Санкт-Петербургский государственный университет

Кафедра системного программирования

Кузьмина Елизавета Владимировна

# Графическая технология управления «умной теплицей»

Бакалаврская работа

Научный руководитель:
к.т.н., доцент кафедры системного программирования
Литвинов Ю.В.

Рецензент:
младший инженер-программист АО «ПФ «СКБ Контур»
Перешеина А.О.

Санкт-Петербург
2018

SAINT PETERSBURG STATE UNIVERSITY

Software Engineering

Elizaveta Kuzmina

# Graphical technology for “smart greenhouse” programming

Graduation Thesis

Scientific supervisor:
C.Sc., Associate Professor Yurii Litvinov

Reviewer:
junior developer at CJSC Production Company “SKB Kontur”
Peresheina Anna

Saint Petersburg
2018

# Оглавление

# Введение

В современном мире набирает популярность концепция Интернета вещей (Internet of Things) [15, 17], что обусловлено растущим числом подключаемых к Интернету устройств [9]. Концепция заключается в том, что можно настраивать совместную работу различных гаджетов, которые оснащены встроенными технологиями для взаимодействия друг с другом или с внешней средой, объединяя их в единую систему с помощью различных технологий передачи данных.

Примером применения такой концепции является «умная теплица», которая должна самостоятельно, без непосредственного участия ее владельца, реагировать на изменения внешней среды. Пользователь может установить в своем парнике и подключить к системе необходимые датчики и актуаторы (исполнительные устройства), а затем создавать сценарии их работы. Например, для того, чтобы закрывать/открывать окна в момент, когда сенсоры передают определенные значения влажности земли, температуры воздуха внутри теплицы или для того, чтобы по заданному расписанию поливать почву.

Уже существуют системы подобного рода, которые позволяют описывать сценарии для «умных теплиц». Но эти системы либо требуют подключения строго определенного ряда приборов и не позволяют подключить другое оборудование, либо подразумевают наличие навыков программирования у владельца парника.

Для того, чтобы даже неподготовленный пользователь смог задавать нужные ему сценарии, необходимо создавать специальное программное обеспечение, ключевая роль в котором отводится системе, предназначенной для установки правил взаимодействия приборов, подключенных к теплице. Нужно предоставить возможность задавать требуемые функции даже пользователям, не знакомым с программированием.

Для этих целей подходит визуальное программирование – способ задавать программу путем манипулирования графическими примитивами вместо применения текстовых языков. Данный подход позволяет

упростить представление объектов, которыми приходится оперировать конечному пользователю при разработке сценария, заменяя код наглядными абстракциями, что облегчает обучение языку и его использование.

Элементы системы и функции при определении правил работы приборов теплицы в таком случае должны представляться в виде блоков. Пользователь размещает блоки на сцене редактора создаваемого программного обеспечения, рисуя в соответствии с желаемым результатом модель, по которой в дальнейшем с помощью генератора системы должен быть создан код программы, отрабатывающей задуманный сценарий.

Генерации кода по визуальной модели в общем случае препятствует семантический разрыв, который не позволяет модели, являющейся по определению упрощением моделируемого объекта, содержать всю необходимую информацию для порождения кода. Но в случае ограниченной области применения языка, что характерно и для «умной теплиц», такое преобразование в программный код возможно благодаря заранее известным характеристикам предметной области. Данный подход называется предметно-ориентированным моделированием (Domain Specific Modeling), а язык, на котором создается модель, в таком случае является предметно-ориентированным языком, то есть создается для конкретной задачи, и в совокупности с методом его применения и средствами инструментальной поддержки, такими как редактор модели, генератор кода, средство проверки ограничений, представляет собой так называемое DSM[1]-решение [8].

Работа в области визуального моделирования проводится в Санкт-Петербургском государственном университете на кафедре системного программирования и включает в себя работу над проблемами предметно-ориентированного моделирования, которая ведется наравне с разработкой инструментальных технологий быстрого создания DSM-решений, так называемых DSM-платформ. Примерами таких платформ являются среда QReal [16], на основе которой создана технология визуального

[1]Domain Specific Modeling

программирования роботов, а также находящаяся в стадии активной разработки система REAL.NET[2] [12] – набор конфигурируемых и переиспользуемых компонентов для создания предметно-ориентированных визуальных языков.

Новая платформа REAL.NET нуждается в апробации, которая проводится путем создания на ее основе визуальных языков и сопутствующих инструментов [13]. Тестированием REAL.NET может послужить решение с ее помощью актуальной задачи создания графической технологии для управления теплицей.

[2]GitHub-репозиторий проекта REAL.NET, URL: https://github.com/yurii-litvinov/REAL.NET (дата обращения: 30.05.2018)

# 1. Постановка задачи

Целью работы является создание прототипа системы автоматического управления теплицей на основе платформы REAL.NET, включающего в себя предметно-ориентированный визуальный язык программирования для задания пользователем сценариев работы приборов. Для достижения этой цели были сформулированы следующие задачи.

- Выполнить проектирование целевой системы и входящего в нее предметно-ориентированного визуального языка.
- Реализовать редактор для моделирования пользователями сценариев работы теплицы.
- Реализовать генератор для преобразования графической модели в сценарий на текстовом языке.
- Провести апробацию прототипа.

# 2. Обзор

## 2.1. Существующие решения

При обзоре существующих решений рассматривались не только «умные теплицы», но также другие системы Интернета вещей, позволяющие автоматизировать работу набора приборов путем задания нужных пользователю сценариев. Целью обзора было рассмотрение способов написания сценариев, но интерес представляли также и механизмы подключения приборов к системам. Были рассмотрены следующие решения.

**HortiMaX-Go! контроллер**

HortiMaX-Go! [2] – это закрытая система, которая включает в себя программное обеспечение для управления микроклиматом и водными ресурсами в полностью автоматическом режиме, а также для сбора статистических данных. Недостатком системы является то, что для ее работы необходимо строго определенное аппаратное обеспечение, включающее в себя как сам микроконтроллер с сенсорным экраном, так и подключаемые к нему специальные коммутаторы «Smart Switch», которые могут выполнять специфичное управление различными устройствами, например, узлами, смешивающими удобрения. А плюсом системы является то, что через «Smart Switch» к контроллеру можно подключить любые необходимые приборы, при этом от человека требуется только правильно присоединить контакты к клеммам коммутатора и присвоить коммутаторам адреса в сети, дальше программное обеспечение сможет самостоятельно определить все подключенные устройства и скорректировать по этим данным интерфейс для их мониторинга.


Рис. 1: Заданное правило в системе HortiMaX-Go!

Задание правил автоматизации производится отдельно для каждого типа устройств. У каждого такого типа оборудования в сценарии сутки делятся на заранее определенное количество отрезков времени, называемых этапами, и каждому этапу соответствуют свои условия срабатывания актуаторов. На рисунке 1 представлено правило для одного из этапов, которое пользователь определил путем выбора граничных значений температуры. Согласно условиям, при температуре ниже 18 градусов в теплице будет включаться отопление, а при 20 градусах – охлаждение, мощность которого увеличится, если температура достигнет отметки в 22 градуса.

**Платформа OpenHAB**

Технология OpenHAB [4] написана полностью на языке Java и позволяет объединить приборы с разными протоколами в единую сеть. За счет того, что платформа является открытой, она активно разваивается и к ней добавлется поддержка различных устройств. Для каждого вида устройства подключается специальный плагин, который приводит интерфейс взаимодействия с прибором к общему виду.

Технология предоставляет мощный инструмент программирования логики с помощью скриптов и правил для задания «умного» поведения устройств. Скрипты и правила описываются на языке Xtend, разработанном командой проекта Eclipse. Скрипты определяются в текстовом файле, который делится на секции: секция импортирования различных типов, секция объявления переменных и секция самих правил. Каждое правило имеет блоки “when” с условиями срабатывания и “then” с блоком команд. То, что сценарии пишутся на специальном текстовом языке, можно считать недостатком для не знакомых с программированием пользователей.

Все подключаемые к системе приборы записываются пользователем в специальный список с указанием типа устройства, его имени, иконки для отображения на экране, канала связи и группы, к которой оно принадлежит по расположению (группой могут быть датчики, расположенные, например, в одной комнате). Для подключения микрокон-

троллеров с операционной системой Linux существует готовый образ системы с OpenHAB – openHABian, который позволяет начать работу без лишних затрат времени на установку программного обеспечения.

**Графический конфигуратор Node-RED**

Инструмент программирования с открытым исходным кодом Node-RED написан на языке JavaScript и позволяет через браузер построить в графическом виде схему взаимодействия устройств между собой и по ней сгенерировать исполняемый код. На сцене располагаются блоки сенсоров и актуаторов, а также функциональные блоки. У каждого из них есть своя панель редактирования свойств и некоторые панели включают окно для написания функций на языке JavaScript. Узлы модели сценария связываются линиями так, чтобы данные, поступающие от сенсоров проходили через функциональные блоки и в результате, модифицированные требуемым образом, оказывались на входе у актуаторов. Передача информации между узлами сценария реализована с помощью JavaScript объектов msg, имеющих перегрузку Payload, которая может быть объектом любого типа.

Существует множество библиотек, которые позволяют расширить возможности инструмента и добавлять блоки в стандартную палитру системы [3]. Например, возможна работа с контактами GPIO (интерфейсами ввода-вывода общего назначения) различных микроконтроллеров для общения с физическими устройствами. В добавленных блоках сенсоров и актуаторов указываются номера портов, к которым подключены приборы. Для беспроводных датчиков есть блоки с возможностью использовать открытый протокол MQTT, который работает поверх стека протоколов TCP/IP. В протоколе MQTT собщения передаются между клиентом (издателем или подписчиком) и брокером – координатором, который устанавливается на сервере. Издатель посылает брокеру сообщение с указанием темы, а брокер отправляет его подписавшимся на эту тему клиентам. При использовании такого подхода для input и output блоков в Node-RED во время моделирования сценария прописываются адреса брокеров и темы.

На рисунке 2 показана модель сценария, аналогичная сценарию на рисунке 1, а также часть панели редактирования свойств блока switch, с помощью которого и заданы граничные значения, поступающие от датчика температуры по MQTT протоколу. Можно сказать, что система является универсальной, но она оказывается довольно сложной для неподготовленного пользователя за счет наличия на палитре большого количества разнообразных блоков, разницы в способах задания для них правил, а также необходимости при использовании многих из них обладать знаниями языка программирования JavaScript.

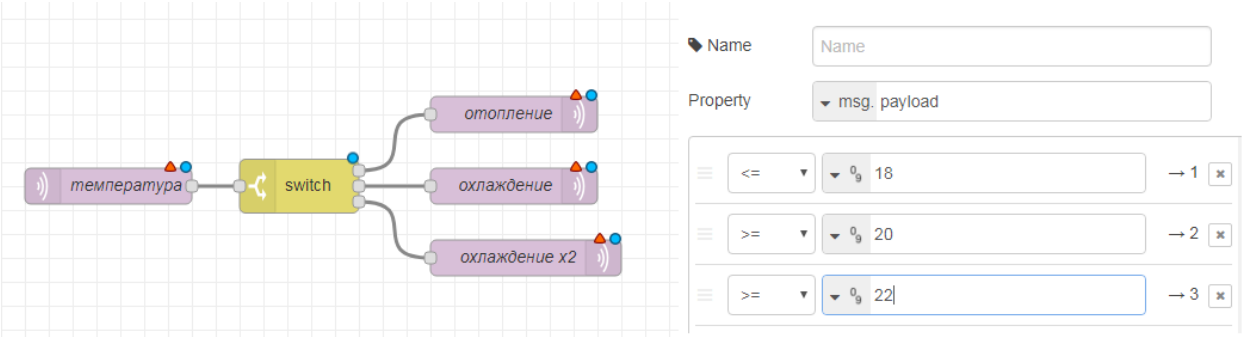


Рис. 2: Заданное правило в системе Node-RED

**Платформа Samsung Artik Cloud**

Облачная платформа Artik Cloud предназначена для объединения всевозможных устройств Интернета вещей. Существуют специальные инструменты разработки для интеграции облачного сервиса с различными языками программирования, такими как Java, Python, C и C++. Но к ним не относятся языки платформы .NET. Для каждого типа устройств определен манифест, в котором указано, какие данные прибор может получать и какие отправлять. С помощью протоколов, таких как уже упомянутый MQTT, устройство обменивается с облаком сообщениями.

Можно создавать правила для запуска действий на основе сообщений Artik Cloud. Правила определяются следующим образом: из выпадающих списков выбираются сенсор, поле (из списка полей, определенном в манифесте устройства), оператор, затем указывается граничное значение. В области THEN выбирается актуатор и необходимая для

выполнения команда. Команда может быть стандартной или пользовательской (написанной с использованием формата JSON). Такой способ задания сценариев предполагает заполнение пользователем пробелов в предоставляемой форме.

Вывод: выявленные при обзоре недостатки систем подтверждают актуальность решаемой задачи. Также, никакие из рассмотреных платформ не могут быть использованы в настоящей работе из-за несовместимости технологий.

## 2.2. Используемые технологии

**Метамоделирование, платформа REAL.NET**

Формально определить любой язык помогает его синтаксис. Он задает правила построения конструкций языка из его элементов. Для определения синтаксиса визуального языка обычно применяют метамоделирование [18]. В этом случае описание проводится посредством использования иного языка, называемого метаязыком. На метаязыке создается метамодель – модель, описывающая множество всех корректных конструкций языка.

Платформа REAL.NET – это среда предметно-ориентированного визуального моделирования, которая разрабатывается на платформе .NET и реализует принцип «глубокого метамоделирования», при котором сущность в модели рассматривается одновременно и как тип, и как экземпляр некоторого типа.

На рисунке 3 представлена архитектура системы REAL.NET. Ее ядром является написанный на языке F# репозиторий, с помощью которого создаются визуальные языки. В репозитории определены методы работы с языками и хранятся их метамодели. При создании нового языка строится новая метамодель с помощью инфраструктурной метамодели, которая содержит элементы, включающие узел, отношения ассоциации и наследования, от которых можно инстанцироваться для создания узлов и отношений нового языка, а также она позволяет до-

бавлять атрибуты создаваемым элементам. Таким образом, в репозитории определяется набор всех возможных сущностей языка.

С репозиторием связаны редакторы и средства проверки ограничений. В настоящей работе используется один из редакторов системы REAL.NET, который реализован на языке программирования C# с помощью графической подсистемы WPF (Windows Presentation Foundation) в составе .NET Framework и библиотеки GraphX [1] с открытым исходным кодом, которая предоставляет широкий спектр инструментов для отрисовки графов. Данный редактор имеет сцену для моделирования сценария, на которую можно перетаскивать объекты метамодели, представленные в палитре на боковой панели. У элементов модели, список которых доступен также на боковой панели, можно менять заданные в той же метамодели свойства с помощью редактора свойств.

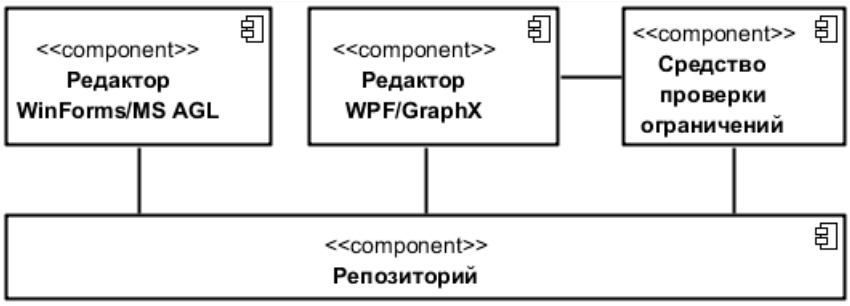


Рис. 3: Общая структура REAL.NET

**Библиотека RX.NET**

RX.NET [5] – библиотека Reactive Extentions для платформы .NET от компании Microsoft. Она упрощает работу с потоками событий и данных. Для этого в ней определяются два главных интерфейса: IObservable и IObserver. IObservable содержит единственный метод, позволяющий подписываться на него. Подписчиком может стать IObserver, у которого определены три метода: для обработки новых поступающих элементов, ошибок и сигналов окончания последовательности. Интерфейс ISubject представляет объекты, которые одновременно являются и IObservable, и IObserver. То есть они могут одновременно быть и подписчиками, и объектами, на которые можно подписаться. В библиотеке реализованы

методы, позволяющие конвертировать в обозреваемую последовательность наборы состояний, например, результаты сгенерированных событий.

**Text Template Transformation Toolkit**

Шаблоны T4 [6] применяются для генерации кода и состоят из текстовых блоков и блоков логики управления. Управляющий код позволяет задавать значения для переменных частей строк и манипулировать условными и повторяющимися частями для формирования конечных файлов. Шаблоны также могут вызываться во время исполнения, для чего им передаются в качестве параметров необходимые переменные. Шаблоны T4 являются надстройкой для Visual Studio, однако есть возможность их использования вне данной среды разработки путем подключения специальных библиотек.

**Библиотека Trik-Sharp**

Библиотека Trik-Sharp [7] предоставляет инструменты для работы с внутренними и внешними периферийными устройствами микроконтроллера TRIK [10]. На центральном процессоре микроконтроллера стоит операционная система Linux, программирование для него возможно на множестве языков, включая языки платформы .NET. Микроконтроллер TRIK имеет 4 порта двигателей и 19 сигнальных портов, а подключение к нему возможно по сети Wi-Fi.

Библиотека Trik-Sharp предназначена для программирования на языках C# и F#. Она дает возможность рассматривать подключаемые сенсоры как Observable в терминах Reactive Extentions, а актуаторы – как Observer, что упрощает работу с ними.

# 3. Архитектура разработанного решения

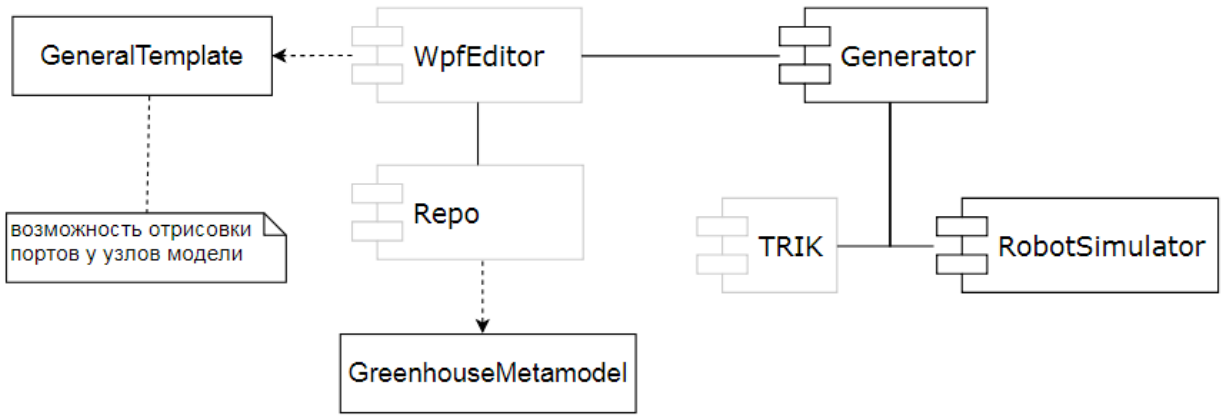


Рис. 4: Диаграмма компонентов прототипа

На рисунке 4 представлена диаграмма компонентов прототипа графической системы. Компоненты, которые требовалось реализовать или модифицировать, выделены черным цветом. Один из них – это метамодель нового визуального языка, реализованная описанным выше образом с помощью инфраструктурной метамодели репозитория системы REAL.NET.

Входящие в метамодель языка «умной теплицы» элементы показаны на рисунке 5. В нее входит единственное отношение ассоциации, являющееся экземпляром инфраструктурной ассоциации. Отношение задается с помощью указания источника и целевой вершины, которые обязаны являться наследниками абстрактного узла метамодели, также являющегося экземпляром инфраструктурного. В метамодель включены сенсоры, которых на данный момент два типа и актуаторы с тремя типами. Атрибутами для них являются номера портов, к которым подключены физические устройства. Потомками абстрактного узла также являются логическая операция, наследниками которой являются операции И и ИЛИ, и интервал, который с помощью атрибутов минимального и максимального значения задает открытый промежуток для проверки соответствия ему входного значения. При этом значение null интерпретируется как отсутствие нижнего/верхнего ограничения.

Также атрибутами для каждого элемента метамодели являются: графическая фигура, задающая его вид при визуальном построении сценария; булевый тип isAbstract, показывающий, можно ли использовать элемент при непосредственном моделировании; тип instanceMetatype – родительский элемент в инфраструктурной метамодели.

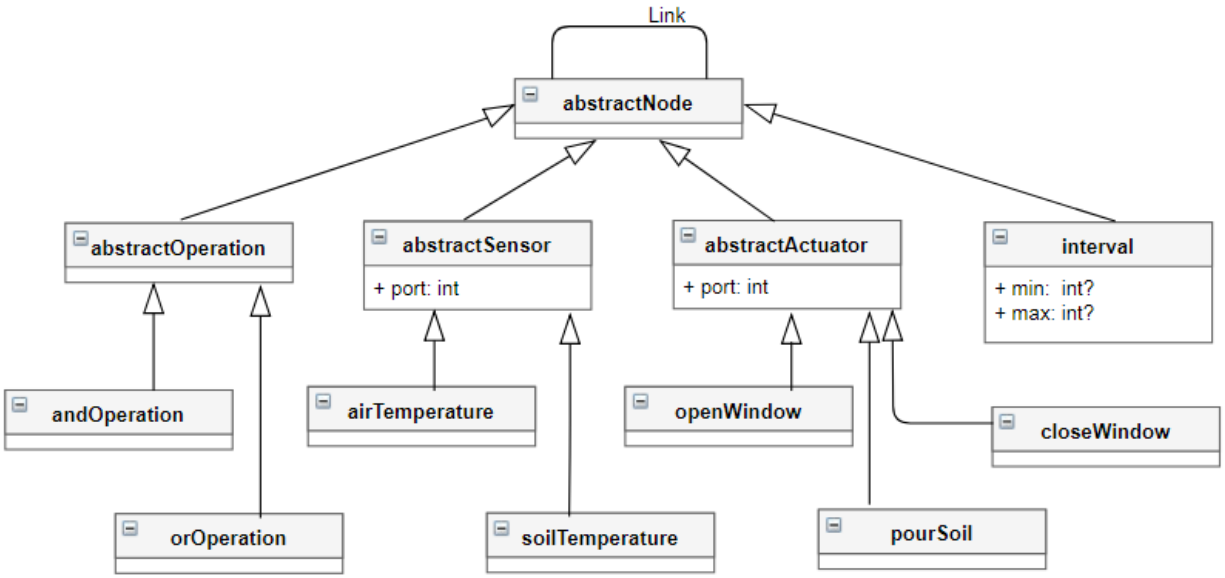


Рис. 5: Метамодель визуального языка для “умной теплицы”

Всеми этими элементами пользователь оперирует для создания модели сценария с помощью компонента WpfEditor – редактора системы REAL.NET, в котором модифицировано визуальное представление вершин для моделирования поведения «умной теплицы». Для создания текстового сценария модель подается на вход генератору, с помощью которого создается исполняемый файл для микроконтроллера. Компонент RobotSimulator предназначен для тестирования сценариев без использования робота.

# 4. Редактор для отрисовки моделей сценариев

Как было сказано ранее, прототип системы управления “умной теплицей” использует WPF/GraphX редактор платформы REAL.NET (рисунок 6).

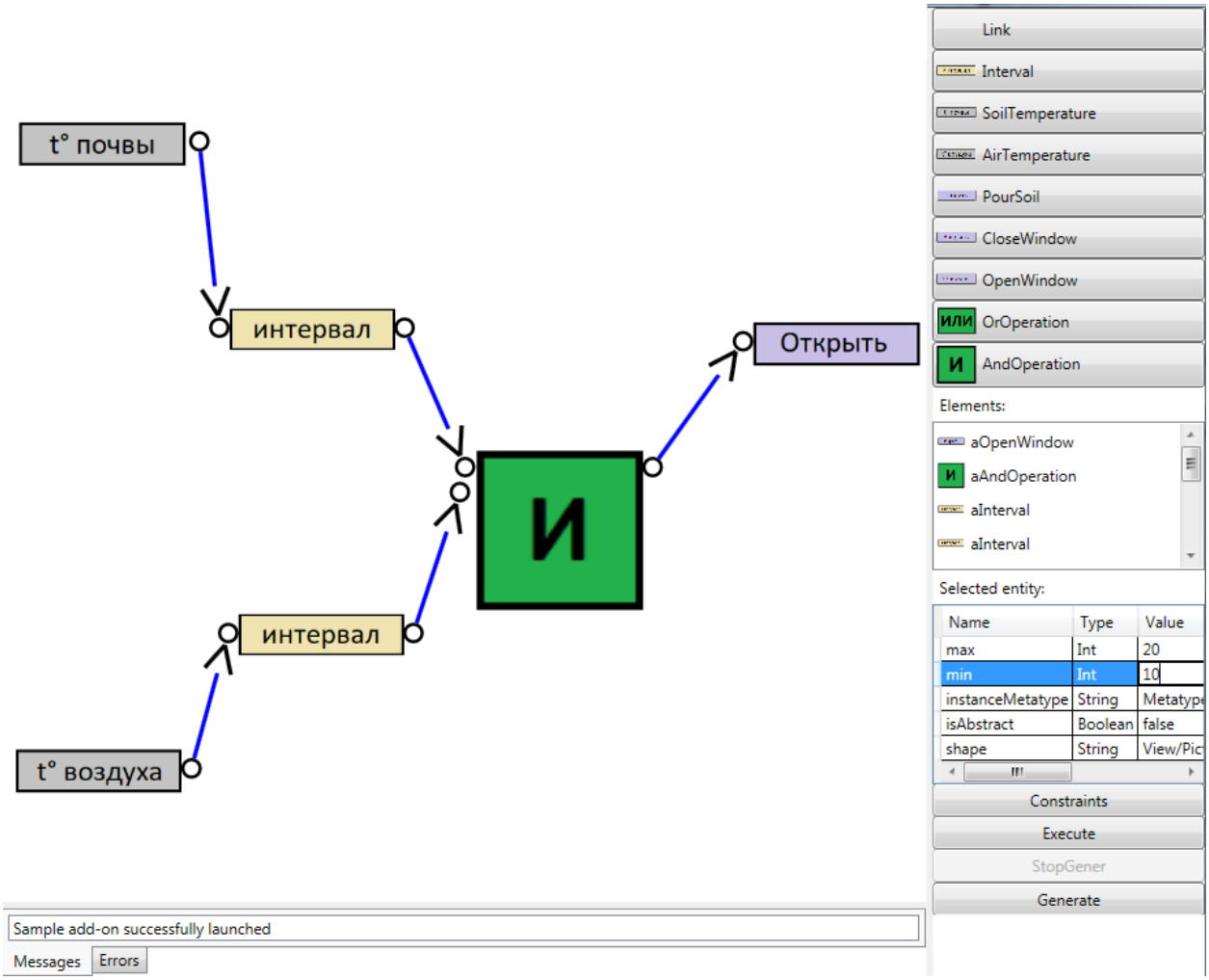


Рис. 6: Редактор платформы REAL.NET

Определяя сценарий работы приборов, пользователь перетаскивает из палитры на сцену все сенсоры и актуаторы, находящиеся в теплице, а также интервалы и логические операции, которые необходимы для определения условий срабатывания актуаторов и формирования более сложных правил. Затем задает в редакторе свойств необходимые значения атрибутов элементов. Поток данных визуализируется с помощью направленных ребер, которые пользователь отрисовывает, выбрав

стрелку в палитре и щелкнув последовательно сначала на источник данных, а затем на их приемник.

В редакторе добавлена возможность отрисовки портов у элементов языка “умной теплицы”. Для этого от класса StaticVertexConnectionPoint библиотеки GraphX унаследован новый класс портов для вершин, в котором добавлены свойства, показывающие является ли порт занятым и для какого типа ребер предназначен – входящих или исходящих.

К узлам модели добавлены панели для отображения портов: с левой стороны – для входящих ребер, а с правой, соответственно, – для исходящих. При этом у сенсоров порты могут располагаться только с правой стороны, а у актуаторов только с левой, так как первые могут только отправлять свои данные, а вторые только получать команды. При добавлении ребер реализована возможность для всех узлов динамически добавлять новые порты, если существующие порты того же типа уже все заняты. Исключение составляет узел ”интервал”, который может принимать на вход единственное ребро.

После того как сценарий смоделирован, пользователь нажимает на кнопку Generate на правой панели для генерации по модели кода. При нажатии с помощью описанного далее генератора создается исполняемый файл, который в дальнейшем можно загрузить на контроллер.

# 5. Генератор сценариев

Генератор для получения по графической модели сценария для контроллера реализован на языке C# с помощью уже рассмотренных технологий T4 и RX.NET.

При генерации сценариев для «умной теплицы» используется шаблон времени исполнения, в качестве параметра ему передается модель из репозитория. В начале генерируемого кода для каждого узла модели создается экземпляр соответствующего класса: сенсор, актуатор, интервал или логическая операция. Объекту присваивается индивидуальный номер, который используется, во-первых, для задания его имени (приведенный к типу строки идентификатор конкатенируется со строкой “element”), а во-вторых, для того, чтобы блоки операций могли хранить словарь со значениями от всех подключенных к ним узлов, где этот самый номер является ключем. Словарь для логических операций формируется изначально путем рассмотрения всех источников входящих в них ребер. Далее у всех узлов задаются свойства, значения которых также определяются из модели.

Затем каждый элемент рассматривается как Subject в терминах Reactive Extentions, который генерирует события для своих подписчиков – всех элементов сохраненной модели, к которым от него направлены ребра, а также подписывается на сигналы от элементов, являющихся источниками для входящих в него ребер, и обрабатывает их. Сенсоры модели имеют только исходящие ребра, но подписываются на события реальных сенсоров, так же как не имеющие исходящих ребер актуаторы модели публикуют события для реальных актуаторов.

Работа с реальными устройствами микроконтроллера TRIK проводится с помощью библиотеки Trik-Sharp. При этом для каждого сенсора и актуатора модели создаются соответствующие объекты классов библиотеки, обрабатывающие и посылающие информацию на номера портов, указанных в свойствах элементов. Созданные на их основе Observer и Observable объекты общаются с узлами уже построенной модели.

Все наблюдатели модели обрабатывают событие только в том слу-

чае, если передаваемое значение отличается от предыдущего значения того же узла-источника. Таким образом, физический актуатор срабатывает только в том случае, если до соответствующего актуатора модели дошло значение True от какого-либо связанного с ним узла, который до этого передавал значение False.

На рисунке 7 представлена самая маленькая из возможных моделей, а на приведенном далее листинге кода – часть сгенерированного по этой модели сценария. Узлу «температура воздуха» передается значение с физического датчика температуры, далее значение переходит на вход к узлу «интервал», где проверяется, принадлежит ли оно промежутку (5, 15), а затем значение операции проверки принадлежности передается узлу «Открыть», который в случае получения True отправляет команду реальному актуатору.

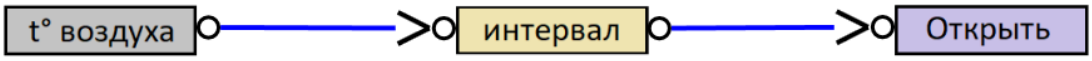


Рис. 7: Пример модели

```
element0 = new Actuator(0);
element0.Port = 0;
IObservable<int> observable0 =
    System.Reactive.Linq.Observable.FromEventPattern<int>(
                h => element0.Event += h, h => element0.Event -= h)
                .Select(e => e.EventArgs).Synchronize().DistinctUntilChanged();
IObserver<int> observer0 = Observer.Create<int>(x => element0.Action(x));
ISubject<int> reactElement0 = Subject.Create<int>(observer0, observable0);

element1 = new Interval(1);
element1.Min = null;
element1.Max = null;
IObservable<int> observable1 =
    System.Reactive.Linq.Observable.FromEventPattern<int>(
                h => element1.Event += h, h => element1.Event -= h)
            .Select(e => e.EventArgs).Synchronize().DistinctUntilChanged();
IObserver<int> observer1 = Observer.Create<int>(x => element1.Action(x));
ISubject<int> reactElement1 = Subject.Create<int>(observer1, observable1);

element2 = new Sensor(2);
```

```
element2.Port = 0;
IObservable<int> observable2 =
    System.Reactive.Linq.Observable.FromEventPattern<int>(
            h => element2.Event += h,h => element2.Event -= h)
                .Select(e => e.EventArgs).Synchronize().DistinctUntilChanged();
IObserver<int> observer2 = Observer.Create<int>(x => element2.Action(x));
ISubject<int> reactElement2 = Subject.Create<int>(observer2, observable2);

var sub0 = reactElement1.Subscribe(reactElement0);
var sub1 = reactElement2.Subscribe(reactElement1);
```

# 6. Апробация

Для апробации был создан симулятор, имитирующий работу сенсоров, которые в случайный момент времени выдают случайные значения, и был проведен эксперимент с участием человека, не знакомого с основами программирования. Ему была предложена задача со следующей формулировкой.

Задача. При значении температуры воздуха больше 20 градусов на датчике, подключенном к порту номер 1, открыть окно с помощью актуатора на порте с номером 5, а при значении меньше 20 градусов – закрыть окно при помощи актуатора на порте номер 4.

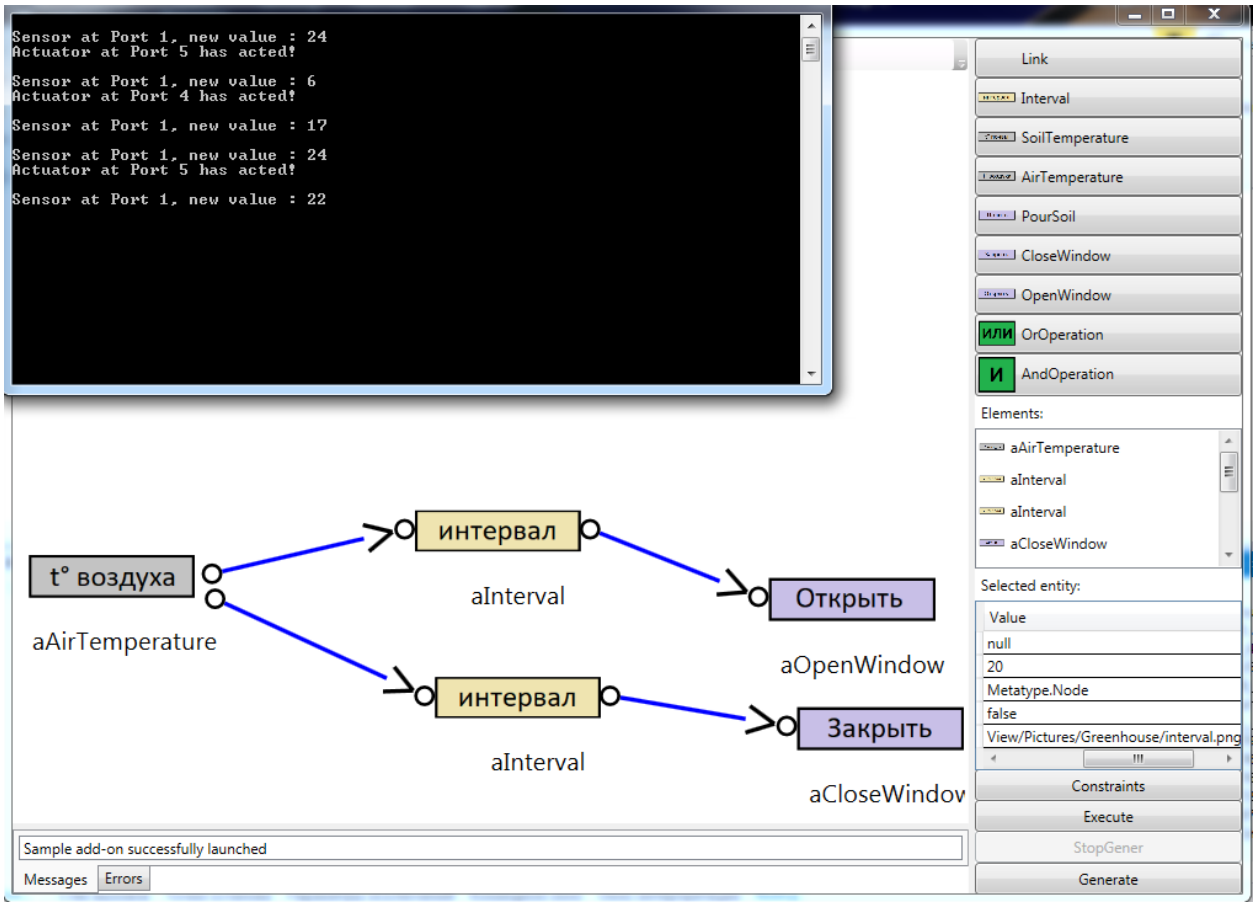


Рис. 8: Часть сгенерированного по модели кода сценария

После объяснения правил работы с системой и приведения примеров ее использования, пользователь потратил 3 минуты на моделирование в редакторе требуемого сценария.

На рисунке 8 представлен скриншот получившейся модели и запу-

щенного по ней сценария, сгенерированного с помощью симулятора робота.

Также было проведено тестирование с использование микроконтроллера TRIK. На основании подключенных к нему двух датчиков и одного актуатора в редакторе была построена модель сценария с указанием значений для срабатывания прибора. Сгенерированный на основании этой модели код был успешно запущен на микроконтроллере.

# Заключение

В ходе данной работы были получены следующие результаты.

- Спроектирована архитектура прототипа графической технологии управления «умной теплицей» и нового визуального языка.
- Реализован редактор на основе редактора системы REAL.NET для построения пользователем модели сценария работы теплицы.
- Реализован генератор с помощью библиотеки RX.NET и шаблонов T4, преобразующий созданную модель в сценарий на языке C#.
- Проведена апробация прототипа.

Основные результаты работы были доложены на конференции "Современные технологии в теории и практике программирования" [11]. Результат работы доступен по ссылке [14].

# Список литературы


[1] GraphX Library. — URL: https://github.com/panthernet/GraphX (online; accessed: 25.05.2018).

[2] HortiMaX-Go! — URL: http://www.hortimax.com (online; accessed: 25.05.2018).

[3] Node-RED. — URL: https://flows.nodered.org (online; accessed: 14.04.2018).

[4] OpenHAB. — URL: https://www.openhab.org (online; accessed: 14.04.2018).

[5] RX.NET Library. — URL: https://github.com/Reactive-Extensions/Rx.NET (online; accessed: 14.04.2018).

[6] Text Template Transformation Toolkit. — URL: https://msdn.microsoft.com/ru-ru/library/ee844259.aspx (online; accessed: 25.05.2018).

[7] Ttrik-Sharp Library. — URL: https://github.com/kashmervil/trik-sharp (online; accessed: 25.05.2018).

[8] Д.В. Кознов. Основы визуального моделирования. — БИНОМ. Лаборатория знаний, Интернет-Университет Информационных Технологий, 2008.

[9] Интернет вещей и межмашинные коммуникации. Обзор ситуации в России и мире // Мобильные телекоммуникации. — 2013.

[10] Контроллер TRIK. — URL: http://blog.trikset.com/p/blog-page_21.html (дата обращения: 30.05.2018).

[11] Кузьмина Е.В. Литвинов Ю.В. Графическая технология управления «умной теплицей». — Сборник материалов конференции «Современные технологии в теории и практике программирования» – СПб.: Изд-во Политехн. ун-та, 2018. — С. 20–22.

[12] Литвинов Ю.В. Кузьмина Е.В. Небогатиков И.Ю. Алымова Д.А. Среда предметно-ориентированного визуального моделирования REAL.NET // Всероссийская научная конференция по проблемам информатики СПИСОК-2017. — 2017. — URL: http://spisok.math.spbu.ru/2017/txt/SPISOK-2017.pdf (дата обращения: 14.04.2018).

[13] Небогатиков И.Ю. Литвинов Ю.В. Создание визуального предметно-ориентированного языка программирования дронов для симулятора AirSim. — Сборник материалов конференции «Современные технологии в теории и практике программирования» – СПб.: Изд-во Политехн. ун-та, 2018. — С. 66–68.

[14] Репозиторий проекта. — URL: https://github.com/elizakuz/REAL.NET (дата обращения: 30.05.2018).

[15] С. Дроздов С. Золотарев. Eurotech, «интернет вещей» и «облако устройств». — Control Engineering Россия, № 8(78) '2012, 2012. — С. 18–24.

[16] Терехов А.Н. Брыксин Т.А. Литвинов Ю.В. и др. Архитектура среды визуального моделирования QReal. — Системное программирование. Вып. 4. – СПб.: Изд-во СПбГУ, 2009. — С. 171–196.

[17] Черняк Л. Интернет вещей: новые вызовы и новые технологии // Открытые системы. СУБД, № 4. — 2013.

[18] Ю.В. Литвинов. Методы и средства разработки графических предметно-ориентированных языков. — URL: https://disser.spbu.ru/files/disser2/727/disser/dissertation_Litvinov_4-12-2015.pdf (дата обращения: 21.04.2018).