Reo Coordination Model for Simulation of Quantum Internet Software

Ebrahim Ardeshir-Larijani[1,2](✉) and Farhad Arbab[3,4]

[1] Department of Computer and Data Science, Faculty of Mathematical Sciences, Shahid Beheshti University (SBU), Tehran, Iran

[2] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

e.a.larijani@ipm.ir

[3] Centrum Wiskunde and Informatica (CWI), Amsterdam, Netherlands

[4] Leiden University, Leiden, Netherlands

farhad.arbab@cwi.nl

Abstract. The novel field of quantum technology is being promoted by academia, governments and industry. Quantum technologies offer new means for carrying out fast computation as well as secure communication, using primitives that exploit peculiar characteristics of quantum physics. While building quantum computing devices remains a challenge, the area of quantum communication and cryptography has been successful in reaching industrial applications. In particular, recently, plans for building quantum internet have been put into action and expected to be launched by 2020 in the Netherlands. Quantum internet uses quantum communication as well as quantum entanglement along with classical communication. This makes design of software platform for quantum networks very challenging and a daunting task. Seamless design and testing of platforms for quantum software, thus, becomes a necessity to develop complex simulators for this kind of networks. In this short paper, we argue that using coordination models such as Reo can significantly simplify the development of software applications for quantum internet. Moreover, formal verification of such quantum software becomes possible, thanks to the separation of concerns, compositionality, and reusability of Reo models. This paper introduces an extension of a recently developed simulator for quantum internet (SimulaQron) by incorporating Reo models extended with quantum data and operations, along with classical data. We explain the main concepts and our plan for implementing this extension as a new tool: SimulaQ(reo)n.

Keywords: Quantum communication · Quantum information
Quantum networks · Reo Coordination Model

The first author is supported in part by a grant from the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Iran.

© Springer Nature Switzerland AG 2018

1 Introduction

As quantum technologies emerge rapidly, designing reliable hardware and software for hybrid quantum/classical systems poses significant challenges both theoretically and experimentally. Nevertheless, specific quantum networks have been built in various cities around the world and already a satellite has been launched to provide secure quantum communication. Using such networks demands rigorous analysis and verification before they can be trusted in safety- and security-critical applications. One way to achieve this goal is to develop a dedicated simulation toolset before actual quantum devices get deployed. SimulaQron [2] is an example of such a tool that is able to model the behaviour of local simulators or even actual quantum devices in a hybrid quantum/classical network which is called quantum internet [4]. The tool itself can be thought of as a platform for developing software applications for quantum internet, and is designed to offer ease-of-use and clarity in that regard. However, simulating complex interactions in quantum networks needs incorporation of coordination models for the same reasons as in the case of classical networks (e.g., compositionality and reusability), even more strongly so, because in quantum networks, primitives with non-local (entanglement) effects play a critical role. Currently, the majority of research in quantum programming focuses on sequential programs and efficient simulation of sequential quantum algorithms (e.g., see [10]).

In this short paper we pursue two objectives: first, to bring the problem of coordination in quantum internet to the attention of the computer science community, especially those active in the field of coordination models [15]. Second, we explain the principles of extending Reo [3,13,14] coordination model and language to support modeling of the behaviour of quantum networks. One milestone toward our second objective consists of automatic generation of executable code for protocols over quantum internet. To this end, the current Reo compiler has to be modified in order to support quantum data types, operations and primitives, as we explain later in this paper. To our knowledge this paper presents the first work on coordination of quantum software components.

The rest of this paper is organized as follows. In Sect. 2, we present the necessary background on quantum information processing. In Sect. 3, we review the Reo coordination concepts. We describe the SimulaQron tool in Sect. 4. In Sect. 5, we present the principles of extension of Reo to support coordination in the quantum setting, particularly in connection with SimulaQron. Finally, we conclude the paper in Sect. 6 with plans for future work.

2 Quantum Information

This section provides a concise introduction to quantum information processing (QIP). For more details, we refer to [1]. The basic unit of quantum information is a *qubit* (quantum bit). A qubit can be in a *basis* state, represented by $|0\rangle$ or $|1\rangle$. These basis states correspond to the classical states 0 and 1. However, a qubit may be in a *superposition* of states, described by $\alpha|0\rangle + \beta|1\rangle$, with $|\alpha|^2 + |\beta|^2 =$

1 where α and β are complex numbers called *amplitudes*. More generally, we consider a state of n qubits, whose general form is $|\psi\rangle = \alpha_0|00\ldots0\rangle + \ldots + \alpha_{2^n-1}|11\ldots1\rangle$ with $\Sigma_i|\alpha_i|^2 = 1$.

The state of a single qubit is an element of a two-dimensional complex vector space, called *Hilbert space*. Multi-qubit state spaces can be constructed by tensor products, e.g., $|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle$ defines an n-qubit basis state $|00\ldots0\rangle$.

There are two kinds of operations on quantum states: unitary operations and measurements. A unitary transformation is an invertible linear operation on the Hilbert space. In a two dimensional Hilbert space, measurement randomly projects the state onto one of a pair of orthogonal subspaces, with a probability determined by the amplitudes. A measurement, thus, produces classical information as a result. For example, if the state $\alpha|0\rangle + \beta|1\rangle$ is measured in the standard basis, then the classical result is 0 with probability $|\alpha|^2$ or 1 with probability $|\beta|^2$. Moreover, measurement of a quantum state (in the standard basis) permanently changes it to $|0\rangle$ or $|1\rangle$, respectively.

An important phenomenon in quantum physics is *entanglement*. A multi-qubit state is entangled if it cannot be decomposed as a tensor product of simpler states. An example is the two-qubit state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, which is known as an EPR pair. This pair is one of a set of four important two-qubit entangled states, termed *Bell states*. In this state, if the first qubit is measured in the standard basis, then the overall state collapses to either $|00\rangle$ or $|11\rangle$, which also determines the state of the second qubit. Therefore, there is a correlation between the two entangled qubits even when they are separated by a distance.

3 Reo Coordination Model

Reo [3] is a language for exogenous coordination of software components, wherein protocols are defined as graphs of primitives called channels. In Reo, graphs of channels, called connectors, are defined compositionally. Recently, a new textual syntax together with a versatile compiler for this syntax have been added to the set of Reo tools [9]. The simplest form of connectors are channels that connect two ends by defining a relation on the observable data exchanged at those ends. This relation imposes a constraint on the flow of data between those end points. Channels constitute the edges of Reo connector graphs on whose nodes channel ends coincide. Reo allows arbitrary user-defined channel types, but only two types of channel ends can exist: a source end through which data enters into a channel, and a sink end through which data leaves a channel. Compositions of these two types of channel ends yields only three types of nodes: source, sink, and mixed. A source node consists of one or more source channel ends; a sink node consists of one or more sink channel ends; and a mixed node consists of one or more source and one or more sink channel ends. Components can perform I/O operations on only source and sink (but not mixed) nodes of a connector. A data item written to a source node gets replicated to every source channel end of the node, only when all of them are able to accept; a source node, thus, performs a form of synchronous broadcast of its incoming data-flow stream onto its outgoing data-flow streams. A take operation on a sink node non-deterministically selects a data item available at one of the sink channel ends of the node and leaves the others intact; a sink node, thus, performs a non-deterministic merge of its incoming data-flow streams onto its outgoing data-flow stream. A mixed node repeatedly performs an atomic operation that combines the behaviour of a sink and a source node: in each iteration, it non-deterministically selects a data item from one of its sink channel ends and replicates it onto all of its source channel ends, all in one atomic operation.

We now informally explain the behaviour of some of the channels in terms of constraints that they impose on data-flow. For formal definition of constraint automata as operational semantics of Reo language, see [3]. The $\mathit{Sync}(a, b)$ channel, gets data items trough its end a and synchronously (i.e., atomically) outputs them through its end b. Similarly, the $\mathit{LossySync}(a, b)$, accepts data through its end a and atomically, either loses the data or outputs them through its end b. A $\mathit{FIFO}(a, b)$ channel synchronously accepts a data item, d, through its channel end a and stores it in its internal buffer, which has the capacity to hold a single data item. The channel then offers the data item in its buffer through its channel end b and clears its buffer when b dispenses the data item. A $\mathit{Filter}[P](a, b)$ channel behaves almost exactly as a $\mathit{Sync}(a, b)$ channel, except that it passes only those data items from a to b that match its pattern parameter, P. The channel accepts any data item that it receives through a, and either loses the data item if it does not match P, or passes it through b if it does match P. A $\mathit{Transformer}[f](a, b)$ channel behaves like a *Sync* channel, except that it applies the unary function f to every data item that it passes from a to b. The channel silently loses all data items taken from a that are not in the domain of f.

We use two specific variants of the *Transformer* channel to express quantum computing protocols, where instead of the function f we use either a unitary operation U_f that operates on qubits, or a projective measurement operator. In the latter case, we get a classical bit as an outcome, and a distorted qubit (depending on the outcome). Thus, evaluating a function by a unitary operator is a reversible action, whereas measuring qubits, is irreversible.

4 SimulaQron

Motivated by the plan to establish a prototype for quantum internet, researchers have proposed SimulaQron [2] as a platform for developing quantum internet software. With SimulaQron it is possible to simulate the behaviour of a quantum network, where each node may have a share of a quantum entanglement as well as the ability to perform quantum operations on qubits. The back-end of the SimulaQron at each node of the network consists of two main entities: a virtual node and a CQC (classical and quantum combiner) interface. The virtualization of nodes allows us to use different quantum simulators on the network. A virtual node a quantum register, simulated qubits and virtual qubits. A quantum register interacts with the local simulator. Simulated qubits are objects that enable us to manipulate qubits without interacting with quantum registers directly. Finally, virtual qubits are objects with pointers to simulated qubits, some of which may be owned by other virtual nodes, i.e., their pointers may refer to simulated qubits in other virtual nodes. To model entanglement, which excludes the possibility to simulate qubits separately, SimulaQron allows merging virtual nodes in such a way as to place all simulated qubits that are entangled together, in one virtual node. The quantum registers of the merged virtual nodes must be merged as well.

The CQC back-end is an interface for specification of interaction with a quantum network. It enables simulation of sending and receiving qubits to/from a quantum network, command type messages, and information for entanglement management. Figure 1, illustrates the position of the CQC interface in the overall architecture of SimulaQron. For more details see reference [2].

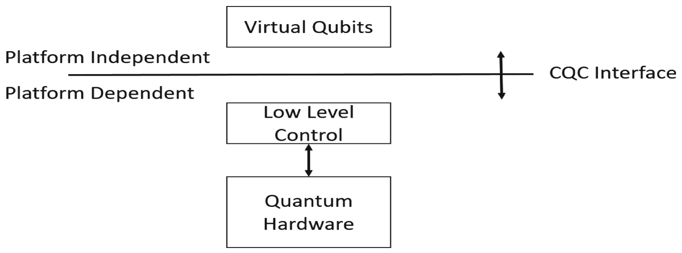


Fig. 1. CQC interface

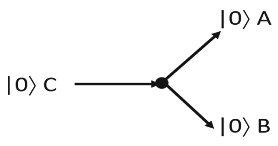
5 Extension: SimulaQ(reo)n

Since quantum entanglement cannot be simulated locally, interdependence of qubits becomes implicit in current models and languages used to express quantum computing. Reo connectors can serve as a middleware that explicitly expresses entanglement, quantum and classical communications, and the protocol for their coordination, all in one structure. Extending Reo with quantum computing primitives can offer a high-level tool for simulating complex interactions among nodes in the quantum networks introduced in the previous section.

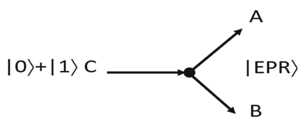
In this work we propose the design of a special coordination layer for quantum components, which relates Reo type connectors with the CQC back-end. To realize this coordination layer, we must extend Reo to support quantum data and operations. However, quantum extension seems incompatible with the semantics of some primitives in classical Reo. For example, Fig. 2 shows two instances of a simple Reo connector, called *replicator*. A replicator consists of three *Sync* channels and its behaviour in the classical data domain is to replicate data that arrive on node C atomically through nodes A and B. However, the no-cloning theorem [1] in quantum mechanics states that no physical process can duplicate a quantum state. Therefore, when qubits arrive at C, two cases need carefull consideration.

Consider a replicator with a fan-out of 2, (similar to the replicators in Fig. 2), we describe the behaviour of this replicator in terms of a quantum operation that is called controlled-NOT ($CNOT$).[1] For $d \in \{0, 1\}$, when a qubit in the state $|d\rangle$ arrives at the source node of this replicator, the replicator creates a qubit in the initial state $|0\rangle$, and subsequently performs the controlled-NOT operation $CNOT(|d\rangle|0\rangle)$. This results in a two qubit system in the state $|dd\rangle$, which is a separable state. Thus, each of the channel ends A and B in the Fig. 2(a), receives a qubit in the state $|d\rangle$, which allows the local "downstream" simulators to manipulate their corresponding qubits separately. On the other hand, if the incoming qubit is in a superposition state, e.g., $|d\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$, the $CNOT$ operation creates an entangled state, e.g., the EPR state $\frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$. Entangled states are not separable, meaning that we cannot assign local states to the qubits arriving at channel ends A and B, in Fig. 2(b). Instead, if at later stage, one measures either of the qubits coming out of nodes A and B, the observed outcome of either $|00\rangle$ or $|11\rangle$ will be the same (correlated) at both ends. This instance of replication demonstrates that local "downstream" quantum simulators cannot always operate on quantum states in a distributed manner: such cases require an entanglement/virtualization management layer.

The idea of using the $CNOT$ operation is taken from the work of Altenkirch [5] in quantum functional programming. To implement the replicator of Fig. 2 in Reo, we place a filter channel before every source node in order to distinguish between classical and quantum data. For every quantum data item, we create a qubit in the initial state $|0\rangle$ and add a transformer channel to perform the $CNOT$.



(a) replicator producing separable states



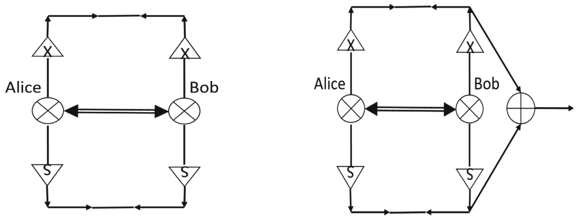
(b) replicator producing entangled states

Fig. 2. Replicator connector

Quantum Key Distribution (QKD) is an example of an industrialized quantum protocol, which can be integrated into a classical network. We now analyze a version, introduced by Ekert [11], where Alice and Bob share classical keys

[1] This two qubits operation consists of a control and a target qubit. If the control gate is set in the state $|1\rangle$, a quantum flip operation (also known as the Pauli X) is applied to target qubit. [1].

using entanglement. In this protocol Alice and Bob share pairs of entangled qubits. Then each party randomly decides on applying quantum measurement in standard basis (S) or X basis (where bases are entangled EPR states) on its share. For those bases that both parties agree, measurement outcomes are in fact shared keys. We illustrate this protocol using Reo connectors in Fig. 3. These connectors use quantum channels (depicted as double line arrows) to produce entangled pairs. The symbol $\bigotimes$ represents Reo's standard exclusive router. Two kinds of transformers, X and S represent quantum measurements in different bases. If we are interested only in sharing keys without external observation, Fig. 3(a), specifies the necessary interaction between parties. However, we often need to know the statistics of cases of agreement between Alice and Bob. To obtain this information, we compose the connector in Fig. 3(a) with a simple circuit that "taps" the flow of data in the protocol circuit and diverts it to a monitor, as in Fig. 3(b). Here $\bigoplus$ represents a component that merely monitors the number of agreements between Alice and Bob. This composition of an external monitor is a desirable feature in the sense that components (Alice and Bob) do not need to be modified while exogenously, we are able to count the number of times they agree on their choices of quantum measurement.



(a) QKD (b) QKD with monitor

Fig. 3. Connectors in Reo

It is also possible to consider local quantum simulations as web services provided to the network, by adopting Reo based orchestration techniques introduced in [6], in the quantum setting. Here the goal would be to develop and implement proxies between Reo connectors (e.g., Fig. 3(b)) and network nodes (e.g., CQC back-end).

Similar to classical Reo, the formal semantics of quantum primitives may be described by (an extension of) constraint automata, where the data domain is extended to include a quantum data type (qubits). The set of states in this case, may include description of quantum states. This is in particular important in the case of *FIFO* channels. The constraint on the data-flow in this channel type is specified by the value in its memory, which may be a quantum state. However, in the generalization of this channel, e.g., $FIFO^n$, where the memory has n cells, we may have both quantum and classical data types. We leave the exact definition of quantum constraint automata and its composition for future work.

There are several case studies at the frontline of implementation of a distributed quantum networks such as quantum leader election, quantum byzantine agreement, and quantum dining philosophers. These are examples where quantum solutions are often faster and simpler (e.g., deterministic) compared with their classical counterparts. For instance, dining philosophers (DP) is a classic problem in distributed system [8] where the effectiveness of exogenous coordination can be neatly demonstrated (see Sect. 7 of [7]). In the quantum version of the DP problem [12], an entangled state $|0^n\rangle + |1^n\rangle$ gets distributed among n parties (this can be done by each philosopher sending an EPR pair to its neighbours). Then each party needs to do internal quantum operations and measurements to (I) run a fair leader election, and (II) form two groups for breaking symmetry. We envisage that adding a Reo connector to generate a coordination layer, separating it from internal actions of each party, simplifies the implementation of quantum DP on quantum internet infrastructure.

6 Future Work and Conclusion

In this short paper we argued that using a coordination model to implement quantum internet software can play an important role in realizing such technology in near future. We explained how Reo coordination concepts can be extended to the setting of the so-called quantum internet. The main line for future work is to formally define the coordination layers for quantum components, and to express its (operational) semantics in an extended version of constraint automata. This must be followed by automatic code generation using an extension of the current Reo compiler [9] to generate executable code for the CQC back-end of the SimulaQron tool. Generating code for this back-end raises the question of existence of specific optimization methods for the Reo compiler, given the non-locality of quantum primitives in a distributed system.

It is also crucial to collaborate with experimental teams to accurately incorporate their requirements and levels of abstractions needed for coordination of quantum software components.

Using formal verification schemes developed for coordination formalisms such as Reo in distributed quantum programming presents an important line for future work. Full implementation of quantum algorithms for dining philosophers and Byzantine agreement on SimulaQron using the extension presented in this paper is an interesting line of future work.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
2. Dahlberg, A., Wehner, S.: SimulaQron - A simulator for developing quantum internet software. Quantum Sci. Technol. **4**(1) (2019). https://doi.org/10.1088/2058-9565/aad56e
3. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3), 329–366 (2004)

4. Castelvecchi, D.: The entangled web. Nature **554**, 289–292 (2018)
5. Altenkirch, T., Grattage, J.: A functional quantum programming language. In: 20th Annual IEEE Symposium on Logic in Computer Science, LICS 2005, pp. 249–258 (2005)
6. Jongmans, S.-S.T.Q., Santini, F., Sargolzaei, M., Arbab, F., Afsarmanesh, H.: Automatic code generation for the orchestration of web services with Reo. In: De Paoli, F., Pimentel, E., Zavattaro, G. (eds.) ESOCC 2012. LNCS, vol. 7592, pp. 1–16. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33427-6_1
7. Arbab, F.: Composition of interacting computations. In: Goldin, D., Smolka, S.A., Wegner, P. (eds.) Interactive Computation, pp. 277–321. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-34874-3_12
8. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Informatica **1**, 115–138 (1971)
9. ReoLanguage GitHub repository. https://github.com/ReoLanguage/Reo. Accessed 23 Mar 2018
10. Microsoft Quantum Dev Kit (2018). https://www.microsoft.com/en-us/quantum/
11. Ekert, A.K.: Quantum cryptography based on Bell's theorem. Phys. Rev. Lett. **67**, 661–663 (1991)
12. Aharonov, D., Ganz, M., Magnin, L.: Dining Philosophers, Leader Election and Ring Size problems, in the quantum setting. arXiv: 1707.01187 (2017)
13. Arbab, F.: Puff, the magic protocol. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 169–206. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24933-4_9
14. Arbab, F.: Proper protocol. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods. LNCS, vol. 9660, pp. 65–87. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30734-3_7
15. Arbab, F.: What do you mean, coordination? In: Bulletin of the Dutch Association for Theoretical Computer Science, NVTI, pp. 11–22 (1998)