Programming and Simulation of Quantum Search Agents

Matthias Klusch and René Schubotz

***Abstract*— The extension of classical agents by the ability to perform quantum computation and communication provides an efficient and secure solution to applications such as information search and service matchmaking. In this paper, we propose a hybrid architecture for quantum computational agents, and demonstrate its principles by means of a simple type-I quantum search agent based on quantum pattern matching. Finally, we present preliminary results of the comparative evaluation of its implementation using different quantum programming languages and simulators.**

I. INTRODUCTION

Quantum computing technology based on quantum physics promises to eliminate some of the problems associated with the rapidly approaching ultimate limits to classical computers imposed by the fundamental law of thermodynamics. Quantum computing (QC) devices have been physically implemented since the late 1990's by use of, for example, nuclear magnetic resonance, and solid state technologies. Rapid progress and current trends in nanoscale molecular engineering, as well as quantum computing research carried out at research labs across the globe could make it happen to let us see increasingly sophisticated quantum computing devices in the era 2020 to 2050. The implied major research challenge of agent based computing in such environments is how to make the most of the potential of quantum computing and communication? We acknowledge that any answer to this question at the very moment will be, of course, highly speculative; though work in this direction already started such as in [7]. We build upon this work and focus more on its engineering aspects by means of programming and simulation of a special kind of quantum computational agents, that is type-I quantum search agents. Key idea is to appropriately extend one prominent generic agent architecture, namely InteRRap, to the case of a type-I QC agent that is supposed to run on a hybrid quantum computer, and to show its feasibility by instantiating the respective QuantumInteRRap architecture for a programmed quantum pattern matching (QPM) based type-I quantum search agent. The remainder of the paper is structured as follows. After a brief introduction to quantum computation in section II, we comment on a recently proposed classification of QC agents and a quantum programming design flow in sections III-A, respectively, III-B. Based on this work, we present our quantum extension of the InteRRap architecture for QC agents in section III-C, and describe a slightly improved version of the quantum pattern matching algorithm of [9] in section IV-A. The architecture and benchmarking results of the QPM based type-I quantum search agent simulated on different quantum simulators are presented in sections IV-B and V.

M. Klusch is with the German Research Center for Artificial Intelligence, Multiagent System Group, Saarbrücken, Germany klusch@dfki.de

R. Schubotz is with the Saarland University, Computer Science Department, Saarbrücken, Germany schubotz@gmx.de

II. QUANTUM COMPUTING IN VERY BRIEF

Quantum computation is built on the concept of the *qubit*. Any isolated physical 2-observable quantum system is appropriate to realize a single qubit. In mathematical terms, a qubit ψ is associated to its state space, a complex 2-dimensional Hilbert space $H_2 = span\{|0\rangle, |1\rangle\}$ with orthonormal computational standard basis. Any quantum state $|\psi\rangle$ of ψ is described by a *coherent superposition*, $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \quad |\alpha_0|^2 + |\alpha_1|^2 = 1$. The state space $H_2^{\otimes n}$ of a physical system composed of n single qubits ψ_i is the n-folded *tensor product* of the state spaces of its n constituting qubits $H_2^{\otimes n} = \bigotimes^n H_2$. Such systems can be regarded as n*-qubit register* Ψ with 2^n computational basis states. If a state $|\Psi\rangle$ of a n-qubit register can be written as a product of its constituting qubits in the form $|\Psi\rangle = \bigotimes_i (\sum_j \alpha_{i,j} |j\rangle)$, then $|\Psi\rangle$ is called separable. Non-separable composite states are known as *entangled states*, allowing non-local effects of instantaneous state changes between spatially separated but entangled quantum states upon measurement. A *projective measurement* of Ψ is described by a set of pairwise orthogonal subspaces $W_1, \ldots, W_m$ satisfying $H_2^{\otimes n} = \bigoplus_{k=1}^m W_k$ and results in $j \in \{1, \ldots, m\}$. Let $\{|\Phi_l^j\rangle\}$ define an orthonormal basis of subspace W_j, then the operator $P_j = \sum_{i=1}^{dim(W_j)} |\Phi_i^j\rangle \langle\Phi_i^j|$ projects Ψ on the subspace W_j. The probability of measuring $j \in \{1, \ldots, m\}$ is given by $\langle\Psi| P_j |\Psi\rangle$. Time evolution of Ψ is described by *unitary transformations* in its state space $H_2^{\otimes n}$. Any non-measuring quantum operation is *inherently reversible* since any unitary transformation U has an inverse. For a comprehensive introduction to quantum computation, we refer the reader to [12].

III. AGENTS ON QUANTUM COMPUTERS

A *quantum computational agent* (QC agent) [7] is an intelligent software agent that is able to perform both classical and quantum computing to accomplish its goals individually, or in joint interaction with other QC agents. The future quantum internet is expected to consist of networked classical and quantum computers, and populated with QC agents that operate on quantum computers and communicate with each other according to the quantum communication model of either physical direct quantum transmission, or quantum teleportation, or superdense coding.

A. Classification of QC Agents

According to [8], QC agents can be classified based on the used quantum communication model. *Type-I QC agents* communicate over classical channels and are restricted to local quantum computing. Communication between type-I QC agents has to be additionally secured which is not necessary in case of inherently secure type-II QC agents. With respect to the classification of different types of *agent autonomy* in [4], it has been shown in [7] that the self-autonomy of individual type-I QC agents remains intact. The ability to autonomously reason about sets of goals, plans, and motivations for decision-making is not affected, since quantum computational effects are restricted to local quantum machine components. *Type-II QC agents* can be distinguished depending on whether entangled qubits for quantum communication are shared between communicating agents, or not. Communication between *type-IIa QC agents* is based on direct quantum particle transmission. In contrast to type-IIb QC agents, entangled qubits are not shared. Using the entangled qubits at its disposal, a *type-IIb QC agent* is able to transmit superdense coded information to its communication partner, alternatively messages can be teleported. Type-II QC agents greatly benefit from computational advantages of quantum computing and communication, but at the cost of limited self-autonomy, due to non-local effects of quantum entanglement.

B. Quantum Computing Design Flow

Quantum computers are highly fragile artifacts due to their delicate machinery and require precise monitoring of their state and operation. The proposal of a hybrid master-slave architecture in [7] comes naturally, and casts a classical machine (CM) to control the timing and sequence of quantum operations carried out in a quantum machine (QM). As depicted in figure 1, the CM consists of a CPU

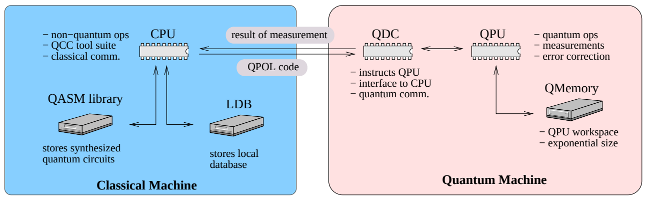
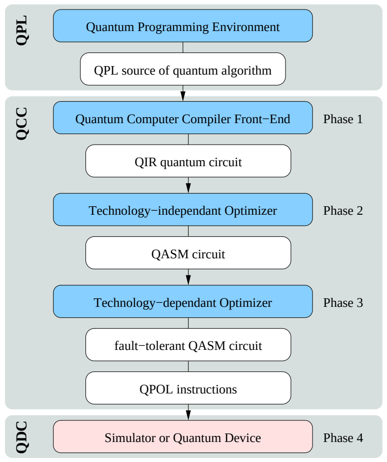


Fig. 1. Master-slave hybrid quantum computer

for high-level dynamic control and scheduling of the QM components. The CM provides a *quantum programming language*[13], [2], [14] (QPL) compiler suite that translates quantum algorithms into low-level quantum device instructions. It interfaces to the quantum machine via the quantum device controller (QDC), an instruction processing unit that distributes technology-dependent machine code to a quantum processing unit (QPU).

In [16], Aho et al. envision a layered software architecture for quantum computing design tools, and propose a simple hierarchy with interfaces between QPL, QPL compilers, optimizers, simulators, and layout modules. Key concept of this architecture is a four-phase computer-aided design flow (see figure 2), that ensures an interoberable tool hierarchy by appropriate intermediate representations of quantum algorithms between the different phases. The flow's first three phases are handled by a *quantum computer compiler* (QCC) residing on a classical machine, whereas the final phase mounts the QDC and implements the quantum circuit on the underlying quantum device. A top level quantum programming environment



QPL
Quantum Programming Environment
QPL source of quantum algorithm
QCC
Quantum Computer Compiler Front−End
Phase 1
QIR quantum circuit
Technology−independant Optimizer
Phase 2
QASM circuit
Technology−dependant Optimizer
Phase 3
fault−tolerant QASM circuit
QPOL instructions
QDC
Simulator or Quantum Device
Phase 4

Fig. 2. Quantum design flow phases on a classical computer

offers a high-level QPL, and thereby provides mathematical abstractions of quantum mechanics and linear algebra in a hardware and technology-independent fashion. Such a QPL is expected to be based on familiar concepts from classical programming and software engineering, e.g., work-flow control, type systems, code reusablility and modularization. QPL source programs containing high-level primitives for logical quantum operations and classical work-flow statements, are passed to the QCC during the first phase of the design flow. The QCC maps QPL sources into a technology-independent *quantum intermediate representation* (QIR), i.e., a quantum circuit using gates from a universal set. Since this process is similar to the front-end processes of classical compilers, approved algorithms for lexical, syntactic and semantic analysis can be used. During the second phase, a technology-independent circuit optimizer translates the QIR into a circuit of elementary quantum gates. The resulting *quantum assembly language* (QASM) representation is optimized according to various cost functions. QASM circuits consist solely of qubits, classical registers, single-qubit gates, CNOT gates, and measurement operations. Any quantum circuit can be constructed in QASM, but efficient synthesis algorithms [15] are still a wide area of research. At the earliest in the third phase, knowledge of the physical layout and technology-dependent limitations enters the design flow. The third phase consists of two subphases. During the first subphase, a technology-specific optimizer maps the QASM representation from the second phase into a QASM circuit with gates drawn from a fault-tolerant discrete universal set. The second subphase outputs *quantum physical operations language* (QPOL) and precisely describes the execution of a given QASM circuit on a specific technology. QPOL includes physical operations and technology-specific modules. During the last phase, the QCC interfaces with the QDC. Utilizing device-specific tools and modules, the QDC translates a final QPOL program into quantum machine instructions and distributes them to a QPU or a quantum simulator. In order to minimize quantum decoherence caused by imperfect control over qubit operations, measurement errors, number of entangled qubits, and the physical limits of the quantum system, fault tolerance and error correction can be added at multiple phases of the design process.

C. Generic QC Agent Architecture QuantumInteRRap

In [10], an architecture for multi-agent systems is presented. The proposed model *InteRRap* combines both the reactive and the deliberate paradigm, and explicitly represents knowledge, plans and strategies. Including a mechanism for devising joint plans, knowledge about protocols and communication strategies, InteRRap is suitable for describing high-level interactions of autonomous and intelligent agents. In the following, the InteRRap architecture is embedded in the context of QC multi-agent systems. Figure 3 shows the high-level components of the *QuantumInteRRap* architecture and their basic interplay. Following the InteRRap model, the basic building blocks of a QuantumInteRRap agent are a hierarchical structured *knowledge base* (KB) and the *agent control unit* (ACU). The KB consists of the agent's *world model*, the agent's *local goals*, the *behavioural knowledge*, the *local planning knowledge* and the *cooperative planning knowledge*. The *world model* is organized in a taxonomical fashion, and contains the agent's beliefs and knowledge about the state of the world. Besides information about its macroscopic environment, a QuantumInteRRap agent has beliefs and knowledge about the quantum realm. The agent's *behavioural knowledge* is defined by *patterns of behaviour* (PoB). PoB represent the elementary problem-solving facilities of QC agents. On the one hand, PoB provide information regarding their suitability, applicability, and expected utility under certain conditions. Inter alia, the declaration of a PoB contains information concerning *a priori* evaluation, external preconditions and postconditions, termination conditions, failure conditions, or execution of corresponding lower-level procedures, enabling the plan-based modules to reason about which PoB to perform, and which not. On the other hand, there is a class of PoB that are not represented in a declarative manner. These PoB can be considered as precondition-action-postcondition triples, and are linked to precompiled procedures that can be activated in for routine behaviour. *Quantum patterns of behaviour* (qPoB) declare more or less complex quantum operations, e.g., error correction, specific quantum circuits, measurements in various bases, etc. It is important to realize, that qPoB encapsulate quantum operations by mathematical abstractions of quantum mechanics and linear algebra, and do not restrict quantum operations to a fixed word size. QPL source code attached to a qPoB is transformed into device-specific instructions. The agent's *local planning knowledge* provides, i.e., local skeletal plans from a plan library, and specific knowledge for standard from-scratch planning. QuantumInteRRap represents plans as tree structures whose leaf nodes contain only (q)PoB. QuantumInteRRap plan structures provide a suitable high-level description of quantum algorithms abstracting away from quantum technology-specific details. Finally, the agent's *cooperation knowledge* contains, i.e., joint plan structures, negotiation protocols, and cooperation-specific partner information. In particular, type-IIb QuantumInteRRap agents possess knowledge about quantum cooperation strategies, e.g., distribution of entangled qubits, quantum leader election, or quantum distributed consens, and high-level quantum communication, e.g., superdense coding, quantum teleportation, and direct qubit transmission.

The ACU comprises the *world interface* (WIF), the *behaviour-based layer* (BBL), the *local planning layer* (LPL) and the *cooperative planning layer* (CPL). As adumbrated, the control components exchange goals, plans and information via communication. In detail, the *WIF* provides the agent's means for perception, actuation and communication according to the classification in section III-A. Any information the agent receives or perceives need to be transformed into an explicit representation for storage in the agent's world model. In consequence of the deployed communicative facilities, *type-I QC agents* do not suffer from incomplete quantum knowledge. *Type-IIb QC agents* may obtain quantum knowledge by classical communication assuming a non-antagonistic agent scenario, whilst *type-IIa QC agents* are inherently uninformed about received but not yet measured quantum bits. The *BBL* implements the agent's basic behaviour, its execution and decision component. The BBL has access to a set of executable (quantum) patterns of behaviour which can be activated by external stimuli or by the LPL. The BBL has the task to select and execute suitable and applicable (q)PoB that contain activation calls to WIF modules, or to the LPL. In order to reasonably choose (q)PoB, the BBL needs to maintain a goal priorization hierarchy. The goals of the agent are generated from its world model, and correspond to (q)PoB. The *LPL* consists of a plan generator , a plan evaluator, a resource handler, a control component, and an interface to the BBL. The LPL is able to generate and to control the execution of local single-agent plans, to interpret an incoming plan structure from another agent, to return plan structures to the BBL in order to tell another agent a plan for a certain goal, and to decide which plan to choose from a set of alternative plans. The resource handler interfaces to planning-specific resources, e.g., planning knowledge, and can be commisioned to provide a certain resource to a LPL component. The plan generator has access to a plan library and a standard from-scratch planning algorithm. The plan generator either chooses a suitable plan from its library, or it devises a plan from scratch. The plan evaluator associates plans with their expected utilities, and therefor capacitates the plan interpreter to decide amongst a set of alternative plans. By means of a plan interpreter, the control component decomposes a devised plan, and uses its interface to instruct the BBL to activate

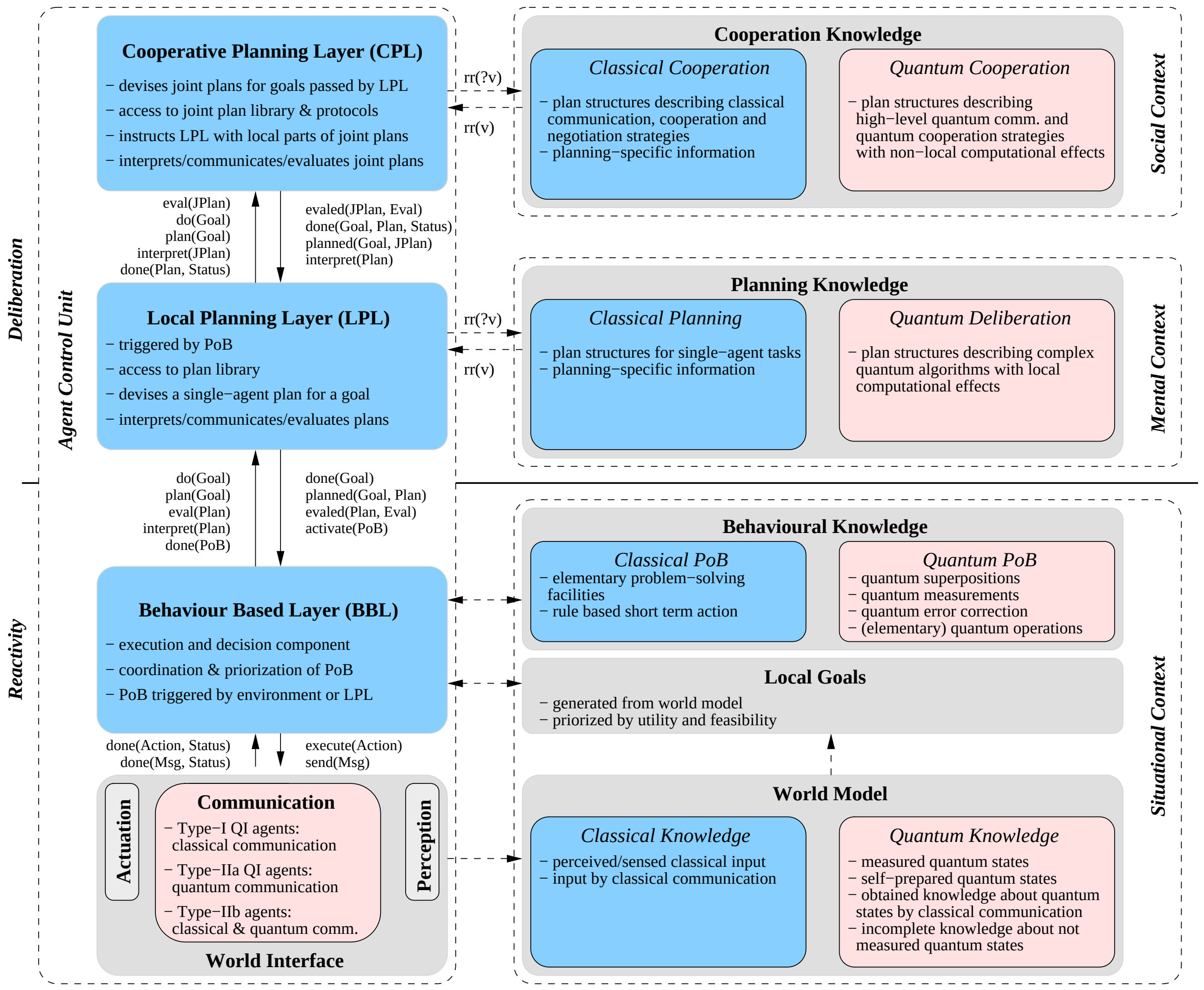

Cooperative Planning Layer (CPL)
− devises joint plans for goals passed by LPL
− access to joint plan library & protocols
− instructs LPL with local parts of joint plans
− interprets/communicates/evaluates joint plans
rr(?v)
rr(v)
Cooperation Knowledge
Classical Cooperation
− plan structures describing classical communication, cooperation and negotiation strategies
− planning−specific information
Quantum Cooperation
− plan structures describing high−level quantum comm. and quantum cooperation strategies with non−local computational effects
Social Context
eval(JPlan)
do(Goal)
plan(Goal)
interpret(JPlan)
done(Plan, Status)
evaled(JPlan, Eval)
done(Goal, Plan, Status)
planned(Goal, JPlan)
interpret(Plan)
Deliberation
Agent Control Unit
Local Planning Layer (LPL)
− triggered by PoB
− access to plan library
− devises a single−agent plan for a goal
− interprets/communicates/evaluates plans
rr(?v)
rr(v)
Planning Knowledge
Classical Planning
− plan structures for single−agent tasks
− planning−specific information
Quantum Deliberation
− plan structures describing complex quantum algorithms with local computational effects
Mental Context
do(Goal)
plan(Goal)
eval(Plan)
interpret(Plan)
done(PoB)
done(Goal)
planned(Goal, Plan)
evaled(Plan, Eval)
activate(PoB)
Reactivity
Behaviour Based Layer (BBL)
− execution and decision component
− coordination & priorization of PoB
− PoB triggered by environment or LPL
Behavioural Knowledge
Classical PoB
− elementary problem−solving facilities
− rule based short term action
Quantum PoB
− quantum superpositions
− quantum measurements
− quantum error correction
− (elementary) quantum operations
Local Goals
− generated from world model
− priorized by utility and feasibility
Situational Context
done(Action, Status)
done(Msg, Status)
execute(Action)
send(Msg)
Actuation
Communication
− Type−I QI agents: classical communication
− Type−IIa QI agents: quantum communication
− Type−IIb agents: classical & quantum comm.
Perception
World Interface
World Model
Classical Knowledge
− perceived/sensed classical input
− input by classical communication
Quantum Knowledge
− measured quantum states
− self−prepared quantum states
− obtained knowledge about quantum states by classical communication
− incomplete knowledge about not measured quantum states

Fig. 3. QuantumInteRRap architecture

a (q)PoB. The *LPL* consists of a plan generator , a plan evaluator, a resource handler, a control component, and an interface to the BBL. The LPL is able to generate and to control the execution of local single-agent plans, to interpret an incoming plan structure from another agent, to return plan structures to the BBL in order to tell another agent a plan for a certain goal, and to decide which plan to choose from a set of alternative plans. The resource handler interfaces to planning-specific resources, e.g., planning knowledge, and can be commisioned to provide a certain resource to a LPL component. The plan generator has access to a plan library and a standard from-scratch planning algorithm. The plan generator either chooses a suitable plan from its library, or it devises a plan from scratch. The plan evaluator associates plans with their expected utilities, and therefor capacitates the plan interpreter to decide amongst a set of alternative plans. By means of a plan interpreter, the control component decomposes a devised plan, and uses its interface to instruct the BBL to activate a (q)PoB. Again, in consequence of the deployed communication facilities, *type-II QC agents* may perform quantum operations on received quantum systems, whereas *type-I QC agents* can only operate on self-prepared quantum systems. The *CPL* implements mechanisms for devising joint plans based on the goals of the agent. It has a notion of protocols and communication strategies and access to a joint plan library. Similar to the LPL, the CPL consists of a joint plan generator , a joint plan evaluator, a resource handler, a control component, an interface to the LPL, and in addition a joint plan translator. The CPL is able to device and to execute joint plans for goals passed to it by its next lower layer, to interpret an incoming joint plan structure from another agent, to return joint plan structures to the LPL in order to tell other agents a joint plan for a common goal, and to decide which plan to choose from a set of alternative plans. The resource handler interfaces to planning-specific resources, e.g., planning knowledge, or negotiaton protocols, and can be commisioned to provide a certain resource to a CPL component. The resource handler looks up for cooperative planning knowledge in the agent's local KB or in appropriate parts of the knowledge bases of cooperating agents. The joint plan generator has access to a joint plan library and a standard from-scratch planning algorithm. The plan generator either chooses a suitable plan from its library, or it devises a plan from scratch. The joint plan generator is based on the agent's local goals and on the goals of agents participating in an interaction. The joint plan evaluator evaluates a joint plan that has been generated by the plan generator or that has been proposed by another agent, and associates joint plans with their expected utilities. Since joint plans are subject to negotiation, agents need to argue about the appropriateness of joint plans. Depending on the applied evaluation strategies, agents may commit to solutions that maximize local or global utility. By means of the joint plan translator, the control component translates a joint plan into single-agent plans which guarantee satisfaction of joint plan constraints during local plan execution. The CPL of *type-IIb QC agents* is able to devise joint plans involving quantum communication and quantum cooperation strategies. To this end, joint plan structures containing appropriate (q)PoB are generated and executed on the participating QC agents' hybrid machines. Since qPoB describe quantum operations in an abstract fashion, type-IIb QC agents based on different quantum technologies are enabled to cooperate.

IV. A Type-I Quantum Search Agent

Quantum computers can be of avail for certain search problems. Exploiting the principles presented in chapter II, efficient quantum algorithms can be composed of sequenced unitary operations interleaved with classical work-flow statements and a finalising measurement operation. Section IV-A explicates a *quantum pattern matching algorithm* [9] and provides the grounding for a *type-I quantum search agent* in section IV-B.

A. Quantum Pattern Matching

The problem of determining the *closest match* with a given pattern $p \in \Sigma^M$ of size $M \ll N$ in an unstructured database string $w \in \Sigma^N$ of size N has been solved in [9] by extending Grover's quantum search algorithm [5][6]. The query complexity of QPM has been proven to be $O(\sqrt{N-M})$ allowing for a significant speedup by an order of magnitude compared to classical matching approaches such as *approximate swapped matching*[1] in $O(N\log(M)\log(\min(M,|\Sigma|)))$. Applying a *compile once, run many* approach, the QPM algorithm enables to search for an arbitrary large number of distinct patterns in a given database. To this end, the i-th position of w is encoded by $|i\rangle \in H_2^{\otimes\lceil log(N)\rceil}$. Hence, the quantum state

$$|\chi\rangle = \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle$$

superposes all positions of database string w. For the purpose of actually generating this superposition from a simple initial state, we propose to apply the methods proposed in [17]. For each symbol $\sigma \in \Sigma$, a symbol oracle Q_σ is given by

$$Q_\sigma|\chi\rangle = (\mathbf{1} - 2\sum_{1\le j\le N}|j\rangle\langle j|)\,|\chi\rangle$$

for positions j of w satisfying $w_j = \sigma$. The algorithm is constituted by iterating the phase shifts induced by a symbol oracle Q_{σ_l} for a *random symbol* σ_l, $0 \le l < M$ of the pattern followed by Grover's amplitude amplification through operator

$$P_\psi|\chi\rangle = (2|\psi\rangle\langle\psi| - 1)\,|\chi\rangle\,, \quad |\psi\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$$

In order to apply the effects of Q_{σ_l} to the correct starting positions $|k\rangle$, $0 \le k \le N-M$ of matching database substrings $\langle w_k ... w_l ... w_{k+M-1}\rangle$, the states $|i\rangle$ corresponding to the positions of w need to be permuted in each iteration. This can be done reversibly using

$$P_\pi^l|\chi\rangle = P_\pi^l\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle = \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i+l \mod N\rangle$$

If $M \ll N$, sampling randomly over M elements will lead to searching with high probability over all symbols of the pattern. On average, a position with a partial match of $M' \le M$ individual symbols will experience $\frac{M'}{M}$ phase shifts.

Algorithm 1 Quantum pattern matching

Input: $w \in \Sigma^N, p \in \Sigma^M, n = \lceil\log(N)\rceil$
Output: $m \in \mathbb{N}$
Quantum Variables: $|\chi\rangle \in H_2^{\otimes n}$
Classic Variables: $r, i, j \in \mathbb{N}$

```
 1: Choose $r \in [0, \lfloor\sqrt{N-M+1}\rfloor]$ uniformly
 2: Set $|\chi\rangle = \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1}|k\rangle$
 3: for $i = 1$ to $r$ do
 4:     Choose $j \in [0, M-1]$ uniformly
 5:     Set $|\chi\rangle = P_\pi^j|\chi\rangle$
 6:     Set $|\chi\rangle = Q_{\sigma_j}|\chi\rangle$
 7:     Set $|\chi\rangle = (P_\pi^j)^{-1}|\chi\rangle$
 8:     Set $|\chi\rangle = P_\psi|\chi\rangle$
 9: end for
10: Set $m$ to the result of the measurement of $|\chi\rangle$
```

B. Type-I Quantum Search Agent Architecture

How to model a type-I quantum search agent (QSA) by use of the QuantumInteRRap architecture presented in section III-C. According to section III-B, the QSA is supposed to run on a hybrid quantum computer which is, e.g., provided with a unstructured classical *local database* (LDB). For the sake of simplicity, we consider this LDB as a long static string $w \in \Sigma^N$ of size N. Figure 4 illustrates how a *Type-I QSA agent* can be modeled as a *QuantumInteRRap* agent.

The plan library of the QSA agent contains a plan structure for database management given by

```
update_db =
[ rr(update(?w)),
  activate(store(w)),
  activate(parse(w, ?a)),
  activate(synth(w, a)) ]
```

Once a database setup or update is perceived, the BBL asks the LPL to devise a local plan for this situation. The

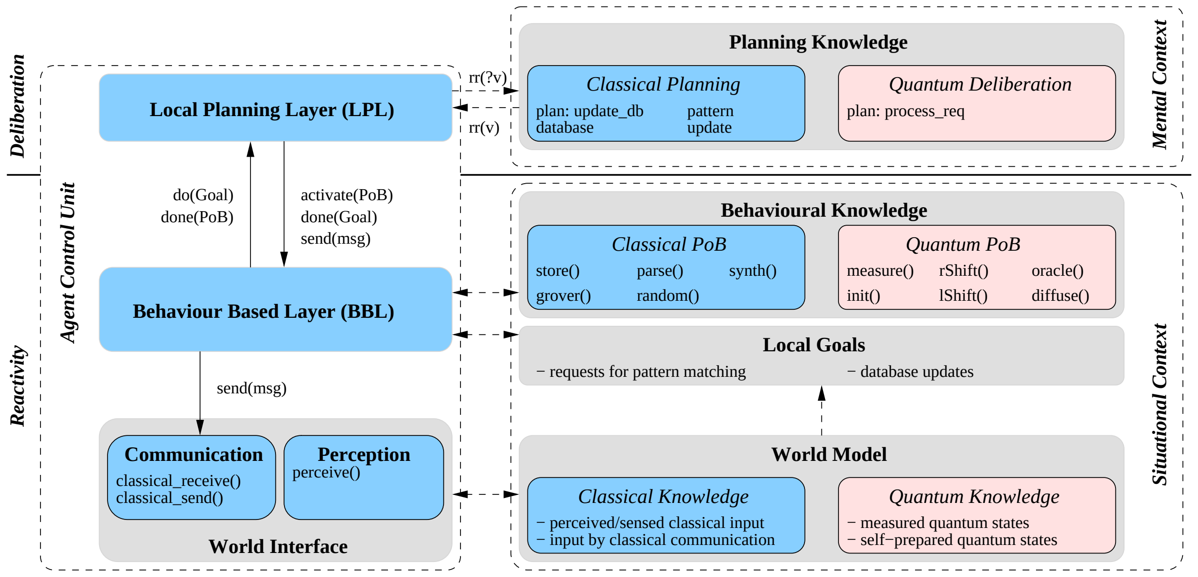


Fig. 4. QuantumInteRRap architecture of a QPM based type-I quantum search agent

LPL selects the appropriate plan structure $update\_db$ from its plan library and extracts the new database string from the world model using its resource handler. Now, the LPL instructs the BBL to update the LDB using PoB $store$. When control comes back to the LPL, it asks the BBL to activate PoB $parse$ that extracts the symbol alphabet and further information for oracle synthesis from the LDB. Finally, the LPL commands the BBL to synthesize oracle circuits for each symbol from the database alphabet. The resulting QASM circuits are stored for further usage. Oracle synthesis is executed by pattern $synth$.

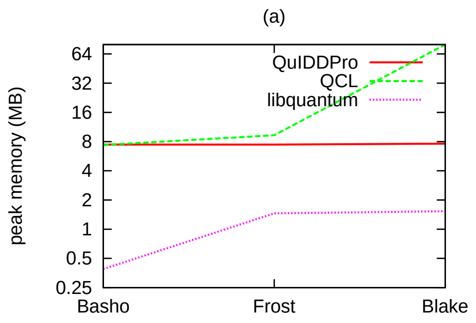
A plan for query processing is given by

```
process_req =
[ rr(database(?db)),
  rr(pattern(?r)),
  activate(grover(db, r, ?g)),
  activate(init(db)),
  while (i=0; i<g; i++) {
    activate(random(r, ?p)),
    activate(rShift(p)),
    rr(pattern_symbol(r, p, ?s)),
    activate(oracle(s)),
    activate(lShift(p)),
    activate(diffuse(db))
  },
  activate(measure, ?result),
  rr(substring(result, ?match)),
  send(match) ]
```

Upon receipt of request s containing some pattern $r \in \Sigma^M$ from another agent A_1, the BBL asks the LPL to process the query. To this end, the LPL devises and controls the execution of a local plan according to the plan structure $process\_req$ from its plan library. Since aspects of the QPM algorithm depend on database size and pattern p, the LPL performs respective requests to the world model. Now, the LPL asks the BBL to compute the correct number of grover iterations using PoB $grover$, and afterwards commands the BBL to initialize a quantum register of appropriate size via qPoB $init$. The plan interpreter of the LPL now decomposes the while-loop of the devised plan, and iteratively instructs the BBL the activate the specified (quantum) patterns of behaviour. Pattern $random$ uniformly chooses an index of the query pattern for which the correct permutation $rShift$, and $lShift$, and the correct precompiled symbol oracle operation $oracle$ need to be applied. The last step of a while-loop iteration is the application of qPoB $diffuse$ that implements Grover's diffusion operation. The interaction with the quantum machine is finalized by a projective measurement of the quantum register resulting in the database index of the closest matching substring to the requested pattern. Finally, the LPL extracts the corresponding substring from the LDB, and by passing the substring to the lower level of the WIF, the requesting agent A_1 receives the computed closest match to its query string, or a failure message in case of the unsuccessful search.

V. Simulation and Benchmarking

Quantum computer simulators enable the simulation of computational operations designed for quantum machines on classical computing machinery. To demonstrate the operability of our type-I QSA, we have programmed and simulated it by use of open source quantum simulators *QuIDDPro 2.1* [20], [19], [18], *QCL 0.6.1* [13] and *libquantum 0.2.4* [3]. For extensive review of quantum computer simulators, we refer the reader to [21]. Memory and runtime performances for simulations on a classical computer with query patterns of size $|p| = 4$ and various database strings are displayed in figure 5. We simulated the QSA for three different use case scenarios: LDB set up with Matsuo Basho's *Frog*



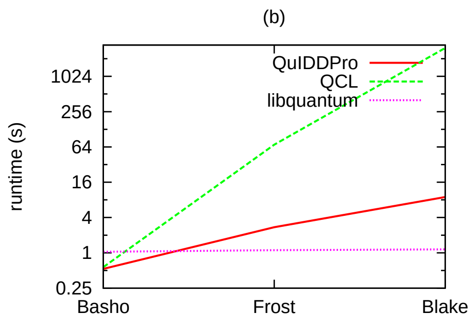


Fig. 5. Comparing peak memory (a) and runtime (b) of Qsa on selected quantum computing simulators

Haiku encoded in 6 qubits, LDB set up with Robert Frost's *Fire and Ice* encoded in 8 qubits, and LDB set up with William Blake's *The Tyger* encoded in 10 qubits. We strongly highlight, that our comparative evaluation soley investigates selected quantum computer simulators with respect to memory and runtime performance on classical machines. Using a not yet implemented quantum computing device, QPM's quantum query complexity of $O(\sqrt{N-M})$ will allow for a significant speedup of our QSA.

VI. Related Work

To the best of our knowledge, the presented architecture, programming and simulation of a type-I quantum search agent is unique as there are no alternative QC agent implementations available yet. Our work combines related work from quantum computing and agent based computing, and builds upon existing work on QC agents in [7]. In particular, we did exploit an improved version of the recently proposed quantum pattern matching algorithm in [9] for speeding up the local search process of a type-I quantum search agent. The architecture of such an agent on a hybrid quantum computer is then proposed to be an appropriately extended version of the classic InteRRap agent architecture in [10] for QC agents on hybrid quantum computers. The programming of this special kind of QC agent is given by means of three different existing quantum simulators, and demonstrated by example.

VII. Conclusion

In this paper, we presented a hybrid generic architecture for QC agents as an extension of the known InteRRap architecture, discussed basic engineering aspects of how to realize QC agents on hybrid quantum computers, instantiated the QuantumInteRRap architecture by concrete means of a quantum pattern matching based type-I quantum search agent, showed the results of comparative run time performance testing of its simulation with different quantum simulators, and demonstrated its functionality also by example. The theoretical performance of the QPM based quantum search agent over classical edit based string matching provides strong evidence for the expected significant speed up of a service matchmaking process of type-I quantum matchmaker agents compared to the classical case. However, the quantum realization of semantic service matchmaking remains one open problem. Our ongoing research focuses on solving this problem by feasible type-II QC agents for service selection, that is the programming and simulation of type-II quantum matchmaker agents with QuantumInteRRap architecture and by use of the same quantum simulators.

References

[1] Amir. Approximate swapped matching. *Inf. Process. Lett.*, 83(1):33–39, 2002.
[2] Bettelli. Toward an architecture for quantum programming, eur. phys. j., 25:181–200, 2003., 2003.
[3] B. Butscher. Non-technical description of libquantum, enyo.de/libquantum/.
[4] C. Carabelea, O. Boissier, and A. Florea. Autonomy in multi-agent systems: A classification attempt. In Nickles et al. [11], pages 103–113.
[5] L. K. Grover. A fast quantum mechanical algorithm for database search, arxiv.org/quant-ph/9605043, 1996.
[6] L. K. Grover. From schrödinger's equation to the quantum search algorithm. *American Journal of Physics*, 69(7):769–777, 2001.
[7] M. Klusch. Toward quantum computational agents. In Nickles et al. [11], pages 170–186.
[8] M. Klusch. Coordination of quantum internet agents. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 1221–1222. ACM, 2005.
[9] P. Mateus and Y. Omar. Quantum pattern matching, arxiv.org/quant-ph/0508237. 2005.
[10] J. Müller and M. Pischel. The agent architecture interrap: Concept and application, technical report rr-93-26, dfki saarbrucken, 1993, 1993.
[11] M. Nickles, M. Rovatsos, and G. Weiß, editors. *Agents and Computational Autonomy (AAMAS 2003)*.
[12] M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge Univ. Press, Cambridge, 2000.
[13] B. Oemer. Quantum programming in qcl, master thesis, technical university of vienna, computer science department, 2000.
[14] P. Selinger. Towards a quantum programming language. *Mathematical. Structures in Comp. Sci.*, 14(4):527–586, 2004.
[15] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum logic circuits, 2004.
[16] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov. A layered software architecture for quantum computing design tools. *Computer*, 39(1):74–83, 2006.
[17] C. A. Trugenberger. Phase transitions in quantum pattern recognition, arxiv.org/quant-ph/0204115. 2002.
[18] Viamontes. Gate-level simulation of quantum circuits, arxiv.org/quant-ph/0208003, 2002.
[19] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2:347, 2003.
[20] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the density matrix representation. *Quantum Information and Computing*, 5:113, 2005.
[21] J. Wallace. Quantum computer simulators - a review version 2.1, citeseer.ist.psu.edu/wallace99quantum.html.