WE LOOK AFTER THE EARTH BEAT

ASTRA 2015

Robot Control Operative System Forum

Andrea Biggio

HOW TO INCREASE RE-USABILITY OF A ROBOT CONTROL SW ARCHITECTURE

12/05/2015

Ref.: 83230913-DOC-TAS-EN-002

OPEN

ThalesAlenia Space

A Thales / Finmeccanica Company

Contents

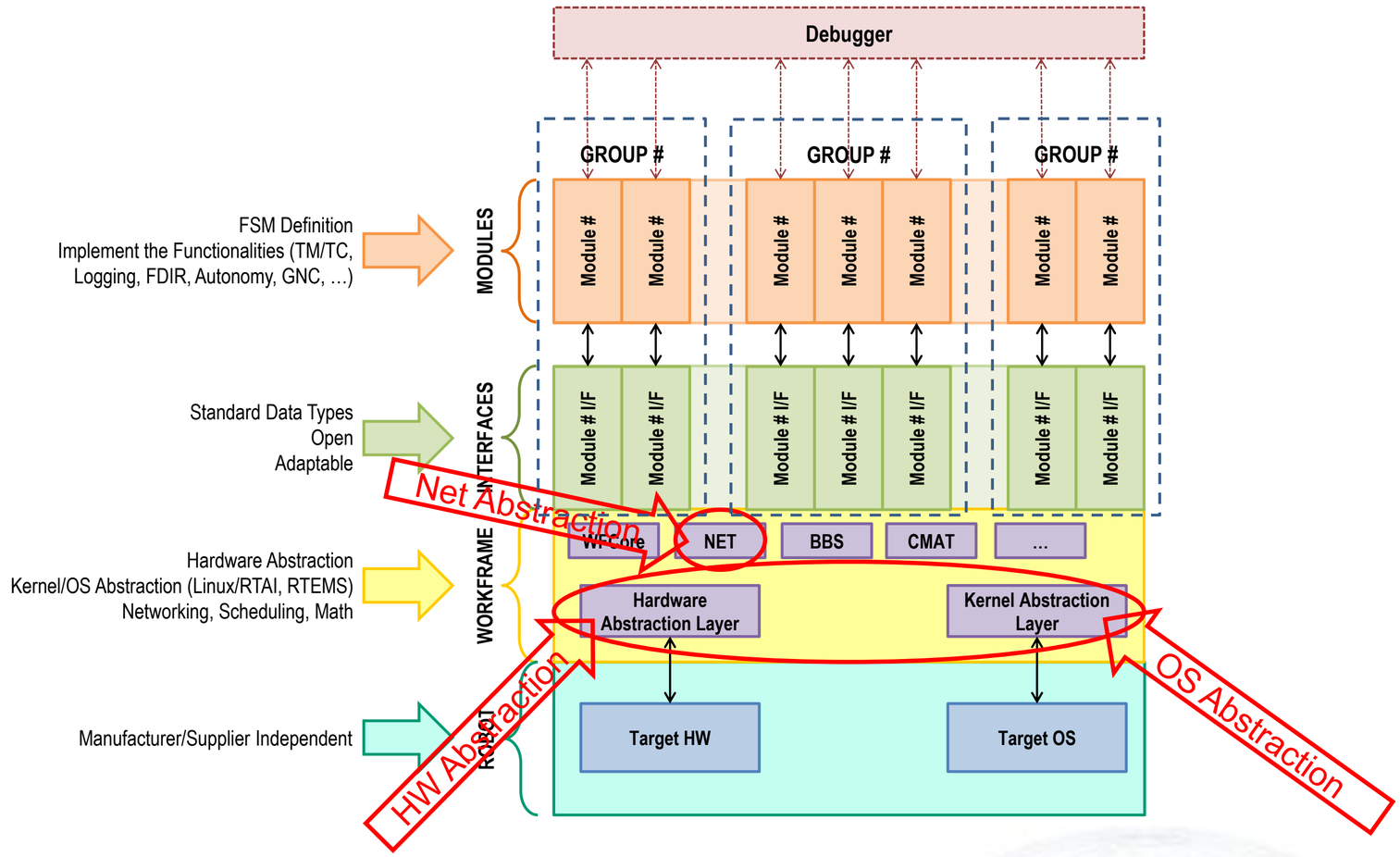
- Architecture and Context
- (RCOS) Re-Usability Principles
- Reusability and RTOS abstraction
- Kernel Abstraction
- Network Abstraction
- Hardware Abstraction

The activities subject of this presentation have been performed in the frame of STEPS program - Systems and Technologies for Space Exploration - a research project co-financed by Regione Piemonte (Piedmont Region) within the P .O.R.-F.E.S.R. 2007-2013 EC program.

In collaboration with University of Genova – DIBRIS – G.R.A.A.L. Laboratory

Architecture and Context

- Validation and Verification of a generic robotic system / modules
- Study and Integrate robotic technologies (e.g. sensors, algorithms)
- Ease interaction with project partners



Designed to minimise the re-coding of high-level layers of the RCOS

(RCOS) Re-Usability Principles

- Re-Usability is not limited to the code
- Well defined interfaces (API, data, …)
- Generality
- Encapsulation
- Modularity
- Well documented
- Link test to code
- Type Independent
- Abstraction
- …

RCOS code should be extended, not modified

Reusability and RTOS abstraction

- Reusability can be increased by providing an abstraction from the underlying RTOS used.
- This allows an application to be moved from e.g. an x64 system running RTAI to one running RTEMS on a PowerPC
- Definition of a minimum set of APIs that constitute the *Kernel Abstraction Layer*
- Which RTOS to support?
  - QNX and VxWorks are widely used in commercial applications
  - RTEMS is also used within the space community
  - RTAI is one of the preferred academic choices

KAL shall implement the overlapping funtions of the different RTOS, but not restricting the use of RTOS-specific APIs

Kernel Abstraction Layer

- Supporting different RTOS such as RTAI, RTEMS, QNX and VxWorks poses different problems

- Abstraction of resource identifiers (mailbox example)
  - RTAI uses 6 characters to identify a mailbox
  - RTEMS and VxWorks use an unsigned integer
  - QNX uses an array of chars up to PATH_MAX

- To ensure portability between systems none of these resource IDs can be used directly

Identical functionalities have different RTOS-dependent implementations

Kernel Abstraction Layer (mailbox example)

- Behaviour is almost always slightly different between different RTOS.
- Mailbox/queue example:
  - RTEMS only provides *rtems_message_queue_send()* that does not block if the queue is full
  - RTAI provides both *rt_mbx_send()*, which is blocking, and *rt_mbx_send_if()*, which is non-blocking
  - QNX provides a single *mq_send(),* however its behavior can be changed using *mq_setattr()* to set specific flags for blocking/non-blocking behaviors

Identifying the set of features common to all the RTOS is not straightforward

Kernel Abstraction Layer (multiple processes example)

- RTEMS executive is fairly different from RTAI/QNX/VxWorks, as everything is run from the same unique rtems_init() function.
- You can’t have multiple processes, only threads
- If you want portable processes, you can’t call them all as main()

Having portable processes means having portable modules

Interoperability between Networked Hosts

- Reusability can be also a further problem when multiple hosts come into play
- If I have an host with RTAI KAL and another with RTEMS KAL, can they talk to each other?
- Data structures are represented differently in memory between different systems, as well as endianness
- This is a common and standard problem in networking
  - Need to marshal/serialize data when transferring it between systems
  - Needs to be transparent to the user and possibly without too much “boilerplate” code

Network abstraction is needed for inter-module and inter-RTOS communications

Reusability and Hardware

- Reusability of software code is a challenge whenever hardware is changed
- Need for a clean separation and abstraction between general purpose code and the APIs that interact with the underlying hardware

- Definition of a Hardware Abstraction Layer (HAL)

- Creating such APIs is extremely difficult, as different hardware has different capabilities.

- What is the minimum set of features that a certain devices must possess? Where do you draw the line?

RCOS HAL is complimentary to the RTOS HAL

Possible Solution for Hardware/Software Reusability

- Definition of a communication protocol and its data structures independent from the specific hardware
- A dedicated task (HAL-task) handles the interaction with the hardware (initialization, reading and writing to it, etc.) and answers to the above protocol providing data in a hardware-independent format
- The other task do not directly access the hardware but rather query this HAL-task
- Changing the hardware can be done by simply using/querying another HAL-task, without rewriting the general purpose part of the code

HAL-tasks are a simpler way to manage sensors and actuators.

Contacts and References

Author Contacts:

- TAS-I: Andrea Biggio, andrea.biggio@thalesaleniaspace.com; +39 011 7180423

References:

- Biggio A. et Al (2015), Design and Implementation of a Robot Management Framework and Modular GNC for Robotic Space Exploration. *ASTRA 2015.*
- Biggio A. et Al. (2015), Validation and Verification of Modular GNC by means of TAS-I Robot Management Framework in outdoor ROvers eXploration facilitY. *ASTRA 2015.*
- Biggio A. et Al (2012), Test Bench for Robotics and Autonomy: Advancements in Navigation for Space Exploration. *i-SAIRAS 2012.*
- Simetti E. et Al (2011), A new software architecture for developing and testing algorithms for space exploration missions. *Intelligent Service Robotics volume 4(2)*, Springer-Verlag, pp135-146.
- Simetti E. et al (2010), A Portable Object Oriented SW Framework for Real-Time Control of Robot and Multi-Robot Systems. In *Control Themes in Hyperflexible Robotic Workcells* (Eds. F.Basile, P.Chiacchio), CUES, pp129-143.
- Anguswamy R. (TBW), A Study of Factors Affecting the Design and Use of Reusable Components, TBW
- Anguswamy R. (2013), Factors Affecting the Design and Use of Reusable Components, *PhD Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University.*
- Anguswamy R., Frakes W.B. (2013), Reuse Design Principles, *DReMeR '13 - International Workshop on Designing Reusable Components and Measuring Reusability Picture held in conjunction with the 13th International Conference on Software Reuse.*