Issue Date:	2022-02-xx	CEN-CLC/FGQT	N164
Deadline:	n.a.	Supersedes:	
Status:	FOR APPROVAL		

TITLE	**Modularity and layering of software stack**
PROJECT	FGQT Roadmap
REFERRING TO	
SOURCE	Michele Amoretti (CINI, Italy)
CONTACT	michele.amoretti@unipr.it

With valuable input from and supported by:
Rob van den Brink (Delft Circuits)
Oskar van Deventer (TNO)
Noel Farrugia (University of Malta)
Marco Gramegna (INRIM / EMN-Q)
Thomas Länger (StandICT)
Antonio Manzalini (TIM)
Vicente Martin (UPM)
Niels Neumann (TNO)
Homer Papadopoulos (NCSR Demokritos)
Momtchil Peev (Huawei Technologies Duesseldorf GmbH)
Nicolas Spethmann (PTB)
Frank Wilhelm-Mauch (UdS / FZ Jülich)

ABSTRACT

During FGQT18 in Berlin, a breakout group worked together to build consensus on a layered view of quantum software, specialised by application area. A report of that discussion was presented as N148a. In this contribution, a detailed presentation of the software stack for quantum computing is presented, aiming to replace the full section 7.1.5 of the Roadmap Document (N020g, or its successors).

1. Situation sketch

Quantum computing is an area covering very different implementations and each implementation has its own dedicated solution and maturity. The aim is a stack of software and hardware layers, where higher layers are more agnostic for these differences than lower layers.

As illustrated in Fig. 1, which comes from the Strategic Research Agenda of the Quantum Flagship [1], designing and developing software is intended as a cross-cutting activity with respect to the key application areas of the Quantum Flagship: Communication, Computing, Simulation, and Sensing & Metrology.

We propose to replace the present section 7.1.5 of the roadmap document (N020g, or its successors) with the text in this contribution.

Proposal for inclusion in section 7.1.5

7.1.5 Modularity and layering of software stack

According to the Strategic Research Agenda of the Quantum Flagship [1], designing and developing software is intended as a cross-cutting activity with respect to the key application areas of the Quantum Flagship: Communication, Computing, Simulation, and Sensing & Metrology (Fig. 1).

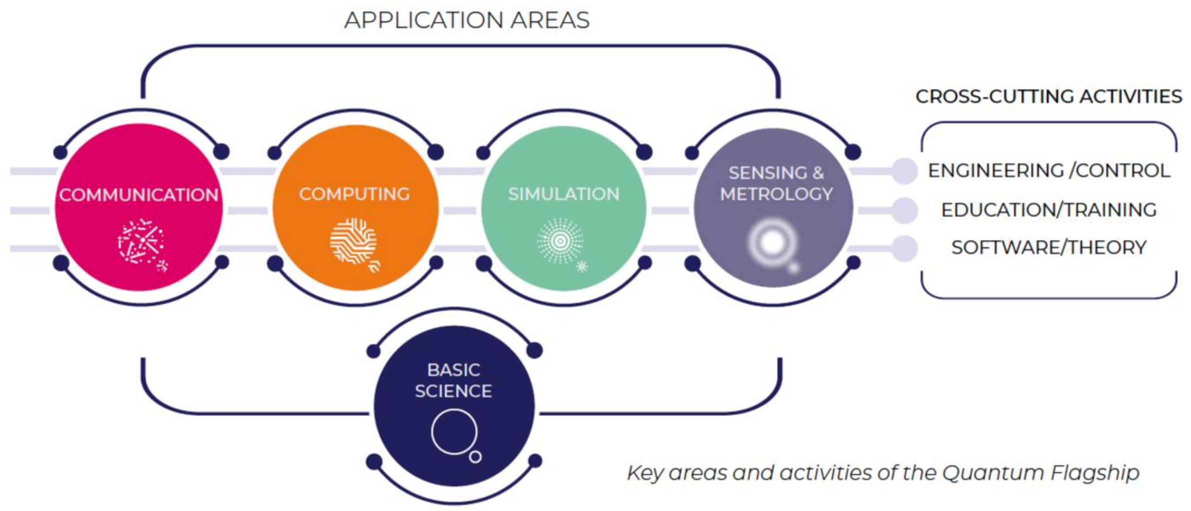

Figure 1 – Areas and cross-cutting activities of the Quantum Flagship [1].

Broadly speaking, quantum software is all the software that is related to quantum technology systems, including quantum programs executed on quantum devices, control software, and more.

To the best of our knowledge, the Quantum Software Manifesto [2] is the earliest (and most widely supported) European-level effort to optimally represent software in the Quantum Flagship initiative. The Manifesto emphasizes the importance and urgency of

- quantum software research,

- integrated approach to hardware and software R&D,
- collaboration between industry and academia,
- educating more quantum programmers.

A convenient way of specifying quantum software is via a stack of layers, specialised by application area. The layers are chosen in such a manner that the functionality of each layer can be described in an independent manner, as shown in Fig.2. The interworking between these layers can be described through well-defined interfaces.

In the following subsections, the layers of the software stack are described and discussed, using a bottom-up approach (i.e., from the control software to the applications / services supporting use cases).

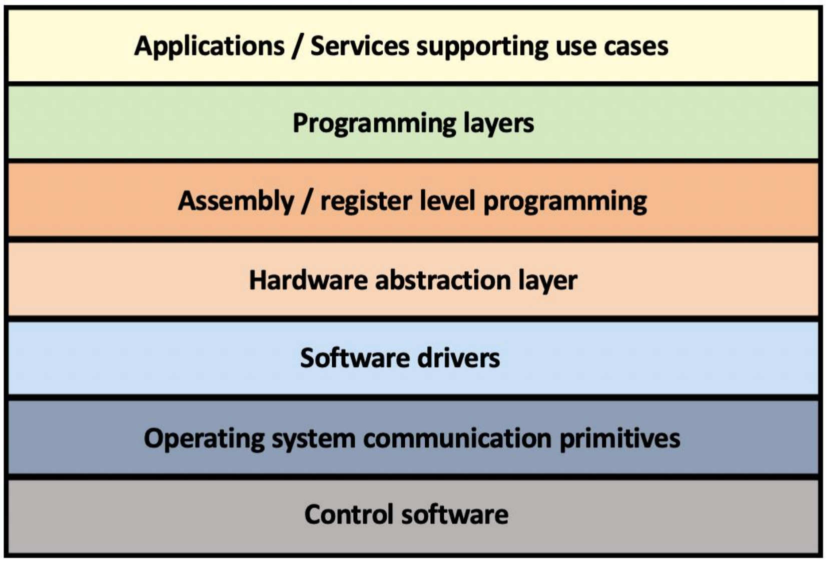
Applications / Services supporting use cases
Programming layers
Assembly / register level programming
Hardware abstraction layer
Software drivers
Operating system communication primitives
Control software

Figure 2 – A possible break-down of software layers for quantum computing.

7.1.5.1 Control software

The purpose of control software for quantum computing is to deliver high-performing qubit operations to higher level of abstraction in the quantum software stack with minimal user intervention.
Control software may include calibration means, low-level code to translate instructions from higher software layers into commands for guiding the control electronics/optics, and comprises the techniques used to define error-robust physical operations and associated supporting protocols designed to tune-up and stabilize the hardware.
Control software for quantum hardware is typically stored on classical computers, i.e., there is a very strict separation between the place where the control software is stored and the quantum registers.
~~Control software is usually deployed as firmware, i.e., embedded in hardware. If the firmware is stored in a memory chip EEPROM or flash, it may be reprogrammed through a special procedure.~~
In the long term, control software may work in concert with Quantum Error Correction (QEC), which is supposed to lay at the assembly / register level programming layer, to provide broad coverage of various error types. More specifically, control software could improve the efficiency of QEC, i.e., reduce resource overheads required for encoding, by homogenising error rates and reducing error correlations.

7.1.5.2 Operating system communication primitives

A quantum computer must be provided with an operating system (OS), which is a resource manager for the underlying quantum hardware, provided with built-in networking functions allowing multiple users and applications to use the resources as remote clients. To an application, it appears as if it has its own resources and is protected from other applications. Applications can make use of facilities only as offered by the OS. For example, the OS provides communication primitives (e.g., based on the POSIX standard for the sockets interface [3]) and only by means of these primitives should it be possible to pass messages between client applications and the quantum computer.

~~It is quite common that the communication primitives provided by the OS are based on the simple message-oriented model offered by the transport layer, which is the last part of the TCP/IP network protocol stack and turns the underlying network into something that can be used by an application developer. Special attention has been paid to standardising the interface of the transport layer. The sockets interface, introduced in Berkeley Unix in 1970 and later adopted as a POSIX standard with only very few adaptations, includes the following primitives: `socket` (to create a new communication endpoint), `bind` (to attach a local address to a socket), `listen` (to announce willingness to accept connections), `accept` (to block the caller until a connection request arrives), `connect` (to actively attempt to build a connection), `send` (to send some data over the connection), `receive` (to receive some data over the connection), and `close` (to release the connection) [3].~~

7.1.5.3 Software drivers

In the layered view illustrated in Fig. 2, software drivers are components that are plugged into the operating system and allow hardware-abstraction programs to call the control software of the underlying quantum hardware. If the hardware changes, the software drivers must change as well.

7.1.5.4 Hardware abstraction layer

The hardware abstraction layer (HAL) should allow high-level quantum computer users, such as application developers, platform and system software engineers, cross-platform software architects, to abstract away the quantum hardware implementation details while keeping the performance. The hardware may change, but the QASM-like programs (belonging to the upper assembly / register level programming layer) should be still able to work.

Among all software layers for quantum computing, this is the one that requires the most urgent standardisation effort. The hardware abstraction layer should provide Application Programming Interfaces (APIs) to the upper layer, decoupling from the different types of quantum hardware technologies (e.g., superconducting qubits, trapped ions, molecular magnets) and concerning:

- predefined gate names (short and long)
- gate matrices
- quality parameters / element-level characterization
- calibration, characterization, and control
- benchmarking

Within the Internet Group of GSMA [4,5], the Quantum Networking and Services (QNS) group has recently established a new work-item on the Quantum-HAL for both Quantum Computing and Networking. The main motivation is that, today, one major obstacle hindering the exploitation of quantum technologies is that the industry has not yet consolidated around one type of quantum hardware technology. In this scenario, a Quantum-HAL - for Quantum Computing and Networking - would allow Applications and Services Developers to start using the abstractions of the underneath quantum hardware (even if today under consolidation): this would simplify and speed-up the development of quantum platforms, services, and applications.

In fact, for example, a Quantum-HAL would provide unified northbound quantum Application Programming Interfaces (APIs) for the higher layers, decoupling from the different types of quantum hardware technologies (e.g., trapped ions, superconducting qubits, silicon photons qubits) for Quantum Computing and Networking.

This is important also for another reason: any quantum node or system of a Quantum Computing and Networking infrastructure required to be properly configured, managed, and monitored to effectively operate and support interworking with classic nodes/systems. There is a need of defining data model representing various aspects of the networked quantum sub-systems and components (or devices). This will bring interoperability and plug-and-play capabilities. As an example, an option could be using the YANG data model, which is standard language widely used in the telecommunication domain for describing and managing devices on a network. This is under study also in the IEEE P1913 project [6].

In general, the activities of definition, modelling, and standardization of a Quantum-HAL - for Quantum Computing and Networking – require coordinated and joint efforts including, where appropriate, existing projects, industry bodies and standard fora (e.g., CEN-CENELEC, GSMA, ETSI, IEEE, etc…) active in the area.

7.1.5.5 Assembly / register level programming

This layer concerns QASM (i.e., quantum assembly) languages that describe quantum computations according to one specific model (e.g., circuit model, measurement-base model, quantum annealing model), with a per-architecture instruction set.

An example is OpenQASM [7], which targets IBM Q devices and enables experiments with small depth quantum circuits. OpenQASM represents universal circuits over the CNOT plus SU(2) basis with straight-line code that includes measurement, reset, fast feedback, and gate subroutines. OpenQASM possesses a dual nature as an assembly language and as a hardware description language.

A different example is NetQASM [8], which is a platform-independent and extendable universal instruction set with support for local quantum gates, classical logic, and quantum networking operations for remote entanglement generation. NetQASM consists of a specification of a low-level assembly-like language to express the quantum parts of quantum network program code.

Due to the huge diversity of quantum computing architectures, it is not likely that a unique, widely accepted QASM would emerge and later become a standard.

7.1.5.6 Programming layers

The specification of quantum algorithms using QASM languages is not easy for programmers. Indeed, QASM programs are usually generated by a software library, from a piece of code written in a common programming language, such as Python.

In general, the programming layers include all the languages, libraries, and software development facilities (e.g., software development kits, debugging tools, quantum compilers) used by developers for coding quantum algorithms or high-level applications that use predefined quantum algorithms as subroutines.

Quantum compilation is the problem of translating an input quantum circuit into the most efficient equivalent of itself, considering the characteristics of the device that will execute the computation and minimizing the number of required two-qubit gates. In general, the quantum compilation problem is NP-Hard. The most advanced quantum compilers are noise-adaptive, i.e., they take the noise statistics of the device into account.

7.1.5.7 Applications / Services supporting use cases

To effectively support industrial and research use cases, quantum applications must be executed in suitable environments. Currently, some vendors provide access to quantum devices via user-friendly cloud platforms. The quantum programs must be locally compiled for a specific device and submitted for batch processing to the remote platform. However, other paradigms are emerging. For example, the Quantum Internet will enable networked quantum applications, whose execution will involve multiple quantum nodes and will be characterised by interleaved classical and quantum message passing.

References

[1] European Quantum Flagship, *Strategic Research Agenda*, March 2020
[2] Quantum Software EU Platform, *Quantum Software Manifesto*, 2017
[3] IEEE/Open Group 1003.1-2017, Standard for Information Technology—Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7
[4] GSMA, IG.11 – Quantum Computing Networking and Security, July 2021
[5] GSMA, IG.12 – Quantum Networking and Service, December 2021
[6] IEEE P1913 Project Software-Defined Quantum Communication Working Group, IEEE-P1913-Summary-2018-v0.1.pdf (comsoc.org)
[7] IBM, OpenQASM 3.x Live Specification, https://qiskit.github.io/openqasm/index.html, 2022
[8] A. Dahlberg et al., NetQASM -- A low-level instruction set architecture for hybrid quantum-classical programs in a quantum internet, https://arxiv.org/abs/2111.09823, 2021