AN ADAPTABLE ON-BOARD APPLICATION IN C++

Gaetano Montalto [*], Alessandro Pasetti[**], Nicola Salerno [*]

[*]*Alenia Spazio S.p.A., via Saccomuro 24, 00131 Rome, Italy*
[**]*P&P Software GmbH, Peter-Thumb Str. 46, D-78464 Konstanz, Germany*
montaltog@roma.alespazio.it, pasetti@pnp-software.com, salernon@roma.alespazio.it

Abstract

Although C is well-established as a programming language for on-board applications, the use of its successor C++ is still resisted owing to safety concerns and the difficulty of accommodating its object-oriented features within current software qualification processes. While acknowledging these problems, this paper emphasizes the potential advantages of C++ for on-board software deriving from its support for advanced adaptation techniques that are essential to making software reuse possible. The discussion is made in the context of a project jointly done by Alenia Spazio / Rome (I) and P&P Software (D) for the development of the software for the TTC (Telemetry and TeleCommand) Modem of a digital transponder (TTCM-SW project). A prototype for this application has been entirely developed in C++ using software framework technology.

The paper is ivided into two parts. The first part (sections 1 to 3) presents the rationale for the use of C++ in on-board applications. It argues in particular that this language is, at least in the short term, the ideal implementation medium for a framework approach which is in turn regarded as the most effective way of increasing the level of software reuse. The second part of the paper (section 4) describes the TTCM-SW project and the experience gained from the use on this project of C++ and of framework technology. The paper closes (section 5) with an overview of how the C++ adaptable components have been organized as a repository of reusable building blocks available on a company-wide basis.

1. Software Adaptability – Why?

There are two main reasons why adaptability is an important attribute of software for on-board applications, Firstly, *adaptability fosters reusability* and reusability is in turn important because it is the single most effective way of reducing software costs. To reuse a piece of software means to use it in an operational context other than that for which it was originally intended. Since different operational contexts inevitably impose different requirements, a piece of software is reusable only to the extent that it can be adapted to meet different sets of requirements. Adaptability is therefore the essential pre-requisite for reusability.

Secondly, *adaptability helps model requirement variability* within a single project. In space applications, it is often impossible to freeze all requirements at the beginning of the development process. Usually, some requirements can only be formulated in an incomplete way or must remain completely open, for others it is known that they will be subject to variations, and still others will have to be modified in response to changes at system level. Adaptation mechanisms can help minimize the cost of incorporating such requirement changes into a piece of software late in the development process.

2. Software Adaptability – How?

Adaptability must be built into an application at (at least) three levels. At *implementation level*, adaptability requires the presence of adaptation mechanisms that allow the behaviour of a piece of software to be tuned to match different requirements. At *architectural level,* adaptability requires an application to be designed in such a way as to allow its structure to be modified to match different sets of requirements. At *requirements level*, adaptability requires a project to be specified in such a way as to allow its technical specification to match different customer's (or market's) baselines in the same domain. The implementation language has an impact on the first two levels which are therefore discussed in greater detail below.

An important constraint in the case of on-board applications is that adaptation must be achieved without touching the source code of the software that is being adapted. Changes in the source code require the software to be fully re-qualified. The cost of doing so will often be comparable to the cost of re-developing the software *ex novo* which would therefore destroy the advantage of adaptability.

At implementation level, the adaptation mechanisms that have been proposed over the years range widely in complexity and effectiveness. At one extreme, the simplest kind of adaptation mechanism consists in parameterizing a piece of software (often a single routine) by leaving the value of some key variables open and by setting it either at run-time (e.g. by passing it as a procedure parameter) or at compile time (e.g. through pre-processor macros). At the other extreme, some languages (e.g. Smalltalk) achieve complete adaptability by allowing the code of a running application to be dynamically modified. Intermediate mechanisms include Ada-style generics, C++ templates, inheritance, object-composition, and dynamic class loading.

The adaptation mechanisms available to a particular application are a function of the programming language. In this respect, the two traditional languages used in on-board applications – C and Ada83 – are very limited. C basically only allows adaptation through variable parameterization[1] while Ada83 adds some limited type parameterization through the use of the generic mechanism. Both are unable to model any but the simplest kinds of requirement changes. They are in particular inadequate to support the implementation of design patterns [Gam95] that have proven to be the most powerful means to build adaptability into a piece of software.

The key role of adaptability in making software reusable and the inadequacy of traditional languages to support it, probably explain why the level of reuse has until now been so low in space applications. C++ is in this respect far superior because it supports very flexible template programming and it supports both inheritance and object composition. Template, inheritance and object composition are essentially equivalent and differ primarily in the fact that the former two mechanisms allow static adaptation whereas the latter allows dynamic adaptation.

The project described in the second part of this paper relies on inheritance and object composition. These two mechanisms are illustrated in Figure 1.

[1] Technically, C also allows more complex forms of parameterization through the use of function pointers. This is not considered because, once function pointers are allows, C can become essentially equivalent to (but far less safe than) C++

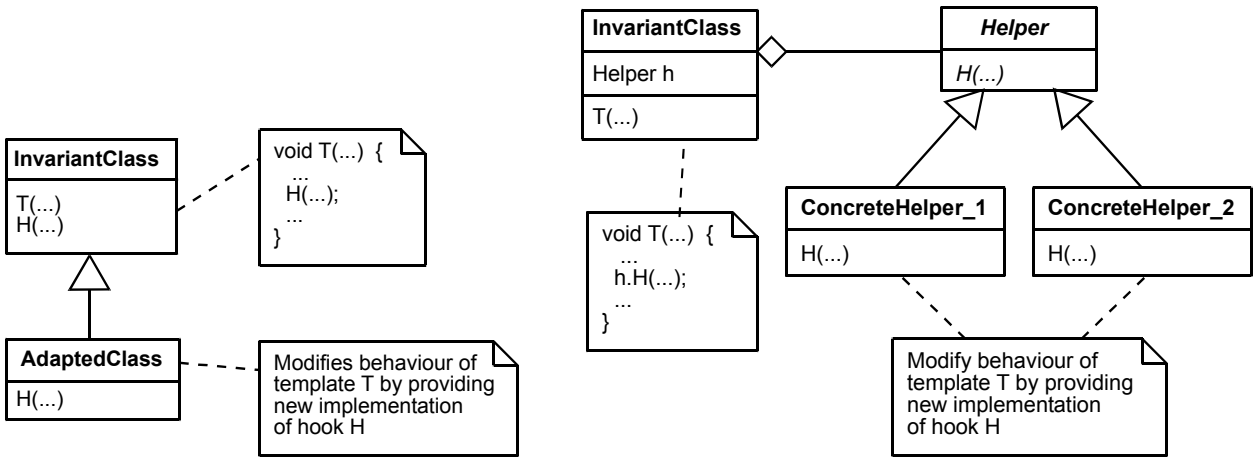

Figure 1: Adaptation Mechanisms for C++ Components

The left-hand side of Figure 1 shows adaptation by inheritance whereas the right-hand side shows adaptation through object-composition. In both cases, an invariant class (the class to be adapted) implements some behaviour in a so-called *template method* T(). The template method is parameterized by a so-called *hook method* H(). Adaptation is achieved by replacing the hook method. In one case, its replacement is achieved by overriding the default implementation proposed in the invariant class with a new implementation in a derived class. In the other case, the hook method is implemented by a plug-in helper class and its replacement is achieved by changing helper class. Note that the replacement of the hook method is achieved without touching the code of the invariant class.

Figure 1 shows a simple case with just one template and one hook method. In practice, a realistic adaptable class will normally have multiple template and hook methods and will therefore offer a high degree of behaviour adaptability and much flexibility in which parts of its behavior are adapted.

From an architectural point of view, adaptation can take place at several levels of granularity. At one extreme, adaptation can be aimed at individual procedures that are designed to be adaptable through, for instance, the use of different values for their parameters or through the use of template parameters. At the next higher level, adaptation could be aimed at clusters of related procedures typically implemented as C++ or Java classes which can be adapted through, for instance, inheritance or template parameters. At a still higher level, adaptation can be aimed at a system of classes together with their mutual relationships. This last type of adaptation is realized through the use of software frameworks [Fay95]. It is by now recognized and accepted that adaptation, to be effective, must be realized at such a system level using framework technology because lower-level adaptation introduces costs that rapidly outweigh the benefits of adaptation.

Use of framework technology has an impact at programming language level, too, because software frameworks - at least in their modern, object-oriented, variant - rely on object composition and inheritance as adaptation mechanisms and use of these mechanisms requires support at language level. Among mainstream languages, the ideal implementation language is probably Java which supports both adaptation mechanisms and which offers a

dedicated construct to represent *abstract interfaces*. This is important because a software framework often uses abstract interfaces to model the points of adaptation where the framework must be adapted to match the needs of individual applications in its domain [Pas01]. The typical class structure of an application built with the help of an object-oriented framework tends to be as shown in Figure 2: an upper layer of abstract interfaces and abstract classes that are provided by the framework and a bottom layer of concrete classes that specialize the abstract classes or implement the abstract interface and implement the application-specific part of the application behaviour.

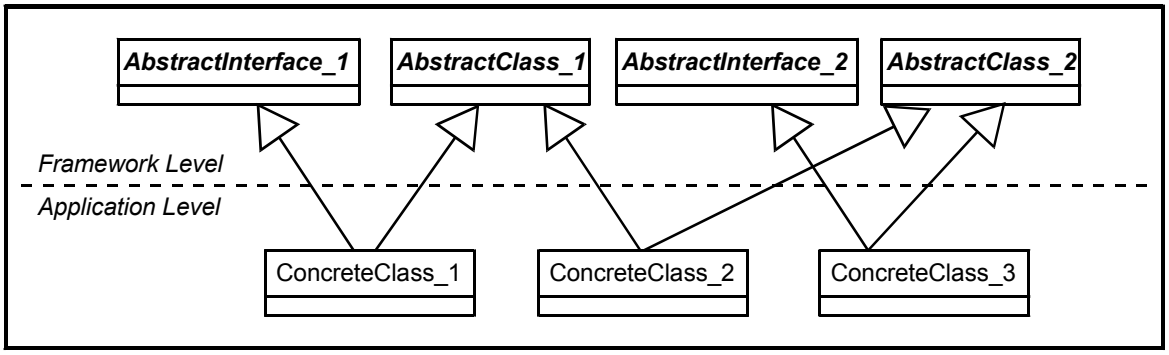

Figure 2: Adaptation Mechanisms for C++ Components

The class structure shown in Figure 2 can be most naturally and safely implemented in Java. Unfortunately, Java is not yet ready for use in on-board systems [Cec02]. In the interim, C++ is regarded as the most appropriate solution. It lacks an explicit support for abstract interfaces but these can be implemented as pure virtual classes and class structures as shown in Figure 2 are fully and naturally supported. Ada95, on the other hand, is somewhat inadequate because it does not easily support complex class trees.

3. C++ for On-Board Systems

The previous section has illustrated the advantages of C++ for on-board applications. In particular, it has argued that C++ is a desirable choice because it supports implementation of advanced adaptation techniques which are especially suitable to foster software reuse. On the down side, C++ is not a very robust language and this is a serious concern in the on-board domain where reliability and predictability of behaviour are essential requirements.

Language assessments however are necessarily comparative in character and the suitability or otherwise of C++ must therefore be decided with reference to other languages that are known to be suitable for on-board applications. These reference languages are at present C and Ada both of which are widely used in satellite applications. From a robustness point of view, C++ is clearly inferior to Ada in both its Ada83 and Ada95 incarnation. This section will therefore concentrate on C as a reference against which the suitability of C++ should be judged.

In the specific context of the TTCM-SW project, which provided the initial motivation for the work described in this paper , there is also a second and more pragmatic reason for ignoring Ada. Ada was never considered as a practical alternative to C++ because of lack of trained

personnel and because of its high development costs. Additionally, this project was intended to contribute to the development of long-term reusable assets and Ada was felt not to be an appropriate choice because there are doubts about its long-term viability.

When compared against C, there are four perceived drawbacks to the use of C++: (1) lack of coding guidelines suitable for on-board application, (2) lack of language subsets suitable for on-board applications, (3) high memory footprint, and (4) difficulty of reconciling object-oriented constructs with the ECSS-E40 qualification process. Note that tool support is not an issue because practically all mainstream commercial tools – for design, UML modelling, code analysis, etc – that support C also support C++.

The experience done by Alenia-Spazio in the TTCM-SW project and the experience done by P&P Software in other projects is that only the fourth drawback is real. The first three drawbacks are due to misconceptions about or misuse of C++. The four drawbacks are examined more in detail in the following subsections.

3.1 Coding Guidelines for C++

Several guidelines for the use of C++ in mission- and safety-critical projects exist. One of the oldest dates from 1996. It was produced by NIST and specifically targeted the use of C++ for safety-critical applications. It is publicly available from [Bin96]. These guidelines also include a collection of techniques that can be incorporated into C++ programs to make them safer. Guidelines for C/C++ more specifically targeted at the space industry have been produced both by CNES [Cne96] in 1996 and more recently by ESA [Bss00] in 2000. Additionally, a currently on-going TRP study by ESA investigates software robustness issues and should, among other things, produce an updated set of coding rules for C/C++.

For the work described here, the ESA rules were used as baseline. However, a tailoring was made to adapt them to the purposes of the project. Tailoring consisted both in rejecting a small subset of the rules proposed in the ESA document and in proposing new additional rules. This type of tailoring is important because the ESA rules appear to have been implicitly written for an object-based (rather than *object-oriented)* programming paradigm and are not specifically targeted at the case of *on-board* applications. Among these new rules, some of the most significant cover the following topics:

- *Object Creation*: all objects are created during the application initialization phase. No dynamic creation of objects, either on the heap or on the stack, is allowed. This ensures that the pool of objects used by the application is fixed and stable. Compliance with this rule can usually be statically checked by the constructor if the class destructors are declared `private` or `protected`.
- *Object Manipulation*: objects are always and exclusively manipulated through their pointers. In particular, assignment of objects (either direct or indirect by passing an object as a method parameter) is forbidden. Compliance with this rule can be checked by implementing the copy constructor and class assignment operator to throw an assert error.
- *Pointer Manipulation*: use of pointers to primitive types is minimized and in particular is forbidden as method parameters. This also means that arrays are never passed as method parameters.
- *Error Handling*: use of the C++ exception mechanism is forbidded by the adopted language subset (see section 3.2). Coding rules were instead defined to enforce uniform

treatment of error conditions through the generation of standard error reports that are stored in a globally accessible repository.

- *Use of Assertions*: standard C++ assertions are used to check the legality of all method parameters and, where appropriate, to check the value of method invariants.

Adherence to the above rules eliminates many types of common C/C++ errors. In particular, the first three rules taken together virtually eliminate the problem of memory leaks and dangling pointers that so often plague C/C++ applications.

Although these were not used in this project, many commercial code-analysis tools exist that can verify that the coding rules adopted in the project are complied with.

3.2 Language Subset

Among mainstream languages, C++ is probably the most complex. Its high level of complexity and certain ambiguities in its definition make it difficult to construct fully compliant compilers. On-board applications should therefore be restricted to small, safer and unambiguous language subset. The project described here used the Embedded C++ [Emb02] language subset as a starting point for defining a more specific language subset that, though very narrow, was found to provide sufficient flexibility to support the required degree of adaptability. The only syntactical features of the C++ language that were used are:

- *Classes*: classes are used to encapsulate logically related data and the functions to manipulate them.
- *Virtual Methods*: virtual methods are used as an adaptation mechanism to allow the default behaviour encapsulated in a framework class to be adapted to the needs of a specific application.
- *Pure Virtual Methods*: pure virtual methods are used to mark adaptation points for which no default implementation is provided.
- *Single Inheritance*: single inheritance is used for two purposes: to provide concrete implementations for a pure virtual class and to adapt a concrete class by overriding one or more of its virtual methods.
- *Method Inlining*: method inlining is used to improve execution speed.
- *Constant Methods:* constant methods are used as a safety feature to prevent some methods from modifying class variables.
- *Constant Variables:* constant variables are used as a safety feature to enforce the non-modifiability of some variables.

Note that all the above constructs belong to the safe and unambiguous part of C++.

3.3 Memory Footprint

C++ applications - especially in the desktop world - sometimes have large memory footprints. This has three causes: the presence of exception handling code, the inclusion in the executable module of information for run-time type identification (RTTI), and the comparatively large size of C++ libraries. However, most compilers (and in particular the gcc compiler used for the ERC32 processor) allow the exception handling code and the RTTI information to be left out and on-board applications only need the core C++ library that, in the

gcc case, has a rather modest size[2]. Figures for the memory footprint of the TTCM-SW application are given in the next section and they demonstrate that a C++ application for space can be made to fit in 32 kBytes of PROM or less. There is no reason why C++ cannot be made as efficient in its memory usage as C.

3.4 ECSS-E40 Qualification Process

The ECSS-E40 qualification process is inadequate to support framework-based software development as understood here for two reasons. First, the ECSS-E40 standard has been defined with an object-based model for the target application. Thus, issues that are specific to object-oriented development such as the use of inheritance or dynamic binding are not well covered. Second, the ECSS-E40 standard is primarily aimed at the development of one-off applications. The development of artifacts like software frameworks that are intended as architectural skeletons from which applications can be instantiated and that include abstract constructs like abstract interfaces and design patterns are poorly covered by the standard.

These and other issues are part of the broader problems of adapting software certification processes to the use of object-oriented features and to framework technology. These problems are still open. The adaptation to object-orientation in particular is currently under discussion in the US (see for instance [Oot02]). However, neither of these problems is in any way specific to C++. The same problem apply to any attempt to use object-oriented techniques in a mission-critical context, regardless of the language that it used.

3.5 Summary

The use of a general purpose language in an on-board context always requires some precautions being taken. The considerations advanced in this section can be summarized by saying that users who are prepared to accept C as an on-board language have no reason to reject C++. C++ can be made (at least) as robust and as efficient as C.

By way of conclusion, it may be noted that Honeywell have recently completed the qualification to level A according to the DO-178B standard of an RTOS written in C++ [Cof02]. This is incontrovertible evidence that C++ can be used for even the most critical types of applications.

4. The TTCM-SW Project

This project aimed to develop the software for the prototype of a TTC Modem of a digital transponder. Alenia Spazio, as part of a drive to encourage the software reuse in the on-board applications, have started this as a pilot project where framework technology is used from the beginning of the software lifecycle.

The TTCM-SW software was developed by instantiating a software framework specifically aimed at the digital transponder domain (the TSP Framework). This was the second framework used at Alenia-Spazio. The first one was developed for antenna pointing controllers (the APC Framework, see [APC01]). Both frameworks were provided by P&P Software as customization of one of their internal products. The remainder of this section discusses the use of the TSP Framework to instantiate the prototype TTCM-SW application.

[2] Of required, this can be further reduced by recompiling the library to leave out exception handling and RTTI code.

A digital transponder is one of the elements of the TM/TC channel which allows communications between the satellite's host computer and ground, as simplified Figure 3. A digital transponder consists of more elements (modules); one of these is the TTC Modem where the TTCM-SW runs. It is in charge to perform the following main tasks:

- DSP algorithms implementation (such as FFT and so on)
- TM/TC handling (from/to host/ground)
- Internal ASIC's set-up
- Hardware initialization & health-checking
- FDIR (Failure Detection Isolation & Recovery)
- Software tracing

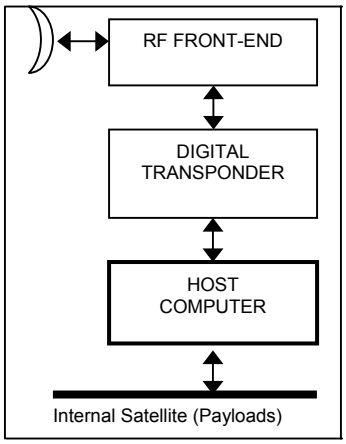

Figure 3: Structure of a Digital Transponder

The prototype TTCM-SW discussed here is based on a Nios™ CPU development board featuring the Nios™ embedded processor (see [Alt02]). The TTCM-SW interacts with the environment consisting of the DSP section of the digital transponder, quite fully implemented by means of an ASIC component (other parts consist of standard digital circuitries). The interaction with ASIC and DSP's circuitry is performed by exchanging data through proper interfaces (a set of memory mapped registers) as shown in Figure 4.

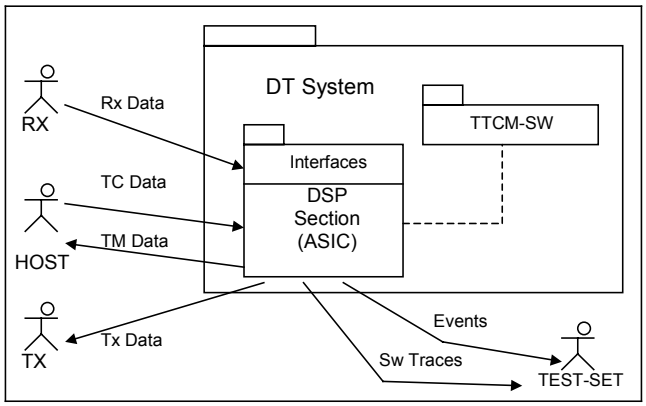

Figure 4: Interaction of TTCM-SW with its Environment

The Figure shows how various "actors" interact with the digital transponder (DT) "System". They can be identified as follows: RX (Front-End, Rx side), TX (Front-End, Tx side), HOST and TEST-SET. The **RX actor** sends, periodically, a "data sample ready" message to the

TTCM-SW that - in turn - configures properly the RX itself; the **TX actor** is configured directly by the TTCM-SW. The **HOST actor** sends to the TTCM-SW one or more telecommand messages and receives from the TTCM-SW a set of telemetries. The **TEST-SET actor** (relevant to the external test environment) receives from the TTCM-SW one or more "sw-traces" (i.e. byte streams generated for diagnostic and/or test purposes) and captures messages relevant to occurred "events".

The functionalities of the TTCM-SW that were developed with reusable building blocks provided by the TSP Framework include:

- Implementation and management of the state machines
- Connections with the ASIC and other peripherals
- Management of event reports
- Implementation and management of the on-board database

The telecommand and telemetry management have not yet been implemented (in this type of application they are very simple) but the necessary components too could be provided by the TSP Framework.

The TCMM-SW had to satisfy very demanding memory and CPU constraints. In the past, this type of application was coded in assembler and had to fit on PROM chips of 8 kBytes. At present, the smallest PROM chip is 32 kBytes. The table below gives the PROM memory footprint for two recent examples of transponder applications developed in assembler and for the TTCM-SW application developed in C++ using the TSP Framework:

Application #1 *(Assembly)*	**Application #2** *(Assembly)*	**TTCM-SW** *(Framework / C++)*
6 Kbytes	8 Kbytes	20 Kbytes

The table indicates an expansion factor of about 4 in going from assembler to framework and C++. This expansion is partly accounted for by the use of a higher level language and partly by differences in requirements among the three applications.

Transponder software operates on a cyclical basis. In the TTCM case, the cycle frequency can be up to 10 kHz. Tests with the prototype TTCM-SW application done on the target processor running at 33 MHz indicate that the cycle frequency could be increased up to 47 kHz before overruns occur. Thus, any overheads that may have been introduced by the use of C++ and of framework technology can be comfortably accommodated with the target application requirements.

It can thus be concluded that the tests performed on the prototype TTCM-SW indicate that the cost in terms of memory and CPU usage of using of framework technology implemented in C++ is very limited and remains compatible with even severely resource-constrained applications.

5. Conclusions

The TTCM-SW prototype represents an instance of systematic software reuse. The TSP Framework, from which the prototype application was instantiated, is available on the intranet of Alenia-Spazio and is available to all Alenia engineers irrespective of their location (Turin, Rome and L'Aquila). The TSP Framework, together with its sister APC Framework, can thus be seen as a ***software framework repository*** that makes available ***adaptable building blocks*** for on-board applications. This is illustrated in Figure 5.

It must be stressed that the building blocks provided by the frameworks are more than just components. They also include abstract interfaces and design patterns. Crucially, the framework repository also makes available documentation items that are ready to be directly incorporated within the documentation of the target applications. This approach was successfully tested in the TTCM-SW project where the preparation of the software documentation was made significantly more efficient and faster by the possibility of drawing on ready-made documentation items that are provided by the TSP Framework repository.

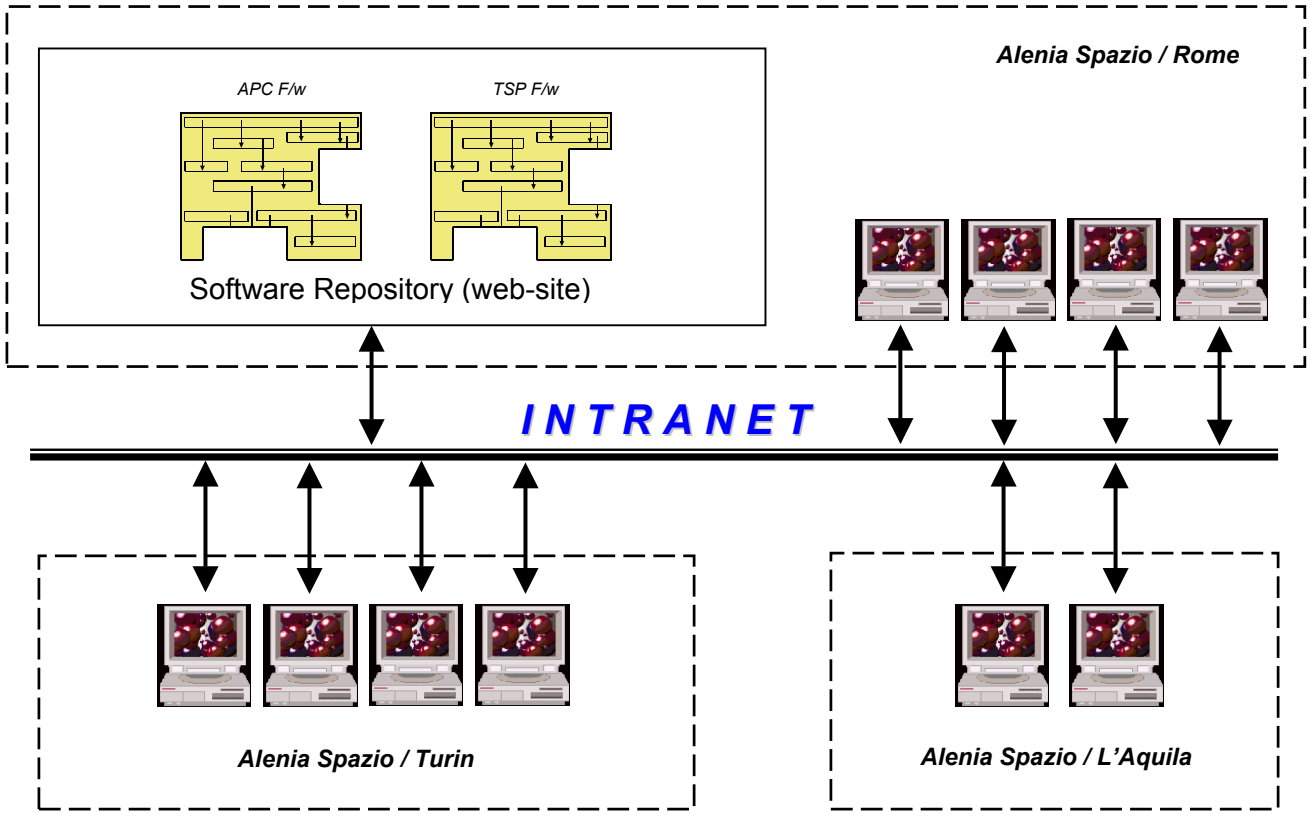

Figure 5: Framework Repository on the Alenia Spazio Intranet

The use of C++ played a crucial role in realizing the concept outline above. The switch from older languages like C and Ada83 is essential to effect a transition from a modular approach of software development to an adaptable approach that puts the emphasis on the development of adaptable artefacts – in particular adaptable components and architectures.

C++ provides the necessary support at language level for achieving the required degree of adaptability and the experience described in this paper demonstrates that C++ is a viable choice for on-board applications.

6. References

[Alt02] Altera, *NIOS 32-Bit Programmer's Reference Manual*, Version 2.2, 2002

[Bin96] D. Binkley, *C++ in Safety Critical Systems,* NIST Report IR 5769, [available for download from: hissa.nist.gov/sw_develop/ir5769/ir5769.l.html]

[Bss02] Esa Board for Software Standardization and Control (BSSC), *C and C++ Coding Standards*, BSSC(2000)1, Issue 1

[Cec02] V. Cechticky, A. Pasetti *Real-Time Java for On-Board Systems,* Proceedings Data System in Aerospace (DASIA2002) Conference; May 2002, Dublin, Ireland

[Cof02] D. Cofer, M. Rangarajan *Formal Modeling and Analysis of Advanced Scheduling Features in an Avionics RTOS*, p. 138, LNCS Vol. 2491, Springer-Verlag

[Cne96] MPM-53-00-15, *Regles essentielles pour l'utilisation du Langage C++,* Ed. 1, Rev. 0, June 1996

[Emb02] Embedded C++ Web Site, http://www.caravan.net/ec2plus/

[Fay99] M. Fayad, D. Schmidt, R. Johnson (eds) *Building Application Frameworks – Object Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1995

[Gam95] E. Gamma et. al., *Design Patterns*, Addison Wesley, 1995

[APC01] M. Montalto, A. Pasetti, N. Salerno, *Application of Software Framework Technology to an Antenna Pointing Controller,* Proceedings of the Data System In Aerospace (DASIA2002) Conference; May 2002, Dublin, Ireland

[Oot02] Object Oriented Technology in Aviation (OOTiA) Workshop, http://shemesh.larc.nasa.gov/foot/

[Pas01] A. Pasetti et al., *An Object-Oriented Component-Based Framework for On-Board Software*, Proceedings of the Data Systems In Aerospace (DASIA2001) Conference, Nice, May 2001,

[Pas02] A. Pasetti, *Software Frameworks and Embedded Control Systems*, LNCS Vol. 2231, Springer-Verlag, 2002