PISA: PI SOFTWARE ARCHITECTURE FOR INTEGRATED SATELLITE PAYLOAD CONTROLLERS

F. Pont[1], R. Siegwart[1], D. Akuatse[2], O. Moulin[2], and R. Vitulli[3]

[1]*Autonomous Systems Lab, Swiss Federal Institute of Technology, Lausanne, Switzerland, Email:frederic.pont@epfl.ch*
[2]*Syderal SA, Neuchatel, Switzerland, Email: daniel.akuatse@syderal.ch*
[3]*European Space Agency, ESTEC (TEC-MMA), Noordwijk, The Netherlands*

ABSTRACT

With the growing amount of data produced by instruments and the increasing importance of the payload element that delivers the added value of the missions, a satellite can be regarded as a distributed system with a platform integrating all the traditional on-board control functions (Attitude and Orbit Control and Data Handling Control) and collaborating with the payload through the spacecraft bus. In PISA, we investigate tools and methods to develop a payload controller based on a single embedded computer integrating instrument control software components developed by several PI teams. A hardware demonstrator based on PowerPC running RTAI Linux is also presented.

Key words: Integrated Payload Controller, Software Framework, Real-time Systems, RTAI Linux.

1. INTRODUCTION

The trend for designing satellites is to embed an ever-bigger payload component in the on-board system. As a result, the cost and criticality of the system shifts towards the payload element that delivers the added value of the mission. In this context, the role of the platform part of the satellite is to operate as a dependable carrier of the value-added embedded in the payload. Hence, a satellite will have to be regarded as a distributed system with a platform integrating all the traditional on-board control functions (Attitude and Orbit Control and Data Handling Control) and collaborating with the payload through the spacecraft bus [1].

In order to maximize the total return cumulated during satellites missions' lifetime, and as the amount of data produced by instruments increases faster than the downlink capacity, the need for increased on-board data processing capabilities increases. This means that more intelligence and autonomy capabilities must be available, to provide relevant information rather than large amount of data to be processed on Earth, for example by executing on-board data filtering, compression or selection before transmission.

If the payload offers more services, its complexity inevitably increases and the usage of an operating system or even higher-level structures like software frameworks is the best solution. New integrated payload controllers will be based on a single compact computer instead a many less powerful computers dedicated to controlling a single instrument [2]. In this integrated approach, the payload controller will host several software components, each of them controlling one instrument or sub-instrument, and each of them potentially being developed and tested by different teams (PIs). Integrating onto a single embedded computer software parts produced by different teams is a complex and challenging task. PISA aims at exploring tools and methods to facilitate this development, testing and integration, and to ensure that a robust and flexible software system can be created to control all instruments from a single embedded computer.

The remainder of this paper is organized as follows. In section 2 we present the objectives that have been set for this project. Section 3 describes in more details the internals of PISA, including the overall software architecture, the software parts that compose PISA and the hardware demonstrator. Section 4 contains the conclusions and outlook.

2. OBJECTIVES

In this context of new requirements and expectations for integrated satellite payload controllers, the PI Software Architecture (PISA) presented in this paper aims at proposing new techniques and tools to facilitate the development and testing of software components controlling instruments, and for integrating and controlling those software components onto a single embedded computer.

In particular, the following objectives have been defined:

- Facilitate the development and testing of instrument control software components by PI in a provided development environment and testing framework on a

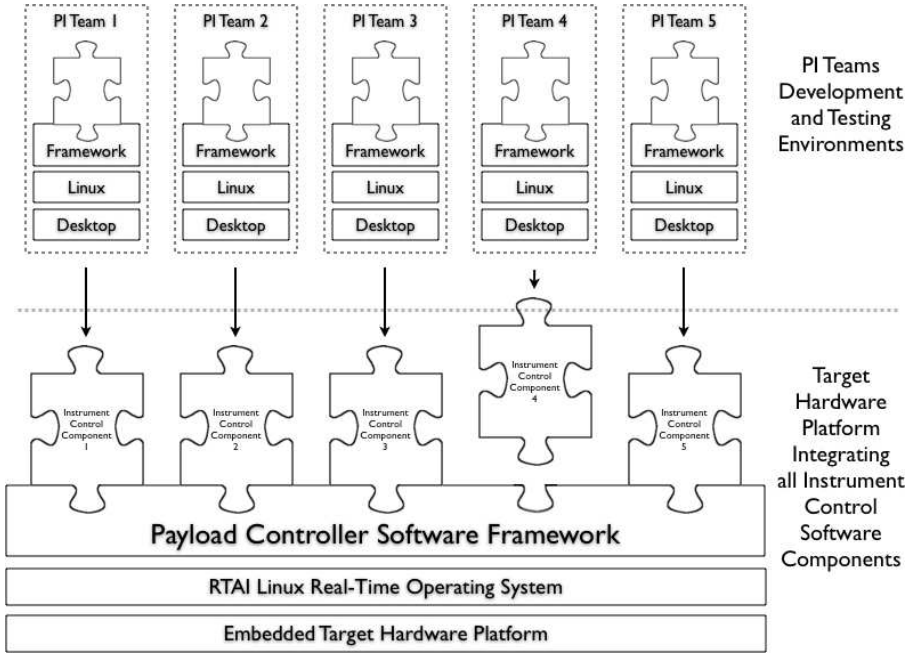

Figure 1. PISA integration process overview. Instrument control software components are developed and tested by PI teams, before being integrated onto a single embedded computer by the PISA Integrator.

Linux computer, without access to the target embedded computer. The algorithms developed to control the instrument shall be used without any modification to the source code on the target embedded computer.

- Facilitate the integration of instrument control software components provided by PI teams into a real-time capable software framework onto the target embedded computer. This integration shall happen without any modification to the source code provided by PI teams.
- Ensure robustness of the complete embedded system to a failure of the single instrument control software component.
- Provide in-flight component modification capabilities without any effect on other instrument control software components. The in-flight component modification process shall support loading the new version of a component from the Mass Memory, or via direct download from the Avionics.
- Demonstrate that the proposed solution is able to manage up to roughly twenty low data rate instruments (average between 1kb/sec and 5kb/sec, peak up to 50kb/sec) through a CAN bus, and medium data rate (average between 5kb/sec and 100kb/sec, peak up to 1Mb/sec) instruments through a SpaceWire network.

An overview of the PISA concept is represented in figure 1, with all PI teams developing and testing their own instrument control software component on legacy Linux computers, and later handing it to the PISA Integrator for integration onto a single embedded computer with real-time capabilities.

The concepts developed in the context of this project are demonstrated on a PowerPC target embedded computer running RTAI (Real-Time Application Interface) Linux [3], an extension of the Linux kernel that provides hard real-time capabilities. Moreover, PISA relies on GenoM [4], an open-source component-based software framework used in several robotics research laboratories on complex mechatronic systems, from indoor mobile robots to autonomous cars, as well as prototypes of space exploration rovers.

3. PISA OVERVIEW

3.1. Software Architecture

The PI software architecture will allow PI teams (developers) to develop and test software components compatible with the PISA software framework. Each software component will be responsible for controlling a specific instrument or sub-instrument. The role of the PISA Integrator is to receive the software components produced by all PI teams and to integrate them into a software system to running on the target embedded computer. An overview of this process is shown in figure 1.

The PI Software Architecture is composed of software to be used for development and testing of instrument control software components at PI teams premises, and of

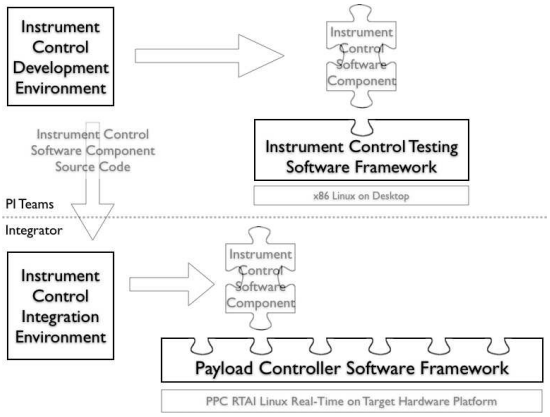

Figure 2. PISA software parts: Instrument Control Development Environment and Instrument Control Testing Software Framework (for PI teams), Instrument Control Integration Environment and Payload Controller Software Framework (for PISA Integrator).

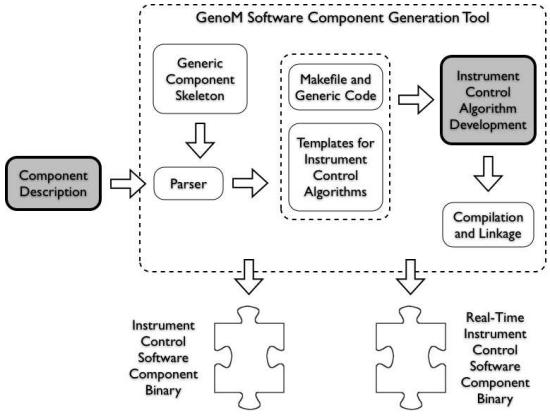

Figure 3. Instrument Control Software Component Generation Process, from the component description file to the generated binaries.

software for integrating and executing those instrument control software components onto a single target embedded computer, as represented on figure 2.

The PISA software can be divided into four main software parts:

- Software to assist PI teams in the development of instrument control software components on an x86 Linux desktop or laptop. This software part is called PISA Instrument Control Development Environment.
- Software to assist PI teams in testing their instrument control software components on an x86 Linux desktop or laptop, with the actual instrument connected to it. This software part is called PISA Instrument Control Testing Software Framework.
- Software to assist the integrator in generating binaries from the instrument control software components source code produced by PI teams for the target hardware platform. This software part is called PISA Instrument Control Integration Environment.
- Software to integrate and execute all instrument control software components produced by PI teams onto the target hardware platform running PPC RTAI Linux. This software part is called PISA Payload Controller Software Framework.

The PISA software frameworks (Instrument Control Testing and Payload Controller) can be defined as partially complete software systems that are meant to be instantiated. A framework defines the architecture for a family of systems and provides the basic building blocks to create them. It also defines the places where adaptations for specific cases can be made, these places are called hot-spots. Instrument control software components can be plugged into hot-spots to be executed. These two frameworks ensure that instrument control software components can be developed and tested by PI without access to the target embedded computer, and later integrated onto the target embedded computer without any modification to component source code.

The Instrument Control Development Environment is based on GenoM, a tool to design and develop complex real-time software architecture. This environment helps PI teams to generate instrument control software component. The detailed generation process is shown on figure 3. The first step of the software component generation process is to write a component description file in a C-like format. In PISA, templates of such description files are provided to PI teams, providing them with mandatory configuration and control functionalities that shall be implemented for each component. Component-specific functionalities are also defined in this file.

Next, this configuration file is processed by the GenoM parser and templates (.h and .c files) are generated. These templates contain empty functions to be completed by developers to implement the functionalities to control their specific instrument using the C programming language. Once the coding of the control algorithms is complete, binaries for a x86 Linux computer can be generated by the environment. Those binaries can be executed in the Instrument Testing Framework and tested. The software component can be tested with the actual instrument if it is available. Otherwise, a simulator for the instrument has to be developed by the PI team. In order to ensure that the code developed by PI teams can be later integrated onto the target embedded computer without any modification, all hardware access shall be performed through Hardware Abstraction Layer (HAL) provided by PISA.

The PISA Hardware Abstraction Layer (HAL) represented on figure 4 ensures that instrument control software components can be developed and tested independently of the underlying hardware devices for CAN, SpaceWire and Mass Memory access. This decoupling is

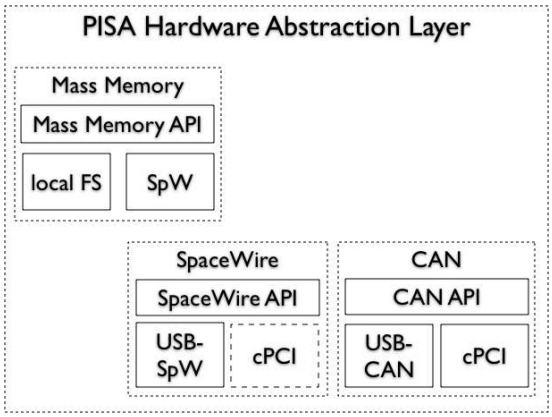

Figure 4. Hardware Abstraction Layer for CAN, SpaceWire and Mass Memory, providing a consistent API for hardware access during development, testing and integration phases.

key for software development by PI at their premises on a standard Linux computer without access to the target embedded computer. As a result, CAN and SpaceWire accesses are abstracted through the PISA HAL and the instrument control algorithms developed by PI teams are totally decoupled from the underlying hardware devices. Moreover, the PISA HAL also abstracts Mass Memory access to ensure that PIs can develop and test Instrument Controller software using a Mass Memory Emulator as if they were using an actual Mass Memory device.

As represented on figure 5, the Payload Controller Software Framework is capable of executing software component with real-time constraints, if it is required for specific instruments. A specificity of RTAI Linux on PowerPC is that real-time components have to be executed in kernel space (See [5] for more information about the Linux user space kernel space duality). The PISA HAL ensures that the exact same API (Application Programming Interface) for accessing CAN, SpaceWire and Mass Memory is available in kernel space and in user space, so that the instrument control software component developed by PI teams can be executed with real-time constraints without modification whenever necessary. Note that usual restrictions about code that can be executed in kernel space apply, and that the robustness of components executed in real-time is limited due to the sharing of a single memory address space with the Linux kernel.

The PISA CAN Hardware Abstraction Layer provides a consistent interface to the following CAN hardware devices:

- USB CAN dongle CPC-USB by EMS Wuensche to be used by PIs during development of their instrument control software component.
- CAN ports embedded on the cPCI 405 PowerPC CPU board to be used on the target hardware system for PISA. Access to this CAN device will be provided both for non real-time (user-space) components and real-time (kernel-space) components.

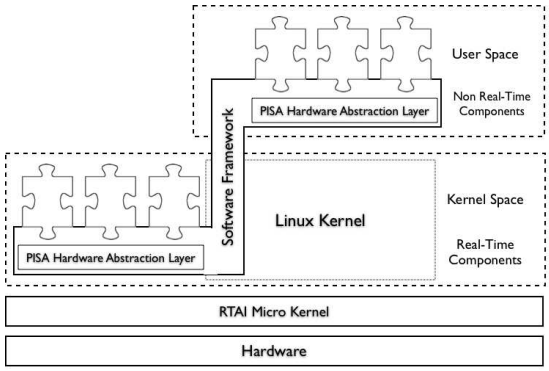

Figure 5. Integrated Payload Controller Architecture with hardware abstraction and real-time capabilities.

The SpaceWire Hardware Abstraction Layer provides a consistent interface to the following SpaceWire hardware devices:

- USB SpaceWire dongle by Star-Dundee to be used by PIs during development.
- cPCI SpaceWire board to be used on the target hardware system for PISA.

The Mass Memory Hardware Abstraction Layer provides a uniformed API to the following mass memory emulators:

- A Mass Memory emulator running on the local host file system to be used by PI during development of an Instrument Control Software Component.
- A Mass Memory emulator accessible through a SpaceWire network to be used on the target hardware. Note that a real Mass Memory device can transparently replace this emulator without any change to the PISA MM HAL for real applications. This emulator allows us to demonstrate the capabilities of the PISA Mass Memory HAL without the need of an actual Mass Memory device.

Each instrument control software component that relies on the PISA HAL will have to provide a mandatory configuration function (in the form of a GenoM configuration request) to ensure that HAL underlying hardware devices or emulators can be selected using a TCL command. One such mandatory configuration function will have to be provided for Mass Memory, SpaceWire and CAN, if these HAL subsystems are used on the component. These configuration functions will be available in the GenoM templates to be used by PI teams when they start development of an instrument control software component.

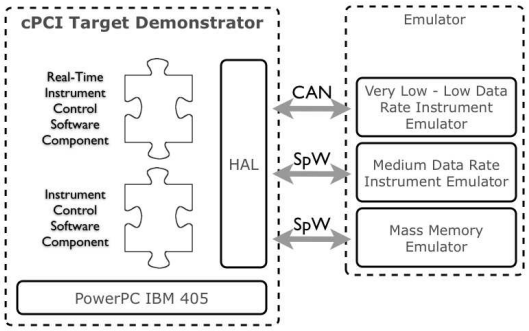

Figure 6. PISA Hardware Demonstrator, based on a PowerPC embedded CPU and an Emulator Unit Tester PC.

The role of the PISA Integrator is to receive the instrument control software component developed by PI teams and to integrate them onto the target embedded computer. The first step is to receive the source code of each software component and to cross-compile them for the target hardware platform. PI teams have developed and tested their software component on x86 Linux computers, and the target hardware in the PISA project is based on a PowerPC.

The Instrument Control Integration Environment is also based on GenoM, and allows the Integrator to produce binaries that can be executed on the PISA Payload Controller Software Framework on the target embedded computer, with real-time constraints whenever required.

As represented on figure 5, the Payload Controller Software Framework integrates all instrument control software components produced by PI teams. The framework can execute software components with real-time constraints if necessary, and provides in-flight component modification capabilities. Moreover, it provides a TCL-based (Tool Command Language) interface for controlling all software components with a high-level script language that can be used by scientists or PI teams to describe experiments scenarios in a convenient manner.

3.2. Hardware Demonstrator

For the development and the demonstration of the PISA project, a PowerPC platform running RTAI Linux has been selected. The complete hardware setup is shown in figure 6. The hardware demonstrator includes two CAN buses and two SpaceWire links. In order to demonstrate the PISA functionalities, instrument emulators will be developed and run on the Unit Tester PC.

The first instrument emulator will be connected to the target embedded computer through a CAN bus, emulating a very-low to low data rate instrument. The second instrument emulator will be connected via a SpaceWire network and will emulate a medium data rate instrument. A Mass Memory emulator will also be developed and run on the Unit Tester PC.

4. OUTLOOK

The goal of the PISA project is to demonstrate the feasibility of an integrated approach in satellite payload controller by facilitating the development of instrument control software components by PI teams and supporting the integration of all component onto a single embedded computer. With PISA, we are demonstrating the feasibility of such a concept, including real-time capabilities, resilience to a failure of a software component and providing the in-flight software component modification capabilities.

Moreover, PISA provides a solution that allows PI teams to focus on their speciality and on their instrument, freeing them from the burden of developing and delivering a complete embedded system. With PISA, they can rely on the development tools and environment that is provided to develop and test their instrument on a standard Linux computer before delivering their instrument control software component to the PISA Integrator, an embedded system expert who takes care of handling specific problems.

Another advantage of this integrated approach it that it allows PI teams to take advantage of an increased processing power that has to be shared among all instruments that would not be available in a dedicated payload controller approach, allowing them to increase the on-board autonomy and intelligence, sending to the Earth more valuable data.

Next steps for PISA includes the application of the proposed concepts for developing a control software component for an actual instrument, and a possible port on a lighter operating system frequently used like RTEMS, possibly on a LEON processor.

REFERENCES

[1] Compact dpu software development. Statement of work, European Space Agency ESA-ESTEC, 2004.

[2] Ph. Armbruster. European space technology harmonisation: On-board payload data processing systems. Technical dossier, European Space Agency ESA, 2003.

[3] RTAI: Real-time application interface. http://www.rtai.org.

[4] S. Fleury, M. Herrb, and R. Chatila. GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 842–848, Genoble, France, September 1997.

[5] D. Bovet and M. Cesati. *Understanding the Linux Kernel, Second Edition*. O'Reilly and Associates, Inc, 2003.