A Cross-Level Traceable Modeling for Hybrid Quantum–Classical Software

Samuel Sepúlveda[1], **Claudio Navarro**[1], **Ania Cravero**[1], **Enrique Moguel**[2]

[1]Departamento de Ciencias de la Computación e Informática
Universidad de La Frontera – Temuco – Chile.

[2]Quercus Software Engineering Group – Universidad de Extremadura
Cáceres – Spain.

{samuel.sepulveda,claudio.navarro,ania.cravero}@ufrontera.cl,enrique@unex.es

Abstract. *Hybrid quantum–classical software requires engineering approaches that connect stakeholder goals, variability decisions, and architecture under NISQ constraints. We present Quantum Variability Management (QVM), a lightweight, requirements-driven modeling approach that integrates iStar, UVL, and Quantum-UML, through explicit cross-level traceability. QVM defines five rules that relate goals to features, constraints, component stereotypes, and configuration properties such as the provider and qubits. QVM focuses on the requirement-to-variability-to-architecture chain and on the auditable propagation of design changes. We assess the proposal through a scenario-based walkthrough in a logistics case, showing how changes in strategic intent, backend selection, and hardware quality assumptions can be propagated consistently across the three levels. The proposal provides a foundation for controlled reconfiguration in HQC systems, while broader user studies, multi-domain replications, and full tool integration remain future work.*

1. Introduction

Quantum computing (QC) is transitioning from theory to practice in domains such as cryptography, drug discovery, materials science, and machine learning [Stirbu and Haghparast 2023]. However, current NISQ devices remain constrained by noise, errors, and calibration drift, while quantum algorithm design demands specialized expertise, raising the entry barrier for many practitioners [De Stefano et al. 2022].

Hybrid quantum-classical software (HQC) is growing in the NISQ era, yet engineering practices remain fragmented in *requirements*, *variability*, and *architecture* [Preskill 2018]. Current approaches remain insufficient to provide a compact requirement-to-architecture *approach* with explicit *cross-level traceability* under latency, decoherence, and noise [De Stefano et al. 2022]. As a result, design decisions made at the goal level rarely propagate in a verifiable way to variability choices and their architectural realization [Zhao 2020]. Addressing this fragmentation requires a unifying abstraction to capture stakeholder intentions and guide cross-level design.

Furthermore, recent surveys argue that the agent-oriented paradigm remains a fertile abstraction to integrate cognitive, social, and collaborative capabilities in complex software ecosystems and is also used as a starting point in end-to-end development proposals

[Wang et al. 2024, Navarro et al. 2025]. This perspective reinforces our choice of agent-oriented requirements as a natural entry point to link stakeholder intentions and *goals* with downstream design decisions in HQC settings.

In this regard, we argue that HQC systems require a requirement-driven, multiview, and *traceable* design approach that connects *goals, features, and quantum-aware architecture* [Sepúlveda et al. 2024]. Building on this premise, we ask how such an approach can be defined to effectively link *intentional goals*, *variability* decisions, and *architectural* elements, while preserving traceability and enabling impact analysis. As an initial proposal, we outline a modeling approach that addresses this RQ through a lightweight process instantiation and a minimal illustrative case.

To answer the RQ, we present Quantum Variability Management (QVM), a lightweight three-level modeling approach (iStar, UVL, and Quantum-UML). QVM contributes: (i) a conceptual model based on variability that connects goals, features, and architectural elements; (ii) five traceability rules linking goals to features and component stereotypes; (iii) a running example illustrating feasibility through auditable propagation and controlled reconfiguration; and (iv) a positioning discussion of strengths, novelty, prototype scope, and limitations.

Our contribution is methodological. We claim that QVM offers a disciplined way to make explicit the assumptions of the HQC design, to reason about reconfiguration, and to preserve traceability between the goals, variability, and architectural views.

The paper is structured as follows. Section 2 introduces the background; Section 3 presents QVM; Section 4 illustrates it through a running example; Section 5 discusses limitations; and Section 6 presents the conclusions.

2. Background

This section outlines the foundational concepts that underpin QVM: goal modeling and iStar, UVL variability management, and HQC architectures with Quantum–UML.

2.1. Goal Modeling and iStar

Agent orientation is a paradigm in software engineering that emphasizes autonomous agents as the primary entities for modeling and designing software systems. Agents are viewed as self-directed units that perceive, reason, and act upon their environment to achieve goals aligned with their roles or commitments. Over time, agent orientation evolved into a family of modeling frameworks that extend intentional reasoning to system-level analysis, most notably the iStar framework introduced by Yu [Yu 2001].

The iStar provides a concise notation for modeling intentions and dependencies in socio-technical systems [Yu 2001]. It represents how actors (human or organizational) depend on each other to achieve goals, perform tasks, exchange resources, or satisfy quality attributes expressed as softgoals. Two complementary perspectives are defined: the *Strategic Dependency (SD)* view, which captures actors and their interdependencies, and the *Strategic Rationale (SR)* view, which details how each actor operationalizes its internal goals through refinements, tasks, and contribution links. These views allow analysts to reason about autonomy, responsibility, and collaboration among stakeholders.

The updated iStar 2.0 language [Dalpiaz et al. 2016] refines the syntax and semantics of the original framework, clarifying the definitions and relationships of the constructs, while improving the support of the tools and the consistency of the methodological.

Following the classical distinction between the *system as–is* and the *system to–be*, Goal-Oriented Requirements Engineering (GORE) [Van Lamsweerde 2001] describes how goals identified in the current situation can be analyzed and transformed into functional and non-functional requirements that characterize the desired future system. Through this intentional modeling process, the elicited elements can be progressively elaborated into tasks and responsibilities that express and operationalize the envisioned change, while maintaining alignment between organizational objectives and technical design.

In the context of QVM, iStar 2.0 can serve as the intentional backbone for expressing actors such as business owners, operations teams, and quantum service providers, together with their goals (for example, *optimize routing*), softgoals (e.g., *quantum advantage*, *low latency*, *cost awareness*), and delegated responsibilities. Such intentional models may provide a traceable foundation from which variability decisions and hybrid architectural elements can later be derived. This process is particularly useful in HQC contexts, where new capabilities are progressively incorporated while ensuring that their introduction remains consistent with higher-level goals. Figure 1 summarizes this transition from the *as-is* to the *to-be* system (new actors, resources, and softgoals can be incorporated to represent emerging quantum–classical capabilities). This illustrates how GORE modeling could support the identification of requirements and could also serve as an entry point for a model-driven pipeline that leads to variability management and system design.

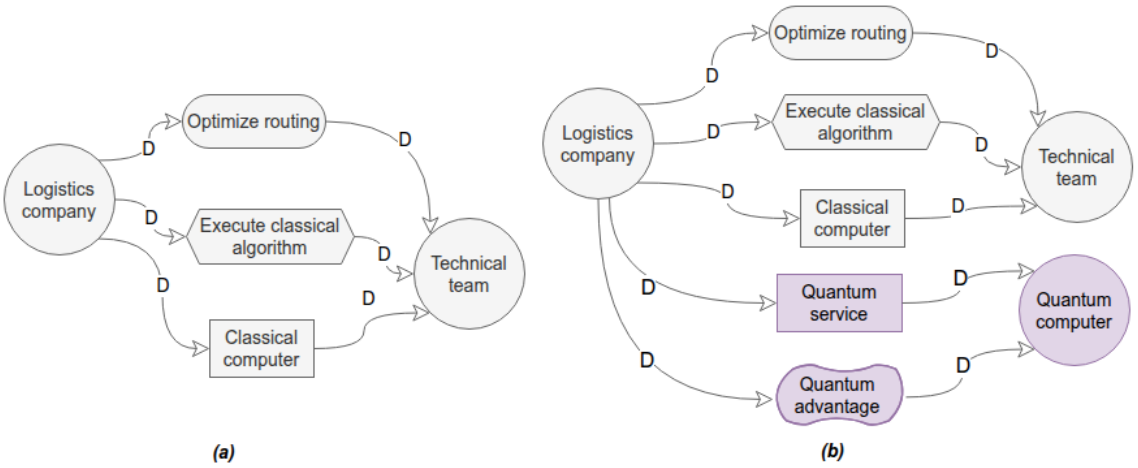


Figure 1. Strategic Dependency (SD) models illustrating the transition from the classical *as-is* scenario (a) to the hybrid *to-be* scenario (b).

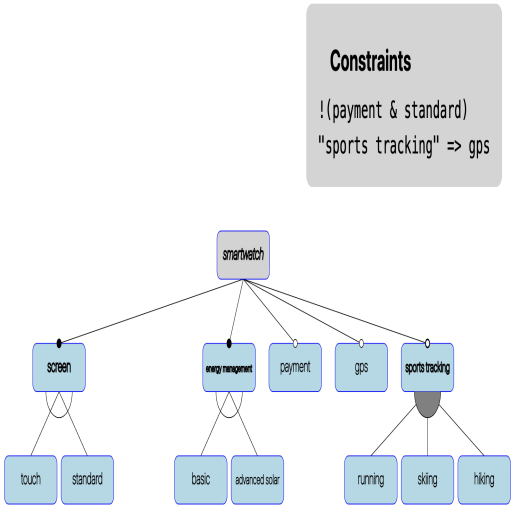
2.2. UVL - Variability Modeling

SPLs manage families of related systems through explicit *variability* modeling: products share a common core and differ by configurable features [Pohl et al. 2005]. Feature models (FMs) capture options and constraints and, in their textual form, are increasingly expressed with the Universal Variability Language (UVL), a pivot notation that eases tool interoperability and human-readable annotations [Benavides et al. 2025].

FMs are the de facto mechanism for capturing variability and are now standard practice in SPLs [Galindo et al. 2015]. FMs are hierarchically structured, with parent–child dependencies and relationship types (mandatory, optional, XOR/OR groups), plus cross-tree constraints (*requires*/*excludes*) encoded as boolean formulas [Lackner and Schmidt 2015].

UVL is a textual language that represents variability, promotes the unification of fragmented SPL notation, and improves scalability, editing, and tool integration [Benavides et al. 2025]. UVL models SPLs as hierarchical trees (root/sub-features) and encodes variability with simple keywords (`mandatory`, `optional`, `alternative`, `or`, `xor`) and propositional constraints [Agarwal et al. 2024]. This language improves scalability and interoperability, automated analyzes (validity check, defect detection), and product derivation in large configuration spaces [Benavides et al. 2025].

Figure 2 shows the FM/UVL model of the *smartwatch SPL*, including its feature hierarchy and cross-tree constraints. The root smartwatch has two mandatory XOR branches: *screen (touch vs. standard)* and *energy management (basic vs. advanced solar)*. Optional features include *payment, gps, and sports tracking*; the latter is an OR-group that requires at least one *running, skiing, or hiking*. Two constraints ensure consistency: `!(payment & standard)` and sports `(tracking => gps)`. Thus, each valid configuration selects exactly one screen type and one energy strategy, while optional capabilities are combined subject to these rules. The full example is available in [Benitez et al. 2025].



(a) Smartwatch FM

```
1  namespace smartwatch
2  features
3      smartwatch {abstract true}
4          mandatory
5              screen
6                  alternative
7                      touch
8                      standard
9              "energy management"
10                 alternative
11                     basic
12                     "advanced solar"
13         optional
14             payment
15             gps
16             "sports tracking"
17                 or
18                     running
19                     skiing
20                     hiking
21 constraints
22     !(payment & standard)
23     "sports tracking" => gps
```

(b) UVL representation

Figure 2. SPL Smartwatch modeling - FM and UVL

In HQC domains, variability spans both algorithmic and platform choices (e.g., QAOA vs. Grover; SpinQ/IBMQ vs. local simulators), which are encoded as mutually exclusive alternatives or dependent configurations in the FM. In the QVM context, feature-level decisions (algorithm/hardware/backend) become first-class and traceable, supporting controlled evolution toward heterogeneous hybrid deployments.

2.3. QSE and Hybrid Architectural Modeling

HQC systems operate in hybrid architectures in which classical components orchestrate business logic and data management, while quantum components execute domain-specific

kernels on remote QPUs [Leymann and Barzen 2021]. Quantum Software Engineering (QSE) adapts processes and notation to this setting [Piattini et al. 2021]. To model such architectures, Quantum-UML introduces stereotypes for qubits, gates, and quantum services within class, component, and sequence diagrams [Pérez-Castillo and Piattini 2022], allowing classical and quantum elements to coexist in a single model.

HQC applications comprise classical programs, workflows, and data processing combined with quantum circuits [Weder et al. 2022]. Unified stacks are needed for interpreted and compiled flows, abstracting late-binding steps (e.g., transpilation under changing hardware calibration) while exposing configurable controls [Shehata et al. 2025]. Despite process adaptations (e.g., activity diagrams for quantum parallelism; tailored RE for quantum-specific concerns [Khan et al. 2022]), many hybrids remain framework-coupled, and task identification often depends on expert intuition [Maio et al. 2025].

HQC architectures take advantage of quantum accelerators, where they add value while maintaining mature classical tooling for data handling, control, and persistence [Sabzevari et al. 2024]. The modular split clarifies responsibility boundaries (classical vs. quantum) and allows pipeline optimizations (e.g., batching, pre/post-processing) that mitigate NISQ limitations [Khan et al. 2022]. In addition, provider-agnostic orchestration and topology/lifecycle, where TOSCA and its runtimes make deployments declarative and portable across providers [Binz et al. 2014].

Quantum-UML profile extends UML with *quantum-aware* stereotypes (e.g., «qubit», «qgate», «qservice») in class, component, and sequence diagrams to co-model classical and quantum concerns [Pérez-Castillo and Piattini 2022]. This profile captures quantum resources, operations, and service boundaries alongside classical controllers, facilitating allocation, interface definition, and the mapping from abstract operations to backend-specific realizations. In HQC settings, this UML profile provides an architectural meeting point where requirements and variability decisions are consolidated under latency, noise, and orchestration constraints.

Quantum-UML offers a shared architectural view to: (i) specify quantum services and interfaces next to classical components; (ii) reason about deployment options and coupling (loose/co-located/on-node); and (iii) connect upstream intent/variability decisions to concrete design elements, improving communication and traceability in multi-disciplinary teams [Piattini et al. 2021, Pérez-Castillo and Piattini 2022, Rallis et al. 2025].

2.4. Challenges and Research Trends

End-to-end traceability—from feature variability to SPL architecture under component addition/removal—remains difficult between phases and abstraction levels [Yousaf et al. 2019]. The definition of FM is socio-technical and demands coordinated, multidisciplinary input [Gómez et al. 2024]. Scalability and constraint management persist in highly variable or IoT-scale settings, which require reliable automated analysis for valid configurations and product counts [Yousaf et al. 2019]. The lack of standardization across notation and tools hampers longevity and interoperability [Agarwal et al. 2024], while classical FMs give limited guidance on concrete variability mechanisms [Kim 2024]. Toolchain fragmentation is a major obstacle in HQC systems. Therefore, QVM offers a unified, hardware-agnostic, and modular approach to improve portability and traceability.

QVM is guided by the vision of connecting *goals, variability, and architecture*

through transformation rules and formal traceability. Although these three pillars (*goals, variability, and architecture*) have been treated in isolation, a coherent, traceable framework that integrates them is still missing. We address this gap with a methodological orchestration that links decisions from requirements to the final HQC architecture.

Despite growing activity around HQC systems, existing efforts typically cover *one* or *two* viewpoints—but not the combined triad of (i) goal-oriented requirements, (ii) variability modeling, and (iii) architecture, with *explicit end-to-end traceability* across all three. To the best of our knowledge, existing work does not provide an explicit and formal end-to-end traceability mechanism that *jointly* connects goals, variability, and quantum-aware architecture.

3. QVM Approach: Overview and Principles

This section presents the QVM motivation, introduces each level and its key elements, and then details the integrated structural view together with its traceability rules and artifacts.

3.1. Motivation

QVM is structured into three complementary levels that allow one to model an HQC system from its inception to its structural design. Each level addresses a different kind of characterization: (i) the intentional level, focused on *goals* and *dependencies* (using iStar); (ii) the variability level, focused on alternative system configurations (using UVL); and (iii) the structural level, where the components of the HQC system are formalized (using Quantum-UML). This stratification enables an integrated treatment of the challenges of synchronization, adaptability, and traceability in the design of HQC software, aligning established software engineering practices with the particularities of the quantum paradigm. The three levels and their relationships, and traceability rules are shown in Figure 3.

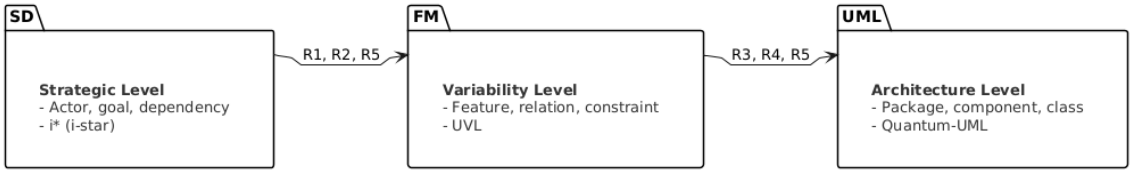


Figure 3. QVM three levels (iStar, UVL, Quantum-UML) and Rules (R1-R5)

3.2. From Goals to Architecture: QVM's Three Levels

Each of the three levels of QVM addresses a complementary facet of the hybrid system (spanning intent, configuration, and structural design) while employing specialized languages tailored to each purpose. Next, we detail each one of the three levels of QVM, according to the pattern: description, used language/model, purpose, justification, key elements, quantum elements previously identified, and the role in the proposal.

3.2.1. Level 1: Intentional - iStar

QVM makes explicit the *goals, actors, and dependencies* that govern the interaction between quantum and classical components, providing a foundation to analyze functional and non-functional requirements under uncertainty and weak coupling.

In QVM, the language/model is *iStar*. Its purpose is to capture goals, dependencies, and *softgoals* that guide the entire process. This choice is justified because it enables reasoning about actors and quality in early phases [Yu 2001]. Key elements include *actor, goal, task, resource, and softgoal*. Quantum-specific concerns are marked with the «quantum» stereotype on tasks or softgoals involving QPUs, fidelity, and related properties. Within the framework, iStar serves as the source of truth from which variability is derived and alignment with business objectives is assessed.

3.2.2. Level 2: Variability - UVL

QVM manages alternative system configurations (e.g., algorithms and platforms) by modeling design decisions that shape system behavior and context adaptation.

In QVM, the language/model is UVL. Its purpose is to provide a textual, tool-friendly representation of HQC options. This choice is justified because UVL is an open standard that supports `constraints` and properties [Benavides et al. 2025]. Key elements include *Feature*, *Group* (AND/OR/XOR), *Constraint*, and *Property*. Quantum-specific aspects are captured via the attribute `isQuantum=true`, the `@quantum` tag, and hardware properties such as `qubits` and `fidelity`. Within the framework, UVL encodes variability and enables the automated derivation of HQC products.

3.2.3. Level 3: Architecture - Quantum-UML

QVM formalizes the structure of the system using packages, components, interfaces, classes, and sequence views, incorporating quantum-specific elements.

In QVM, we use Quantum–UML, a UML profile that has the purpose of realizing the hybrid topology(components, interfaces, and sequences) at the architectural level. This choice is justified because it reuses UML while introducing quantum-aware stereotypes such as «`QAlgorithm`» and «`QService`» [Pérez-Castillo and Piattini 2022]. Key elements include Component, Interface, Class, and Sequence. Quantum aspects are captured through «`Q*`» stereotypes and properties (e.g., `provider=QCA`). Within the framework, Quantum–UML materializes the UVL configuration into implementation-ready artifacts and serves as a basis for code generation.

3.3. Conceptual Model - QVM's Three Levels

Figure 4 introduces the conceptual model of QVM, which integrates the intentional, variability, and architectural layers through a minimal set of classes—Goal, Feature, and Component—and a small number of typed associations. In this model, Goal captures stakeholder intent (from iStar), Feature encodes variability (from UVL), and Component represents architectural building blocks (from Quantum–UML). Associations such as «defines» (target →Feature), «realizes» (Feature→Component), and «tracesTo» (between artifacts) establish an explicit backbone for end-to-end traceability. Properties and tagged values, including «Quantum stereotypes» are attached to features and components to represent configuration parameters (e.g., provider, qubits, fidelity) and to support analysis.

This conceptual model is intentionally minimal, yet extensible. It suffices to express the five traceability rules (R1–R5), to propagate what-if changes across layers, and to

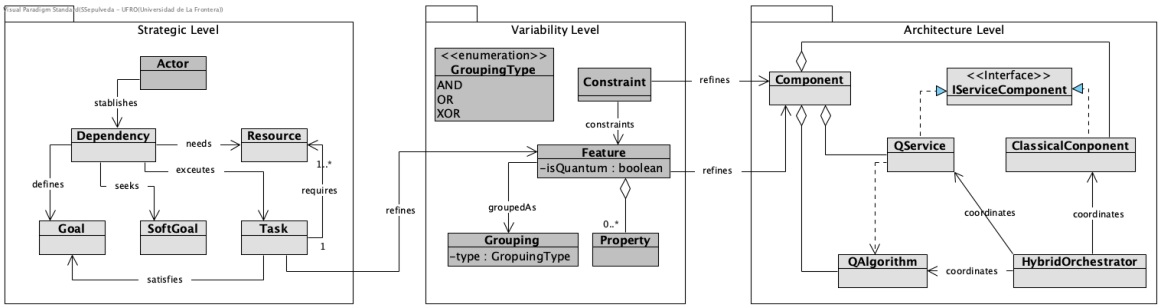


Figure 4. QVM conceptual model with cross-level relations

.

enable automated consistency checks (e.g., verifying that real-world components conform to selected features and that feature selections are justified by goals). This abstraction also acts as the pivot for transformations (iStar SD/SR → UVL → Quantum–UML) and for operational tasks such as product derivation, impact analysis, and code-generation.

3.4. Traceability rules (R1-R5)

QVM establishes end-to-end traceability between levels to preserve semantic coherence and progressive alignment between system goals, variability decisions, and technical implementation in HQC systems.

We define formal mapping rules that link elements across levels, enabling analysts to follow the impact of a strategic goal down to concrete architectural structures and, conversely, to propagate changes upward. As summarized in Table 1, the rules R1-R5 provide cross-level links that support impact analysis and controlled evolution. In particular, R4 carries feature properties (e.g., provider, qubits, fidelity) into class attributes and stereotypes to preserve configuration consistency. Concretely, analysts can run what-if analyzes (e.g., swap *QAOA*→*Grover* or *Simulator*→*Hardware*) and obtain consistent architectural deltas through the rule chain (R1-R5), while R5 is close to the original goal. The rule definitions are provider- and algorithm-agnostic, easing portability across back-ends (e.g., IBM, IonQ) and hybrid patterns (e.g., VQE, QAOA).

Table 1. Traceability rules (R1-R5)

.

Rule	From → To	Purpose
R1	*Goals/Softgoals* → Feature	To preserve intent (e.g., quantum advantage, low latency) as selectable features.
R2	*Task/Dependency* → Constraint	To encode requires/excludes and XOR/OR to bound realizations.
R3	*Feature* → Component/Class	To map choices (algorithm, provider) to concrete design elements (stereotypes).
R4	*Feature Property* → Class Attribute	To carry isQuantum, provider, qubits/fidelity into design.
R5	*Goal* → Component (*conformance link*)	To close the loop for impact analysis and controlled reconfiguration.

3.5. Lightweight process instantiation

Next, we outline a lightweight, repeatable instantiation of QVM that moves from intent to implementation with minimal overhead. The workflow operationalizes the conceptual model and the traceability rules (R1–R5), ensuring that stakeholder goals drive variability decisions and architectural realizations. The result is a pragmatic, end-to-end path that preserves alignment, supports what-if analysis, and enables controlled reconfiguration as platforms and algorithms evolve.

In practice, QVM is enacted as follows: (i) *scope* of the domain and stakeholders; (ii) model *iStar SD to-be goals* and *softgoals*; (iii) *derive UVL* features, groups (XOR/OR), and cross-tree constraints (R1-R2); (iv) *map to* Quantum-UML components and stereotypes (R3-R4); (v) *validate traceability* back to *Goals*, performing a "what-if" analysis (R5); and (vi) *reconfigure* products as platform/algorithm choices evolve.

QVM outputs a coherent set of artifacts across the three levels: (i) *iStar* model captures stakeholder intent and trade-offs; (ii) an UVL model that encodes HQC variability; (iii) Quantum–UML diagrams that realize selected features; (iv) a traceability map (target →feature→component) that supports impact analysis (R1–R5); and (v) product configurations derived from UVL-to-UML mapping. See a summary in Table 2.

Table 2. Summary of generated artifacts and applied rules

Artifact	Purpose and key contents	Rules
iStar Models	Capture stakeholder intent and quality trade-offs; actors, goals, softgoals, tasks, and dependencies.	R1
UVL FM	Encode HQC variability; features, groups, constraints, and properties (`provider`, `qubits`).	R1-R2
Quantum-UML	Architecturally realize selected features; Quantum-UML diagrams and implementation-level.	R3-R4
Traceability Map	Ensure end-to-end alignment; links goal→feature→component with rationale.	R1-R5
Configurations	Instantiate products; support code generation; valid configurations, UVL→UML mappings.	R4-R5

QVM provides a methodological path that begins with hybrid requirements, manages configuration variability, and culminates in a structured design that facilitates implementation and adaptation across multiple platforms. Next, we illustrate the QVM approach, demonstrating the application of the models and rules described above.

4. Running Example - QVM in Logistics Domain

To illustrate QVM in practice, we present a running example of an HQC system. This example traverses the three levels (intentional, configuration, and architectural) and shows how rules R1-R5 preserve the coherence between the objectives of the system and the design/implementation decisions.

We use a simple but representative HQC scenario. The goal is to demonstrate the end-to-end traceability of QVM (iStar → UVL → Quantum-UML). Concretely, the example shows how to (i) encode goals/NFRs as operable constraints; (ii) model typed variability with cross-constraints; (iii) project decisions into component stereotypes; and (iv) respond to changes in goals with controlled, measurable, and explainable impact.

4.1. Domain description

A global logistics company operates multiple regional hubs and local depots. Daily operations include receiving purchase orders, allocating inventory, planning multi-modal shipments (air, sea, rail, truck), and coordinating last-mile delivery. The current platform is classical and distributed (microservices), with batch planning for long-haul routes and near-real-time tracking for last-mile events. The company currently plans routes using classical heuristics executed by its own technical team. Seasonal peaks and disruptions (weather, strikes) stress the routing optimizer and the demand forecaster. To improve performance, it evaluates delegating route optimization to a quantum provider (QaaS) [Leymann and Barzen 2021]. The final solution is an HQC system: classical services handle data ingestion, storage, tracking, and orchestration; quantum services support combinatorial optimization (routes) and estimation (demand).

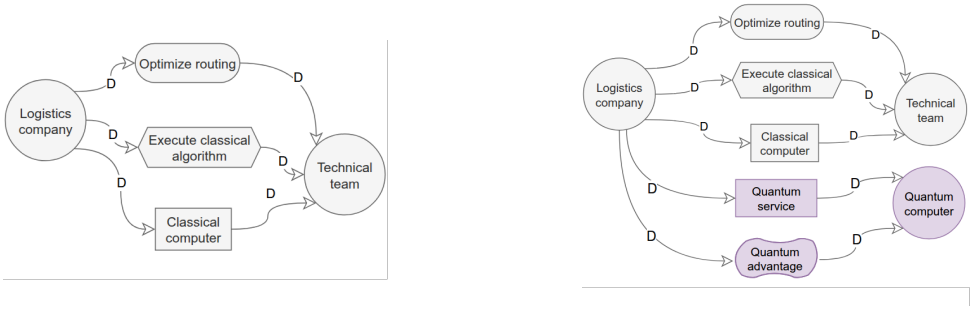
4.2. Level 1: Intentional

We can state four requirements: **R1** Storage & Inventory; **R2** Route Optimization; **R3** Last-mile Tracking; and **R4** Demand Forecasting.

In the *iStar SD to-be* model, **R2** and **R4** introduce quantum services: (i) adiabatic annealing for routing; (ii) gate-based QAOA for demand, both orchestrated by classical microservices. *Softgoals* can include *cost reduction*, *on time delivery*, and *scalability*, constrained by *latency* and *availability*.

The logistics company seeks the quantum advantage for complex routing/optimization tasks. In the *iStar SD as-is* model, all four requirements are realized by classical software. Limitations appear in **R2/R4** (solution quality vs. latency budget). The *goal* evolves to 'Quantum route optimization'. The *softgoal* 'Expected quantum advantage' is introduced, expressing the intention to achieve specific quantum computational benefits.

Figure 5a shows the iStar SD diagram, where the Company depends on the Technical Team for the Task "Execute classical algorithm" to satisfy the *goal* "Optimize routing". Figure 5b introduces the QC agent; the softgoal "Quantum advantage" is added. This *softgoal* reflects the intention to obtain computational benefits by using QC technology ("Quantum computer" and "Quantum service") in complex logistic tasks.



(a) QVM—SD model system as-is **(b) QVM—SD model system to-be**

Figure 5. QVM-SD models

4.3. Level 2: Variability

Applying rules R1–R3, we derive the UVL variability model: the main goal becomes the root feature Optimization, while classical and quantum tasks map (R2–R3) to Classical and Quantum sub-features in an OR-group, enabling hybrid configurations. The *softgoal* "Quantum advantage" further induces the constraint `requires(Optimization, Quantum)` (R2). A summarized UVL model is shown in Listing 1.

Listing 1. UVL model for the logistics domain

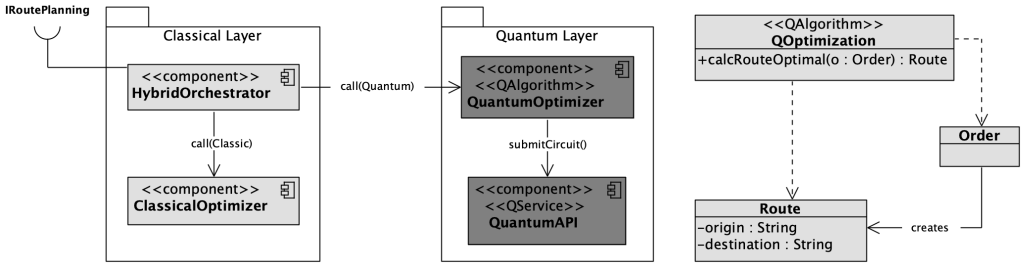
```
namespace Optimization
features
  Optimization {abstract true}
    mandatory
      Hybrid
    optional
      QuantumAdvantage
  Hybrid
    alternative
      Classical
      Quantum
```

```
  Quantum
    attributes
      isQuantum = true
      provider  = "QCA"
      qubits    = 20
constraints
  QuantumAdvantage => Quantum
```

4.4. Level 3: Architectural

Selecting the `Quantum` feature triggers (R4) the components `QuantumOptimizer` «`QAlgorithm`» and `QuantumAPI` «`QService`», while `Classic` activates `ClassicOptimizer`. Map *QAOA/Grover* to «`QAlgorithm`» with `isQuantum=true`; map *Simulator/Hardware* to «`QService`»/`provider`; orchestrate via `HybridOrchestrator`.

Figures 6a and 6b show the component diagram and class diagram, respectively. These diagrams are intended to illustrate the structure of the solution rather than a fully detailed design. Figure 7 complements it with the runtime interaction.



(a) Hybrid component diagram **(b) Hybrid class diagram**

Figure 6. Extracts from a hybrid architectural view for the HQC Logistics domain

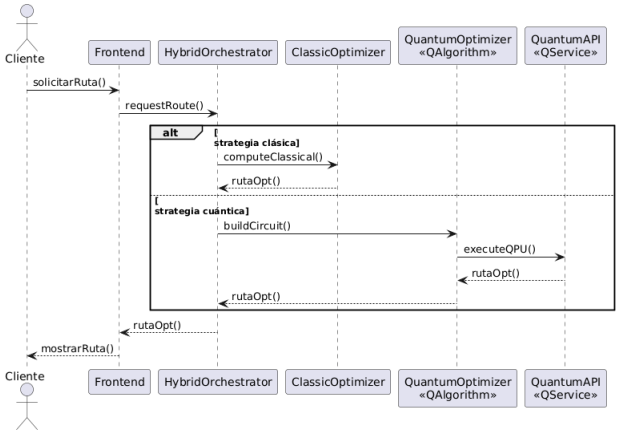


Figure 7. HQC summary for runtime interaction

4.5. Traceability check

QVM traced the evolution from a strategic viewpoint (such as pursuing quantum advantage) to the activation of concrete HQC components. This approach ensures multi-level coherence and enables controlled reconfiguration of the logistics product line. Then, *R5* provides the conformance link from the selected components back to the goals.

The example shows how a strategic goal drives UVL decisions and is realized by a Quantum-UML design. Rules R1-R5 make the path auditable and reproducible. Next, Section 5 examines the strengths and limitations identified during the application of QVM.

5. Discussion

This section reflects on the findings and outlines the strengths and limitations of the QVM proposal. The conceptual results and the running example indicate that it *is feasible* to model requirements in a structured and traceable manner in HQC systems, preserving vertical alignment and enabling controlled reconfiguration. The evidence reported details how QVM reduces cross-view inconsistencies and facilitates *what if* reasoning.

5.1. Strengths

QVM fulfills the objective of the RQ by providing vertical traceability that links *goals, variability, and architecture*, something approaches such as QaaS [Leymann and Barzen 2021] and Q-SDLC [Weder et al. 2022] do not formalize. QVM offers an integrated view from requirements to implementation, enabling impact analysis and supporting controlled system evolution. In this sense, QVM *complements* Quantum-UML by providing requirement-to-variability links (iStar $\rightarrow$ UVL) that are typically left implicit in QSE proposals.

The set iStar /UVL / Quantum-UML allows business stakeholders to reason at their own abstraction level, while variability and architecture engineers work on technical artifacts. Thus, QVM is complementary to lifecycle proposals, focusing specifically on requirement-to-variability-to-architecture traceability.

The role of variability is central. In HQC settings, algorithm families, providers, simulators, QPUs, orchestration modes, and fallback strategies are all design choices with business and technical consequences. Treating these choices as first-class variability elements allows engineers to reason about reconfiguration in a disciplined way. Without this level, it becomes more difficult for the architecture to evolve systematically when hardware conditions or quality priorities change.

5.2. Limitations

We deliberately omitted the iStar Strategic Rationale (SR) view to keep the proposal lightweight and focused on cross-level traceability. However, this view naturally aligns with the current SD-based derivation. Refinement links, contributions, and internal resources can be mapped to UVL capabilities. Incorporating SR in future work would unlock the expressive power of iStar, without altering the core derivation logic already established.

The agent-oriented level was modeled using the standard iStar 2.0 without any domain-specific extensions, ensuring conceptual compatibility and semantic interoperability with existing agent-oriented frameworks. This choice was intentional: domain adaptation to HQC systems was performed at the transformation level rather than through a specialized iStar profile, maintaining the purity of the notation and avoiding premature commitment to any domain-specific semantics.

The use of three modeling languages can increase cognitive load if presented as a monolithic method. QVM addresses this concern by distributing responsibilities across roles and by encouraging incremental adoption. Teams may start with the iStar and UVL

levels plus a compact component view and only then refine the architecture or the optional rationale layer. Templates, rule catalogs, and prototype-supported checks are, therefore, not peripheral aids but adoption mechanisms.

5.3. Threats to validity

We outline the main threats to validity relevant to QVM, and, for each identified limitation, we provide a mitigation strategy.

Integrated tooling. Currently, there is no environment that synchronizes *iStar, UVL, and Quantum–UML. Mitigation*: an incipient prototype validator[1].

Learning curve. Using three modeling languages may increase cognitive load. *Mitigation*: training, pattern catalogs, and automated validators.

Construct validity. Our modeling choices may bias the *goal* $\rightarrow$ *feature* $\rightarrow$ *component* mappings, potentially overfitting the linkage to our preferred languages. *Mitigation*: explicit templates for rules R1–R5 and provide worked examples that make the mapping assumptions transparent, repeatable, and open to scrutiny and adaptation.

External validity. The evaluation focuses on a single domain with a minimal running example, which limits generalizability across industries, quantum providers, and workload profiles. *Mitigation*: multi-domain replications and by introducing provider diversity to stress the approach under heterogeneous conditions.

Conclusion validity. We have not yet conducted a user study or a systematic tool evaluation, which constrains the strength of claims about efficiency gains, error reduction, and usability. *Mitigation*: an upcoming tool-supported pipeline with automated checks to allow controlled experiments and empirical evaluation with practitioners.

6. Conclusion

This paper introduced QVM, a lightweight requirements-driven modeling approach for HQC systems. QVM integrates iStar, UVL and Quantum–UML through five explicit rules (R1–R5) that connect goals, configuration decisions, and architectural elements.

Through the logistics walkthrough and the scenario-based analysis, we showed that QVM can make reconfiguration decisions auditable, expose the role of variability in HQC design, and represent key hardware-related assumptions at the modeling level. At the same time, we claim that QVM has not yet been validated through user studies, multi-domain replications, or a mature integrated environment.

Future work will focus on broader empirical evaluation, richer use of Strategic Rationale links, stronger prototype support, and tighter integration with dynamic hardware information from heterogeneous quantum providers.

Acknowledgments

Samuel Sepúlveda is supported by ANID-Chile Fondecyt Iniciación No. 11240702.

Enrique Moguel is partially funded by the EU through the Interreg Spain-Portugal / POCTEP programme (0289_SER65_PLUS_6_P), and "Next GenerationEU /PRTR", by

[1] `https://github.com/JessusTM/MDD-HQC`

the Ministry of Science, Innovation and Universities (RED2022-134148-T, TED2021-130913B-I00, PDC2022-133465-I00, PID2024-155693NB-C41). Also by the Regional Ministry of Education, Science and Vocational Training of the Regional Government of Extremadura (GR24099).

Thanks to RIPAISC - Red Iberoamericana Para el Avance de la Ing. de Software Cuántico (CyTED 525RT0174).

References

Agarwal, P., Feichtinger, K., Schmid, K., Eichelberger, H., and Rabiser, R. (2024). On the challenges of transforming uvl to ivml. *arXiv preprint arXiv:2401.10911*.

Benavides, D., Sundermann, C., Feichtinger, K., Galindo, J. A., Rabiser, R., and Thüm, T. (2025). Uvl: Feature modelling with the universal variability language. *Journal of systems and software*, 225:112326.

Benitez, F., Galindo, J., Romero, D., and Benavides, D. (2025). Uvl web-based editing and analysis with flamapy.ide. In *Procs. of 19th Int. Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'25)*, page 121–125, NY, USA. ACM.

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2014). Tosca: Portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer, Berlin, Heidelberg.

Dalpiaz, F., Franch, X., and Horkoff, J. (2016). istar 2.0 language guide. *arXiv preprint arXiv:1605.07767*.

De Stefano, M., Pecorelli, F., and Palomba, F. (2022). Software engineering for quantum programming: How far are we? *Journal of Systems and Software*, 190.

Galindo, J., Dhungana, D., Rabiser, R., Benavides, D., Botterweck, G., and Grünbacher, P. (2015). Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology*.

Gómez, J., Ruiz, P. H., Delgado, V. A., and Camacho, M. C. (2024). Collaborative approach for feature models in software product lines. *TecnoLógicas*.

Khan, A. A., Ahmad, A., Waseem, M., Liang, P.-J., Fahmideh, M., Mikkonen, T., and Abrahamsson, P. (2022). Software architecture for quantum computing systems - a systematic review. *Journal of Systems and Software*.

Kim, J. A. (2024). Variability management mechanism for domain engineering and case study in sunroof control domain. *Tehnički glasnik*.

Lackner, H. and Schmidt, M. (2015). Potential errors and test assessment in software product line engineering. *MBT*.

Leymann, F. and Barzen, J. (2021). Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective. *arXiv preprint 2103.04320*.

Maio, V. D., Brandić, I., Deelman, E., and Cito, J. (2025). The road to hybrid quantum programs: Characterizing the evolution from classical to hybrid quantum software. *SIGSOFT FSE Companion*.

Navarro, C., Devia, L., Labra Gayo, J. E., and Cares, C. (2025). An agent-oriented model-driven development process for cyber-physical systems. In *Ibero-American Conference on Software Engineering (CIbSE 2025), Main Track*, pages 150–164. SBC.

Pérez-Castillo, R. and Piattini, M. (2022). Design of classical-quantum systems with uml. *Computing*, 104(11):2375–2403.

Piattini, M.and Serrano, M., Perez-Castillo, R., Petersen, G., and Hevia, J. (2021). Toward a quantum software engineering. *IT Professional*, 23(1):62–66.

Pohl, K., Böckle, G., and van Der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, Berlin, Germany.

Preskill, J. (2018). Quantum computing in the nisq era and beyond. *Quantum*, 2:79.

Rallis, K., Liliopoulos, I., Varsamis, G. D., Tsipas, E., Karafyllidis, I. G., Sirakoulis, G., and Dimitrakis, P. (2025). Interfacing quantum computing systems with high-performance computing systems: An overview.

Sabzevari, M. T., Esposito, M., Taibi, D., and Khan, A. A. (2024). Qcshqd: Quantum computing as a service for hybrid classical-quantum software development: A vision. *QSE-NE@SIGSOFT FSE*.

Sepúlveda, S., Cravero, A., Fonseca, G., and Antonelli, L. (2024). Systematic review on requirements engineering in quantum computing: Insights and future directions. *Electronics*, 13(15):2989.

Shehata, A., Groszkowski, P., Naughton, T. J., Meena, M. G., Wong, E., Claudino, D., da Silva, R. F., and Beck, T. (2025). Bridging paradigms: Designing for hpc-quantum convergence. *Future generations computer systems*.

Stirbu, V. and Haghparast, M. (2023). Quantum algorithm cards: Streamlining the development of hybrid classical-quantum applications. *International Conference on Product Focused Software Process Improvement*.

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Proceedings fifth ieee international symposium on requirements engineering*, pages 249–262. IEEE.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Weder, B., Barzen, J., Leymann, F., and Vietz, D. (2022). Quantum software development lifecycle. In *Quantum Software Engineering*, pages 61–83. Springer.

Yousaf, N., Akram, M., Bhatti, A., and Zaib, A. (2019). Investigation of tools, techniques and languages for model driven software product lines (spl): A systematic literature review. *Journal of Software Engineering and Applications*.

Yu, E. (2001). Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, 43(2):123–132.

Zhao, J. (2020). Quantum software engineering: Landscapes and horizons. *arXiv preprint 2007.07047*.