environment. As an example, we show how to include a computer vision library for 3D scene interpretation named CANDELOR [39].

The FORTE can be extended by functional modules. A module can contain a collection of FB types or an extension of core functionality, located in the modules subfolder of the FORTE source code. Every module folder contains a CMake configuration file which specifies the required information like include directories, link directories and libraries.

Listing 1: CMake configuration file for CANDELOR module

Using the template as described in Section 3 the FB type RANGO as depicted in Figure 6 is designed using the 4DIAC function block type editor. The FB should invoke the CANDELOR object pose recognition algorithm (doOR). As a second function we use the show result (showResult) functionality to visualize the recognition result.
To make sure that the show result can only be called after the doOR algorithm we implemented the FB as a Basic FB type with an internal state machine as presented in Figure 7.

Using the export functionality the skeleton of the FB source code is generated. The implementation of the methods

```c
void FORTE_RANGO::alg_DO_OR (void)
void FORTE_RANGO::alg_SHOW_RESULT (void)
```

is based on the examples in the API documentation [40].

### 5.4. Composite FB using communication protocol

A Composite FB using communication FBs can be used to invoke service functions of external service components which provide a network interface. Additionally, the composite network can be extended with FBs for data pre and post processing as well as for status evaluation and error handling. Figure 8 shows a Composite FB communicating with an external ROS based component. The ROSCOM FB itself is implemented as composite FB to realize a communication structure as explained in Section 4.3.

### 6. Example application

For the evaluation of the proposed development approach we used an example as described in the following. The overall task is to grasp defined objects in an unstructured environment with a robotic arm and to transport them with a mobile platform to a specific target position. A 3D sensor, which is controlled using a ROS package, is used to record a point cloud of the scene.

This requires the 3D sensor component to be integrated as described in Section 5.4. To enable object recognition and planning of collision free grasping paths, the CANDELOR system as well as a manipulation planner is integrated as external library according to Section 5.2. Using the YARP communication layer, the collision free manipulation path is sent to the Robot Control component which handles the communication with the robot. In our example, the communication between the Robot Control component and the robot is implemented using Modbus/TCP. A standard PLC system is used to open and close the gripper. This requires the activation of PLC I/O using generic PLC FBs as described in Section 4.4. Using these FBs enables changes of PLC systems without requiring changes in the control application.

The grasped part is placed on the mobile platform which is controlled by several ROS nodes. The control application which coordinates the components is generated using the 4DIAC configuration tools as described in Section 3. The components and their relations are depicted in Figure 9.

### 7. Lessons learned and Conclusion

This section provides a short overview on the experiences we gained during the evolution of the presented development approach.

Every control application builds on component FBs which encapsulate component functionality. Hence, to ensure correct behavior and stability of a control application, thorough tests and revision cycles are necessary to be performed for every component FB. In this context, early tests of component FBs can be performed using software mock objects instead of the “real” system components. Unexpected behavior of complex control applications, resulting from faulty components or component FBs, is always hard to resolve.
In order to reduce the complexity in error and exception handling in control applications, every component should provide its own, internal error handling routines. Errors which cannot be resolved by the component itself are reported to the control application through simple error codes. In general, positive and negative status values are forwarded to the status data outputs of the component FB (see Figure 2).

A high-level view on components and their capabilities like described in Section 3 is necessary to derive an interface for component integration, which is open and generic. Moreover, component abstraction forms the fundament for simplified application modeling based on workflow models.

With the experiments gained in the Locobot project we could show the suitability of IEC 61499 – Function blocks and 4DIAC Integrated Development Environment (4DIAC-IDE) for the integration of complex, modular systems. The high flexibility of the 4DIAC runtime environment (FORTE) enables the fast integration of components, as well as communication layers to support communication via fieldbuses and framework borders.

Future work topics include the improved integration of simulation software into the development process. Early simulation is an important step to the verification of control application models prior to commissioning of the real system.

Acknowledgment

This work has been supported by "LOCOBOT - ToolKit for Building Low Cost Robot Co-Workers in Assembly Lines" (FP7 NMP under Grant No. 260101) and by “DARWIN - Dexterous Assembler Robot Working with Embodied Intelligence” (FP7 ICT under Grant No. 270138) from the European Commission under the 7th Framework Programme (FP7).

References
