DEVELOPMENT OF A CONTROL SOFTWARE FOR A PLANETARY EXPLORATION ROBOT WITH ESROCOS

Malte Wirkus[*], **Moritz Schilling**[*], **and Benjamin Kisliuk**[**]
[*] *Robotics Innovation Center, DFKI, Germany*
[**] *Forschungsgruppe Planbasierte Robotersteuerung, DFKI, Germany*

ABSTRACT

ESROCOS is a H2020 research project and the name of an open-source software system for the development of robotics software, which was created during the project. It provides a model-driving development workflow targeting especially the space robotics domain by focusing on the creation of applications with stringent Reliability, Availability, Maintainability and Safety (RAMS) requirements. ESROCOS comprises of different tools with TASTE as the core technology. To validate the project results, a one-week test campaign has been carried out in September 2018. During the test campaign, a control application for a planetary exploration rover was integrated on a space rover prototype. In this paper, we explain the control application and the features of ESROCOS relevant for its realization. From the experience of the validation test, we discuss the suitability and usability of the ESROCOS tools for the given task with the motivation to provide ideas for further improvement.

1. INTRODUCTION

While robotic systems already proved to be an adequate tool for supporting humanity in the exploration of space, additional functionalities and more autonomy are required to fully leverage the potential of robotic systems [2]. However, additional capabilities for robotic systems come at the price of increased software complexity. Especially in the case of the space domain, high requirements for Reliability, Availability and Safety (RAMS) are imposed on hard- and software, resulting in high efforts and costs for developing and validating software. The research oriented robotics community faces the increasing complexity with an open source approach: Robot software development frameworks managed to establish a large collection of tools and re-usable software components for robots in the academic world. These software components are usually supposed to mature over time due to massive testing instead of a verification process as in space applications. In addition, parts of the frameworks internal code already do not apply to the RAMS requirements for space, such that for space robotics a similar collection of re-usable software components is missing.

The European project European Space Robotics Control and Operating System (ESROCOS)[1] [1] aims to provide an open-source framework to assist in the development of flight software for space robots with space-grade RAMS properties. The resulting ESROCOS software system [6] comprises of different mature and novel tools with the TASTE software [7] at its core. TASTE allows the specification and generation of correct-by-construction software for heterogeneous embedded system. ESROCOS aims to establish a standard workflow and infrastructure for robot software development in the European space industry and – in the longer run – a pool of state-of-the-art software solutions for common algorithms, driver and tools needed in space robotic software applications.

To validate the project results, a one-week test campaign has been carried out in September 2018. In this test campaign, a control application for a planetary exploration rover was integrated on the robotic system *Bridget* (cf. figure 2). We give details about the test campaign in section 4 after introducing different robotics framework and ESROCOS in section 2 and 3. With the intention to identify demand for certain functionalities and provide ideas for further improvement, a subjective evaluation of the suitability and usability of the ESROCOS tools that have been used during the test campaign is given in section 5. With section 6 we conclude the paper with a some closing remarks.

2. STATE OF THE ART

In the robotics community, the use of software component systems has gained a lot of interest in recent years. The idea of having standardized interfaces

[1]ESROCOS Website: `http://www.h2020-esrocos.eu`

and data types to enable the development of *compatible* off-the-shelf software components is not new and was first introduced back in 1968 [5]. Nowadays, there exists a wide variety of frameworks to choose from. To just name a few, there are Orocos-RTT, ROS or ROCK, which will be introduced in the following paragraphs.

Since its release in 2009 Robot Operating System (ROS) [8] managed to gather a large user and developer community and became the presently most prominent framework for robot software development. By now, it provides a plentiful repository of ready-to-use software components and a rich development environment. ROS provides a publisher/subscriber middleware, to which so-called *nodes* publish their topics and/or subscribe to topics they expect to retrieve data from. The topics of a node are advertised by the nodes at runtime at a central name service, but otherwise are hidden in the implementation code. ROS provides a mechanism for renaming topic of a node without changing its code to allow constructing composite software systems from components, which were developed individually of each other. ROS does not provide a model description of node interfaces or behavior. The knowledge about the provided topics must be retrieved from documentation or by running them and using framework tools that retrieve a list of advertised topics. This practice is unfavorable for the space domain, since it impedes the possibility of constructing, analyzing and validating of component networks offline.

The software framework Robot Construction Kit (ROCK) [4] improves on this situation by providing a model-based component development workflow. Each component has a model assigned, which describes typed input and output ports, as well as properties, functions and different runtime states of the component. Consistency between the model and the actual implementation of a component is ensured by using the component model already in the development process for the auto-generation of framework-specific code. The underlying runtime and communication infrastructure in ROCK is the Orocos Real-Time Toolkit (RTT), a software system for component based real-time applications with support for standard and real-time Linux distributions and different communication schemes such as direct memory access, POSIX message queues or CORBA. Like ROS, ROCK provides support for letting software components communicate via network, but does not provide means for defining the systems network topology as part of the deployment process.

The development of The Assert Set of Tools for Engineering (TASTE) [7] dates back to 2008 and its development is steadily driven forwards under the coordination of ESA since then. It is designed as a development environment for distributed embedded real-time systems that facilitates established and mature technologies such as Architecture Analysis & Design Language (AADL), Abstract Syntax Notation One (ASN.1) and Specification and Description Language (SDL). Originally designed for satellite systems, an initiative called *SARGON* started in 2016 to use TASTE for the development of robotic applications for the first time. In contrast to the other frameworks mentioned before, TASTE provides a hardware model of the robotic system to account for glue code generation, compilation and validation. The hardware model allows specifying multiple execution units and their communication (e.g. buses) between each other. Combined with the assignment of the software components to their execution units, different checks, e.g. for worst-case execution time or schedulability are possible.

In the next section, we describe TASTE and the extensions that have been added with the ESROCOS framework in more detail.

3. ESROCOS AND TASTE

TASTE comprises of tools for modeling, analyzing and compiling applications composed from multiple software components. A key feature of TASTE is the support of interactions of components on distributed and heterogeneous hardware platforms. Hence, control over data type modeling and serialization and cross compiling capabilities in the build process are provided for a selection of hardware platforms relevant for the space domain. For modeling data types, the ASN.1 format is used. Component networks and hardware are modeled with a visual editor for AADL models. For the communication between the components, TASTE uses the middleware PolyORB-Hi-C.

TASTE used to follow a bottom-up procedure allowing modeling and implementing data types and components from scratch, with no means for software re-use. With ESROCOS, component re-use capabilities were added by extending TASTE with features to im- and export component interfaces and source code. To allow for compatibility of software components that are developed independent from each other, a set of common data types for robotics applications (*base types*) were defined and are part of the ESROCOS framework. To support developers with distributing their software components, a mechanism for registering new software packages in the ESROCOS system and their automated retrieval from public source code repositories was added.

Much of the data robots produce and process is framed by the geometric coordinate space it is expressed in. Thus, to combine data samples they have to be transformed into a mutual coordinate system. ESROCOS provides the *transformer* library to configure a graph with named geometric frames as nodes and frame transformations generated by user code as edges. Transforms between arbitrary frames within the specified graph can be queried and the Transformer resolves the geometric operations required for calculating the requested transform. Similar functionality is provided within ROCK with the *transformer* and in ROS with the *tf* mechanism. The implementation used in ESROCOS is free of dynamic memory allocations to account for space software development standards.

Much of the capabilities of intelligent systems derives from the combination and fusion of the available data. However as sensors and sensing methods vary quite a lot the sensory data varies in size, frequency and effort to process. The more distributed a robotic system is, the more important it gets to be able to process the produced data in an orderly manner to get meaningful results. The *stream aligner* supports developers with the processing of multiple asynchronous data streams by buffering and aligning the data streams. Matching tuples of samples for a given time point can thus be retrieved from the stream aligner.

Data Logging is a crucial capability when developing robotic applications that rely on sensor data processing for autonomous tasks. Not only do log files prove invaluable during the engineering and debugging process of the software, but they also allow for the documentation of experiments. A software library for performant data logging is provided with ESROCOS.

A feature unique to TASTE compared to other component development frameworks is the possibility to analyze the real-time behavior and resource utilization of the software. ESROCOS complements these capabilities with including the Behavior, Interaction, Priority (BIP) tool [3]. BIP offers additional possibilities to analyze the software and verify properties at a behavioral level, and can be used to generate correct-by-construction software components by modeling their behavior as automata and translating them to executable code.

In the test application presented in the next section, all of the above features are put to test in order to identify strengths and weaknesses of the ESROCOS system. It is to note, that ESROCOS provides more interesting features for software development for space robots. The interested reader is invited to read [6] for a complete overview of the ESROCOS system.

4. VALIDATION TEST SCENARIO AND APPLICATION

To assess the functionality of the ESROCOS system, the selected test scenario and application should be suitable to test specific features of the framework rather than solving a particular robotics tasks. For the validation test campaign, we implemented a teleoperation application with additional functionality for data processing, actuator control, visualization and logging. The application was designed to validate in particular the following aspects of the ESROCOS system: a) Feasibility to develop a robot control system. b) Compatibility of the software packages *stream aligner* and *transformer* with space representative hardware. c) Actuator control from space representative hardware via Controller Area Network (CAN) protocol. d) Feasibility to develop a sensor-processing pipeline including camera image processing, geometric transformations and visualization. e) Possibility to incorporate BIP-modeled fault detection routines. f) Transmission and reception of telemetry data and commands between a remote computer and a robot. g) Ability to record log data and use data from log files as data source within a control application.

Figure 1 shows the application, which is divided into seven separate subsystems that in some cases are realized by multiple software components. The software components are distributed over different execution hardware present in the target hardware system comprising the rover *Bridget* with a few modifications and an external computer (cf. figure 2). Figure 1 shows two versions of a similar control system. The difference is that in the version on the right the *joystick control* subsystem is replaced by a log file *replay* subsystem. The remaining functionality is the same. The following subsections explain the software and hardware setup in more detail.

4.1. Software subsystems

All colored blocks in figure 1 refer to subsystems implemented using the ESROCOS tools and communicating using the TASTE middleware. The gray blocks refer to hardware devices not modeled within TASTE.

The *joypad control* and *rover control* subsystems in figure 1 (left) realize a tele-operation application using a gamepad. The application provides three different kind of control: 1. Generation of rover motion (driving velocities and steering) by using the left analogue stick. 2. Generation of joint velocities for the pan/tilt unit of the rover using the right analogue stick of the gamepad. 3. Mapping of individual button press events to toggling rover configurations (switch on/off lights, toggle point turn or Ackerman drive mode).

The joystick control subsystem is separated into a gamepad driver component that reads the USB connected device and outputs a structure containing the raw values read (such as axis values, button states). The raw data is processed by individual components

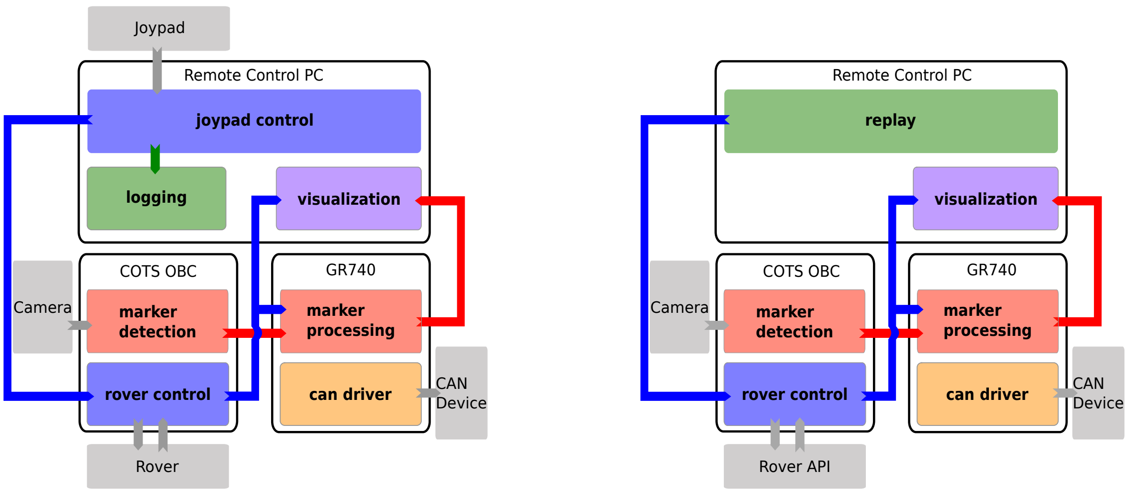

Figure 1. Coarse overview about the software distributed among the hardware platforms with schematic data flow. Left side shows for manual control and data logging. Right side for log data replay.

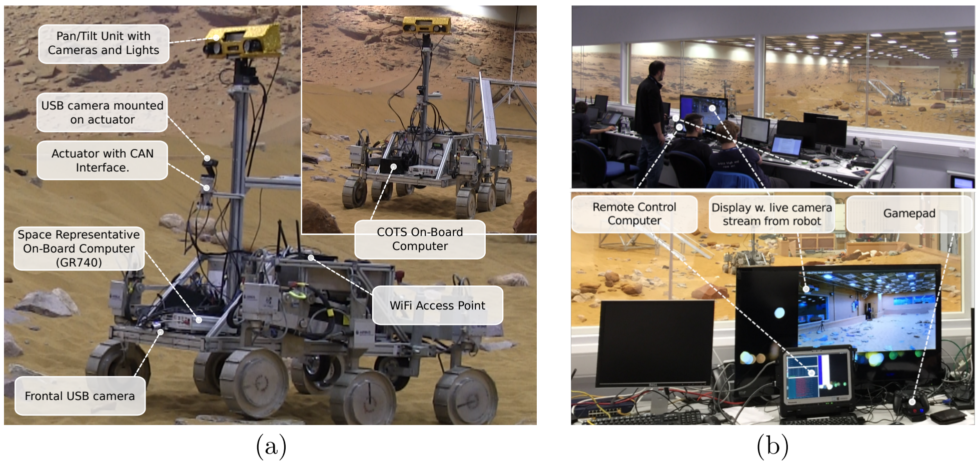
Pan/Tilt Unit with Cameras and Lights
USB camera mounted on actuator
Actuator with CAN Interface.
Space Representative On-Board Computer (GR740)
COTS On-Board Computer
WiFi Access Point
Frontal USB camera
(a)
Remote Control Computer
Display w. live camera stream from robot
Gamepad
(b)

Figure 2. The Mars rover test system Bridget in the Mars Yard (a) and control room (b).

that translate the data into ESROCOS base types to represent reference driving motion commands for the rover or joint commands for its pan tilt unit.

The motion commands from the gamepad control subsystem serves as input to the *rover control* subsystem, which contains a driver component for the Bridget rover and a BIP-modelled watchdog component. The rover driver is realized by a TASTE component wrapping a C++-API for the Bridget rover. With the API telemetry data, including joint state and localization information is received from the rover's internal software and commands for the rover's devices are issued. The wrapper component exposes dedicated API functions to the TASTE modeling environment for interfacing with other components. The watchdog component monitors the incoming rover motion control commands and forwards them to the rover driver, or sends a stop command if no new commands have been received for some time.

The two different versions of the software system illustrated in figure 1 were used in conjunction to test logging of data and the use of log data for online control. The logging subsystem in the version on the left receives a copy of the rover motion command, provided by the joystick control subsystem, and writes the samples to a file using the logging library provided with ESROCOS. In the version on the right, this log file is read and the samples from the log file are fed into the *rover control* subsystem, effectively replacing the tele-operation.

The *marker detection* and *marker processing* subsystems implement an image-processing pipeline with 3D pose reconstruction and further processing of the resulting geometric transformations. For object detection and pose reconstruction, the ArUco fiducial marker recognition system[2] (marker detection) is used to determine the 6D poses of printed markers, which are hidden in the scene.

[2] Available as OpenSource at `https://www.uco.es/investiga/grupos/ava/node/26`

The poses of the marker detection are expressed in the camera frame and fed into the *marker processing* subsystem. This subsystem also receives self-localization information of the rover expressed in the rover's world origin frame that is placed at the location where the rover was started. The samples of both data sources are processed by stream aligner and transformer components that calculate the global pose of the marker. The global pose of the marker is displayed alongside with the localization data of the rover by the visualization subsystem on the operator's screen. The visualization subsystem therefor uses the Vizkit3D software, which is integrated into ESROCOS.

Decoupled from the remaining subsystems, the *can driver* subsystem consists of a motion pattern generating component and a component wrapping a CAN API shipped with RTEMS called *GRCAN*. To receive telemetry, the can driver sends a Telemetry Request frame to the device, which is formatted as presented in table 1. The actuator in turn responds with the current state encoded in a Telemetry Response frame. A new reference is given to the motor controller by sending a Tele-Command frame.

In addition to the mentioned subsystems, a camera stream was established to provide a live image data feed using the software FFMPEG[3]. Data from a camera in Bridget's sensor head is streamed to the operator's remote control computer to provide direct visual feedback to the operator controlling the robot.

4.2. Target Hardware

The software system was implemented on the Mars rover breadboard *Bridget*, which was provided by Airbus Defence & Space in Stevenage, UK through the OG6-Facilitators project[4]. For the ESROCOS validation tests, the rover was equipped with additional cameras, a space-representative GR740 board hosting a LEON4 processor, as well as a normal Linux computer (cf. figure 2 (a)).

The tests have been conducted in the *Mars Yard* facility of Airbus DS, a laboratory consisting of a large sand box with several stones and slopes, with controllable lighing conditions. Using the Wi-Fi access point on the rover and another one in the control room (cf. figure 2 (b)) of the Mars Yard, a wireless network communication between the control room and the rover was created.

The *Remote Control Computer* in the control room was used as a terminal for the operator to the rover and therefore equipped with I/O devices such as a display, mouse, keyboard and a gamepad. This computer served as the source of control commands either originating from the gamepad or log file. The on-board Linux computer hosted the component wrapping the C++API for Bridget, the watchdog component as well as the marker detection component. On the GR740 the *can driver* subsystem as well as the transformer and stream aligner components were executed.

TASTE distinguishes between the system used for software development and the systems that executes the resulting software. In a graphical AADL editor, the deployment targets are specified on the development system (in our case a Virtual Machine that is used to distribute TASTE). TASTE then uses cross-compiling to generate the binaries for the respective target systems. For the case of a Linux target, the resulting executable binary contains the polyorb-HI-C middleware to establish communication with the other subsystems, the data type serialization / deserialization methods and the application itself. For the case of the GR740, TASTE generates a binary that in addition also includes the RTEMS operating system to handle memory and process management. For executing a TASTE application each generated executable file needs to be distributed to the corresponding execution hardware, where the executable files then have to be started manually.

5. EXPERIENCES AND RESULTS

In this section, we want to elaborate on a few aspects we have experienced during developing the application and conducting the test campaign.

While using TASTE and the ESROCOS extensions for implementing the application described in the previous section, we were confronted with certain issues impeding the development process. This section now details these issues following no particular order and explains how we dealt with them. Both, TASTE and ESROCOS are work-in-progress, and will receive further maintenance and extensions in future. While TASTE was in the past years already steadily extended and maintained, and to our knowledge will be continued to be similarly maintained, further development of the ESROCOS system will continue in research projects subsequent to ESROCOS that recently have started. Therefore this section can be understood as feedback on the current state of the system and, where applicable, ideas for further improvements. The experiences reported here have been made with TASTE 9.1 64 Bit version released as Debian Image and ESROCOS packages in a version that later became the ESROCOS_FP_1.0 repository tag.

Accessibility General information about the research project ESROCOS can be retrieved from the

[3]Online available at `https://www.ffmpeg.org/`

[4]Project website: `https://www.h2020-facilitators.eu/`

Message	CAN identifier	RTR Bit	Length	Data					
Telemetry Request	0x7C0	1	0	-					
Telemetry Response	0x1A0	0	6	τ_0	τ_1	p_0	p_1	v_0	v_1
Tele-Command	0x182	0	3	m	v_0	v_1			

Table 1. Format of the CAN messages. τ denotes motor torque, p and v joint positions and speeds. The control mode identifier m can be used to switch between position, velocity or torque control mode.

official project website[5], where also a link to more technical information is placed. The technical information is presented as a Wiki on a GitHub page[6]. The documentation provides a few tutorials and instructions such as installation instructions, but the content provided there does not yet cover all features of ESROCOS. The user will not be able to retrieve information on how to use certain features of ESROCOS. For example, instructions for installing and using BIP or the mixed criticality systems approach are not giving here. Further information can be retrieved from the project deliverable *D4.4-RCOS APIs and Tools*[7], but this document is difficult to find, as it is not properly advertised as source of documentation. This effectively hides strong features of ESROCOS from the user and could easily be overcome by migrating information from the PDF to the online reference and in addition by providing simple use-case examples. For in-depth information about the individual tools, references to the documentation resources of the particular software could be placed.

For TASTE there are multiple ways to retrieve a working environment: It could be installed via Docker, Virtual Box image or from source code. The ESROCOS installation instructions refer to the installation by downloading the Virtual Box image. This is an uncommon method for software distribution, but it provides a very easy way to retrieve a working development environment. For TASTE this kind of distribution is feasible, since unlike other robotic software development frameworks, with its cross-compiling capabilities, TASTE makes a real distinction between the system which is used for software development and the target system for execution. For installing ESROCOS on top of the Virtual Box image, a few simple steps are documented in the Wiki. While the steps are easy to execute it might be more convenient if ESROCOS came already pre-installed with the Virtual Box image.

ESROCOS uses the software package management and compilation system *Autoproj*[8]. The system allows the retrieval and installation of individual software packages with simple commands such as `aup tutorial/cam_capture` and `amake tutorial/cam_capture`. The tools allows to give the user access to a potentially large pool of re-usable software components, but currently the pool of available components is very limited. In addition, some of the ESROCOS tools are not yet integrated into Autoproj.

Data Types TASTE uses ASN.1 for modeling data types that can be used on component interfaces. The ASN.1 types are then converted into structures in the target programming language. New developers might be confused that the name of the type in the target language is different from the model, e.g. in the case of C++ they are prefixed with *asn1Scc*. Also, the experience working with the ASN.1-generated types can be frustrating, since the generated structures are very low-level and cumbersome to work with. In practice, we were using the ASN.1 types only for the external component interfaces and convert them inside the user code of a component to a more programmer-friendly representation. For that purpose, ESROCOS provides the *base-support* package, a software library that allows converting between ASN.1-generated types and corresponding C++ data types used in ROCK.

Another problem regarding data types is that large data structures cannot be transmitted by the middleware. This problem came apparent for us when wanting to transmit camera images. In the case of marker detection, we overcame the problem by embedding the camera driver into the marker detection component such that no transfer of an image is necessary. In the case of providing the operator an overview from the rover's perspective, with the camera stream realized with FFMPEG, we chose a solution independent from ESROCOS. To realize transport of images using TASTE, ESROCOS provides a specific pair of components (*imagetransfer*) that splits camera images into transferable 32kB chunks and re-assembles them after the transfer. A solution for transferring larger structures should be found within TASTE to overcome the need for such workarounds.

A problem hindering the growth of a larger set of reusable components and compatible components is the way in which array structures are treated in TASTE. Dynamically sized structures are not supported since they are forbidden in safety critical space applications. Instead, TASTE allows the specification of fixed sized arrays with ASN.1. ESROCOS proposes to split the type specification into two parts: a) the re-usable base-types with a size-variable as a placeholder for the dimensionality of array-types, and b) an additional ASN.1 file contain-

[5]`https://www.h2020-esrocos.eu/`

[6]`https://github.com/ESROCOS/esrocos.github.io/wiki`

[7]PDF online available at `https://www.h2020-esrocos.eu/wp-content/uploads/ESROCOS_D4.4_RcosApisAndTools_V1.2.pdf`

[8]`https://www.rock-robotics.org/documentation/autoproj/`

ing the size-variable specifications. While the base-types are supposed to remain untouched by the user, the size-variable file is edited in a user project. The problem with this approach is that it is only possible to give only one size per type. This results in a waste of resources and the need for workarounds to express the amount of used elements in an array structure, what in turn is a potential source of mistakes.

Application Modeling Modeling an application with TASTE works very well in general. While readability for large software systems could be improved TASTE's graphical user interface already does a great job in abstracting technical details to simplify software modeling. In particular, we were impressed how well the development of programs for the GR740 went. Since we have never worked with such a system before, we expected more problems here. The only hurdles we had to take were that by default TASTE configures very small stack sizes for each processing unit and we had to increase these values in Concurrency View of TASTE, and that not all properties used in TASTE are reflected by the user interface. Some properties are handled by so-called *TASTE-directives*, a piece of text following a formatting convention, which is then evaluated in the build process. In our case, we had to set some compiler options for the RTEMS compiler via TASTE-directives. The presence and the need of the compiler flags need to be known by the user. This kind of expert knowledge could be better reflected by the GUI.

In our test application, we were confronted with the fact that we wanted to have multiple provided interface (PI, callable interface of a component) being connected to a single requesting interface (RI, calling interface). The GUI prevents this as it only allows one-to-one connections. In our case, one-to-many connections would have made a lot of sense since we were building a data flow system, where having multiple consumers of the same data sample is a common pattern. We overcame the problem by adding dedicated dispatcher components that receive a sample and forward it to multiple RI's. Our test application already contains three dispatcher components for three different types. In our opinion there should not be a strict rule to forbid one-to-many connections and the dispatching should be automatized.

The mechanism for component re-use currently implemented in TASTE does not support multiple instances of the same component in an application. This is a problematic issue for numerous applications, for instance with redundant hardware where the same driver component with different configuration is supposed to be reused. A workaround is the creation of copies or symbolic links of the component's source code files.

To use the data logging library in TASTE for replay, is wrapped by a TASTE component, where the call of a periodic PI is mapped to reading the next sample from the file. The frequency of data reading from the log-file is controlled by the triggering rate of the PI. To replicate the original data it must be taken care, that the triggering is configured to the same frequency as the original data was produced. Thus, only data from source of a fixed frequency can currently replicated. An improvement would be if the log-file replay component would be send samples based on the timestamps present in the log-files.

Build System Programs in TASTE are compiled by executing an automatically generated shell script that triggers a number of processes such as type conversion, glue code generation and compilation. The overall process is rather slow. For a new project consisting of a single C++ component with no user code and a single periodic interface, that is compiled for a Linux target the first call of the build-script already takes 21 seconds (Virtual Box with default settings hosted on a notebook with i7-6600U CPU). In a second call of the build script where the project did not change in the meantime, the execution time drops to 12 seconds. These are values that are still okay to work with, but with multiple components and deployments compilation easily gets displeasingly slow. A project with 15 trivial C++ components that are distributed to five Linux executables takes already about 2 minutes of compilation time, or 1 minute in second run respectively.

The main part of the compilation time comes from the overhead introduced by the framework rather than the user code. Especially for the case, where a user wants to compile and debug his/her code, faster compilation times are essential to avoid user frustration. Improved mechanisms to identify which parts of the system have actually changed (user code vs. deployment specification vs. component interfaces) could help skipping a number of compilation actions and thus improve on the compilation times.

Extentability During preparing the validation test application, we found interest in the idea of extending TASTE in two directions. One was to create a tool that can connect and configure certain software components automatically. The other was to extend the bus systems known by TASTE by an additional one. We had to abandon both ideas due to difficulties identifying how to integrate features that are on the level of processing AADL models in TASTE. For the automated component wiring, we would have needed to parse, process and emit AADL and then pass a modified version of the AADL file to TASTE. Already finding a AADL parser/emitter library was problematic since the web does not reveal much information about processing AADL models.

We have found a documented feature in Ocarina[9] for converting AADL to a XML file representation, but for us it was not working with TASTE Interface-Views. Similarly was the case for the extension with a new bus system. It was easy to identify where to place the AADL files that model the new system but about the software integration we found no information. These examples give us the impression, that there is a lot of potential of the TASTE infrastructure still unused by setting the entrance barrier to work with the model-based back-end too high. Due to the sound design of many parts of the system our impression is, that by putting more focus on hands-on tutorials and on tooling, users of the system could already be put in the situation to integrate the hardware or modeling features they need. This could have a big effect on the growth of the systems community and application domains.

6. CONCLUSIONS & FUTURE WORK

We have shown that ESROCOS and TASTE are principally suited to setup a working robot control application deployed to heterogeneous distributed execution hardware. Due to the mature visual modeling tools of TASTE, the software system was comparably easy to set up and built.

During the development, some issues have come up that where described in the previous section. Some of them point to a lack of documentation, demand an improved degree of integration, or missing convenience features. We believe that these things can easily improve with further maintenance. Others are of technical nature and here we want to stress three critical issues with the potential to impede the applicability of the ESROCOS system to more complex robotics applications and therefore should be dealt with in particular: 1. Multiple instances of components should be possible. 2. The current situation how array-like data structures and their dimensions are handled are not satisfactory. 3. Transfer of large data structures should be supported by the framework.

With the named issues solved and the further maturing of the framework tools and workflow, ESROCOS could become an even more valuable tool for the development of robotics applications. Since the evolution of ESROCOS continues in the second phase of the Space Robotics Technologies SRC, we are optimistic for the future of the system.

[9]Distributed with TASTE and also available online at `https://github.com/OpenAADL/ocarina`

ACKNOWLEDGEMENTS

The authors would like to thank all team members for their contribution to the development of the RCOS and the demonstration application. For their support and guidance in the H2020 ESROCOS activity, we would like to thank the European Commission / Research Executive Agency and the members of the PERASPERA Programme Support Activity (ESA as coordinator, ASI, CDTI, CNES, DLR and UKSA). The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730080 and 821858.

REFERENCES

[1] M. M. Arancón and G. Montano et. al. ESROCOS: A Robotic Operating System for Space and Terrestrial Applications. In *Proceedings of the 14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 20017)*, number 1, 2017.

[2] M. Bajracharya and M. W. Maimone et. al. Autonomy for Mars Rovers: Past, present, and future. *Computer*, 41(12):44–50, 2008.

[3] A. Basu and S. Bensalem et. al. Rigorous System Design Using the BIP Framework. *IEEE Software*, pages 41–48, 2011.

[4] S. Joyeux, J. Schwendner, and T. M. Roehr. Modular Software for an Autonomous Space Rover. In *Proceedings of the 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014)*, Montreal, Québec, Canada, 2014.

[5] D. McIlroy. Mass Produced Software Components. In P. Naur and B. Randell, editors, *Software Engineering, Report on a conference sponsored by the NATO Science Committee*, Garmisch, 1968. Scientific Affairs Division, NATO.

[6] M. Munoz Aracon, M. Wirkus, K. Hoeflinger, N. Tsiogkas, S. Bensalem, O. Rantanen, D. Silveira, J. Hugues, M. Shilton, and H. Bruyninckx. ESROCOS: Development and Validation of a Space Robotics Framework (in press). In *Proceedings of the 15th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA 20019)*, 2019.

[7] M. Perrotin and E. Conquet et. al. The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software. *European Congress on Embedded Real-Time Software (ERTS 2010)*, pages 1–10, 2010.

[8] M. Quigley and B. Gerkey et. al. ROS: an open-source Robot Operating System. In *In Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2009.