ORES
Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.415.2

АРХИТЕКТУРА МНОГОПОТОЧНОГО ПРИЛОЖЕНИЯ УПРАВЛЕНИЯ ПРОМЫШЛЕННЫМ ОБОРУДОВАНИЕМ НА C++/QT

Фролов Денис Александрович,
Национальный исследовательский университет «МИЭТ», Москва, Зеленоград
e-mail: den.frolov2004@mail.ru

Аннотация

Рассматривается архитектура многопоточного приложения управления промышленным оборудованием, реализуемого на C++ и Qt. Предложено функциональное разделение графического интерфейса, сетевого обмена с программируемым логическим контроллером, получения кадров от камеры и вычислительной обработки по отдельным потокам выполнения. Описано применение worker-объектов QObject с переносом в QThread, передача данных через очередные сигнально-слотовые соединения, организация владения ресурсами и обработка ошибок. Сформирован порядок корректного запуска и завершения рабочих потоков, исключающий обращение к удалённым объектам и принудительную остановку потоков. Проведено качественное сравнение вариантов параллельного выполнения и сформирован набор сценариев функциональной проверки. Предлагаемый подход направлен на сохранение отзывчивости интерфейса, локализацию отказов подсистем и повышение сопровождаемости программного обеспечения.

Ключевые слова: C++, Qt, многопоточность, QThread, QObject, worker-объект, промышленное оборудование, архитектура программного обеспечения, потоковая принадлежность.

ARCHITECTURE OF A MULTITHREADED C++/QT APPLICATION FOR INDUSTRIAL EQUIPMENT CONTROL

Denis A. Frolov,
National Research University of Electronic Technology (MIET), Moscow, Zelenograd, Russian Federation

ABSTRACT

The architecture of a multithreaded industrial equipment control application implemented in C++ and Qt is considered. A functional separation of the graphical user interface, network communication with a programmable logic controller, camera frame acquisition, and computational processing into dedicated execution threads is proposed. The use of QObject worker objects moved to QThread, data transfer through queued signal-slot connections, resource ownership, and error handling are described. A controlled startup and shutdown sequence is defined to prevent access to deleted objects and avoid forced thread termination. Implementation

approaches are qualitatively compared, and a set of functional verification scenarios is formulated. The proposed approach is intended to preserve interface responsiveness, localize subsystem failures, and improve software maintainability.

Keywords: C++, Qt, multithreading, QThread, QObject, worker object, industrial equipment, software architecture, thread affinity.

Введение

Приложение управления промышленным оборудованием одновременно обрабатывает действия оператора, сетевой обмен с контроллерами, данные камер, события приводов и вычислительные задачи. Выполнение всех операций в главном потоке приводит к задержкам обработки интерфейса. Даже кратковременный блокирующий вызов может создать впечатление отказа системы и затруднить реакцию оператора на изменение технологического состояния.

Qt предоставляет платформенно-независимые средства многопоточности и передачи событий между потоками [1]. Объекты QObject обладают потоковой принадлежностью: события и очередные вызовы обрабатываются в потоке, которому принадлежит объект [2]. Нарушение этого правила при работе с сетевыми клиентами, таймерами, камерами и моделями данных способно приводить к гонкам, зависаниям и ошибкам освобождения ресурсов.

Описание архитектуры программной системы должно связывать её компоненты, интерфейсы и значимые свойства [7]. Для рассматриваемого приложения к таким свойствам относятся временная эффективность, надёжность, сопровождаемость и способность восстанавливаться после отказов подсистем [9]. При этом программный интерфейс оператора не должен подменять функции безопасности, выполняемые контроллером и аппаратными средствами; такое разграничение соответствует общим принципам проектирования связанных с безопасностью частей систем управления [8]. Современные исследования промышленных программных систем подтверждают целесообразность разделения получения, обработки и визуализации данных между самостоятельными функциональными компонентами [10, 11]. Для параллельного опроса оборудования и повышения отзывчивости программного интерфейса применяются многопоточные схемы сбора данных [12].

Цель исследования

Цель исследования состоит в формировании архитектуры многопоточного приложения на C++ и Qt, разделяющей графический интерфейс и взаимодействие с промышленным оборудованием, определяющей правила межпоточной передачи данных и обеспечивающей контролируемый жизненный цикл рабочих потоков.

Материалы и методы исследования

В качестве исходной модели рассматривается настольное приложение, включающее графический интерфейс, клиент сетевого обмена с ПЛК, подсистему получения кадров от промышленной камеры и модуль длительной обработки данных. Метод исследования основан на функциональной декомпозиции подсистем, анализе потоковой принадлежности объектов Qt, сопоставлении вариантов организации параллельного выполнения и сценарной проверке запуска, обмена данными, обработки ошибок и завершения приложения.

Архитектурные решения сопоставлялись с рекомендациями документации Qt по потокам, QObject, QThread, синхронизации и QtConcurrent [1–6], а также с актуальными научными работами по архитектуре систем диагностики промышленного оборудования,

высокоскоростному сбору данных и многопоточному мониторингу производственных машин [10–12]. Количественное измерение производительности в рамках работы не проводилось; результатом является архитектурная схема, правила взаимодействия и набор проверяемых критериев.

Главный поток отвечает за виджеты, модель представления и обработку пользовательских событий. Сетевой обмен с ПЛК выполняется в отдельном потоке с собственным циклом событий. Подсистема камеры использует самостоятельный поток получения кадров. Длительные вычисления или подготовка данных выделяются в вычислительный worker-объект либо выполняются средствами пула потоков. Такое разделение соответствует архитектурному подходу, при котором сбор, обработка и представление промышленных данных реализуются раздельными компонентами [10, 11].

Разделение выполняется по характеру ресурса и требуемому контексту выполнения, а не по числу классов. Объекты, работающие с одним последовательным каналом оборудования, целесообразно размещать в одном потоке, чтобы не вводить дополнительную синхронизацию. Графические объекты и операции изменения виджетов остаются только в главном потоке. Такое представление фиксирует компоненты и их связи в форме, согласующейся с принципами описания программной архитектуры [7].

Основная схема реализации состоит в создании QObject-наследника, содержащего операции подсистемы, и переносе этого объекта в QThread методом moveToThread(). После запуска потока слот worker-объекта выполняется в его потоке, если вызов поступил через очередное соединение. Сам объект QThread принадлежит потоку, в котором был создан, поэтому размещение рабочих слотов непосредственно в наследнике QThread требует отдельного контроля контекста выполнения [3].

Worker-объект создаётся без родителя, переносится в рабочий поток и связывается с сигналом завершения для удаления методом deleteLater(). Контроллер верхнего уровня передаёт команды сигналами и получает результаты также сигналами. При этом перенос QObject с родителем невозможен, а дочерние объекты должны создаваться в том же потоке, в котором будет выполняться worker; эти ограничения следуют из правил thread affinity [4].

Для небольших неизменяемых структур применяется копирование через очередные сигналы. Крупные кадры или массивы данных требуют явного владения: объект передаётся через умный указатель, неизменяемый снимок либо ограниченную очередь. Передача ссылки на изменяемый контейнер без синхронизации недопустима.

Состояние оборудования хранится в модели верхнего уровня и обновляется атомарными сообщениями. Рабочий поток не обращается к виджетам, а интерфейс не вызывает напрямую методы сетевого сокета или камеры, принадлежащих другому потоку. При интенсивном потоке событий обновления агрегируются: интерфейсу передаётся последний снимок параметров с ограниченной частотой, а не отдельный сигнал для каждого регистра или кадра. Аналогичные многопоточные схемы параллельного опроса каналов и передачи результатов в программный интерфейс применяются в системах мониторинга производственного оборудования [12].

Мьютексы применяются только к данным, которые действительно изменяются несколькими потоками. Предпочтительно организовать владение так, чтобы объект изменялся одним потоком, а остальные потребители получали копии. Длительное удержание блокировки в главном потоке недопустимо. Использование механизмов синхронизации должно учитывать повторную входимость классов и область защищаемых данных [5].

Ошибка оборудования преобразуется в сообщение подсистемы и передаётся контроллеру приложения. Рабочий поток может инициировать восстановление соединения, однако решение о повторном запуске технологической операции принимается

на уровне логики процесса. Коды ошибок и диагностическая информация не должны теряться внутри worker-объекта. Функции аварийной остановки и аппаратные блокировки остаются в ПЛК и специализированных цепях безопасности [8].

Потоки запускаются после создания всех сигнально-слотовых соединений. При завершении приложение сначала запрещает новые команды, затем направляет рабочим объектам запрос остановки, прерывает блокирующие операции, вызывает quit() для циклов событий и ожидает завершения методом wait(). Принудительный terminate() не используется как штатный способ остановки, поскольку он способен оставить ресурс и внутреннее состояние объекта в неопределённом состоянии [3].

Порядок уничтожения особенно важен для камер, сетевых клиентов и графических ресурсов. Устройство закрывается в потоке, которому принадлежит его объект; затем worker удаляется, завершается QThread и только после этого уничтожается управляющий объект верхнего уровня. Последовательность корректного завершения рабочих потоков изображена на рисунке 1.

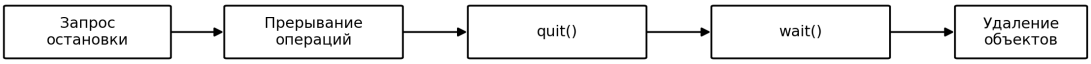

Рисунок 1. Последовательность корректного завершения рабочих потоков

Результатом функциональной декомпозиции является архитектура, в которой интерфейс, последовательные каналы оборудования и вычислительные задачи имеют определённые контексты выполнения. Наиболее подходящим вариантом для длительно живущих подсистем с таймерами, сетевыми сокетами и сигналами является QObject + moveToThread. Наследование от QThread сохраняет смысл для самостоятельной функции, реализуемой в run(), а QtConcurrent и пул потоков удобны для конечных вычислительных задач без постоянной объектной модели [6].

Таблица 1.

Сравнение вариантов организации параллельного выполнения

Подход	Преимущества	Ограничения	Рекомендуемое применение
Один поток	Минимальная сложность реализации	Блокирующий ввод-вывод останавливает GUI	Только быстрые неблокирующие операции
Наследник QThread	Прямой контроль функции run()	Сложнее использовать цикл событий и QObject	Изолированная самостоятельная функция
QObject + moveToThread	Сигналы, таймеры, сокеты, явная потоковая принадлежность	Требует контролируемого жизненного цикла	Длительно живущая подсистема оборудования

Подход	Преимущества	Ограничения	Рекомендуемое применение
QtConcurrent / пул потоков	Удобный запуск конечных задач	Нет постоянной объектной модели подсистемы	Вычисления и пакетная обработка

Источник: составлено автором на основе рекомендаций Qt и результатов исследований [1–6, 11, 12].

Сравнение по таблице 1 показывает, что выбранный worker-подход уменьшает число прямых межпоточных вызовов и позволяет связать жизненный цикл подсистемы с жизненным циклом её объектов. При этом он не устраняет необходимость проектировать ограниченные очереди данных и явную обработку остановки. Для оценки архитектуры сформированы сценарии, проверяющие отзывчивость интерфейса, корректность освобождения ресурсов и отсутствие обращений к объектам после завершения их потоков. Эти критерии связаны с характеристиками временной эффективности, надёжности и сопровождаемости программного продукта [9].

Сценарная проверка охватывает одновременную работу интерфейса и оборудования, задержку ответа ПЛК, интенсивный поток кадров, остановку во время активного обмена, ошибку worker-объекта и повторный запуск подсистемы. Для диагностики потокам присваиваются имена, а в журнал ключевых операций записывается идентификатор текущего потока. Это позволяет обнаруживать прямой вызов метода объекта из чужого потока и некорректную последовательность завершения.

Таблица 2.
Сценарии проверки многопоточной архитектуры

Сценарий	Проверяемый критерий
Задержка сетевого ответа	GUI продолжает обрабатывать пользовательские события
Интенсивный поток кадров	Очередь ограничена, использование памяти не растёт неограниченно
Остановка во время обмена	Ресурсы закрываются в потоках, которым принадлежат их объекты
Ошибка worker-объекта	Диагностическое сообщение передаётся контроллеру приложения
Повторный запуск подсистемы	Не остаются старые соединения, таймеры и потоки

Источник: составлено автором с учётом архитектурных решений и практик проверки, описанных в [10–12].

Заключение

Архитектура многопоточного приложения управления промышленным оборудованием должна строиться на функциональном разделении ресурсов и явной потоковой принадлежности объектов. Worker-объекты, очередные сигнально-слотовые соединения, ограниченные очереди данных и последовательное завершение потоков позволяют сохранить отзывчивость интерфейса и снизить риск ошибок синхронизации.

Разделение подсистем упрощает автономную проверку, замену оборудования имитаторами и локализацию отказов. Дальнейшая оценка архитектуры должна включать количественные нагрузочные испытания на целевой аппаратной конфигурации.

Список литературы:

1. Qt Group. Multi-threading in Qt // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/threads.html (дата обращения: 25.07.2026).
2. Qt Group. Threads and QObjects // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/threads-qobject.html (дата обращения: 25.07.2026).
3. Qt Group. QThread Class // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/qthread.html (дата обращения: 25.07.2026).
4. Qt Group. QObject Class: Thread Affinity // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/qobject.html#thread-affinity (дата обращения: 25.07.2026).
5. Qt Group. Reentrancy and Thread-Safety // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/threads-reentrancy.html (дата обращения: 25.07.2026).
6. Qt Group. Qt Concurrent // Qt 6 Documentation. URL: https://doc.qt.io/qt-6/qtconcurrent-index.html (дата обращения: 25.07.2026).
7. ISO/IEC/IEEE 42010:2022. Software, systems and enterprise — Architecture description. Geneva: International Organization for Standardization, 2022.
8. ISO 13849-1:2023. Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design. Geneva: International Organization for Standardization, 2023.
9. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Geneva: International Organization for Standardization, 2023.
10. Bolanowski M., Paszkiewicz A., Żabiński T., Piecuch G., Salach M., Tomecki K. System Architecture for Diagnostics and Supervision of Industrial Equipment and Processes in an IoE Device Environment // Electronics. 2023. Vol. 12, no. 24. Art. 4935. DOI: 10.3390/electronics12244935.
11. Turkmanović H., Karličić M., Rajović V., Popović I. High Performance Software Architectures for Remote High-Speed Data Acquisition // Electronics. 2023. Vol. 12, no. 20. Art. 4206. DOI: 10.3390/electronics12204206.
12. Wang Y., Cai Z., Huang T., Shi J., Lu F., Xu Z. An Intelligent Manufacturing Management System for Enhancing Production in Small-Scale Industries // Electronics. 2024. Vol. 13, no. 13. Art. 2633. DOI: 10.3390/electronics13132633.

References:

1. Qt Group. Multi-threading in Qt. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/threads.html (accessed 25 July 2026).
2. Qt Group. Threads and QObjects. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/threads-qobject.html (accessed 25 July 2026).
3. Qt Group. QThread Class. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/qthread.html (accessed 25 July 2026).

4. Qt Group. QObject Class: Thread Affinity. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/qobject.html#thread-affinity (accessed 25 July 2026).
5. Qt Group. Reentrancy and Thread-Safety. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/threads-reentrancy.html (accessed 25 July 2026).
6. Qt Group. Qt Concurrent. Qt 6 Documentation. Available at: https://doc.qt.io/qt-6/qtconcurrent-index.html (accessed 25 July 2026).
7. ISO/IEC/IEEE 42010:2022. Software, systems and enterprise — Architecture description. Geneva: International Organization for Standardization, 2022.
8. ISO 13849-1:2023. Safety of machinery — Safety-related parts of control systems — Part 1: General principles for design. Geneva: International Organization for Standardization, 2023.
9. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Geneva: International Organization for Standardization, 2023.
10. Bolanowski M., Paszkiewicz A., Żabiński T., Piecuch G., Salach M., Tomecki K. System Architecture for Diagnostics and Supervision of Industrial Equipment and Processes in an IoE Device Environment. Electronics, 2023, vol. 12, no. 24, article 4935. DOI: 10.3390/electronics12244935.
11. Turkmanović H., Karličić M., Rajović V., Popović I. High Performance Software Architectures for Remote High-Speed Data Acquisition. Electronics, 2023, vol. 12, no. 20, article 4206. DOI: 10.3390/electronics12204206.
12. Wang Y., Cai Z., Huang T., Shi J., Lu F., Xu Z. An Intelligent Manufacturing Management System for Enhancing Production in Small-Scale Industries. Electronics, 2024, vol. 13, no. 13, article 2633. DOI: 10.3390/electronics13132633.