Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Интеграция Qt/QML с OCaml: вызов обработчиков

Михайлов Илья Игоревич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования Д. С. Косарев

Санкт-Петербург
2023

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. Существующие аналоги 5
2.2. JSX 6
2.3. Предыдущая работа 6
2.4. qmlsc 7
2.5. Инструменты 7
2.6. lablqml 8

3. Реализация **9**

3.1. Meta-Object System 9
3.2. Вызов OCaml кода через динамические слоты 10
3.3. Добавление нового синтаксиса 11

4. Тестирование **13**

Заключение **14**

Список литературы **15**

Введение

OCaml – это функциональный язык программирования семейства ML со статической типизацией. С проектированием графического интерфейса пользователя (GUI) в данном языке все не так просто: в opam – менеджере пакетов для OCaml – множество библиотек с базовыми наборами инструментов, но проектирование полноценных динамических GUI-интерфейсов на них затруднительно. Вне opam'а существуют GUI-фреймворки специально для OCaml, биндинги к популярным графическим библиотекам, а также синтаксические расширения языка OCaml, позволяющие проектировать WEB-приложения, но не во всех таких решениях графический интерфейс можно описывать предметно-ориентированным языком разметки.

Удобно проектировать графический интерфейс с помощью встроенной прямо в язык программирования разметки. Примером такой разметки являются HTML-теги в синтаксическом расширении JSX для JavaScript, где с помощью данных тегов можно описывать графический интерфейс прямо в JavaScript.

Фреймворк Qt/QML позволяет проектировать кроссплатформенные приложения, используя язык разметки QML для описания интерфейса пользователя (UI), и имеет богатый набор инструментов для создания приложений любой направленности. QML компилируется[1] в C++, и как следствие, открывается доступ к множеству библиотек на C++.

Вдохновляясь примером JSX предлагается интегрировать в OCaml предметно-ориентированный язык разметки QML из фреймворка Qt/QML.

[1]href=https://www.qt.io/blog/the-new-qtquick-compiler-technology (дата обращения 31.05.2023)
Дата сборки: 1 июня 2023 г.

1. Постановка задачи

Целью работы является улучшение интеграции Qt/QML с OCaml, а именно добавление возможности вызывать OCaml из сгенерированного кода на QML. Для ее выполнения были поставлены следующие задачи:

1. спроектировать интерфейс для взаимодействия между динамическими объектами C++ и QML-фронтэндом;
2. реализовать вызов OCaml из C/C++;
3. добавить поддержку OCaml обработчиков сигнатуры (unit -> unit) в разметку;
4. провести тестирование.

2. Обзор

Целью обзора является поиск и описание существующих библиотек и фреймворков для создания UI и Web-приложений, похожих технологий, расширяющих синтаксис языка для вызова функций, а также инструментов для реализации собственного решения.

2.1. Существующие аналоги

Ниже приведены найденные решения, которые могут описать спектр возможностей для проектирования графических приложений:

- LablGTK3[2];
- lablqml[3];
- Bogue[4].

LablGTK3 – решение, представляющее собой интерфейс для взаимодействия между OCaml и GTK (GIMP tool kit) – кроссплатформенного фреймворка для создания GUI на основе виджетов. Является популярным инструментом для проектирования GUI, в силу общей известности библиотеки GTK. Нет встроенного языка разметки.

lablqml – представляет собой интеграцию QtQuick с OCaml. Главной особенностью является возможность описать обработчики сигналов для контролов (объектов QML) на языке OCaml, тогда как описание GUI происходит обособленно от среды OCaml на языке QML.

Bogue – библиотека для создания GUI виджетов, написанная с нуля на основе SDL2, главной особенностью которой является легковесность, отсутствие интеграций в другие библиотеки для UI, ускорение работы графического процессора (GPU acceleration), благодаря SDL2. В данном проекте не используется язык разметки графичесеского интерфейса.

[2]https://garrigue.github.io/lablgtk/ (дата обращения 31.05.2023)
[3]http://kakadu.github.io/lablqml/ (дата обращения 31.05.2023)
[4]http://sanette.github.io/bogue/Principles.html (дата обращения 31.05.2023)

Также был осуществлен поиск решений, которые являются синтаксическими расширениями для OCaml и позволяют проектировать Web-приложения:

- ReasonML[5] и Melange[6];
- ReScript[7];
- ReveryUI[8] и Brisk[9].

ReasonML является языком программирования, который представляет собой синтаксическое расширение языка OCaml и напоминает C и JavaScript. Благодаря интеграции с JS (компилятор js_of_ocaml) и встроенной поддержкой JSX, становятся доступны многие популярные библиотеки для JS (например, ReactJS). ReScript также является расширением синтаксиса языка OCaml, отличающимся от ReasonML, и собственным компилятором в JavaScript.

2.2. JSX

JSX – синтаксическое расширение для языка JavaScript, позволяющее описывать UI непосредственно в коде JavaScript с помощью HTML-тегов, которые раскрываются в компоненты библиотеки ReactJS. Преобразование кода выполняет компилятор Babel.js. Предлагается использовать данную идею для проектирования Qt/QML приложений прямо внутри OCaml, т.е. реализовать язык разметки, который можно будет перевести в вид, понятный Qt/QML.

2.3. Предыдущая работа

В предыдущей работе по интеграции Qt/QML с OCaml была добавлена возможность написания разметки графического интерфейса внутри OCaml с помощью создания синтаксического расширения, а также

[5]https://reasonml.github.io/docs/en/what-and-why (дата обращения 31.05.2023)
[6]https://jchavarri.github.io/melange-docs/ (дата обращения 31.05.2023)
[7]https://rescript-lang.org/ (дата обращения 31.05.2023)
[8]https://www.outrunlabs.com/revery/ (дата обращения 31.05.2023)
[9]https://github.com/briskml/brisk (дата обращения 31.05.2023)

добавлена возможность генерации кода на QML из нового синтаксиса. В данной работе будут использоваться результаты практики предыдущего семестра.

2.4. qmlsc

Для компиляции QML в C++ может использоваться Qt Quick Compiler[10], который состоит из:

- QML type compiler;
- QML script compiler:
  - qmlcachegen;
  - qmlsc[4].

2.5. Инструменты

2.5.1. Формальная грамматика и LL(1)-анализатор

Формальная грамматика [1] – это набор из четырёх компонентов:

1. набор терминалов;
2. набор нетерминалов;
3. набор правил вывода;
4. начальный символ - специальный нетерминал.

LL(1)-анализатор[11] – это синтаксический анализатор для некоторых контекстно-свободных грамматик, где буквы L означают, что анализ входных данных идёт слева направо и строится левосторонний вывод.

[10] https://doc.qt.io/qt-6/qtqml-qtquick-compiler-tech.html (дата обращения 31.05.2023)
[11] https://en.wikipedia.org/wiki/LL_parser

2.5.2. Camlp5

Camlp5 [5] – препроцессор для OCaml [3], который использует LL(1)-анализатор и который позволяет расширять язык новым синтаксисом.

В Camlp5 используется парсер с заглядыванием вперед на одну лексему. Это необходимо учитывать в написании решения, так как рассмотрения лишь одной лексемы из потока не всегда будет достаточно для распознания нужной синтаксической конструкции.

2.5.3. Dune

Dune – система сборки проектов для OCaml. В данной задаче используется для создания проекта, состоящего из нескольких модулей, а также для использования препроцессорной обработки и подключения C/C++ модулей, что Dune позволяет удобно делать.

2.5.4. OCaml API для C

OCaml предоставляет интерфейс для взаимодействия с C. В данной работе будет использоваться данный API для определения функций из OCaml в C и вызов OCaml функций из C.

2.6. lablqml

lablqml предоставляет биндинги к основным функциям фреймворка Qt/QML, которые позволяют создать основу приложения непосредственно из OCaml.

3. Реализация

3.1. Meta-Object System

Механизм слотов и сигналов в фреймворке Qt/QML реализован с помощью moc (Meta-Object Compiler), который раскрывает Qt макросы (Q_OBJECT, Q_INVOKABLE) класса QObject во вспомогательные функции. Meta-Object Compiler не позволяет добавлять сигналы и слоты во время выполнения программы, из-за чего предлагается реализовать механизм добавления слотов и сигналов динамически. Это необходимо для избежания кодогенерации C++.

Был найден пример реализации динамических объектов [12], но некоторые, необходимые для данной работы макросы не поддерживаются приведенным решением. Например макрос Q_INVOKABLE позволяет вызывать C++ функцию из разметки QML[6]. Использование данного макроса в проекте заключается в вызове методов connect наследника класса DynamicQObject из QML:

Листинг 1: Пример сигнатуры Q_INVOKABLE

```
Q_INVOKABLE bool callEmitDynamicSignal(char *signal, void **
   arguments);
Q_INVOKABLE bool callConnectDynamicSlot(QObject *obj,  QString
   signal, QString slot);
Q_INVOKABLE bool callConnectDynamicSignal(QString signal, QObject
    *obj, QString slot);
```

Листинг 2: Пример вызова метода из QML

```
    signal qml_singal1()
    Component.onCompleted: {
        wrapper.callConnectDynamicSlot(root, "qml_signal1()", "
           caml_slot1()");
    }
```

[12] https://doc.qt.io/archives/qq/qq16-dynamicqobject.html (дата обращения 31.05.2023)

Вызов методов динамичесского объекта осуществлен через класс-обёртку, который хранит ссылку на DynamicQObject и который в свою очередь компилируется moc'ом, в отличие от DynamicQObject.

3.2. Вызов OCaml кода через динамические слоты

Необходимо интегрировать lablqml со своим решением, для создания основы приложения на Qt/QML непосредственно из OCaml.

Чтобы создать экземпляр DynamicQObject'а и добавить в него динамические слоты, необходимо определить OCaml функцию, тело которой реализовано на C/C++, и которая вернет указатель на объект[2]:

Листинг 3: Декларация функции

```
external create_func_stub : string → (unit → unit) → t = "
  caml_create_func"
```

Листинг 4: Примерная реализация функции на C/C++

```
extern "C" value caml_create_func(value _name, value _func)
{
    _ans = caml_alloc_small(1, Abstract_tag);
    CamlDynamicQObj *camlObj = new CamlDynamicQObj(nullptr);
    CamlSlot *camlSlot = new CamlSlot(camlObj, _func);
    QString slotSignature = QString(String_val(_name));
    camlObj → addCustomSlot(slotSignature, camlSlot);
    *((Wrapper **)&Field(_ans, 0)) = new Wrapper(0, camlObj);
    CAMLreturn(_ans);
}
```

После передачи указателя на OCaml функцию в класс CamlSlot, необходимо подключить динамический слот к некоторому QML сигналу (пример приведен в листинге 2). Когда будет вызван сигнал, выполнится функция CamlSlot::call, где с помощью caml_callback вызывается OCaml код:

Листинг 5: OCaml callback

```
void CamlSlot::call(QObject *sender, void **arguments)
{
    caml_callback(_camlSlot, Val_unit);
}
```

Таким образом, можно создавать простейшие OCaml обработчики, с помощью передачи в функцию caml_create_func функций, с сигнатурой вида (unit -> unit).

3.3. Добавление нового синтаксиса

В предыдущем семестре был разработан прототип парсера разметки. Так как необходимо добавить новый синтаксис, был переписан парсер, который раскрывает OCaml обработчики из разметки в вызов функций из своей библиотеки а также из lablqml API.

Листинг 6: Новый синтаксис

```
ApplicationWindow {
    id: root;
    Button {
        slot onClicked: {
            Printf.printf "Hello from OCaml"
        }
    }
}
```

Листинг 7: Синтаксис после обработки препроцессором

```
ApplicationWindow {
    id: root
    signal qml_singal1()
    Component.onCompleted: {
        wrapper.callConnectDynamicSlot(root, "qml_signal1()", "
           caml_slot1()");
```

```
    }
    Button {
        onClicked: qml_signal1()
    }
}
```

4. Тестирование

Было сделано несколько тестовых программ для проверки корректности работы библиотеки. Измерение покрытия не было произведено из-за несовместимости утилиты для измерения покрытия и препроцессора camlp5.

Заключение

В результате работы были выполнены следующие задачи:

- спроектирован интерфейс для взаимодействия между динамическими объектами C++ и QML-фронтэндом;
- реализован вызов OCaml из C/C++;
- добавлена поддержка OCaml обработчиков сигнатуры (unit -> unit) в разметку;
- проведено тестирование.

Ссылка на репозиторий GitHub: https://github.com/mikhaylovilya/ocaml-syntax-extension

Список литературы

[1] Compilers: Principles, Techniques, and Tools (2nd Edition) / Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. — Addison Wesley, 2006. — August. — ISBN: 0321486811.

[2] Interfacing C with OCaml. — URL: https://v2.ocaml.org/manual/intfc.html (дата обращения: 31.05.2023).

[3] Minsky Yaron, Madhavapeddy Anil, Hickey Jason. Real World OCaml - Functional Programming for the Masses. — O'Reilly, 2013. — P. I–XXII, 1–483. — ISBN: 978-1-4493-2391-2.

[4] Ulf Hermann. — URL: https://www.qt.io/blog/author/ulf-hermann (дата обращения: 31.05.2023).

[5] camlp5. — URL: https://camlp5.github.io (дата обращения: 31.05.2023).

[6] qml book. — URL: https://qmlbook.github.io/ (дата обращения: 31.05.2023).