ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA003

INTEGRATING MOBILE DEVICES INTO CNAO'S CONTROL SYSTEM, A WEB SERVICE APPROACH TO DEVICE COMMUNICATION*

C. Afonso[1], C. Larizza, University of Pavia, Pavia, Italy
S. Foglio, S. Gioia, M. Necchi, Centro Nazionale di Adroterapia Oncologica, Pavia, Italy
S. Toncelli, Consultant on behalf of Centro Nazionale di Adroterapia Oncologica, Novara, Italy
L. Casalegno, Consultant on behalf of Centro Nazionale di Adroterapia Oncologica, Como, Italy
[1] also at Centro Nazionale di Adroterapia Oncologica, Pavia, Italy

Abstract

The Italian National Hadrontherapy Center (CNAO) is a cancer treatment facility that employs a synchrotron to accelerate charged particle beams.

The configuration and support environment of CNAO's control system is responsible for managing the repository, configuring the control system, as well as performing non-real time support operations. Applications in this environment interface with the relational repository, remote file systems, as well as lower level control system components. As part of the technological upgrade of the configuration and support environment, CNAO plans to integrate mobile applications into the control system.

In order to lay the groundwork for the new generation of applications, new communication interfaces had to be designed. To achieve this, a web services approach was taken, with the objective of standardizing access to these resources. In this paper we describe in detail the update of the communication channels. Additionally, the solutions to challenges encountered, such as access management, logging, and interoperability, are presented.

CURRENT CONFIGURATION AND SUPPORT ENVIRONMENT

The current physical architecture of the control system is presented in Figure 1. This figure displays the physical levels of the control system, specifying the hardware equipment and role of each level [1]. Additionally, the diagram presents the data transfer periodicity of the communications between components.

The first layer contains the several types of applications. The largest component present in this level is the collection of WinCC SCADA (Supervisory Control and Data Acquisition) [2] applications. Also present are Virtual Instruments, written in LabVIEW, which provide a graphical interface used by operators to manage subsystems that are currently not integrated into the SCADA. Additionally, the first layer also contains the configuration and support applications with a graphical user interface.

In the second layer, the WinCC SCADA system applications are responsible for obtaining data and alarms from the third layer, as well as delivering these to the first level's SCADA terminals. Operational data from the accelerator is archived at this level in a SCADA database. This layer is also where the repository and repository services reside. The repository is an Oracle DB cluster containing data for running the facility. This data includes the physical characteristics of the accelerator, configuration settings for software applications, accelerator settings for all currently available charged beams, and information to link patient identifiers to their scheduled treatments.

The third layer is responsible for performing data transfer between the second and forth layers through the OPC and OPC-UA protocol. Finally, the fourth layer, the closest to the accelerator equipment, possesses the strictest real-time requirements, and contains hardware and software to manage each subsystem.

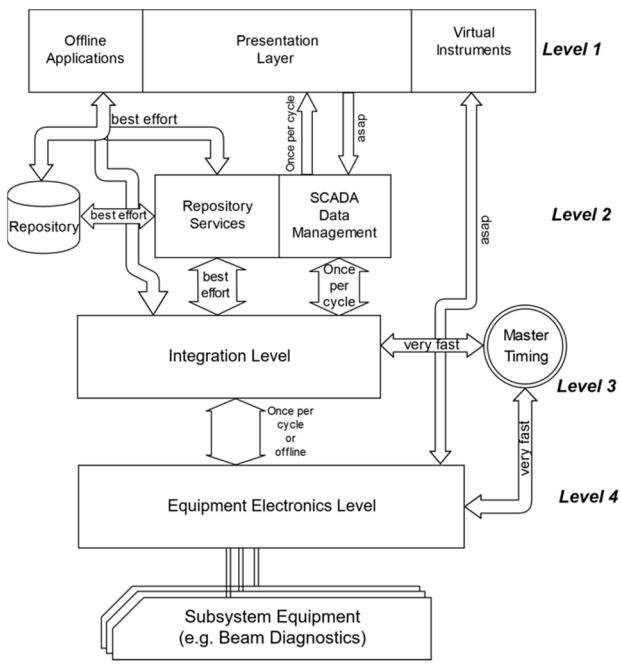

Figure 1: CNAO's Control system, adapted from [1].

The Data transfer frequency between the fourth and third layer is synchronized with events generated by the Timing System. Namely, once per acceleration cycle, the data in the fourth layer is refreshed, and can safely be read during the time period between the two synchronization events. From the third layer upwards, the periodicity

* 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.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA003

of data access depends on the type of application. Data access to the SCADA supervisor possesses real-time constraints and occurs once per cycle. Offline applications and repository services typically access the third layer through OPC-UA without real-time constraints.

The configuration and support environment is a set of C# applications in the first and second layer of the control system, represented in Figure 1 as the "Offline Applications" and "Repository Services". This environment's applications manage the repository and perform support roles in the facility. There are currently over 60 applications in the configuration and support environment. Previously, these applications could only target desktop workstations.

UPGRADE OF THE CONFIGURATION AND SUPPORT ENVIRONMENT

The project presented in this document is the technological upgrade of the configuration and support environment. The objectives of the upgrade technological upgrade were the following:

- Allowing mobile devices to be used in the control system. In order to achieve this, applications of the upgraded environment should target multiple platforms, namely Windows 10 and Android.
- Performing technological upgrade of the environment. The legacy configuration and support environment was designed in 2003. This project aims at integrating newer development frameworks and libraries into the environment.
- Services designed for these applications should be developed in a multi-platform solution and provide an open interface, consequently allowing them to be reused in applications that target all devices expected in the upgraded environment.
- Aiding the medical certification process through the improvement of the testability and maintainability in applications of the upgraded environment.

In order to fulfil the project's objectives, the technological upgrade has been performed via the following steps:

- Selection of Technologies and libraries to be used in the new applications.
- Design of a product line architecture [3] to be followed by the new applications. The product line architecture defines the general architecture of future applications, as well as variation points to be filled by developers.
- Design and development of services and tools for supporting the future applications.

Development Environment

The Xamarin [4] development framework was chosen for the applications and libraries of the upgraded control system environment. By developing applications using Xamarin with the .Net Standard code sharing method, developers define shared library projects that can only contain multi-platform logic [5]. These libraries can then be used by single-platform projects, in C#, that in this project will target the Windows and Android platforms.

Using this sharing method, the .Net Standard library can contain all code that implements non-platform specific logic, including the user interface. Figure 2 contains a diagram illustrating the dependencies between executable projects and libraries in the development of multi-platform applications for the upgraded environment. However, in the case that these applications are only required to target one of the platforms, only one executable project will be created in the solution.

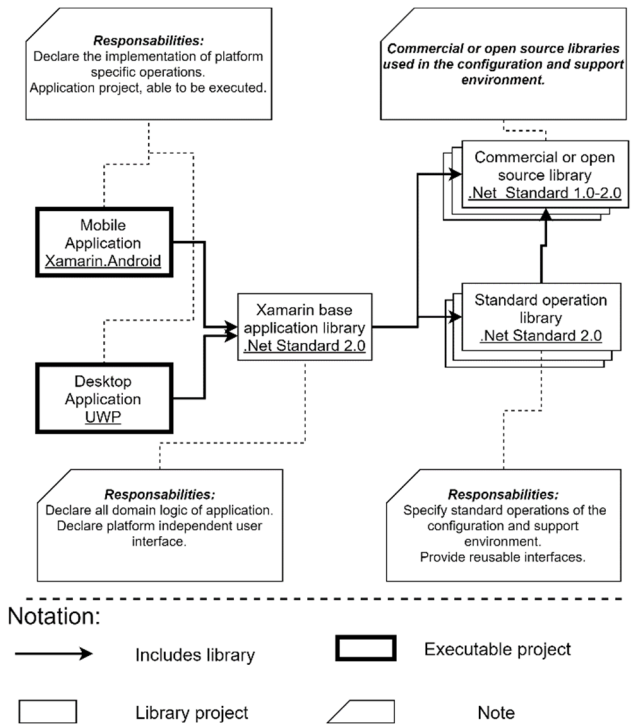

Figure 2: Dependencies between libraries and executable projects in the upgraded configuration and support environment.

Application Architecture

In order to assist control system developers to create applications for the upgraded environment, a product line architecture was designed. This product line architecture defines the general structure of applications in the upgraded environment, as well as several variation points. In these variation points, developers can define their own components to be used, according the application's specific requirements. Figure 3 presents a class diagram of the designed product line architecture, illustrating the classes used by these applications, and the relationship between each software layer.

As defined in this product line architecture, the separation between the presentation and the domain layers is ensured by the usage of the MVVM software pattern [6]. Also, the architecture dictates the usage of the dependency injection pattern [7] for the initialization of services and dependencies. Additionally, a framework library is provided to developers, containing classes for initializing

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA003

the application, as well as base classes for several of the software patterns dictated by the product line architecture.

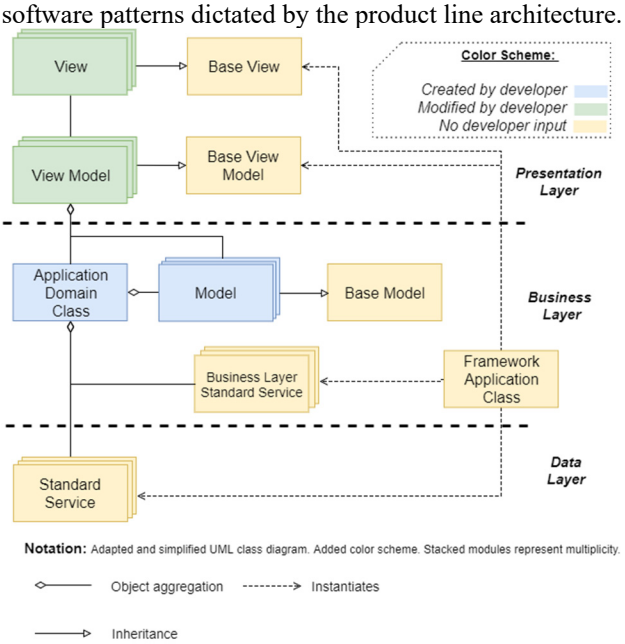

Figure 3: Diagram of the product line architecture for the configuration and support environment applications.

Finally, standard service client library classes are also present in the diagram. These classes are available to product line applications and implement domain and data layer operations. In the next section we describe the web services and client libraries developed during this project.

DEVELOPMENT OF THE STANDARD SERVICES

The standard services were developed to facilitate the communication between the future applications of the configuration and support environment and the resources they depend on. They are composed of several servers and client libraries.

In the legacy environment, the communication between applications and resources was performed through client libraries, sometimes aided by tools present in the desktop workstations they resided in. Because these tools are unavailable to applications running on mobile devices, the standard services had to be developed using a different approach. The following standard service libraries and servers have been implemented as part of this project:

- **RDAS (Relational Data Access Service)**: Web service that exposes a RESTful API for accessing the repository, as well as a client library that performs object-relational mapping and interfaces with RDAS server instances.
- **FDAS (File Data Access Service)**: This service is composed of a web service that exposes a RESTful API to allow applications to access a set of files in the server. Additionally, a client library has been developed to interface with the server.
- **CNAO Identity Provider**: An OpenID Connect [8] Identity Provider that performs authentication of users and clients. A client library designed to interface with the identity provider and manage the tokens obtained.
- **OPCUASiprod**: Standard service library designed to perform communication with OPC-UA servers in the third layer.
- **CnaoLog**: Standard service library designed to perform local and remote logging. Remote logging can currently be performed into RDAS and FDAS server instances.

In this section, we describe the most important design decisions in the development of these standard services. In addition, their expected impact on the software environment is presented.

RDAS

The RDAS is composed of a server and a client library. The server interfaces with the repository and exposes a RESTful API that is accessible to authorized clients. The RDAS client library was designed for two main purposes: implementing the communication with the server and allowing the repository database data to be manipulated in an object-oriented manner.

The RDAS library defines a single interface for accessing the repository, alongside two concrete implementations. The remote implementation requests data from the RDAS server, while the local implementation communicates directly with the repository databases, as shown in Figure 4. The remote implementation does not require the Oracle client application to be installed in the device, and therefore can be used in all the environment's devices. Meanwhile, the local implementation was developed for porting legacy applications with requirements that would not allow them to use the remote interface.

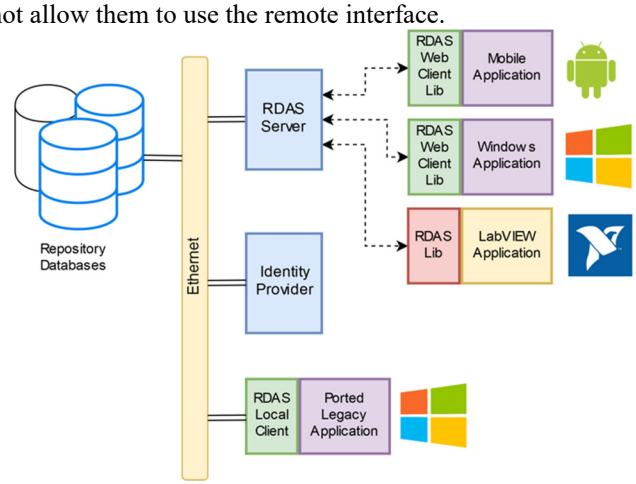

Figure 4: Local and remote access to the repository through the RDAS.

In order to fulfil the second purpose, the library implements several ORM features, allowing CRUD operations in tables to be performed without the developer handling SQL code. The developer can define classes to represent database views, tables, or queries. Afterwards, the developer can perform queries using these classes, and addi-

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA003

tionally, if the class represents a table, also perform insert, update, and delete operations.

FDAS

Configuration and support environment applications are occasionally required to access files in other control system devices. In the legacy environment, applications were installed on machines with access to remote network drives or FTP access, which is unavailable on mobile.

The FDAS service was developed with the purpose of exposing files contained in the server to environment's applications via a RESTful API. In order to allow developers to define which folders are to be exposed, a virtual file system abstraction was defined. The abstraction allows configuration of which folders will be exposed by the FDAS server instance, and how to present them to clients. Additionally, a client library was developed to communicate with the server instances on behalf of clients.

Figure 5 contains a domain model, illustrating the relationships between FDAS servers, the client libraries, and the identity provider.

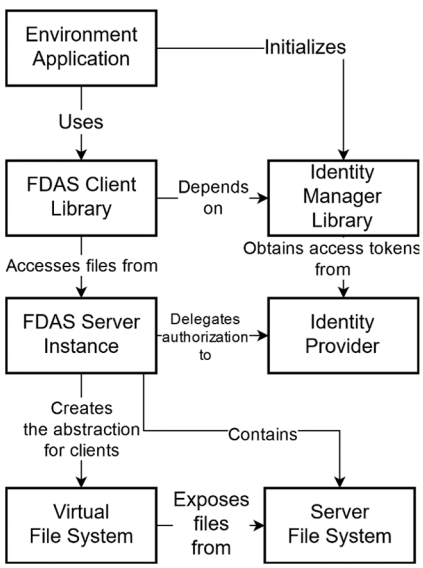

Figure 5: FDAS domain model.

Authorization and Authentication Service

As a part of the effort to standardize the access to resources of the control system, several security requirements were analysed. In legacy environment's applications, authentication was performed by clients via the LDAP protocol [9], and mobile devices integrated into the upgraded environment are not able to access this service. Additionally, in the legacy solution, client applications performed the authorization of users, and it was desirable for resources to also confirm the authentication process. Consequently, we chose the OpenID Connect [8] standard for authentication and authorization in the upgraded environment.

The OpenID Connect standard defines an identity provider entity, to which protected resources delegate their authentication and authorization capabilities.

The identity provider was developed using the IdentityServer4 framework [10], which supplies a base implementation of the standard, as well as several extension points to tailor the identity provider. The following extension points were developed and integrated:

- All configuration data for running the identity provider was persisted in the repository database, and several management classes were defined to obtain this data. This data includes client data, API and identity resources information. In total, 18 new tables were added to the repository's database schema.
- A custom profile service, which obtains and provides claims about the user [11]. Since IdentityServer4 does not define how user data should be structured, when using a custom user repository, developers also have to implement and override all services that consume user data.

Protected resources, such as the RDAS and FDAS server instances were configured to delegate access control to the identity provider. In the resulting access token from an authorization request, the developed profile service includes a permission mask, which denotes the permissions of the respective end-user. The usage of permission masks was ported completely from the legacy environment, and the identity provider obtains them from the repository. By using the permission mask scheme, user permissions from the legacy environment were carried over throughout environment's upgrade.

Finally, a client library was developed to communicate with the identity provider and manage the obtained tokens. This library is also used by other service client libraries to access their respective resources.

Integrated Logging

Event logging in the upgraded environment is performed via a base logging library and a set of optional endpoint libraries, which were developed during this project. The base library defines a façade logging interface, which is implemented by an optional library using the Serilog logging framework [12].

The Serilog framework defines the concept of sinks. Sinks are configured during the application initialization, and later, whenever an event is sent for logging, each sink receives the event and processes it as it sees fit. Two optional libraries implement sinks that log events received into RDAS and FDAS server instances.

Both RDAS and FDAS services can be configured to also act as a centralized logging endpoint. RDAS server instances configured this way log events received into pre-defined tables. Meanwhile, FDAS instances can be configured with a folder path to store the organized logs.

OPC-UA Access

As mentioned previously, applications of the configuration and support environment may communicate with OPC enabled servers in the third layer. During the envi-

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA003

ronment upgrade, no major design changes were required for the OPC client library. Therefore, the legacy OPC interfacing library was ported for the upgraded environment.

INTEGRATION INTO THE ENVIRONMENT

Currently, several applications are being developed for the upgraded environment, following the product line architecture, and using the standard services.

In the same period, developers of software applications in other environment expressed willingness to utilize some services developed for the configuration and support environment. As a result, a project was approved, and is currently underway, to allow LabVIEW applications present in the first level of the control system to consume the web services of the upgraded environment [13].

Impact on the Environment

The expected date for the integration of the first upgraded environment applications into the control system is the beginning of 2020. As a result, evaluating the upgraded environment, in comparison to the former, can only be performed at this moment based on the impact of the newly developed tools, and the expected quality attributes of applications.

The main improvement to the control system has been the addition of the mobile devices as application platforms. In addition, we argue that, overall, the designed product line architecture better promotes maintainability and testability than the legacy design. This has been done by segregating the implementation of the services from the applications themselves and enforcing several loose coupling strategies in the product line architecture.

Finally, resource access control has been improved through the adoption of an authentication and authorization standard that is widely adopted in the industry, allowing us to consider the possibility of making a set of the web services externally available in the future.

CONCLUSION

During the past years, the configuration and support environment of CNAO's control system is being upgraded. In this document, we have presented the service oriented approach for designing the communication between future applications in this environment to other control system components.

As a result, several web services and client libraries were designed and developed. These services, alongside the technologies chosen in the product line architecture, extend range of devices able to operate in the facility's control system, adding mobile devices.

The preliminary results of this project have been positive, with the first environment applications being developed currently. Additionally, a project to allow the services to be consumed by applications in other environments is currently underway, expanding their usage.

REFERENCES

[1] L.Casalegno, M.Pezzetta and S.Toncelli, CNAO General Control System Organization Document, unpublished.

[2] SCADA System SIMATIC WinCC V7, https://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/pages/default.aspx

[3] L. Bass, P. Clements and R. Kazman, Software architecture in practice, Addison-Wesley Professional, 2003.

[4] Xamarin | Open-source mobile app platform for .NET, https://dotnet.microsoft.com/apps/xamarin

[5] Introduction to Portable Class Libraries (PCL), https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/pcl?tabs=windows

[6] The Model-View-ViewModel Pattern – Xamarin, https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[7] M.Seemann, Dependency injection in. NET, Manning New York, 2012.

[8] N.Sakimura, J.Bradley, M.Jones, B.deMedeiros, and C.Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", The OpenID Foundation, specification, 2014.

[9] LDAP user authentication explained - Connect2id, https://connect2id.com/products/ldapauth/auth-explained

[10] IdentityServer4 Framework, https://github.com/IdentityServer/IdentityServer4

[11] IdentityServer4 documentation, https://identityserver4.readthedocs.io/en/release/.

[12] Serilog - simple .NET logging with fully-structured events, https://serilog.net/.

[13] S.Foglio, C.Viviani, and L.Casalegno, Use of the CNAO Query.lvlib in the EasyLoader application, unpublished.