CERTIFICATION OF COMPONENT-BASED PARTICLE THERAPY SOFTWARE

Carlos F. Afonso^{a,b}, Luigi Casalegno^b and Cristiana Larizza^a

^a *Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Lombardy, Italy*

^b *Centro Nazionale di Adroterapia Oncologica, Pavia, Lombardy, Italy*

ABSTRACT

The operation of particle therapy facilities requires complex control systems, often spanning several software development environments, each containing large number of applications. While the component-based development approach brings many benefits, institutions often struggle integrating it during the certification process.

As part of the technological update of one of its control system environments, the Centro Nazionale di Adroterapia Oncologica (CNAO) treatment center chose to develop the software applications of this environment in a component-based approach, while focusing on strategies to also optimize the certification process.

In this document, we examine the challenges of compositional development in the medical software certification process. Additionally, the approach taken by CNAO during the update of one of its control system's environments is presented, along with the evaluation of the expected impact on its certification as medical software.

KEYWORDS

Software Engineering, Medical Software, Certification

1. INTRODUCTION

Under the regulatory environment for medical software in member states of European Union, certification is accredited on the basis of complying with the development process defined in the standards IEC 62304+Amendment1(IEC, 2006, 2015), ISO 14971(ISO, 2007), and ISO 13485(ISO, 2016). Since certification is based around complying with the processes defined in the certification standards, applications that share software components are still required perform the certification tasks independently.

Still, several authors propose architecture strategies for efficient validation and certification of critical software that reuses components (Nadkarni and Miller, 2007; Hatcliff *et al.*, 2012). This document focusses on the component-based approach, which proposes an architecture that emphasizes separation of concerns and well defined interfaces (Liu *et al.*, 2005).

In a particle therapy facility for cancer treatment, such as CNAO, control system software performs a wide range of functions such as the control of the accelerator subsystems, dissemination of treatment parameters, and facility configuration. These software packages are often separated into groups, depending on which equipment they run on top of, or their role in the control system. The slow application environment is one of the software environments in CNAO, it is comprised of several applications that share the same development environment and possess similar constraints. Such applications do not generally possess real time constraints and typically perform tasks such as accelerator configuration, data management, or specialized monitoring. At CNAO, applications in the configuration and support environment are developed by the control systems group, are in general expected to receive a low safety class, but are numerous. The work here presented is a part of the OMA project (Cockcroft Institute, 2016) and aims at the design and development of a software infrastructure that the control systems group will exploit for the development of large amounts of such applications.

In this paper we present in detail the compositional approach used in the development of CNAO's configuration and support environment software, which, in addition to improving the reliability and maintainability of these applications, is aimed at reducing the amount of time required for validation and certification steps of the development process.

2. INTEGRATING REUSABLE COMPONENTS INTO MEDICAL APPLICATIONS

In an ideal world, reusable components would be certifiable and then reused without requiring extra certification steps. However, this is not the case in safety critical software (Rushby, 2002). Certification process standards for medical software dictates that risk analysis has to be performed for the software system as a whole, and that risk originated by a component failure can only be determined once the component has had its role defined in regards to the software architecture (IEC, 2006).

Because of this, separation of concerns between software components is not sufficient to allow for reuse of validation and certification artefacts without taking the entire system architecture into consideration. However, well applied compositional development strategies should be able to facilitate the integration of reusable components into new systems (Åkerholm and Land, 2009), and there are several currently adopted industry practices to facilitate the integration of reusable components in safety-critical systems (Land, Åkerholm and Carlson, 2012).

After reviewing the guidelines provided in certification standards and those presented by several authors, we have developed a strategy for designing the new configuration and support environment, outlined in this document. This approach focuses on enabling software reuse, while minimizing the total effort for certification. Later, we evaluate this approach based on the certification standards.

2.1 CNAO's Component-based Development Approach

In order to minimize the effort and tasks necessary for development and certification of future configuration and support environment applications, two main strategies were applied while designing the infrastructure necessary for the software development. The first is the separation of reusable software elements into well-defined *components*, implemented as *libraries*. The second is the definition of a *reference architecture* that is to be followed by all future configuration and support environment applications.

2.1.1 Libraries as Reusable Components

After careful analysis, the following operations have been designed into libraries, becoming reusable components, referred in this document as *standard service libraries*:

- *External communication*, such as database access, local network filesystem access, and communication with other control system equipment, via the OPC-UA (Mahnke *et al.* 2009) protocol.
- *User authentication and authorization* using the OpenID Connect standard (Sakimura *et al.*, 2014).
- *Other standardized operations*, such as system logging and configuration.

2.1.2 Standardization of Software Architecture

In order to provide a set of software patterns that should be followed by developers, we have defined a three-tiered *reference architecture* for configuration and support environment applications. The reference architecture describes each software layer (Presentation, Model, and View Layers), the relationships between each layer, and what software libraries should be used for each part of the final application. By understanding the reference architecture, the facility's developers are able to understand the architectural patterns enforced in the configuration and support environment, and have shared terminology to use when discussing the application's architecture.

A diagram illustrating a class diagram from the reference architecture is shown in Figure 1(a). In order to describe the proposed architecture, we start from the Presentation Layer. The *Presentation Layer* follows the MVVM (Smith, 2009) software pattern. This pattern decouples presentation layer code from domain classes, and allows model and view model classes to be fully unit tested (Smith, 2009), thus aiding the verification process.

We have developed the Framework2020 set of libraries to support the usage of the reference architecture. It provides dedicated base class definitions for MVVM, as well as access control and initialization of the standard services. This is done to assist developer in complying with the reference architecture, as well as segregate the application domain classes from the services used by them.

In the *Domain Layer*, developers should define domain specific classes to implement the final application's features. Very few constraints are placed upon these classes by the architecture, allowing developers to decide how to organize their application. The Specific Domain Classes address the domain requirements of final application. These classes have access to data via the standard services, present results to users via the presentation layer, or communicate/persist the results using the standard service libraries.

Figure 1. (a) Architecture of a configuration and support environment application (left). (b) Application development lifecycle (right)

2.1.3 Application Skeleton Generator

In order to make the applications development easier and more reliable, developers will be provided with an out-of-the-box configured implementation of the reference architecture, by a configurable skeleton generator, currently in development. It should allow developers to specify the general requirements of their configuration and support environment application, such as the application layout, navigation method and standard services required following the lifecycle depicted in Figure 1 (b). By enforcing the development lifecycle developers have to design their application around a custom-made skeleton that is compliant with the reference architecture.

2.1.4 Evaluation Plan

After reviewing the proposed approach and verifying the usability of the application generator, we have decided to proceed with the development approach, using the previously mentioned reference architecture as basis of the configuration and support environment's new applications. In the following months, after the development of the first applications is concluded, a testing and validation phase begins, in which the medical certification documentation is to be prepared. During this phase, the approach presented in this paper is to be evaluated in comparison to previous medical certification for software projects of the CNAO facility. In particular, the evaluation shall focus on two main aspects: First, instances of reuse of certification artefacts shall be documented, as well as the effort necessary for adapting them. Secondly, the overall certification process shall be compared by querying the certification team about the length and complexity of the certification process.

3. DISCUSSION

The IEC 62304/Amd1 certification standard emphasizes that, when defining the software architecture, strategies for component segregation should be used to avoid unsafe interactions between them. Additionally, when credible arguments can be given to justify that the architecture separates components into separate software items in a correct manner, these software items may receive a different software safety class from the system (IEC, 2006). The certification process has different levels of granularity depending on safety class, and some certification activities may even not be required for low safety levels (Hrgarek, 2012).

By implementing commonly used software operations into reusable components, these become well defined software units, often of the lowest safety class (class A). Thus, several certification tasks are not mandatory for these units, such as the task 5.4.2, entitled “Develop detailed design for each software unit” (IEC, 2006). Additionally, we argue that certification artefacts from developing and certifying reusable components can be the basis for their counterparts in the final application, such as unit-test verification (activity 5.5), and establishment of a software maintenance plan (activity 6.1).

Limitations of the component-based approach were also identified. Namely, the scope of most certification tasks is the software system as a whole, therefore a compositional approach is not able to affect the granularity or necessity of system wide tasks.

Regarding the usage of a reference architecture, we argue that its adoption and enforcement reduces effort required for certification. The importance of knowledge of the system’s architecture in the certification process is highlighted multiple times. Such as in the annex B of the IEC 62304 standard, where it is stated that, concerning software safety classification, risks associated with each software item can only be determined once the software architecture “defines the role of the software item in terms of its purpose and its interfaces with other software and hardware items” (IEC, 2006). Furthermore, in the context of the activity 7.1, which is the analysis of software contributing to hazard situations process activity, the guidelines argue that hazard situations can only be fully identified once the software architecture has been designed, and, only then, proposed risk control measures can be completely evaluated (IEC, 2006).

By defining the reference architecture and enforcing it during the design of applications, developers have access to a ‘used-and-tested’ architecture, accompanied by several internal documents explaining its usage, interaction between components and validation procedures. As developers design configuration and support environment applications using the enforced general architecture, the arguments provided to justify the safe interaction between developer modules and reusable components should be written using the reference architecture’s documentation as basis.

Altogether, the authors argue that a composition-based approach provides clear benefits to the development process and that, when managed correctly, reduce the effort related to the medical software certification process. In the future, we plan to empirically evaluate the impact of the presented development approach during the development and certification of the new Accelerator Cycle Designer application.

4. CONCLUSION

We have analyzed challenges of compositional development in the certification process and presented the approach taken by the CNAO facility during the ongoing technological update of its control system.

While composition-based development provides several benefits, its application in medical software creates a few challenges that must be addressed. With the approach presented in this document, we expect to reduce the effort necessary for complying with the certification process of software applications in the new control system environment at the CNAO facility.

We conclude that, while the great majority of certification tasks must still be performed for component-based applications, we have identified several areas where the certification artefacts of components are able to support the activities required for medical certification of these applications. This will be verified during the validation we have planned in the coming months.

ACKNOWLEDGEMENT

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 675265.

REFERENCES

- Åkerholm, M. and Land, R., 2009, Towards systematic software reuse in certifiable safety-critical systems, *RESAFE-International Workshop on Software Reuse and Safety*, Falls Church, VA, United States.
- Cockcroft Institute, 2016, *OMA Project*, Available at: <https://www.liverpool.ac.uk/oma-project/about-us/> , Accessed: 24 May 2019.
- Hatcliff, J. *et al.*, 2012, Rationale and architecture principles for medical application platforms, *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pp. 3–12.
- Hrgarek, N., 2012, Certification and regulatory challenges in medical device software development, *Proceedings of the 4th International Workshop on Software Engineering in Health Care*, pp. 40–43.
- IEC, 2006, *IEC 62304:2006 - Medical device software -- Software life cycle processes*.
- IEC, 2015 *IEC 62304:2006/Amd 1:2015 - Medical device software - Software life-cycle processes / Amd 1*.
- ISO, 2007 *ISO 14971:2007 - Medical devices -- Application of risk management to medical devices*.
- ISO, 2016 *ISO 13485:2016 - Medical devices -- Quality management systems -- Requirements for regulatory purposes*.
- Land, R., Åkerholm, M. and Carlson, J., 2012, Efficient software component reuse in safety-critical systems--an empirical study, *International Conference on Computer Safety, Reliability, and Security*, pp. 388–399.
- Liu, J. *et al.*, 2005, Component-based Medical and Assistive Devices and Systems, *Proceedings of High Confidence Medical and Systems (HCMDSS) Workshop*. Philadelphia, PA, United States.
- Mahnke, W., Leitner, S.-H. and Damm, M., 2009, *OPC unified architecture*, Springer Science & Business Media.
- Nadkarni, P. M. and Miller, R. A., 2007, Service-oriented architecture in medical software: promises and perils, *Journal of the American Medical Informatics Association*, Vol. 14, No 2, pp. 244–246.
- Rushby, J., 2002, *Modular certification*. Menlo Park, CA, United States. Available at: <https://ntrs.nasa.gov/search.jsp?R=20030001129>.
- Sakimura, N. *et al.*, 2014, *OpenID Connect Core 1.0 incorporating errata set 1*, The OpenID Foundation, specification. Available at: https://openid.net/specs/openid-connect-core-1_0.html (Accessed: 27 March 2019).
- Smith, J., 2009, Patterns-wpf apps with the model-view-viewmodel design pattern, *MSDN magazine*, 72.