CS3307A Design Rationale Document– Deliverable 2

Qlink: An Interactive Mental Model Simulator with Link Prediction

Dhir Pathak - 251236762
dpathak8@uwo.ca

Video Demo: https://youtu.be/uelY68fv51A

Architecture Overview and Design Decisions

Architecture Overview:

1. Domain Layer (**core**/)
    a. **model/:** All the graph entities (MentalModel, Concept, Relationship)
    b. **ai/:** Link prediction algorithms and AI assistant integration
    c. **nlp/:** Natural language processing and command system (to be integrated in the next deliverable)
    d. **persistence/:** JSON serialization and file management/ save and load
    e. **common/:** Shared data structures and utilities
2. Presentation Layer (**ui**/)
    a. **MainWindow**: Primary application interface
    b. **GraphWidget**: Interactive graph visualization
    c. **SuggestionPanel**: Relationship suggestions display
3. Infrastructure
    a. Qt6 framework for GUI
    b. igraph library for graph algorithm implementation
    c. Cohere API integration for concept and relationship explanations

Design Pattern Implementation:

1. Strategy Pattern (Behavioral)
    a. Location: core/ai/ILinkPredictor.h and concrete predictor classes
    b. Rationale: Link prediction in graph theory offers multiple algorithmic approaches, each with different strengths depending on graph characteristics. Rather than hardcoding a single algorithm, I realized runtime flexibility was imperative so I could experiment with different approaches and let users choose optimal strategies for their specific mental models.
    c. The implementation architecture is as follows:
        i. ILinkPredictor defines the strategy contract with predictLinks(), getAlgorithmName(), and getDescription()
        ii. IGraphLinkPredictor provides shared igraph conversion functionality, eliminating code duplication
        iii. Three concrete strategies implement different mathematical approaches:
            1. CommonNeighborPredictor: Counts mutual neighbors (best for dense, clustered graphs)
            2. JaccardCoefficientPredictor: Normalizes by union size (handles varying node degrees)
            3. PreferentialAttachmentPredictor: Favors high-degree connections (this models real world networks)

   d. Design Justification: This pattern is essential for the AI subsystem because graph characteristics vary dramatically between domains. A biology model requires different prediction logic than, lets say a network for research citation. The strategy pattern enables algorithm comparison and selection without architectural changes.

2. Factory Pattern (Creational)
   a. Location: core/ai/LinkPredictorFactory in ILinkPredictor.cpp
   b. Rationale: Creating link predictor instances requires knowledge of algorithm types, parameters, and initialization sequences (knowledge I am still shaky on). Direct instantiation would couple the UI to specific algorithm classes and complicate the addition of new prediction methods, so therefore I prioritized the need for a more centralized and parameterized object creation.
   c. The implementation architecture is as follows:
      i. LinkPredictorFactory::createPredictor() accepts AlgorithmType enum parameter
      ii. Factory encapsulates construction logic and returns std::unique_ptr for automatic memory management
      iii. getAvailableAlgorithms() provides runtime algorithm discovery
      iv. getAlgorithmName() enables UI display without coupling to implementation classes
   d. Design Justification: The Factory pattern is needed for extensibility in the AI system. New algorithms can be added by implementing ILinkPredictor and updating the factory, without modifying existing UI or client code.
3. Observer Pattern (Behavioral)
   a. Location: core/model/MentalModel.h, implemented via Qt signals/slots
   b. Rationale: Makes the mental models collaborative that change frequently as users add concepts, create relationships, and accept AI suggestions. Multiple UI components (graph visualization, suggestion panels, statistics displays) must stay synchronized with model state without tight coupling that would make the system hard to extend.
   c. The implementation architecture is as follows:
      i. MentalModel inherits QObject and emits signals on all state changes
      ii. Signals include conceptAdded, conceptRemoved, relationshipAdded, relationshipRemoved, modelChanged
      iii. ModelChangeEvent provides detailed change context with timestamps and affected entities
      iv. UI components connect via Qt's signal/slot mechanism
   d. Design Justification: Qt's signals/slots implementation provide a very good observer functionality that is better than other custom implementations. This pattern is needed for some collaboration features I hope to further add in the next

deliverable and makes sure the UI stays consistent. By having it be decoupled, we can add new UI components and support future features like change history, collaborative editing, and undo/redo functionality.

4. Command Pattern (Behavioral)
    a. Location: core/nlp/ICommand.h, Commands.h, Commands.cpp
    b. Rationale: Natural language commands will require complex parsing and execution that will really benefit from the encapsulation used here. Future features like macro recording, batch operations, and collaborative editing need operation reification as first class objects.
    c. The implementation architecture is as follows:
        i. ICommand interface defines execute(), undo(), and getDescription() contract
        ii. Concrete commands: AddConceptCommand, RemoveConceptCommand, CreateRelationshipCommand, DeleteRelationshipCommand
        iii. Commands encapsulate operation parameters and maintain undo state
        iv. Framework prepared for CommandHistory and MacroCommand implementations in Deliverable 3
    d. Justification: The Command pattern is needed for the advanced user interactions planned in my roadmap. While basic execute() functionality works now, the pattern's true value is seen with undo/redo, command logging, and NLP integration in Deliverable 3. This early implementation sets the foundation for more sophisticated user operations.

Below are some additional design decisions I made for Deliverable 2:

In my implementation, I ensured to go with a domain first approach so that the main domain logic would be independent, as the core/ directory contains pure logic with zero dependencies on Qt widgets or external services. This allows me to further improve on the ui now that I have a solid core 'backend' making it more maintainable and can let me create unit tests for it without GUI dependencies. I also chose to integrate the igraph library for graph algorithms rather than custom implementations so that I could focus on the design patterns and domain logic rather than mathematical implementation (while still allowing me to pursue the nitty gritty of link prediction algorithms in my common neighbors prediction implementation). Finally, I also used both igraph (link prediction algorithms) and Cohere (NLP for relationship and concept explanations) so the application is equipped with comprehensive AI features. The design philosophy behind this follows modern AI architecture combining both algorithmic AI and generative AI. I used cohere because I had made several projects with it before and had a token I could use already.

As such, my current implementation status is as follows:

1. Working Features Demonstration
    a. Application compiles and runs in Qt Creator with Qt6
    b. Interactive graph visualization with draggable nodes
    c. Mental model loading from JSON files (sample_model.json included)
    d. AI-powered link prediction with confidence scores
    e. Real-time UI updates via signals/slots Observer pattern
    f. Professional igraph algorithm integration (Common Neighbors, Jaccard, Preferential Attachment)
    g. AI assistant with Cohere integration (requires API key configuration)
2. Code Quality
    a. Clean architecture with clear separation of concerns
    b. Modern C++17 features including smart pointers and RAII
    c. Comprehensive header documentation with implementation notes
    d. Consistent coding style and naming conventions
    e. Memory safety through RAII and smart pointer usage
    f. Error handling with custom exception classes
3. Build System Robustness
    a. Created a custom CMake with automatic source file discovery so it can work in different systems,
        i. with cross platform library detection (pkg-config + manual fallback)
    b. Proper dependency management and linking
    c. Clean output directory structure
    d. Environment variable support for configuration

Changes from Deliverable 1:

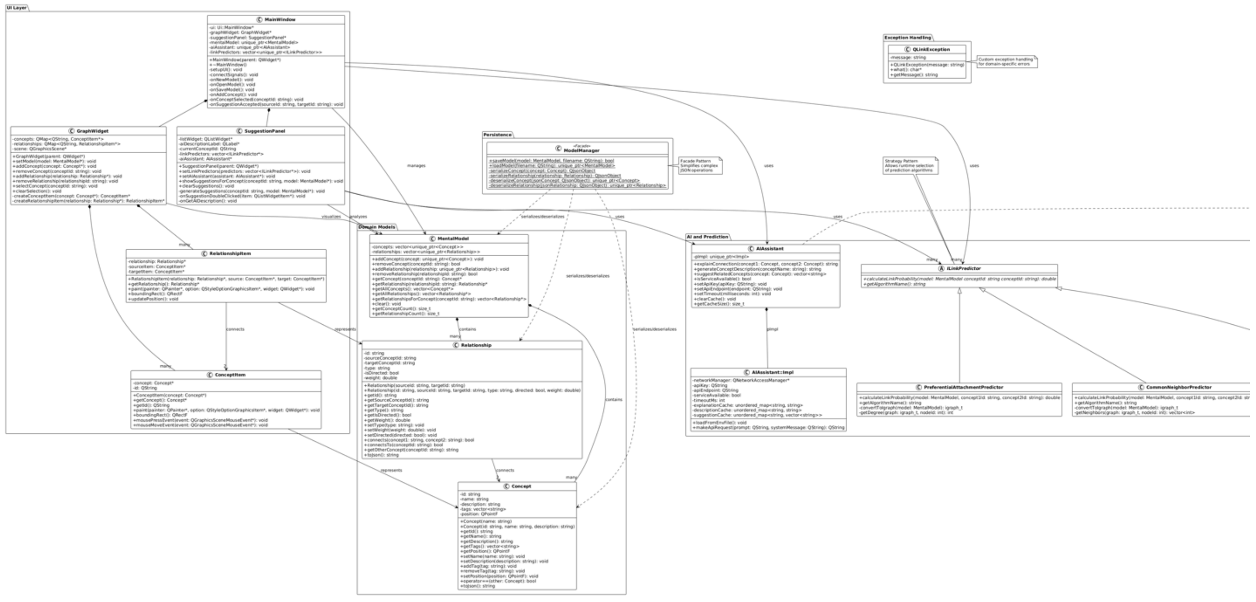
1. Architecture Improvements:
    a. Original: Initially expected to implement a more complex multi layer architecture that had a separate command processing service
    b. Revised: But instead, I streamlined architecture by focusing on the core graph functionality with integrated AI instead
    c. Rationale: The original design was too over engineered for the scope. The revised architecture maintains clean separation while being more implementation focused and achievable within project constraints.
2. Link Prediction Algorithm Integration Update:
    a. Original: Custom link prediction algorithms with basic statistical methods.
    b. Revised: Professional igraph library integration with multiple proven algorithms
    c. Rationale: igraph provides battle tested algorithms used in academic research. This upgrade delivers better prediction accuracy and performance compared to custom implementations.
3. Scope Simplification
    a. Original: I initially had a very ambitious NLP processing with complex command interpretation
    b. Revised: Instead, I am now more focused on core graph functionality with NLP as secondary feature
    c. Rationale: By prioritizing on the working core functionality over ambitious but complex features, I reckoned I could deliver a more solid foundation that can be extended incrementally.

Deliverable 3 Plan:

In the next deliverable, I hope to:

1. Complete the tests for all components (both core and ui).
    a. I currently only have some test skeletons that have not been implemented yet.
2. Add more sophisticated NLP to parse user text and create mental models dependant on their requirements.
    a. This means I need to fully add in CommandFactory.cpp and the factory class for creating commands from natural language CommandFactory.h
    b. And allow for more sophisticated commands (listed above).
3. Update the UI so that it:
    a. Is more modern with more appealing panels
    b. Add the panel for natural language text
    c. Fix the concept bouncer that sometimes constantly moves the nodes of the graph
    d. Fix both spotted bugs in the suggestions panel:
        i. Use Concept name instead of Concept id in generated relationship suggestions
        ii. Use proper algorithm name instead of relationship type in suggestions panel
4. (If I have time):
    a. Look into some more sophisticated algorithms, purely for explorative purposes.

Class Diagram (updated from deliverable 1):


UI Layer
MainWindow
GraphWidget
SuggestionPanel
RelationshipItem
ConceptItem
Persistence
«Facade»
ModelManager
Facade Pattern
Simplifies complex
JSON operations
Domain Models
MentalModel
Relationship
Concept
AI and Prediction
AIAssistant
AIAssistant::Impl
ILinkPredictor
PreferentialAttachmentPredictor
CommonNeighborPredictor
Strategy Pattern
Allows runtime selection
of prediction algorithms
Exception Handling
QLinkException
Custom exception handling
for domain-specific errors
manages
uses
uses
visualizes
analyzes
uses
uses
serializes/deserializes
serializes/deserializes
serializes/deserializes
many
connects
represents
contains
many
contains
connects
represents
many
pimpl
many