System integration: A comparison of the SCATh and the CASCADE system architecture

Vincent Meiser
Zurich University of Applied Sciences
Institute of Mechatronic Systems
Winterthur, Switzerland
Email: vincent.meiser@zhaw.ch

Dejan Šeatović
Zurich University of Applied Sciences
Institute of Mechatronic Systems
Winterthur, Switzerland
Email: dejan.seatovic@zhaw.ch

***Abstract*—With a focus toward system architecture, the SCATh and CASCADE platforms are outlined and a comparison is drawn to show the characteristics of the medical platforms and their differences. In the field of system integration, the architecture is a very basic element of a new system design and development. In the prior project, SCATh, a Windows-based non-real time capable architecture was used. A hard real-time application set will be implemented in CASCADE using ready-to-use frameworks; Robot Operating System (ROS) and Open Robot Control Software (OROCOS) to achieve an accelerated development process. To illustrate the differences between the platforms, their development environment with regard to hard- and software is presented. The crucial elements of the system design, amongst others the multi-tier architecture, are revealed and the consequences for the development process are shown.**

I. Introduction

A European Union research project *Smart Catheterization* (SCATh) [1] was carried out over the last 3 years. Its subject was research on and development of a catheter guidance system for surgical interventions. A challenging aspect was the system integration of multiple, partly independent system components into one operational platform, whereby inhomogeneous software engineering skills and experiences turned out to be major hurdles for a fluent development process. A recently started project named *Cognitive AutonomouS CAtheter operating in Dynamic Environments* (CASCADE) [2] deals with a similar subject, but it is more focused on robotic than medical developments. The task of system integration is even more important in CASCADE, since the expected outcome of the project is an operational prototype. Hence the SCATh platform is presented here mainly with regard to the system architecture to serve as example and to offer background information for prospective developments. A comparison to the CASCADE architecture as currently planned is drawn.

II. SCATh platform

The SCATh platform was meant to serve as an integration framework for the different components developed by each project partner. A straightforward and small-scale environment was one of the targets during architecture design and implementation, to supervise and maintain the development progress of the project without increasing complexity of the development process itself. Hard real-time capabilities were limited to an additional device, a *National Instruments* (NI) real-time target, which was in charge of controlling the catheter and the sensors attached to it.

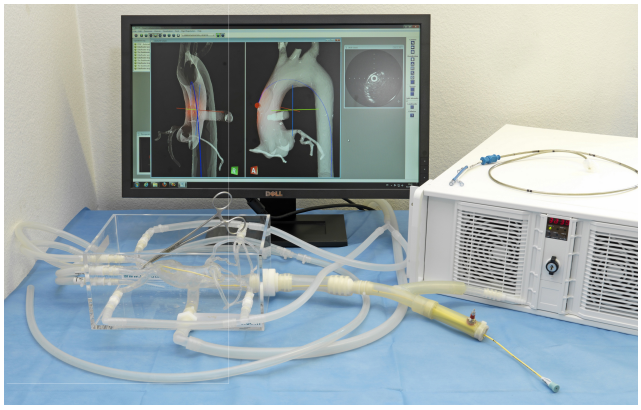

Fig. 1: Basic SCATh Platform Setup

A. Development environment

The target operating system (OS) for the SCATh platform is Windows 7, hence Visual Studio is defined as the integrated development environment (IDE). Several external libraries are integrated into the SCATh application for mathematical calculations, visualization and network communication. The programming language is C++ and Compute Unified Device Architecture (CUDA).

On the hardware side, the SCATh platform consists of an industrial PC for high performance computing (HPC) with sufficient RAM size (24 GB). The HPC is extended with an NVIDIA Tesla card, which adds additional data processing capability with the use of a General Purpose Graphics Processing Unit (GP-GPU). In SCATh this additional computation unit is used for Finite Element Modelling (FEM). The high-performance computer is also called Execution Environment Computer (EEC). The NI real-time target represents the second hardware component. A *LabView* application was implemented to forward measurement data provided by catheter sensors to the EEC.

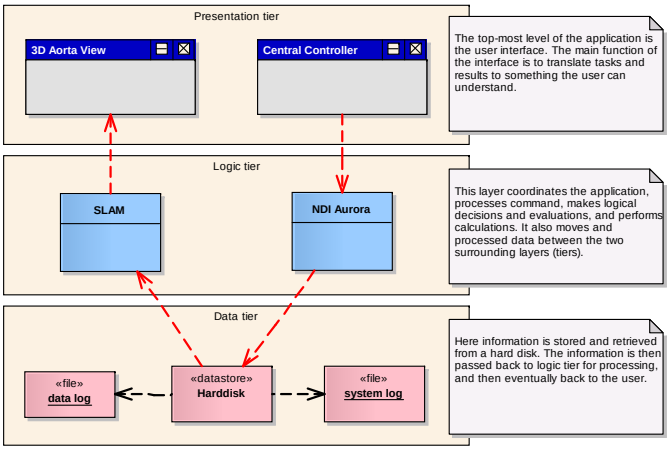

Fig. 2: Three-tier Architecture with Presentation, Application (Logic) and Data tier

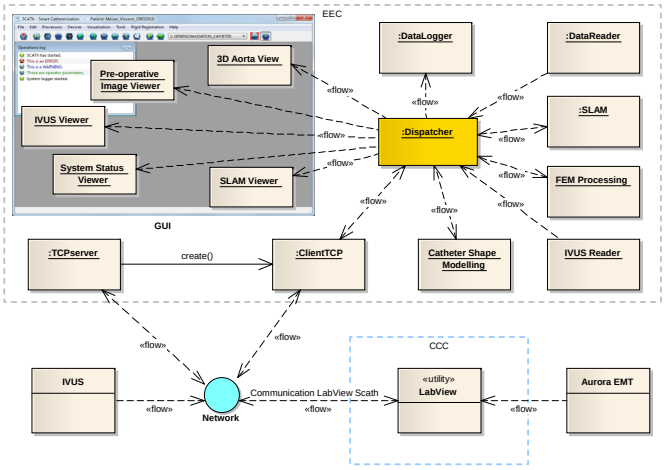

Fig. 3: Data Flow within SCATh Platform

B. System architecture

The SCATh system architecture consists of state-of-the-art software components, an abstract interface pattern as well as a provider-subscriber pattern [3], [4]. The design is based upon a three-tier architecture consisting of a data tier (data model), an application tier (logic) and a presentation tier (GUI). The three-tier architecture gathers logically separated parts of the software and minimises their dependencies. Fig. 2 displays a three-tier architecture.

The pattern of providing information to an arbitrary amount of subscribers is shown in Fig. 3. A *Dispatcher* instance controls all data transfer going throughout the SCATh platform. Subscribers such as *3D Aorta View*, *SLAM* or *FEM Processing* can subscribe for specific data types to receive them or can produce data and send it to the Dispatcher. This forwards the data to all components according to the subcriptions.

The most challenging part of the framework design is to provide a flexible architecture that is able to survive significant requirement changes. This goal has been fully achieved since requirement changes do not or only mariginally affect the system and framework architecture. This way, basic constraints for a robust system, reliability and stability, can be assured.

C. Outcome and experiences

The described SCATh platform provides a user friendly and flexible software framework, which is used by all project partners. It is designed to integrate a wide range of position, force and vision sensors with minimal implementation time. All partners used the framework for design and implementation of their own components. Integration of third-party libraries and components is mostly effortless. Therefore, the project schedule was not affected by late requests for additional components. The key to success of the SCATh platform lies in the approximately uniform level of understanding of essential parts of the framework, which means similar level of knowledge in areas of software engineering, implementation, etc. The usability study of the final SCATh platform in a clinical environment was successfully performed.

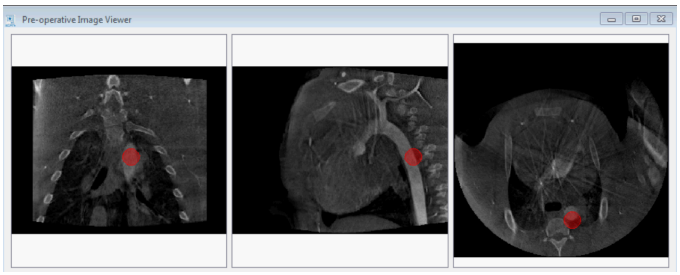

Fig. 4: SCATh Pre-operative Image Viewer, realized using *GDCM* library to cut slices from CT/MR: In automatic mode the viewer shows the appropriate slices of the CT/MR volume at actual catheter tip position.

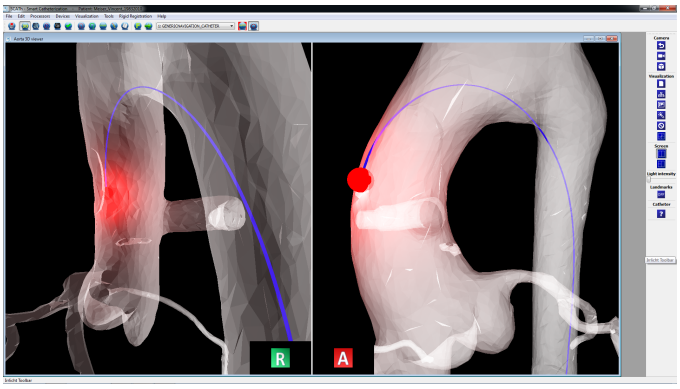

Fig. 5: SCATh three-dimensional Aorta Viewer, realized using *Irrlicht* graphics engine: The viewer displays a 3D model of a human aorta. If the catheter tip is approaching to a safety critical volume, a colored signal is sent to the operator.

Fig. 4 and Fig. 5 depict two graphical elements of the SCATh platform, which will be transfered to the CASCADE project to the greatest extent. This will be issued in the second part of this paper when it comes to the description of the platform's differences.

III. CASCADE SYSTEM ARCHITECTURE

The following section provides an outlook on the architecture planned for the CASCADE project. For multiple reasons, in particular the real-time capabilities, the architecture of CASCADE is distinctly different to the SCATh architecture. The software and hardware needed by the CASCADE system should be as flexible as possible without impinging on the project goals and targets. Therefore all system parts are encapsulated into modules (components). Components are completely independent applications, they only depend on a minimum set of other modules. It is possible to build and maintain a variety of configurations of the CASCADE system which consist of a complete set or subset of the available modules. The modules can be defined as components for example, as described in [5], [6]. The advantage of component-oriented programming is that every component is reusable and self-contained. Components are based strictly on a producer-consumer pattern, where the interface represents the one and (usually) only dependency between two different components.

A. Hard real-time constraints

Since CASCADE has strong hard real-time constraints such as catheter control and safety components, the platform requires a real-time capable operating system. Linux itself does not offer real-time capabilities, but if it is combined with a real-time patch, e.g. Xenomai, Linux OS can serve as a real-time system. Furthermore, only a few of the CASCADE components have hard real-time constraints, others are of soft real-time or non real-time type. A challenging task is the synchronization of these components without computing performance loss and real-time constraints violation. Both aspects increase the complexity of the system and its architecture, and prolong the first development phase of setting up the development environment. For better understanding, the environment is explained subsequently in the system design (section III-B).

The development of software frameworks for surgical robotics is a topical subject in several other research projects. Amongst others, the *EuRoSurge* project aims at developing a conceptual integration platform for Computer and Robot Aided Surgery (CRAS) [7].

B. System design

The CASCADE platform, which is under continuous development over the duration of the project, is facing several challenges. For example, the reliable communication between and coordination of a large number of distributed components and hard real-time constraints upon the execution of several components. Here, the ROS and OROCOS robotic frameworks (Meta Operating Systems) come into the picture. The CASCADE platform will be built on top of these frameworks and make use of their functionality whenever it makes sense and simplifies the development process. Considering modularity, independence, robustness and ease of development, a component-based architecture is chosen as the architecture under development. Components or nodes are separate processes that ideally only fulfill one single isolated task.

The system design is based upon a three-tier architecture as already described in section II-B. Individual CASCADE components are allocated to one particular tier. This enables a clear and logical separation of system parts, which simplifies the development process and encapsulates independent parts. The system design is displayed in Fig. 6.

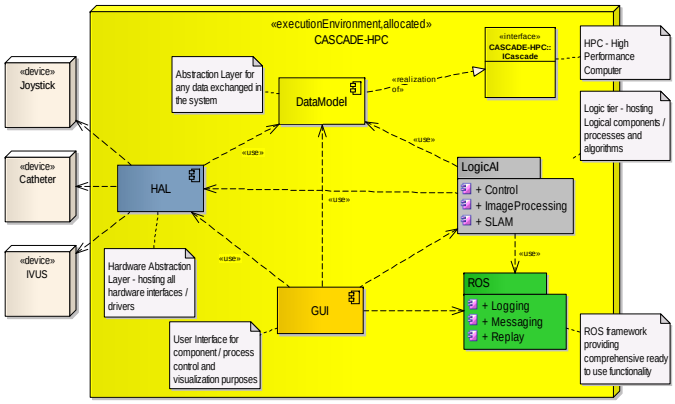

Fig. 6: CASCADE System Design

The data model defines the interface between the different components. Any data exchange within the platform will be performed in accordance to this interface. Since it is planned to use the ROS messaging system, the data model will implement functionality (create ROS messages, etc.) to use the ROS framework. Data exchanged between components within the CASCADE interface can range from position coordinates and force values all the way up to imaging data. The data model represents a single interface for the CASCADE platform. There is no other component that provides interface functionality.

The logic tier contains the processing logic of the components. As mentioned earlier, this layer is meant to be independent from other shared libraries. The logic tier provides processing components based on the interface defined by the DataModel.

The GUI represents the user interface and provides a visualization front end for the operator (surgeon). It is a layer on top of the DataModel and the Logic layer, which are hidden from the user. The GUI operates in detached mode; it fulfills no real-time constraints and is designed for user interaction, with sufficient performance for user-friendly operability.

As can be seen in Fig. 6, the logic tier and the GUI are dependent on the ROS framework. Especially within these tiers, the ROS functionality (messaging system, logging, replay, etc.) as well as the OROCOS functionality (hard real-time task definition and execution) is used. Furthermore, the architecture foresees a pre-definition of all system components considering their real-time characteristics and needs by the system integrator. The partners responsible for one or multiple components can start their development upon these pre-defined parts. This reduces the need for high-level understanding of certain frameworks and so decreases the development time. Prior to this component definition, the requirements of all system parts and subsequently the specification are essential steps in the development process.

TABLE I: Development Environment - Software

Project	*SCATh*	*CASCADE*
OS	Windows XP	Linux (Ubuntu LTS 12.04)
IDE	Visual Studio	Eclipse
Programming language	C++, CUDA	C++, CUDA
Compiler	MVC++	g++
Libraries	Irrlicht, OpenCV Qt, GDCM	OpenCV, rviz Qt, any graphics engine
Frameworks		ROS, OROCOS Xenomai RT-kernel

TABLE II: Development Environment - Hardware

Project	*SCATh*	*CASCADE*
Devices	High-performance computer/IPC	High-performance computer/IPC
	NI real-time target (PXI)	NI real-time target (PXI/cRIO)
	Epiphan VGA2Ethernet frame grabber	opt: Epiphan VGA2Ethernet frame grabber
		FPGA, ...
Internal memory	24 GB	24..64 GB
Graphics card	NVIDIA Tesla C2075	NVIDIA Tesla C2075, ...
Communication	Ethernet	EtherCAT, ...

C. Development environment

The development environment is distinctly different from the one in SCATh. A hard real-time capable OS was chosen to fulfill the constraints. The ROS framework is used for messaging, logging and replay, and the OROCOS connected to the Xenomai RT-kernel will provide hard real-time task definition and execution. A listing of software elements is given in Table I.

With the change of OS, the external libraries (Qt, OpenCV, GDCM, etc.) have also become partly obsolete. To use SCATh components as the *Pre-operative Image Viewer* and the *3D Aorta View* in the new CASCADE platform, larger modifications are necessary. On the other hand, this enables a review and improvement step on the way to a highly stable and robust CASCADE platform. To complete the development environment used in the SCATh and the CASCADE project, a listing of hardware equipment is given in Table II. The hardware components are mostly similar except for the real-time specific system parts, EtherCAT, FPGAs, etc. The purchase of hardware is still pending. To fulfill all real-time constraints and run the appropriate components within one system, it might be necessary to use a high-end HPC, such as Intel i7-39xx, Xeon E3-xxxx or E5-xxxx, combined with sufficient RAM size.

IV. Conclusion

Although the SCATh and CASCADE system architectures have several similarities, there are distinct differences between the platforms (architectures). The basic elements of a component-based system built upon a three-tier architecture are similar. However, the SCATh platform was developed in a Windows environment without hard real-time capabilities (except for the LabView part), whereas the CASCADE platform is designed to completely fulfill the given hard real-time constraints. Due to this fact, the CASCADE architecture is fundamentally different and more complex than that of SCATh. The platform development effort with regard to the architecture will be much higher, since more frameworks and external libraries will be integrated to one system. Once this first hurdle is overcome and the basic system setup is complete, a well-defined and structured architecture will be available for the given tasks to move towards project targets.

References

[1] "SCATh smart catheterization, the SCATh project," http://www.scath.net/, July 2013. [Online]. Available: http://www.scath.net/

[2] "CASCADE cognitive autonomous catheter operating in dynamic environments, the CASCADE project," http://www.cascade-fp7.eu/, July 2013. [Online]. Available: http://www.cascade-fp7.eu/

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[4] H. Gomaa, *Software modeling and design : UML, use cases, patterns, and software architectures*. Cambridge University Press, 2011.

[5] "Nodes - ros wiki, [online]. available: http://www.ros.org/wiki/nodes, [accessed: 23-may-2013]." [Online]. Available: http://www.ros.org/wiki/Nodes/

[6] K. Andy Ju An Wang, *Component-Oriented Programming*. John Wiley & Sons, Inc. All rigths reserved, 2005.

[7] P. Fiorini, "Linking patient safety and automation in robotic surgery: the eurosurge project," in *ICRA2012 Workshop Modular Surgical Robotics: how can we make it possible?* Altair Robotics Laboratory, Department of Computer Science, University of Verona, Italy, 2012.

Nomenclature

ROS	Robot Operating System
OROCOS	Open Robot Control Software
SCATh	Smart Catheterization
CASCADE	Cognitive AutonomouS CAtheter operating in Dynamic Environments
OS	Operating System
HPC	High Performance Computing
GP-GPU	General Purpose Graphics Processing Unit
FEM	Finite Element Modelling
EEC	Execution Environment Computer