CFE EXTENSION FOR THE USE OF ROS APPLICATIONS IN THE SPACE MISSIONS

Tatsuhiko Saito[1], Hiroki Kato[1], Daichi Hirano[1], Kosei Iwabuchi[2], Shinobu Kawaguchi[2]

[1] *Japan Aerospace Exploration Agency(JAXA), 2-1-1, Sengen, Tsukuba, Ibaraki, 305-8505, Japan, E-mail: saitoh.tatsuhiko@jaxa.jp, kato.hiroki@jaxa.jp, hirano.daichi@jaxa.jp*

[2] *Systems Engineering Consultants Co., LTD., Setagaya Business Square, 4-10-1, Yoga, Setagaya, Tokyo, Japan, E-mail: iwabuchi@sec.co.jp, s-kawaguchi@sec.co.jp*

Abstract

This paper presents the space robotics software platform that allows developers to easily build their own Robot Operating System (ROS) programs for future space mission using autonomous robots. We realized this platform by extending an existing reliable spacecraft's framework (Core Flight System (CFS)) and middleware (core Flight Executive (cFE)) developed by NASA's Goddard Space Flight Center. We newly created "ROS/cFE interface library" and "CFS convertor." In the platform, the "ROS/cFE interface library" enables the development and running of ROS applications on cFE. The "CFS convertor" automatically translates ROS source files into compatible codes with cFE libraries without complex and time-consuming porting task. We verified the platform on Linux and the reference board, and confirmed that porting ROS applications worked properly on cFE.

1 INTRODUCTION

In previous space missions, we had to develop specific embedded systems and applications for spacecrafts, which made it difficult to reuse existing software. In addition, integration tests were only possible after combining all system with the OS and an onboard computer, while developers could conduct unit tests for each application on independent desktop computers. Such processes resulted in inefficient software verification. Therefore, a space software platform offering high reusability and verifiability is preferred. The Core Flight System (CFS) and its middleware the core Flight Executive (cFE) were developed on such demands by NASA's Goddard Space Flight Center (GSFC)[1, 2].

The Robot Operating System (ROS)[3, 4] is open-source middleware for robot applications. A large number of ROS packages are provided as the robotics software components, such as sensors, actuator controls, and object recognitions. ROS users can easily combine and integrate these packages into their own robots. ROS was previously implemented for space use in Robonaut2[5] and Astrobee[6] in the International Space Station (ISS), where astronauts can help robots in case any troubles occur. However, a fully autonomous robot mission by spacecrafts without such human supervisory control requires advanced emergency detection, which is not implemented in the original ROS system.

In this paper, we propose a reliable software architecture for ROS that guarantees the emergency detection with the cFE. The proposed architecture for ROS based on cFE allows software developers to use existing application software without any changes and easily test new software under an environment where the essential flight services are integrated in advance. We newly created the " ROS/cFE interface library " and " CFS convertor " so that the ROS applications on cFE work the same way as on ROS. The CFS convertor translates the ROS source, and includes message files into the files that can be built with the cFE and CFS libraries, while maintaining functional compatibility. ROS/cFE interface library defines the wrapper functions and the ROS (topic) message type available on cFE Software Bus (SB). Through this module, each ROS application can communicate with the other applications and cFE. For verification, we deployed the robotic arm control ROS applications used in research regarding space debris removal[7] on cFE. cFE was ported to the RX64M microcomputer with μITRON. We confirmed that the ROS applications successfully send and receive (publish and subscribe) messages via the cFE SB.

2 ROS AND cFE/CFS

2.1 ROS

ROS is an open-source robotics middleware maintained by the Open Source Robotics Foundation. ROS is the most popular robotics mid-

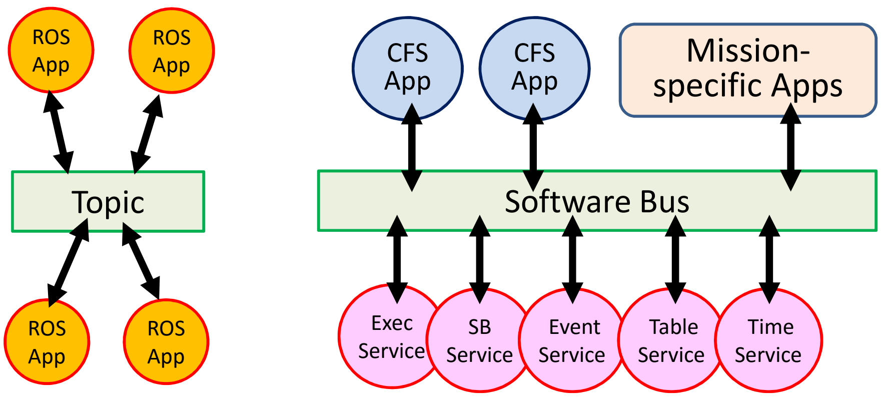

Figure 1: Schematic view of intra-applications communication by publishing/subscribing messages. Left: ROS, right: cFE.

dleware, and the ROS community has numerous users worldwide. The most valuable advantage of adopting ROS into a robotics system is enabling the use of ROS packages. The ROS packages are libraries giving the robotics functions, (e.g. TF package providing coordinate conversion libraries, OpenCV package providing image processing functionalities, PCL providing 3D point cloud processing libraries). Various developing tools are also available for ROS, such as Gazebo (a dynamics simulator) and rviz (a 3D visualization).

Fig. 1 (left) shows a typical structure of ROS architecture. ROS is designed to be operated on a distributed system. Each robotics application works independently in communicating with the other application by publishing/subscribing messages through a "topic", that means the message bus.

The following list shows the popular ROS functionalities.

- Topic: Publishing/Subscribing messages.
- Service: Requesting/replying messages.
- Param (rosparam): Setting and getting initial parameters of the applications.
- Bag (rosbag): Recording and playing published topic messages.

ROS is used not only for the ground-based robots, but also for the robots in the ISS (Robonaut2 and Astrobee), since the functional requirements for the robot used in the pressurized modules of ISS is comparatively close to those for the robots on ground. However, ROS itself is not enough assumed that it is applied as a middleware of the spacecraft system. ROS is not a fault tolerant system and do not provide safety functions (e.g. emergency detection and application monitoring) that are absolutely needed for a mission-critical system. In addition, ROS works on Unix-based OS (especially on Ubuntu PC). Onboard computers used for the spacecrafts do not have sufficient processing capability to operate such a high performance OS.

2.2 cFE/CFS

CFS is an open-source spacecraft software platform developed at GSFC. And a core component of cFS is the cFE that provides the essential flight services described below.

- Executive Service manages cFE, startup, resets and deletes applications, and spawns tasks.
- Software Bus Service provides an inter-application message framework and routes messages.
- Event Service provides the interface for an application's event messages.
- Table Service manages tables.
- Time Service provides spacecraft time and the interface to query time.

These applications and services communicate via the Software Bus (SB) shown Fig. 1 (right) by publishing/subscribing messages. cFE has an event-driven architecture, and tasks are activated based on the messages received. This feature is similar to the publishing/subscribing of topic messages in ROS. CFS has been successfully used in multiple NASA satellites (e.g. LRO, GPM, MMS), Morpheus Lander and by Moon Express in Lunar X-Prize[8]. Due to such guaranteed safety functions as emergency detection and application monitoring, CFS is a reliable framework for space missions.

Given its architectural similarity and high reliability, cFE is the best candidate for ROS applications to run on.

3 PROPOSED SOFTWARE ARCHITECTURE

In order to port the ROS applications on cFE, we developed the following two software modules:

- ROS/cFE interface library
- CFS convertor

3.1 ROS/cFE Interface Library

The entire collection of ROS libraries is too huge to be mounted in the embedded system. We choose the minimum ROS functionalities needed to create applications for space use (i.e. functionalities related to publishing messages, cyclic operations invoked by subscribing messages). The first and second columns of Tab. 1 list the selected ROS functions available in our cFE extension. Similarly, Tab. 2 lists available message types that can be selected. As for porting ROS packages, we port a part of the TF package libraries[9] as shown in Tab. 3, in order to run the sample ROS applications on cFE as described in Subsection 4.1.

The ROS/cFE interface library defines the wrapper functions (listed in the third column of Tab. 1) for the ROS functions and available ROS message types. Inside the wrapper functions, the APIs of cFE Executive Service and Software Bus Services are called (as listed in the fourth column of Tab. 1). By using these wrapper functions, ported ROS applications can send (or receive) messages to (or from) the other applications and services on cFE. ROS/cFE interface library consists of one C++ source file and two header files. The total steps of this library is less than 300 steps.

Fig. 2 shows the a schematic diagram of the overall structure of a system where transcoded ROS applications and the ROS/cFE interface library are built in. The Board Support Package (BSP) contains hardware-dependent driver. The Platform Support Package (PSP) provides hardware-dependent functionalities such as a timer and memory access. OS Abstraction Layer (OSAL) abstracts the software architecture from hardware and enables the system to work on multiples platforms (Linux, vxWorks and RTEMS). The modules described in red letters in Fig. 2 are mission-independent and reusable.

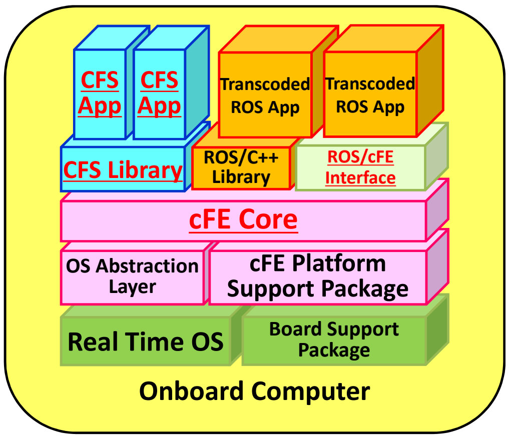

Figure 2: Schematic diagram of the system where the transcoded ROS applications deployed on cFE with ROS/cFE interface library

3.2 CFS Convertor

The proposed CFS convertor can:

- convert the supported ROS functions into the corresponding wrapper functions in porting target source code;
- write the application's startup parameters defined in the roslaunch files directly into porting target source code, as rosparam functionality is not supported in our cFE extension; and
- change the ROS code format (written in C++) into one that can be built with the other cFE/CFS libraries (written in C).

By using the CFS convertor, ROS application developers can easily change their own ROS source and header files into those compatible with cFE/CFS libraries.

3.3 Advantages of our proposed architecture

ROS/cFE interface library and CFS convertor offer developers the following advantages:

- ROS publishing/subscribing functions can communicate with other cFE software components over the cFE SB.
- Existing ROS source files and libraries can be ported and run on cFE without significant modifications.

Table 1: ROS functions available with ROS/cFE interface library

Class	ROS Function Name	Wrapper Function Name in ROS/cFE interface library	Used cFE API
-	init	CONVERT_RosInit	CFE_ES_RegisterApp
			CFE_ES_GetResetType
			CFE_EVS_Register
Time	now()	CONVERT_RosTimeNowToSec	CFE_TIME_GetTime
-	ok()	CONVERT_RosOk	CFE_ES_RunLoop
Rate	sleep()	CONVERT_RosRateSleep	-
Duration	sleep()	CONVERT_RosDurationSleep	-
-	spinOnce()	CONVERT_RosSpinOnce	-
-	spin()	CONVERT_RosSpin	-
NodeHandle	subscribe	CONVERT_RosNodeHandleSubscribe	CFE_SB_CreatePipe
			CFE_SB_Subscribe
	advertise	CONVERT_RosNodeHandleAdvertise	CFE_SB_CreatePipe
Publisher	publish	CONVERT_RosPublisherPublish	CFE_SB_InitMsg
			CFE_SB_SendMsg

Table 2: Topic messages available with ROS/cFE interface library

Message Class	Message Type
std_msgs	header
	Float64MultiArray
	MultiArrayLayout
	MultiArrayDimension
geometry_msgs	Wrench
	WrenchStamped
	Pose
	PoseStamped
	PoseWithCovarianceStamped
	Point
	Twist
	TwistWithCovariance
	Vector3
nav_msgs	Odometry
sensor_msgs	JointState

- The joint test for ported ROS applications and cFE can be conducted on Linux PC.

3.4 Porting Process

This section summarizes the process of porting ROS applications on cFE (Fig. 3). In addition to the ROS/cFE interface library and CFS convertor, we plan to publish a user manual and a guideline for ROS application developers. The user manual gives instructions on how to use the CFS convertor and build transcoded files with cFE/CFS libraries. The guideline describes the functional considerations and limitations in the development of applications built with the ROS/cFE interface library. The porting process using the ROS/cFE interface library and CFS convertor is as follows:

Table 3: TF package files available with ROS/cFE interface library

File Name	Class
Matrix3x3.h	Matrix3x3
MinMax.h	-
QuadWord.h	QuadWord
Quaternion.h	Quaternion
Scalar.h	-
StampedTransform.h	StampedTransform
Transform.h	Transform
Vector3.h	Vector3

1. **Check of the codes**
   ROS users establish a development environment as per the user manual, check whether their porting targets are properly coded, and then modify the code as needed.
2. **Conversion of codes**
   ROS users execute the CFS convertor for their codes and obtain compatible source and header files.
3. **Build**
   Converted source and header files are added to the cFE libraries (and CFS libraries if needed) and ROS/cFE interface library, and then all files are built.
4. **Test on Linux**
   Executable codes after all files are built can be run on a Linux PC, and a joint test of transcoded ROS applications and cFE can be easily conducted.
5. **Cross-compilation**
   After the test on Linux, all files are cross-compiled and embedded to the target platform with a real-time OS (RTOS).

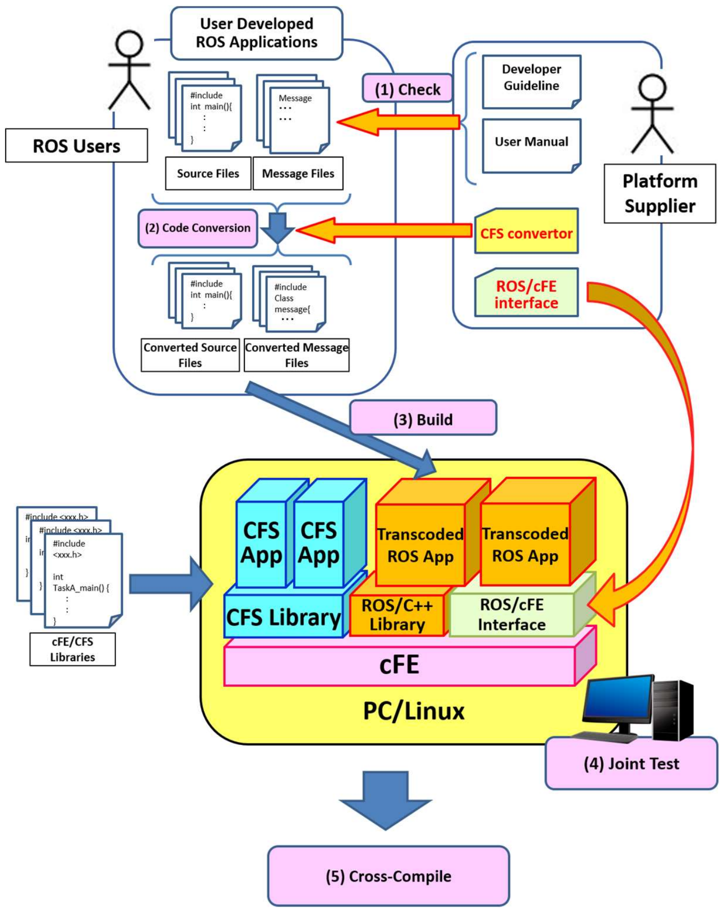

Figure 3: Process of porting ROS applications

4 VERIFICATION OF OPERATION

We verified the performance of the ROS applications ported on cFE. The all tests described in this section are conducted under the following environment:

- OS of development PC: Ubuntu 16.04
- Version of ROS: Kinetic Kame
- Version of cFE: 6.5.0
- Version of OSAL: 4.2.1

4.1 Run on Linux

First, we checked the performance on Linux PC. We chose the ROS applications used for the air-floating robot in the research on the autonomous capture of space debris. The target ROS applications consist of the "main control application" and the "robotic arm control application." The robotic arm control application publishes the joint state including encoder data and motor current,

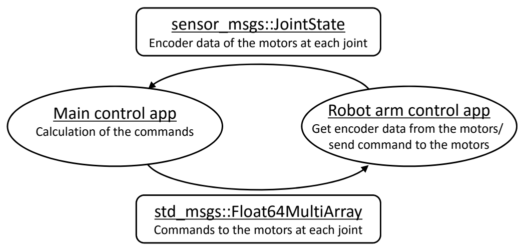
sensor_msgs::JointState
Encoder data of the motors at each joint
Main control app
Calculation of the commands
Robot arm control app
Get encoder data from the motors/ send command to the motors
std_msgs::Float64MultiArray
Commands to the motors at each joint

Figure 4: Message flows between the ROS apps

while the main control application publishes the motor command. These two applications publish/subscribe topic messages to each other at 1 kHz (Fig. 4). We converted the codes and built files with the cFE libraries as described in Subsection 3.4. After the run, we confirmed that the ported ROS applications worked on cFE in the same way as on ROS, and that the messages are exchanged at the configured rate.

We also checked whether each selected ROS application is run not only with other ROS appli-

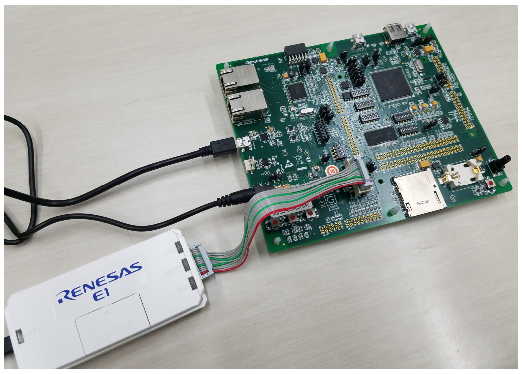

Figure 5: Reference board: Renesas RX64M

cations but also with CFS applications. We conducted testing with the Housekeeping (HK) application. HK application is one of 12 applications published by GSFC (called as CFS applications) for particular onboard functions. It collects logs from the applications, and re-packaged them as telemetry data. We newly created the ROS applications to publish/subscribe messages to/from the HK application, and then confirmed communications between them.

4.2 Reference Board Test

We also verified our cFE extension on the reference board (Fig. 5) assuming that the extension are embedded into a onboard computer in the spacecraft system. The features of the board[10] used for the test are as follows:

- Microcontroller: Renesas RX64m
- CPU core: RX CPU v2 core
- Maximum operating frequency: 120MHz
- Power supply voltage: 2.7 to 3.6 V
- Program Flash: Max 4MB
- Data Flash: 64KB
- SRAM: 552KB (ECC 512KB, Non-ECC: 32KB, Standby RAM: 8 KB)

We used Renesas CubeSuite+ as a developing tool and RI600V4[11] conforming to μITRON4.0 as RTOS.

We created two sample ROS applications which just publish/subscribe uint32 (32-bit unsigned integer) message data each other. We confirmed that these applications deployed on cFE and RTOS successfully communicated via SB on publishing rate up to 100 Hz, which was the operational limit of the board.

5 CONCLUSION

This paper presented the space robotics software platform that allows developers to easily build their own ROS applications. We extended cFE to be capable of ROS functionalities, particularly the publishing/subscribing of the topic messages by building with the ROS/cFE interface library. In addition, we created the CFS convertor to make original ROS codes compatible with the ROS/cFE interface library and cFE/CFS libraries. We verified our extension with the sample ROS applications used for the air-bearing robot and the ROS applications that communicate with the CFS application on a Linux PC. We confirmed that the ROS applications worked correctly on cFE. We also confirmed that the messaging functions of the cFE/ROS interface worked properly on the reference board of a microcontroller running RTOS.

The next step is an on-orbit verification of our cFE extension. For that, further functional improvements are considered, i.e. the expansion of the ROS functionalities supported by the ROS/cFE interface library to rosparam, rosbag and host communication between ROS and cFE.

References

[1] core Flight System. Retrieved April 23, 2018, from https://cfs.gsfc.nasa.gov/.

[2] McComas D (2012) *NASA/GSFC's flight software core flight system*.

[3] ROS. Retrieved April 23, 2018, from http://www.ros.org/.

[4] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R and Ng AY (2009) ROS: an open-source Robot Operating System. In: *ICRA workshop on open source software*, volume 3, Kobe, Japan, p.5.

[5] Farrell L, Strawser P, Hambuchen K, Baker W and Badger J (2017) Supervisory control of a humanoid robot in microgravity for manipulation tasks. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp.3797–3802. doi: 10.1109/IROS.2017.8206229.

[6] Bualat M, Barlow J, Fong T, Provencher C and Smith T (2015) Astrobee: Developing a free-flying robot for the international space station. In: *AIAA SPACE 2015 Conference and Exposition*, p.4643.

[7] Hirano D, Kato H and Tanishima N (2017) Caging-based grasp with flexible manipulation for robust capture of a free-floating target. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, pp.5480–5486.

[8] Prokop LE and Wilmot JJ (2014) *Nasa's core flight software-a reusable real-time framework*.

[9] ROS TF package. Retrieved April 23, 2018, from http://wiki.ros.org/tf.

[10] RX64M. Retrieved April 23, 2018, from https://www.renesas.com/en-us/products/microcontrollers-microprocessors/rx/rx600/rx64m.html.

[11] RI600V4 Real-time OS for RX Family. Retrieved April 23, 2018, from https://www.renesas.com/en-us/products/software-tools/software-os-middleware-driver/itron-os/ri600v4-for-rx-family.html.