САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

АВТОМАТИЗИРОВАННЫЙ АНАЛИЗ ПОВЕДЕНИЯ ПОЛЬЗОВАТЕЛЯ В QREAL:ROBOTS

Дипломная работа студента 545 группы

Кузенковой Анастасии Сергеевны

Научный руководитель	..... / подпись /	ст. преп. Литвинов Ю.В.
Рецензент	..... / подпись /	к.ф.-м.н., доцент Кознов Д.В.
“Допустить к защите” заведующий кафедрой,	..... / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург
2014

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

AUTOMATED ANALYSIS OF USER BEHAVIOUR IN QREAL:ROBOTS

by

Anastasia Kuzenkova

Graduation Thesis

Supervisor	. . . . .	Senior Lecturer Y.V.Litvinov
Reviewer	. . . . .	Associate Professor D.V. Koznov
“Approved by” Head of Department	. . . . .	Professor A.N. Terekhov

Saint Petersburg
2014

Оглавление

Введение

С распространением интерактивных компьютерных систем возросла необходимость в обеспечении удобства их использования. Но свойства и особенности таких систем зависят не только от компонентов программного обеспечения, но и от действий пользователя, а, следовательно, оценка удобства использования должна учитывать и модель самой системы, и аспекты поведения пользователей.

Человеко-компьютерное взаимодействие (Human-Computer Interaction, HCI) — мультидисциплинарная наука, предназначенная давать научно обоснованные рекомендации для проектирования и анализа более удобных компьютерных систем [1]. Человеко-компьютерное взаимодействие стоит на стыке разработки программного обеспечения и психологии, именно эти дисциплины вносят ключевой вклад в развитие HCI. Однако главная цель исследований HCI — создание методов, которые на основе знаний из психологии и разработки программного обеспечения могут применяться для создания интерактивных систем.

Но на практике трудно найти методы, где в равной степени используются знания из обеих областей. В настоящее время существует множество рекомендаций по проектированию интерфейсов на основе психологических исследований, которые покрывают отдельные случаи использования интерфейсов определенных систем, однако в процессе разработки программного обеспечения не всегда могут быть интерпретированы подходящим образом [2].

Главная причина этого кроется в том, что психология и разработка программного обеспечения использует разные «языки» для описания полученных знаний, поэтому основной большой задачей для исследователей HCI становится создание общего языка для этих областей, чтобы появилась возможность интерпретировать и применять знания из психологии в разработке. Наличие формальных методов позволяет проверять и доказывать влияние изменений в системе на удобство использования [1].

Хотя формальные методы широко используются во многих областях разработки программного обеспечения, они редко применяются в области проектирования взаимодействия пользователя с системой [3]. В большинстве случаев, когда речь идет о сложных интерактивных системах, поведение пользователя можно формализовать только с использованием дополнительных условий и ограничений, что делает универсальное моделирование всех аспектов поведения пользователя недостижимой целью. Но область создания формальных методов моделирования поведения

пользователя развивается и может способствовать оценке удобства использования интерфейса.

Отдельный класс систем взаимодействия - средства разработки программного обеспечения. Среди них можно выделить средства визуального программирования [4]. С момента их появления предполагалось, что они окажутся проще в освоении и в использовании в сравнении с текстовыми средствами, поскольку позволяют создавать новые системы путем непосредственного манипулирования определяющими ее компонентами и свойствами. Однако средства визуального программирования не получили широкого распространения, одной из причин этого стало неудобство их использования. В настоящее время актуальной задачей является исследование удобства использования сред визуального программирования с целью определить, как улучшить процесс взаимодействия с пользователем.

Постановка задачи

Целью дипломной работы является создание инструментария для исследования поведения пользователей среды QReal:Robots на этапе создания диаграмм, основываясь на формально описанных сценариях. Для достижения этой цели был сформулирован следующий набор задач.

1) Определить и классифицировать основные действия пользователя при работе с QReal:Robots.
2) Реализовать сбор информации о действиях пользователя во время работы с системой.
3) Построить формальную модель поведения пользователя в системе.
4) Разработать инструмент для описания сценариев создания диаграмм, а также автоматизированного анализа поведения пользователя в рамках этого сценария.
5) Провести эксперименты по анализу поведения пользователя на стадии создания диаграмм.

Глава 1. Обзор

В работе мы исследовали вопрос о формальном описании пользовательских сценариев. Многие подходы носят преимущественно теоретический характер. Для некоторых из них реализованы инструменты, которые позволяют автоматизировать процесс проверки формальных моделей.

Подходы к описанию формальных сценариев поведения пользователя

По результатам изучения существующих механизмов формализации поведения пользователя были определены три наиболее распространенных подхода (Таблица 1).

НАЗВАНИЕ	**КЕМ И КОГДА РАЗРАБОТАН**	**КРАТКОЕ ОПИСАНИЕ**	**ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ**
GOMS	Stuart Card, Thomas P. Moran and Allen Newell, 1983	Модель обработки информации пользователем в процессе взаимодействия с компьютерной системой	Различные текстовые редакторы
XDM	Fiorella de Rosis and Sebastiano Pizzutilo, 1997	Формализм описания поведения пользователя в многопользователь ской системе с помощью сетей Петри	Информационные системы в медицинских учреждениях
ANGEL AND DEMON GAMES	R. Rukienas, 2008	Формальное описание поведения пользователя как переходы между этапом планирования, выбора и действия	Банковские автоматы

Таблица 1. Основные подходы к формальному описанию поведения пользователя в системе

GOMS-модели

GOMS (Goals, Operators, Methods, Selection rules) модель — наиболее известная теоретическая концепция в HCI. Существует несколько GOMS-аналитических методов. Они различаются по форме, предположениям о

модели когнитивной архитектуры пользователя и возможностям. Рассмотрим четыре основных вариации модели семейства GOMS — KLM (Keystroke-Level Model), CMN-GOMS, NGOMSL и CPM-GOMS [5].

Основная идея GOMS-модели — исследовать способ решения задачи в терминах целей, операторов, методов и правил выбора. Мы будем рассматривать GOMS-модели на примере задачи редактирования текста.

Цель — это то, чего пользователь хочет достигнуть. Здесь цель употребляется в своем общем значении. Цели часто разделяются на подцели. Все подцели должны быть достигнуты для достижения общей цели. Декомпозиция целей может происходить многократно, тем самым выстраивая иерархию целей. Однако строгая иерархическая структура не требуется. В некоторых версиях GOMS-моделей несколько целей могут быть активными одновременно, а иногда удобно иметь для некоторых целей «плоскую» структуру, где вообще нет явной иерархии подцелей.

Оператор — действие, выполняющееся для достижения цели. Операторы могут быть перцепционные, когнитивные, моторные, или композиция нескольких из перечисленных. Операторы могут изменить внутреннее состояние пользователя или изменить состояние внешней среды. Важные параметры операторов, в особенности время исполнения, предполагаются независимыми от того, каким образом система или пользователь попали в текущее состояние (то есть не зависит от истории состояний). Время исполнения может быть приближено константой, распределением вероятности или функцией от параметров. Например, время печати одного слова можно приблизить константой, статистическим распределением или функцией от количества букв в слове и времени печати одного символа. В примере, который мы решили рассматривать, операторами могут быть движение курсора мыши, клик мыши, нажатие клавиши delete и другие.

Методы — это последовательности операторов и подцелей, позволяющих достигнуть цели. Если цели имеют иерархическую форму, то соответствующую иерархию будут иметь и методы. Содержание методов зависит от набора возможных операторов и от представленной задачи. Например, метод для достижения цели «Удалить фразу» может выглядеть так: передвинуть курсор в начало фразы, зажать клавишу shift, передвинуть курсор на конец фразы и нажать клавишу delete.

Правила выбора чаще всего содержат более одного метода для достижения цели. Например, вместо метода для удаления фразы, описанного выше, можно использовать и другой: передвинуть курсор в конец фразы, нажать клавишу delete в соответствии с количеством символов во фразе раз.

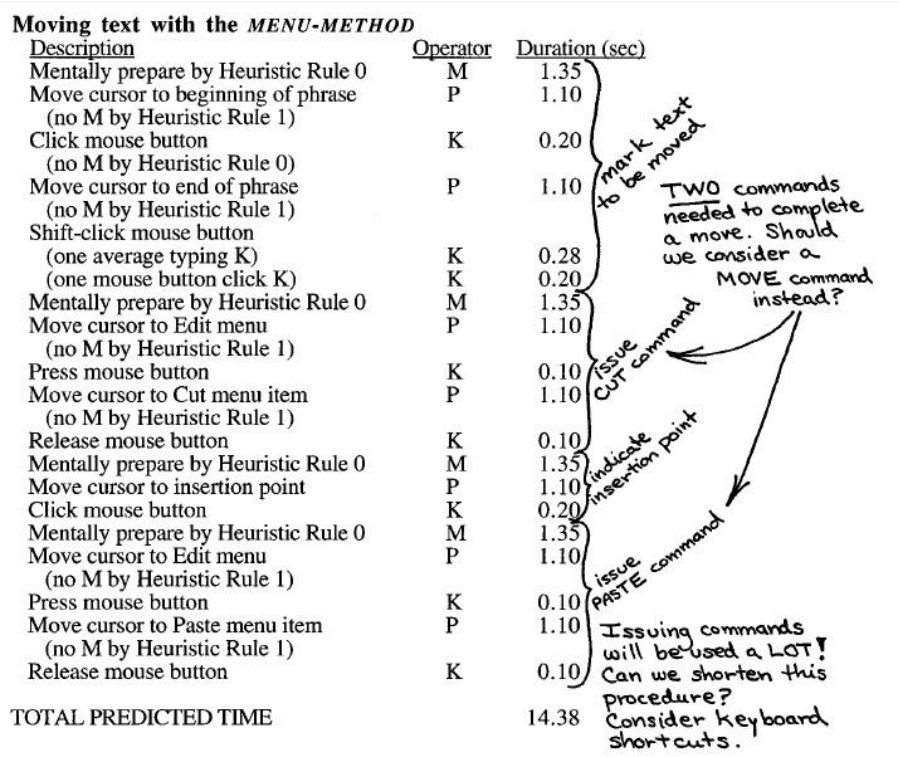
Если больше одного метода применимо для достижения цели, то правила выбора должны дать пользователю понять, какой из методов следует применить. Обычно это правила основываются на свойствах самой задачи. Правила выбора могут способствовать росту уровня владения интерфейсом у пользователя.

Разница между целью и оператором в GOMS-анализе зависит от выбранным исследователем уровнем детализации. Для цели аналитик выбирает методы, состоящие из операторов более низкого уровня, которые позволяют достигнуть ее. В противоположность операторы неделимы и не могут иметь дальнейшую детализацию. Степень детализации зависит от текущей задачи и от характера ее анализа. При этом не требуется все задачи системы приводить к одному уровню детализации — в иерархии некоторые цели могут быть описаны более подробно, если того требует исследование. Таким образом GOMS-модель в смысле детального описания задачи довольно гибкая и предоставляет исследователю простор для выбора описания.

Есть две основные формы GOMS-модели: программная форма и форма последовательностей. Программная форма аналогична параметризированной компьютерной программе. В таком случае у нас есть некоторый набор параметризированных задач. В зависимости от значений параметров генерируется последовательность действий для достижения цели задачи. Такая форма GOMS-модели компактна и дает унифицированное представление о ходе выполнения задачи, которое удобно анализировать. Однако с помощью такого представления нужно добиться генерации сценариев любой задачи рассматриваемой системы. Если система довольно сложна, то создание модели для генерации становится трудоемкой задачей. Форма последовательностей содержит фиксированную последовательность операторов для выполнения определенной задачи. Некоторые условия и параметры также могут быть включены в такую форму модели. Эта форма более наглядна, создать последовательность операторов выбранной задачи несложно, однако, если задач огромное количество, то придется создавать представление для каждой отдельной задачи.

Keystroke-Level Model (KLM) — простейший вариант GOMS-модели. Чтобы оценить время выполнения задачи, аналитик записывает последовательность операторов и затем высчитывает общее время выполнения задачи исходя из времени выполнения операторов. В этом типе модели исследователь должен явно задать метод, который используется для достижения цели. KLM-модель имеет 6 примитивных операторов: K — нажать на клавишу или кнопку мыши, P — наведение курсора мыши, H — перемещение рук на клавиатуру, D — нарисовать отрезок на сетке экрана, M — ментальная подготовка к действию, R — ожидание ответа от системы.

Время выполнения оператора рассчитывается, например, с помощью простейшей функции аппроксимации. Также расстановка ментальной подготовки к действию происходит по определенным эвристическим правилам. Модель изначально была разработана для командных интерфейсов, и претерпела соответствующие изменения в правилах расстановки оператора ментальной подготовки (рисунок 1).



Moving text with the *MENU-METHOD*

Description	Operator	Duration (sec)
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to beginning of phrase (no M by Heuristic Rule 1)	P	1.10
Click mouse button (no M by Heuristic Rule 0)	K	0.20
Move cursor to end of phrase (no M by Heuristic Rule 1)	P	1.10
Shift-click mouse button (one average typing K)	K	0.28
(one mouse button click K)	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu (no M by Heuristic Rule 1)	P	1.10
Press mouse button	K	0.10
Move cursor to Cut menu item (no M by Heuristic Rule 1)	P	1.10
Release mouse button	K	0.10
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to insertion point	P	1.10
Click mouse button	K	0.20
Mentally prepare by Heuristic Rule 0	M	1.35
Move cursor to Edit menu (no M by Heuristic Rule 1)	P	1.10
Press mouse button	K	0.10
Move cursor to Paste menu item (no M by Heuristic Rule 1)	P	1.10
Release mouse button	K	0.10
TOTAL PREDICTED TIME		14.38

Рисунок 1. KLM-модель задачи перемещения фрагмента текста

В основе KLM лежит простая когнитивная архитектура: последовательность этапов обработки информации, в котором каждое действие последовательно совершается пока задача не будет выполнена. Считается, что все действия для выполнения задачи содержатся в определенном ранее наборе примитивных операторов, все когнитивные действия заключены в операторе ментальной подготовки. Так модель накладывает ограничения: выполнение задачи аппроксимируется последовательностью фиксированных действий, где нет параллельных действий, прерываний или чередований разных целей.

Card, Moran, and Newell GOMS (CMN-GOMS) имеет строгую иерархию целей. Методы в ней представляются в неформальной программной форме и могут включать в себя подметоды и условия. Определенная CMN-GOMS модель может предсказывать и время исполнения, и последовательность операторов. Авторы описали модель не с помощью инструкции о том «как построить CMN-GOMS модель», а при помощи нескольких содержательных наглядных примеров (в том числе и на примере редактирования текста), чтобы

далее каждый мог разработать свою собственную CMN-GOMS модель по аналогии. CMN-GOMS модель основана на традиционной модели обработке информации человеком с возможностью параллельной обработки (MHP, Model Human Processor), а именно на двух основных ее принципах: принцип пространства задачи (Problem Space Principle) и принцип рациональности (Rationality Principle). Принцип пространства постулирует, что действия пользователя можно характеризовать как применение последовательности операторов для перевода системы из начального состояния в состояние достигнутой цели. Последовательность операторов, называемая методом, может повторяться неоднократно и в дальнейшем использоваться для достижения похожих целей. Второй принцип (Rationality Principle) говорит, что пользователь будет разрабатывать методы, которые эффективны в терминах текущего интерфейса с учетом человеческих возможностей обработки информации. Поскольку считается, что человек старается быть эффективным сам по себе, выбранные методы сильно зависят от интерфейса системы. Таким образом, при построении GOMS-модели по задаче можно предсказывать важные свойства человеко-компьютерного взаимодействия на основе дизайна системы (рисунок 2).

Недостаток рассматриваемой модели по сравнению KLM состоит в том, что текущая модель не имеет строгой схемы построения, поэтому в ней есть большая доля неопределенности.

Natural GOMS Language (NGOMSL) - это структурированная нотация на естественном языке для представления GOMS-модели и процедура для ее построения. NGOMSL-модель строится в программной форме и предсказывает последовательность операторов, время выполнения задачи и время для изучения методов. Исследователь строит NGOMSL модель сверху вниз, обходом в ширину, начиная с верхних целей, раскрывает их в методы до того, как методы будут содержать только примитивные операторы, обычно операторы «уровня нажатия клавиш». Как и CMN-GOMS модель, NGOMSL модель имеет явную иерархическую структуру целей, поэтому с помощью нее можно описывать большие задачи, такие как совместное написание статьи.

В основе NGOMSL модели лежит представление методов в терминах когнитивной архитектуры, называемой когнитивной теорией сложности (cognitive complexity theory, CCT) [6]. CCT подразумевает собой простую архитектуру последовательных стадий, где оперативная память порождает правило, применяемое к фиксированному состоянию. Эти правила изменяют содержание оперативной памяти или исполняют примитивные внешние операторы, как например нажатие на клавишу. GOMS-методы представлены набором порождающих правил в определенном формате. Изучение всей процедуры состоит из изучения отдельных порождающих правил. Изучение

одной задачи переходит к изучению другой, если текущее правило уже изучено. CCT предоставляет предсказывание и времени выполнения, времени изучения и время перехода между изучениями отдельных правил.

Рисунок 2. CMN-модель задачи перемещения фрагмента текста

NGOMSL изначально определен как высокоуровневая нотация, описывающая содержание CCT-модели. В этой модели методы представлены в программной форме в виде последовательности шагов, которые содержат операторы (операторы уровня нажатия клавиш и внутренние операторы, которые представляют операторы архитектурного механизма CCT, например, добавление и удаление из оперативной памяти некоторой информации или установление подцелей). NGOMSL и CCT модели имеют прямую связь, причем имеется взаимно однозначное отображение одной в другую (рисунок 3).

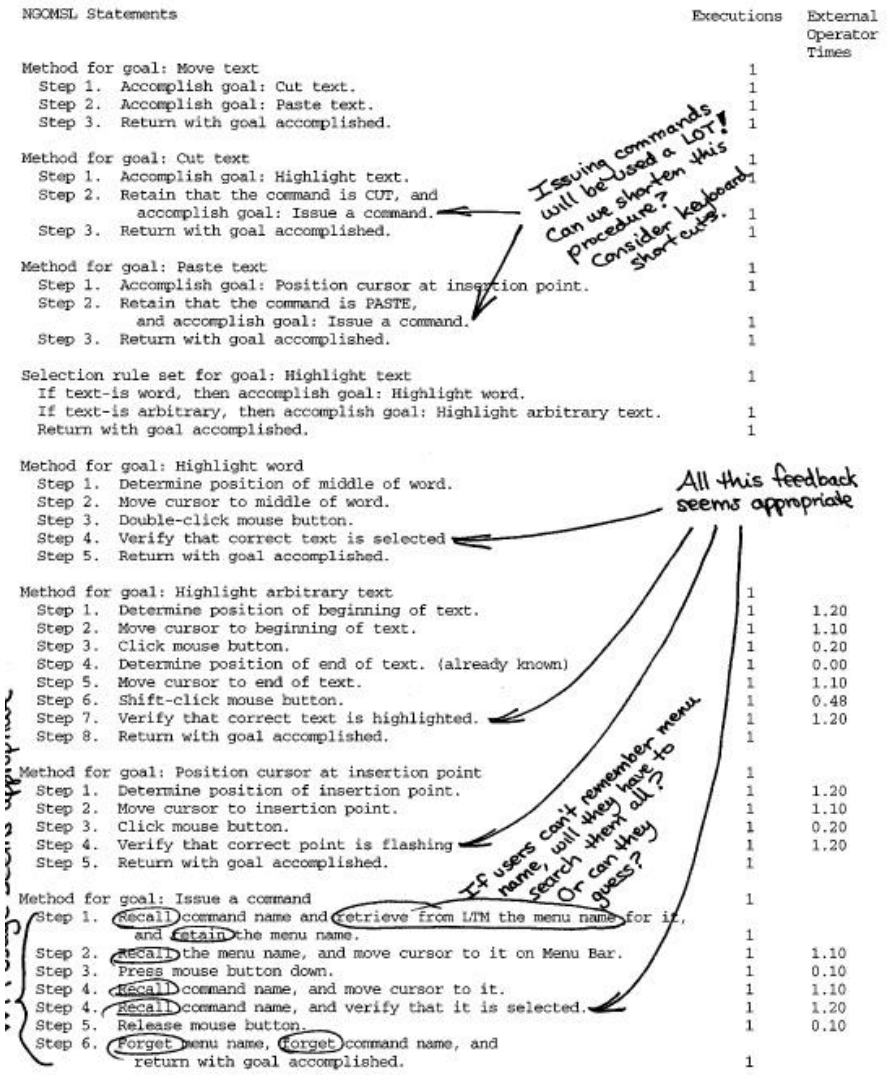


Рисунок 3. NGOMSL задачи перемещения фрагмента текста

Хотя NGOMSL анализ предоставляет многоуровневое описание задачи, тем не менее время на обучение и исполнение методов может быть предсказано, если используемые операторы уже известны пользователю. Таким образом, для уровня примитивных операторов (уровень нажатия клавиш) время может быть предсказано наиболее точно, в то время как для более высоких уровней остается много неопределенностей. CCT модель накладывает ограничение на описание задач в терминах NGOMSL модели: методы могут иметь иерархическую или последовательную структуру, т.е. произвольный порядок, прерывание или повтор действий не допускаются. Также нет возможности предсказывать время выполнения и изучения методов в случае, когда перцепционные, когнитивные и моторные действия частично перекрываются. Например, когда пользователь читает сообщение на экране (перцепционная обработка информации), при этом переносит руку на клавиатуру и извлекает информацию из своей долговременной памяти.

Cognitive-Perceptual-Motor GOMS (CPM-GOMS), как и другие GOMS-модели предсказывает время выполнения задачи на основе анализа

компонентов действий. Однако для CPM-GOMS модели требуется особый уровень анализа, на котором примитивные операторы — это когнитивные, перцепционные и моторные действия. В отличие от рассмотренных ранее вариантов GOMS-модели, модель не предполагает, что операторы обязаны оформляться в виде последовательности, если задача требует параллельного представления когнитивных, перцепционных и моторных действий, то в CPM-GOMS модели это допускается. CPM-GOMS модель представляется в виде PERT-диаграммы[1], где показаны операторы и зависимости между ними.

CPM-GOMS модель непосредственно основана на MHP (традиционной модели обработки информации человеком) [7]. Сначала человек получает сенсорную информацию, которая затем распознается и отправляется в кратковременную память с помощью перцепционных обработчиков информации, а после когнитивный обработчик преобразует полученную информацию и передает моторному обработчику сведения, какие физические действия необходимо совершить. Каждый обработчик совершает действия периодами и параллельно друг другу. Важной ценностью модели стало представление обработки информации человеком с помощью инженерной модели, способной приближенно описать многие особенности человеко-компьютерного взаимодействия. Метод CPM-GOMS модели состоит в прямом применении MHP к анализу задачи путем определения операторов для каждого из обработчиков и определения зависимостей между ними. Именно параллелизм в описании задачи дает преимущество этой модели перед другими (появляется больше детализации).

Чтобы создать CPM-GOMS модель, нужно сначала построить CMN-GOMS модель задачи, которая начинается на любом уровне абстракции, однако заканчивается на так называемом уровне действия, где используются преимущественно перцепционные (например, чтение с экрана) и моторные действия (например, ввод команды). Далее команды этого уровня представляются в виде команд более низкого уровня, действия становятся целями, которые можно достигнуть с помощью операторов MHP-уровня (рисунок 4). У всех операторов есть оценка времени выполнения и ряд оценок, связанных с разными условиями задачи.

[1] Pert-диаграмма, URL: http://bourabai.ru/einf/pert.htm

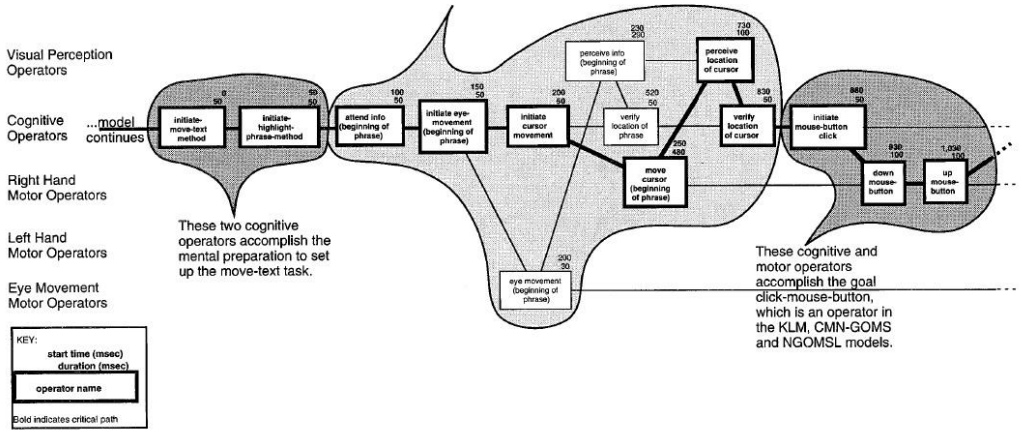


Рисунок 4. Часть CPM-GOMS модели задачи перемещения фрагмента текста

Обобщим особенности рассмотренных моделей. KLM-модель проста в применении, однако предсказывает только время выполнения задачи и только для определенных методов. CPM-GOMS модель предсказывает время для детально определенных, параллельно перекрывающихся действий, однако также для ограниченного количества методов (задач с определенными условиями). CMN-GOMS модель предсказывает время для выполнения параметризованных задач, а NGOMSL модель в дополнение еще предсказывает некоторые аспекты времени обучения методам решения задач. Все особенности моделей связаны с лежащей в их основе архитектурой обработки информации человеком.

XDM

XDM (context-sensitive dialogue modelling) – это визуальный формализм и инструмент для проектирования человеко-компьютерного взаимодействия в системе [8]. XDM соединяет в себе расширенные сети Петри[2] с теорией KLM операторов для описания статических и динамических аспектов взаимодействия в любом контексте, в котором системе нужно работать и делать изменения в интерфейсе проще или автоматически. Метод успешно применялся в нескольких исследовательских проектах, связанных с проектированием информационных систем для медицинских учреждений.

Посмотрим, как применяются сети Петри. В этом случае состояния — это отображаемая информация, переходы — действия пользователя, которые приводят к изменению отображаемого статуса. Путь в сети Петри — последовательность элементарных задач, с помощью которых может быть сделана более сложная задача, дополнительно путь показывает изменение отображаемой информации по ходу выполнения задачи. Выбор между альтернативными путями выполнения представляется в сети Петри переходами, которые имеют одинаковые премножества, то есть несколько

[2] Расширенные сети Петри, URL: http://it.kgsu.ru/SetiPetri/sp017.html

переходов можно запустить одновременно. Альтернативные пути заканчиваются единственным состоянием. Визуальный параллелизм на диаграмме обозначается включением нескольких состояний в постмножество перехода, то есть несколько состояний можно отметить после запуска перехода одновременно. Одновременно активные пути, которые можно выполнить в любом порядке, начинаются из этих состояний. Синхронизация в терминологии этих последовательностей достигается завершением пути с помощью одного перехода. Подзадачи обозначаются с помощью вложенных состояний сети Петри.

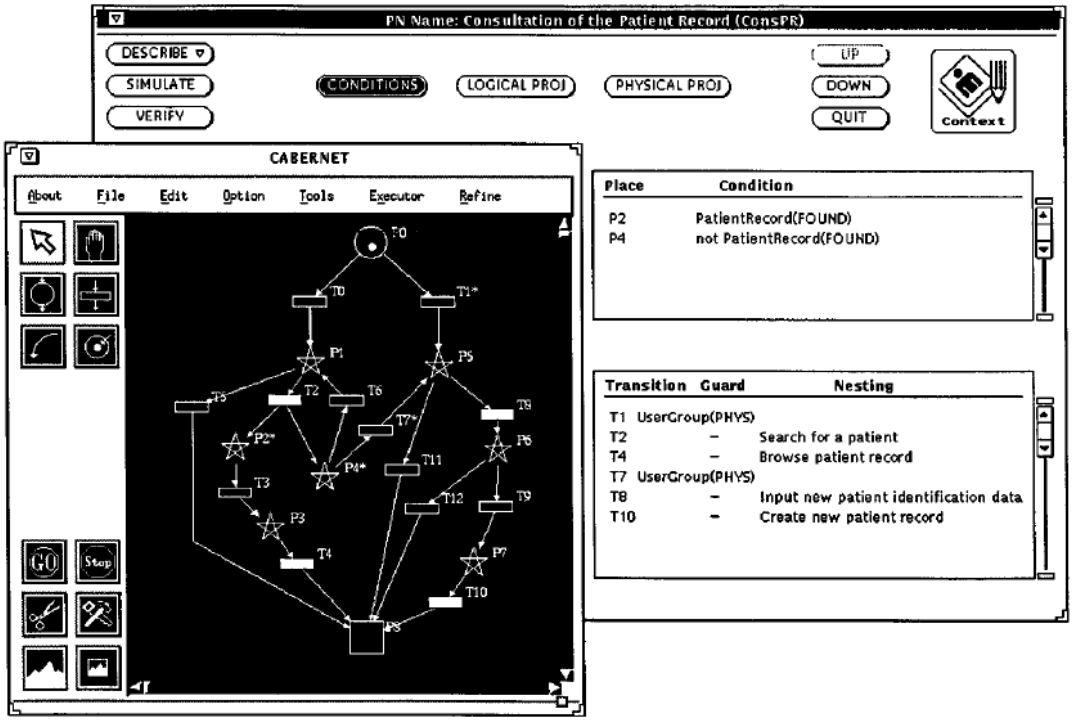


Рисунок 5. XDM: сеть Петри для задачи обновления карточки пациента

Каждая построенная сеть Петри соответствует некоторой задаче. Например, на рисунке 5 переходы T2, T4, T8, T10 вложенные. В нижнем правом углу на примере выше можно видеть названия сетей Петри нижнего уровня. Это обозначает, что задачу «обновление карточки пациента» можно разбить на следующие подзадачи: «поиск пациента», «просмотр карточки пациента», «ввод новых данных о пациенте», «создание новой карточки пациента». Первые две задачи идут последовательно, две оставшиеся — это альтернативные варианты. Также на диаграмме можно видеть некоторые узлы, обозначенные звездами. Это так называемые макросостояния, то есть комбинация элементарных состояний.

Сети Петри позволяют описывать параллельные и асинхронные события, которые часто появляются в пользовательских интерфейсах, а также определять изменение состояний системы как последовательность действий

пользователя. Кроме того, цветные сети Петри дают возможность описывать поведение системы в различных контекстах. Однако чистые сети Петри не могут полностью описать все аспекты взаимодействия, к примеру, как появляется информация о состоянии системы в каждой фазе взаимодействия, какие действия пользователь может совершить в данном случае, какие физические аспекты интерфейса связаны с выбранными методами действий пользователя. В этой ситуации удобно расширить сети Петри этими «статическими» аспектами интерфейса с помощью гипертекста: логические и физические проекции отображаемой информации и действия пользователя, связанные с состоянием диаграммы и переходом. Эти же проекции могут использоваться для описания состояний системы и действий пользователя в интерфейсе произвольной системы.

XDM — инструмент, позволяющий проектировщику формально описывать интерфейс с помощью последовательности шагов

- Описать динамику диалогов с помощью сети Петри.
- Добавить переходы в сеть Петри для воспроизведения иерархии задач.
- Описать статические аспекты взаимодействия (состояния системы и действия пользователя) с помощью логических и физических проекций на состояния диаграммы и переходы.
- Добавить условия на состояния диаграммы и переходы и цвета к разметкам, чтобы реализовать адаптивность интерфейса к контексту.

Эти шаги могут выполняться не строго последовательно. XDM описывает дизайн системы путем нескольких итераций из последовательностей шагов.

Статус отображения на каждой фазе взаимодействия и задачи пользователя можно описать следующими стадиями:

- На первом шаге (логический уровень) проектировщик системы описывает, что может делать пользователь и что система выдает в ответ, т. е задачи, которые может выполнять пользователь в соответствии с каждым переходом, и информацию, которая будет отображаться на экране в соответствии с каждым состоянием.
- На втором шаге (физический уровень) проектировщик определяет, как задачи пользователя и отображения системы реализуются, т.е. определяет действия пользователя, которые соответствуют описанным задачам и расположение отображаемой информации на экране.

Для оценки сложности интерфейса авторы использовали KLM-операторы GOMS-модели. Сложность интерфейса определен как количество,

содержимое и структура информации, требуемой для успешного взаимодействия с системой. Сложность действий и сложность отображения измеряется отдельно. Сложность последовательности действий можно измерить с помощью функции от длины строки, представляющей последовательность переходов, которые ведут от начального к конечному состоянию в сети Петри, представляющей задачу. Если несколько альтернативных путей доступны для достижения конечного состояния в данном контексте, то можно отдельно сравнить их сложность. Можно также сравнивать сложность выполнения одних и тех же задач в разных контекстах, например, для разных типов пользователей. Это сравнение можно использовать для проверки, что для менее опытных пользователей выполнение задач сложнее, чем для профессионалов, а также выявлять ситуации, когда скорость выполнения задач различается значительно. Однако элементарные действия, предназначенные для выполнения простых задач могут иметь разную сложность. Например, выбор нужной кнопки проще, чем заполнение текстового поля n символами. Методы взаимодействия XDM определены как регулярные выражения на языке KLM-операторов. Сложность методов взаимодействия можно оценить с помощью функции от их ожидаемого времени выполнения, от выражения из KLM-операторов, для которых время задано изначально.

Ключевая особенность XDM – это моделирования поведения в зависимости от роли пользователя в системе. Так же инструмент применим исключительно на стадии проектирования интерфейса.

Angel-Demon Games

Авторы подхода Angel-Demon Games [9] разработали абстрактные формальные модели рационального поведения как игры между двумя оппонентами. Интуитивно Ангел абстрактно представляет планируемые аспекты поведения, тогда как Демон - реактивное поведение пользователя. Формализация проводится с помощью фреймворка Mocha[3].

Подход основан на модели пользователя, которая формализует общие когнитивные принципы, например, пользователь входит в систему со знанием задачи, которую ему нужно выполнить, при этом может произвольно выбрать между подходящими возможными действиями для выполнения этой задачи. В случае успешного выполнения задачи подход ограничивается предположением, что недетерминизм в нашей текущей модели пользователя может разрешиться «демонически», т.е. наихудшим способом. Однако такая модель не покрывает более продвинутые аспекты поведения пользователя

[3] Mocha Framework, URL: http://visionmedia.github.io/mocha/

(например, поведение эксперта в использовании системы). Рациональное поведение пользователя включает в себя и планируемое, и реактивное поведение. Общая когнитивная архитектура построена как коллекция агентов, где каждый агент представляет определенный аспект поведения пользователя: реактивный агент, планируемый агент и агент режима. В такой архитектуре взаимодействие моделируется как игра между объединяющимися и/или конкурирующими агентами. Для проверки все агенты делятся на две коалиции: наши агенты представляют ангельские аспекты поведения, другие агенты представляют демонические аспекты. Принадлежность агентов коалиции может зависеть от типа пользователя или от определенных сценариев. Например, если пользователь — эксперт в работе с системой, то разумно предположить, что агент, делающий выбор между реактивным и планируемым режимом поведения относится к «ангельским» агентам, поскольку такие пользователи способны планировать свои действия для выполнения задачи. С другой стороны, новые пользователи могут полагаться на реактивное поведение, поэтому агент в этом случае скорее будет относится к «демоническим».

```
module decision
 external fin : Finished; beh : Behaviour
 interface rel : Relevant; plan : (0..$NACTS)

#foreach j = (1..$NACTS)
atom relevance_$j
  controls rel[$j]
  reads fin, rel[$j]
init
[] true -> rel'[$j] := true
update
[] fin = run & rel[$j] -> rel'[$j] := nondet
endatom
#endforeach

atom planning
 controls plan, rel[0]
 reads fin, plan
 awaits beh
init
[] true -> plan' := 0
update
#foreach k = PREDICTABLE
[] fin = run & beh' = planned -> plan' := $k
#endforeach
[] true -> plan' := 0
endatom
endmodule
```

Рисунок 6. Шаблон агента принятия решения

Для того, чтобы комбинировать и демонический, и ангельский недетерминизм в модели пользователя, используется Mocha Framework. Mocha не поддерживает высокоуровневые спецификации, поэтому когнитивная модель описывается как общий шаблон, который в дальнейшем можно расширить/изменить для получения конкретных моделей пользователя. С помощью Mocha авторы формально определяют три агента: режима, принятия решения и действия.

Агент принятия решения (рисунок 6) определяется таким образом, что недетерминизм разрешается ангельским способом, в то время как агент действия определяется в предположении, что недетерминизм разрешается демонически. С другой стороны, никаких точных предположений о поведении недетерминизма в агенте режима не дается. Более безопасная позиция — иметь слабые предположения о способностях пользователя, поэтому агент режима ассоциируем с демоническим поведением. В этом случае игра займет место между агентом принятия решения с одной стороны и агентом режима и действия с противоположной стороны. Для каждой конкретной задачи и каждого типа пользователя проводится параметризация агентов и назначается способ разрешения недетерминизма. В результате такой подход позволяет выявить проблемы в интерфейсе системы с помощью проверки модели: если пользователь не смог достичь цели в заданных предположениях, значит, интерфейс содержит ошибки. Недостатком такого подхода можно считать сложность в описании каждого аспекта поведения для больших задач: для каждого возможного действия в системе необходимо задать несколько параметров и ограничительные правила для из использования.

Остальные рассмотренные в процессе изучения методы [14], [15], [16], [17], [18] представляют собой синтез нескольких особенностей из перечисленных ранее подходов, и мы не будем останавливаться на них подробно.

Выводы

По результатам исследования существующих подходов можно сделать следующие выводы:

- GOMS-модель позволяет подробно определять сценарии поведения пользователя как с помощью иерархий, так и последовательно. Такой подход хотелось бы использовать и применительно к задачам данной работы. Однако модель задает строгую последовательность действий и тем самым делает модель поведения пользователя сильно ограниченной.
- XDM – эффективный инструмент для проектирования с оценкой юзабилити интерфейса, но применять его для анализа поведения в уже

существующих системах весьма сложно (подход работает для отдельных участков интерфейса).

- Особенность Angel-Demon Games – возможность определять ошибки в интерфейсе. Авторы разделяют стадию планирования и действия в поведении пользователя. В нашем случае можно применить это разделение для классификации действий пользователя. Но предложенный авторами формализм требует трудоёмкого описания поведения пользователя.

Перейдем к рассмотрению системы, для которой мы будем создавать формальные описания сценариев.

QReal:Robots

QReal:Robots – система визуального программирования роботов, разрабатываемая на кафедре системного программирования Санкт-Петербургского государственного университета [10]. Среда позволяет создавать графические программы для роботов Lego Mindstorms NXT 2.0 и исполнять эти программы прямо на компьютере, посылая команды роботу через Bluetooth или USB-интерфейс. Кратко опишем элементы интерфейса QReal:Robots (рисунок 7).

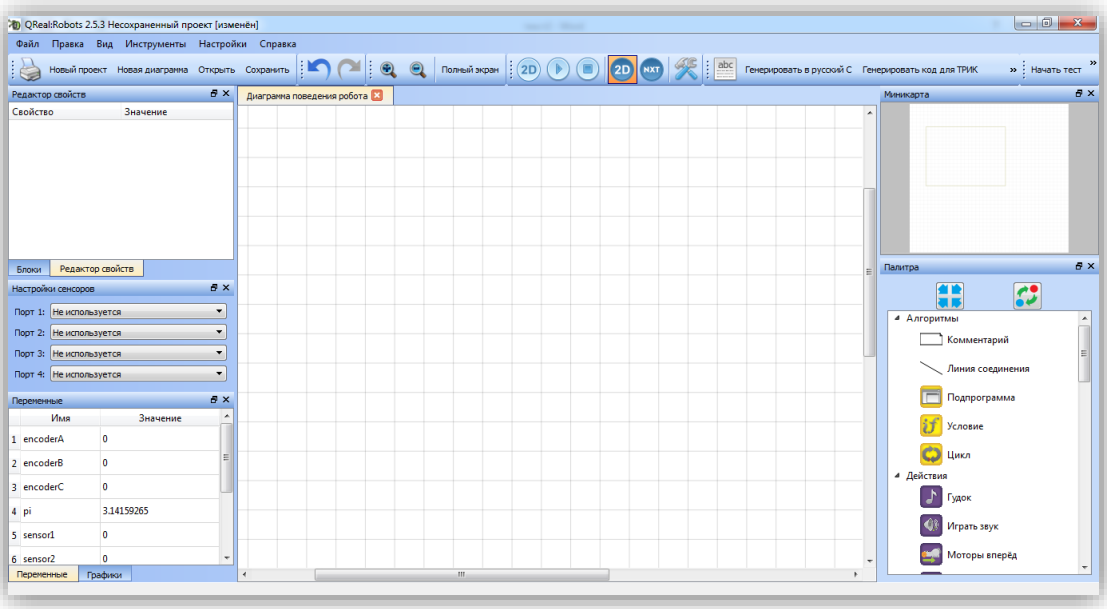


Рисунок 7. Интерфейс QReal:Robots

Главное меню содержит базовый набор операций и настроек среды.

Сцена — главная рабочая область, расположенная в центре, отображает диаграмму, над которой сейчас ведется работа, и позволяет ее редактировать. При исполнении диаграмм текущий интерпретируемый блок подсвечивается на сцене.

Миникарта — отображение сцены в уменьшенном масштабе. Миникарта располагается в правом верхнем углу окна системы.

Палитра элементов располагается справа под миникартой и содержит набор доступных элементов для создания диаграмм.

Редактор свойств расположен в левом верхнем углу и используется для редактирования значений свойств выделенного на сцене элемента.

Список переменных располагается слева под редактором свойств и отображает название и значение всех переменных, используемых во время исполнения диаграммы, а также всех зарезервированных переменных среды.

Настройки сенсоров находятся слева под списком переменных и предоставляют возможность настроить расположение сенсоров на портах робота.

В QReal:Robots существует 2 основных режима работы – запуск программ с помощью 2D-модели робота и запуск программ на реальном роботе. Оба режима работы состоят из трех основных этапов:

- создание программы управления роботом в форме диаграммы;
- настройка конфигурации модели робота (настройки портов и их расположения на роботе), а также настройка его окружения (установка внешних препятствий, линий для проезда и другое, в зависимости от задачи);
- запуск созданной программы на выбранной модели робота.

По необходимости происходит переход снова к первому или второму этапу в случае, если необходимо внести изменения или исправления в программу.

В нашем исследовании мы ограничимся рассмотрением второго этапа – этапа создания диаграмм для управления роботом.

Глава 2. Анализ действий пользователя в среде QReal:Robots

Каждая диаграмма в QReal:Robots представляет собой последовательность блоков, соединенных связями (рисунок 8). Диаграмма начинается блоком «Начало» и завершается блоком «Конец» (блок «Начало» появляется на сцене по умолчанию при создании новой диаграммы).

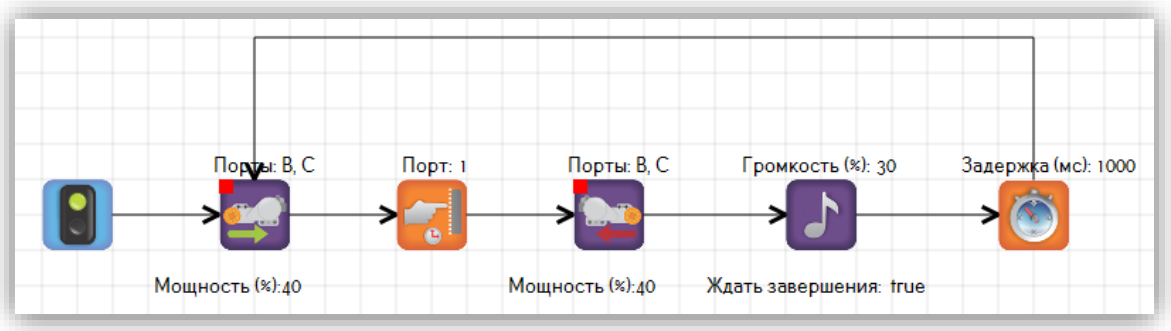


Рисунок 8. Пример диаграммы в QReal:Robots

Создание диаграммы управления роботом включает в себя следующие действия:

- создание элемента на диаграмме (блока или связи между блоками);
- действия с элементом диаграммы на сцене (соединение блоков с помощью связей, перемещение элементов по сцене, удаление и другие);
- изменение свойств элементов;
- действия с диаграммой управления роботом (создание новой диаграммы, сохранение диаграммы и другое).

Рассмотрим подробнее каждое из этих действий для того, чтобы определить базовые действия – атомарные действия, которые в дальнейшем будут использоваться в модели поведения пользователя. Базовые действия неделимы, то есть находятся на нижнем уровне детализации описания.

Создание элемента на диаграмме

В QReal:Robots есть три основных способа создания элемента:

- создание элемента с помощью палитры;
- создание элемента с помощью «линкеров»;
- создание элемента с помощью мышиных жестов.

Создание элемента с помощью палитры – наиболее распространенный способ создания элементов в средствах визуального программирования. Этот способ интуитивно понятен и не требует предварительного изучения

документации для его использования. В этом способе можно выделить два основных шага: поиск элемента в палитре и выбор и перенос элемента из палитры на сцену.

Рассмотрим всевозможные действия с палитрой (рисунок 9), а затем определим, к какому шагу относится каждое из определенных действий.

В стандартном режиме в палитре расположены строки с элементами, каждая из которых представляет собой иконку элемента и его название. Все элементы разбиты на категории. Каждую категорию можно свернуть и развернуть. В верхней части палитры над основной областью находятся две кнопки: кнопка для смены режима отображения элементов в палитре (во втором режиме иконки элементов располагаются по две в одной строке без соответствующих названий) и кнопка для скрытия/раскрытия содержимого всех категорий элементов. Все элементы палитры невозможно сразу просмотреть, для полного просмотра элементов используется полоса прокрутки.

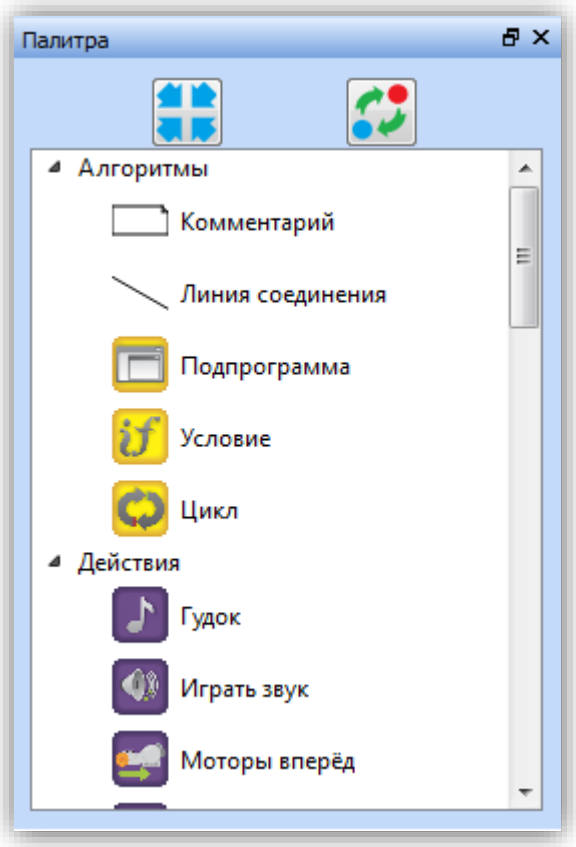


Рисунок 9. Палитра QReal:Robots

Для поиска элемента в палитре пользователь может выполнить следующие действия:

- прокрутить палитру вверх/вниз;

- навести на элемент в палитре/убрать наведение с элемента в палитре;
- нажать на кнопки над палитрой (сменить режим отображения или свернуть/развернуть содержимое всех категорий);
- скрыть/раскрыть содержимое категории элементов.

Чтобы выбранный элемент появился на сцене, его необходимо зажать левой кнопкой мыши и перенести на нужную позицию на сцене, для этого пользователю необходимо выполнить следующие действия:

- навести на элемент в палитре;
- нажать на элемент в палитре правой кнопкой мыши;
- убрать наведение с элемента в палитре;
- ввести элемент на сцену;
- перемещать элемент по сцене;
- бросить элемент в выбранную позицию на сцене.

Отметим сразу, что все определенные выше действия необходимы для достижения цели «Выбор и перенос элемента из палитры на сцену» и обязательно должны следовать в заданном порядке.

Следующие два способа создания элементов не так широко распространены в инструментах визуального программирования и специфичны для QReal:Robots.

Перейдем к созданию элементов с помощью так называемых «линкеров» (рисунок 10). Линкеры – специальные элементы графического интерфейса, ассоциированные с каждым блоком и позволяющие создавать следующий блок для текущего. Линкер появляется справа от блока при его выделении. Зажав правую кнопку мыши, из линкера блока можно «вытянуть» связь. При отпускании правой кнопки мыши происходит следующее: если курсор мыши при отпускании находится на каком-либо блоке диаграммы, связь соединяет исходный и выбранный блок. В противном случае появляется контекстное меню, в котором можно выбрать следующий элемент, который пользователь хочет создать и соединить с исходным (направление связи – от исходного к созданному). Отметим, что для этого метода необходимо, чтобы первый элемент уже находился на сцене (среда позволяет использовать исключительно этот метод создания элементов, блок «Начало» по умолчанию находится на новой диаграмме при ее создании), а также этот механизм удобен в случае, если пользователь сразу может определить, в какой последовательности ему необходимо создавать элементы.

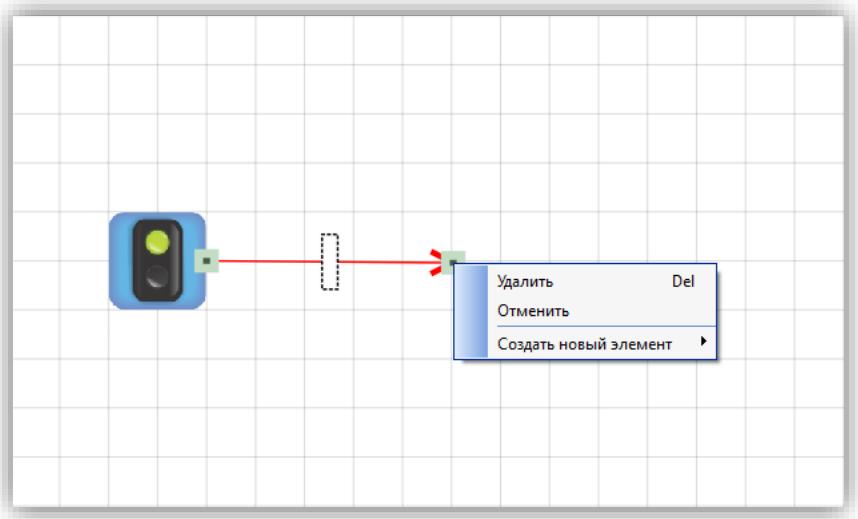


Рисунок 10. Использование линкеров при создании элемента

Действия пользователя для создания элемента с помощью линкера следующие:

- кликнуть на элемент на сцене;
- навести курсор мыши на линкер у выделенного элемента;
- нажать на линкер левой кнопкой мыши;
- вытянуть связь из линкера с зажатой левой кнопкой мыши;
- отпустить левую кнопку мыши;
- навести курсор на пункт контекстного меню с названием элемента;
- кликнуть на пункт контекстного меню с названием элемента левой кнопкой мыши.

Снова все перечисленные действия необходимо выполнить в указанном порядке для достижения цели «Создать элемент» с помощью линкера.

Наконец, рассмотрим создание элементов с помощью мышиных жестов (рисунок 11). С точки зрения разработчиков такой способ создания элементов наиболее быстрый и удобный, однако этот способ требует дополнительной подготовки и практики. Жесты мышью – это определенные траектории, нарисованные курсором с зажатой правой кнопкой мыши. Каждому элементу языка программирования роботов соответствует идеальный жест, с которым сравнивается жест, нарисованный пользователем правой кнопкой мыши и по результатам сравнения на диаграмме создается распознанный элемент. Все доступные идеальные жесты мышью можно открыть и посмотреть в отдельной вкладке:

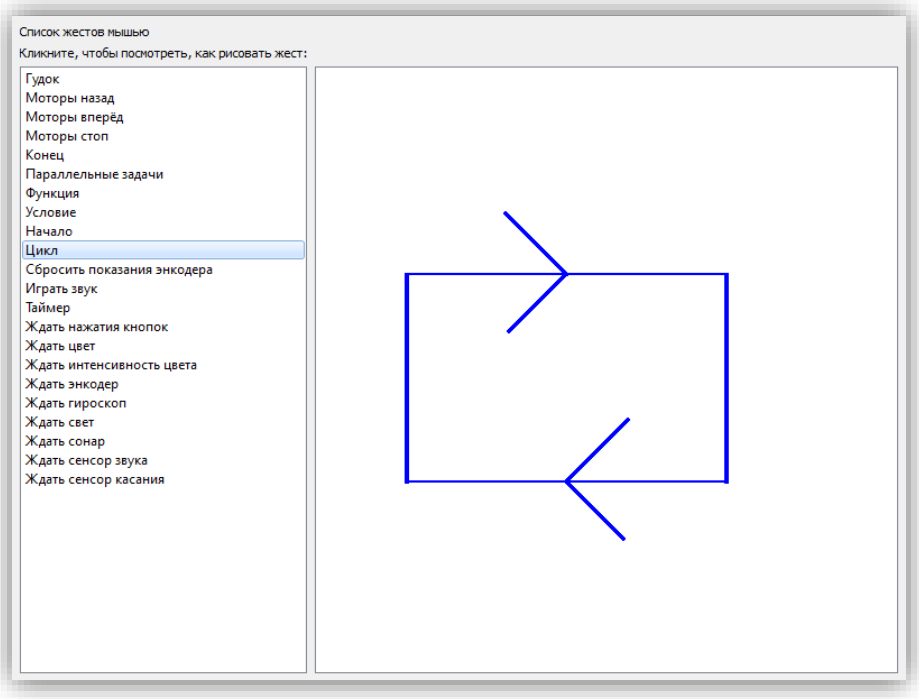


Рисунок 11. Список доступных жестов мышью в QReal:Robots

Действия пользователя для создания элемента с помощью мышиных жестов весьма просты:

- нажать правой кнопки мыши;
- переместить курсор мыши с зажатой правой кнопкой;
- отпустить правую кнопку мыши.

Еще одним способом создания элементов можно считать использование копирования/вставки уже существующего на диаграмме элемента, но в рамках построения текущей модели поведения пользователя мы не будем рассматривать этот способ.

Действие с элементом диаграммы на сцене

Из действий с элементом на сцене обязательными для создания диаграммы служат соединения элементов связями. Способ соединения зависит от способа создания связи.

В случае создания связи с помощью жестов мышью соединение элементов будет необходимым сопровождающим действием, поэтому в этом случае никаких дополнительных действий не требуется. Если линия соединения создавалась с помощью линкера, то возможны следующие варианты.

- Если связь при создании была наведена на существующий элемент или был выбран элемент для создания из контекстного меню, то дополнительных действий не нужно.
- Если при создании связи с помощью линкера при появлении контекстного меню не был выбран элемент для создания, то связь осталась присоединенной началом к исходному элементу, а конец связи остался неприсоединённым. Для завершения соединения необходимо:
    - нажать на конец связи левой кнопкой мыши;
    - перенести конец связи с зажатой левой кнопкой мыши;
    - навести на элемент на сцене;
    - отпустить правую кнопку мыши на элементе на сцене;
    - убрать наведение с элемента на сцене.

Когда линия соединения создавалась с помощью палитры, необходимо присоединить и начало, и конец к нужному элементу, т.е. выполнить перечисленные выше операции для линкера для начала и конца линии соединения.

Другие действия с элементами на сцене необязательны для цели «Создание диаграммы». Среди них можно выделить перемещение элементов по сцене, которое включает в себя:

- навести на элемент на сцене;
- нажать на элемент на сцене;
- переместить курсор мыши с зажатой левой кнопкой мыши;
- отпустить левую кнопку мыши;
- убрать наведение с элемента на сцене.

Возможны также действия с горячими клавишами и контекстным меню, которые позволяют удалять, копировать, вставлять элементы на сцене.

Изменение свойств элементов

Почти у каждого блока диаграммы есть набор свойств, с помощью значений, которых его можно параметризовать. К примеру, для блока «Моторы вперед» это список портов, на которых нужно включить моторы, а также мощность моторов в процентах.

Изменить значение свойства созданного на сцене элемента можно двумя способами: редактировать соответствующую надпись на сцене или использовать редактор свойств. Отметим, что первым способом можно изменить только свойства, значения которых представляют собой редактируемое поле. Если для изменения значения свойства используется, к примеру, выпадающий список, то его можно изменить только в редакторе свойств.

Чтобы изменить значение свойства элемента на сцене, пользователю следует выполнить следующее:

- сделать двойной клик левой кнопкой мыши по нужной надписи на сцене;
- ввести новое значение с помощью ввода с клавиатуры или отредактировать предыдущее, установив курсор в нужное место и используя клавиши клавиатуры Right, Left, Delete и Backspace;
- нажать Enter для сохранения или кликнуть левой кнопкой мыши вне надписи.

Редактор свойств отображает текущие значения свойств выделенного на сцене элемента. Его использование приводит к необходимости в следующих действиях:

- кликнуть левой кнопкой мыши по элементу;
- кликнуть левой кнопкой мыши по полю со значением свойства;
- ввести новое значение с помощью ввода с клавиатуры или отредактировать предыдущее, установив курсор в нужное место и используя клавиши Right, Left, Delete и Backspace;
- нажать Enter для сохранения или кликнуть левой кнопкой мыши вне надписи.

Действие с диаграммой управления роботом

Прежде чем создавать непосредственно диаграмму, нужно создать новый проект или открыть существующий и создать в нем новую диаграмму. Для этих действий есть соответствующие пункты в главном меню (меню «Файл»), а также на панели инструментов. По завершению создания диаграммы ее можно сохранить. Если она создавалась в рамках нового проекта, то нужно выбрать название файла для сохранения и место расположения на диске. Помимо этих действий возможны и другие, связанные с отображением диаграммы и работой над ней в целом (например, включить/выключить отображение сетки для выравнивания элементов). В работе с диаграммой можно выделить такие действия:

- навести курсор мыши на пункт меню (пункт на панели инструментов);
- кликнуть на пункт меню левой кнопкой мыши (пункт на панели инструментов);
- нажать на пункт меню левой кнопкой мыши (это относится к пунктам, связанным с переключением режима).

Классификация базовых действий пользователя

После определения базовых действий пользователя в QReal:Robots мы обозначили несколько критериев, по которым распределили все базовые

действия по нескольким категориям. Такое распределение позволяет проанализировать поведение пользователя на более высоком уровне абстракции и определить участки взаимодействия, которые вызывают затруднения у пользователя.

Первый критерий – блок интерфейса, с которым взаимодействует пользователь. В результате анализа процесса создания диаграмм было обозначено четыре основных блока: палитра, сцена, редактор свойств и меню (таблица 2). Остальные блоки интерфейса не задействуются непосредственно при рисовании диаграмм. Таким образом, все базовые действия были распределены на четыре категории: действия с палитрой, действия со сценой, действия с редактором свойств и действия с меню (главное меню и панель инструментов).

Второй критерий – тип задачи пользователя. Можно определить две основные задачи: подготовка к действию и совершение действия. Подготовкой можно считать такие действия, как «Навести курсор мыши на элемент» или «Прокрутить палитру», то есть те действия, которые не приводят непосредственно к созданию элемента или изменения его свойств. Остальные действия можно отнести к категории «Совершить действие». В этом случае распределения базовых действий можно рассматривать соотношение времени, затраченное на подготовку и на действие само по себе.

БЛОК ИНТЕРФЕЙСА	**БАЗОВЫЕ ДЕЙСТВИЯ**
ПАЛИТРА	Прокрутить палитру Нажать на кнопку в палитре Навести на элемент в палитре Убрать наведение с элемента в палитре Нажать на элемент в палитре Ввести элемент из палитры на сцену Свернуть группу элементов в палитре Развернуть группу элементов в палитре
СЦЕНА	Бросить элемент из палитры на сцену Передвигать элемент из палитры по сцене Навести на элемент на сцене Убрать наведение с элемента на сцене Нажать на надпись на сцене Переместить элемент по сцене Кликнуть на элемент на сцене Нажать на линкер у элемента на сцене Вытягивать линк из элемента на сцене Отпустить элемент на сцене Выбрать из контекстного меню Нажать горячую клавишу на сцене Начать рисовать жест мышью Продолжить рисовать жест мышью

	Закончить рисовать жест мышью Нажать клавишу на надписи на сцене
РЕДАКТОР СВОЙСТВ	Изменить значение свойства элемента в редакторе свойств
МЕНЮ	Нажать на пункт меню Кликнуть на пункт меню Навести на пункт меню

Таблица 2. Результат классификации базовых действий в QReal:Robots

Глава 3. Сбор информации о действиях пользователя

Для сбора информации о базовых действия пользователя в QReal:Robots был реализован специальный модуль, который обеспечивает сохранение данных: названия действия, его параметров и длительность выполнения в миллисекундах. Информация по умолчанию фиксируется в течение каждой сессии работы с системой, записывается в отдельный файл и сохраняется на диске по завершению сессии.

Среда QReal:Robots создана на основе DSM-платформы QReal. Архитектура QReal модульная, в ее состав входит абстрактное ядро, реализующее абстрактную функциональность предоставляемых платформой инструментов (создание проектов, диаграмм, элементы меню, панели инструментов, действия с элементами на диаграммах и т.д.), а специфика конкретных языков определяется в подключаемых модулях (например, вид и свойства элементов конкретного языка).

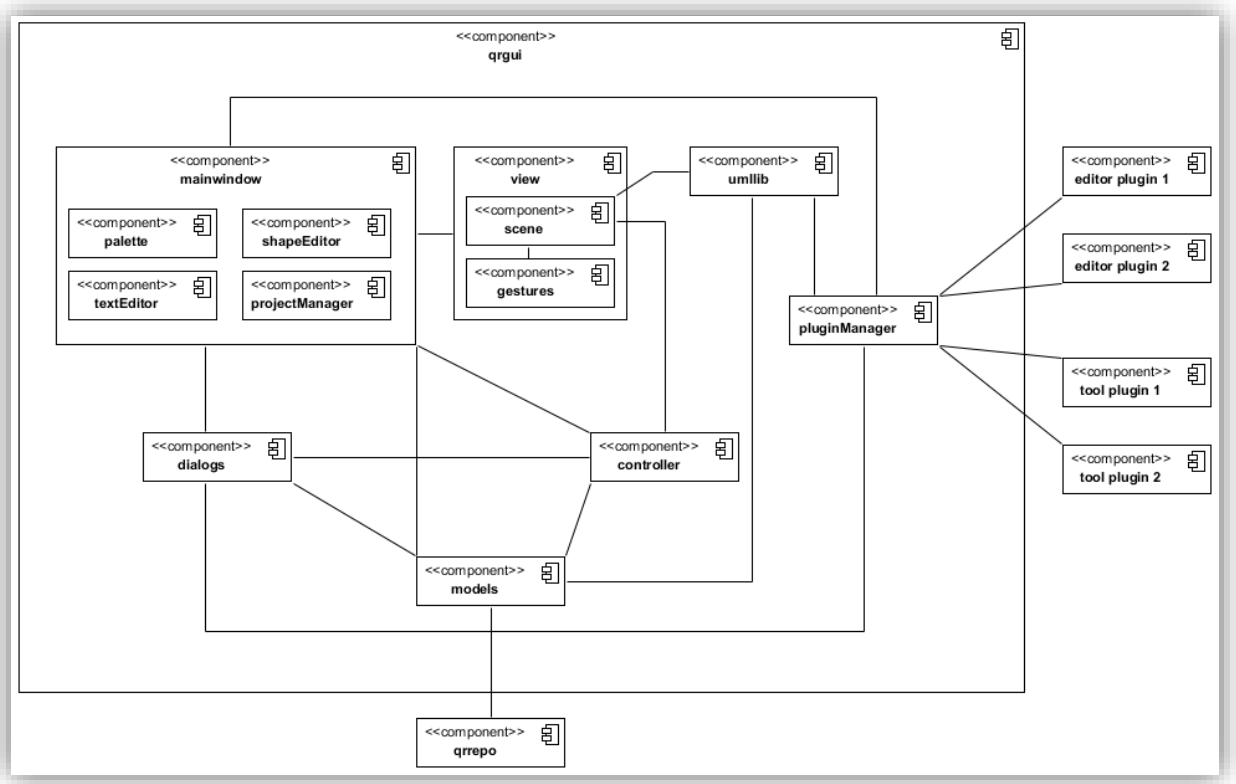


Рисунок 12. Архитектура QReal

На рисунке 12 представлена общая архитектура любого DSM-решения, созданного на базе QReal. На основе анализа действий мы определили компоненты, которые осуществляют обработку интересующих нас действий пользователя. Основной графический интерфейс реализован в компоненте mainwindow. В этой компоненте был реализован сбор действий пользователя с палитрой, с меню и панелью инструментов. Действия с элементами на сцене

распределены между компонентой view, которая реализует функциональность сцены, и компонентой umllib, реализующей функциональность абстрактного элемента на сцене.

Глава 4. Формальная модель описания сценариев поведения пользователя

Формальную модель мы будем использовать для описания так называемого «идеального» поведения пользователя – то есть модель содержит только необходимые для достижения цели действия.

Рассмотрим построение формальной модели на примере создания простой программы: робот издает звуковой сигнал на максимальной громкости. Неформально создание диаграммы (1) можно описать так: создать блоки начало, гудок, конец, соединить их связями и у блока гудок установить громкость на максимальную мощность (рисунок 13).

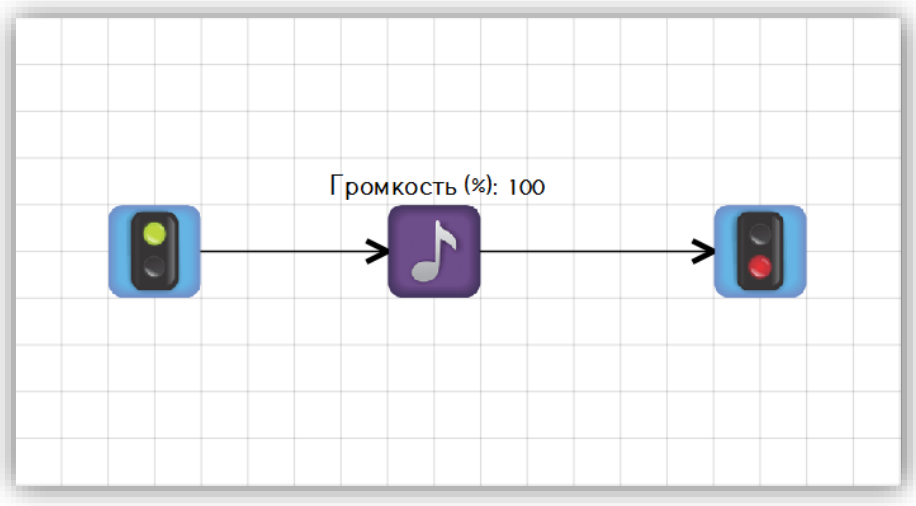


Рисунок 13. Простейшая диаграмма в QReal:Robots

Неважно, в каком порядке создаются элементы, в каком порядке соединяются связями и в какой момент будет установлено свойство, важно, чтобы в результате мы получили диаграмму, которая реализует описание программы. Каждый элемент можно создавать одним из трех перечисленных ранее способов, а для редактирования свойства есть два способа. Таким образом, формально создание диаграммы можно описать в виде набора действий так:

Создание диаграммы (1) = {создать блок «Начало», создать блок «Гудок», создать блок «Конец», создать линию соединения, создать линию соединения, соединить линией блок «Начало» и блок «Гудок», соединить линией блок «Гудок» и блок «Конец», установить свойство «Громкость (%)» блока «Гудок» в 100}.

В действительности порядок перечисленных действий не совсем произвольный: например, мы не можем установить значение свойства у блока «Гудок» до того, как его создать. Однако в этом случае мы можем

воспользоваться тем, что система не позволит совершить эти действия в недопустимом порядке и не будем дополнительно задавать в модели порядок следования этих действий.

Каждое из действий во множестве «Создание диаграммы (1)» мы можем определить так:

Создать блок «Начало» = {Создать блок «Начало» с помощью палитры} или {Создать блок «Начало» с помощью линкеров} или {Создать блок «Начало» с помощью мышиных жестов}.

Далее выбранный способ уже представляется в виде последовательности необходимых действий, которая определялась ранее в главе 1.

Глава 5. Инструмент для описания и анализа пользовательских сценариев

В рамках дипломной работы был разработан инструмент (на языке C++ с использованием инструментария Qt 5[4]), который позволяет создавать составные действия пользователя, создавать сценарии поведения, загружать информацию о реальном поведении пользователя при работе системы и анализировать ее (рисунок 14).

Основные задачи, которые помогает выполнять инструмент:

1. Определять проблемные участки взаимодействия пользователя с системой и выявлять последовательности действий, которые необходимо оптимизировать в интерфейсе.
2. Определять, какая последовательность действий пользователя приводит к ошибке в системе.
3. Выявлять участки взаимодействия, где пользователь не знает, что делать дальше.
4. Сравнивать разные пути выполнения одних задач по длительности, точности и другим показателям.

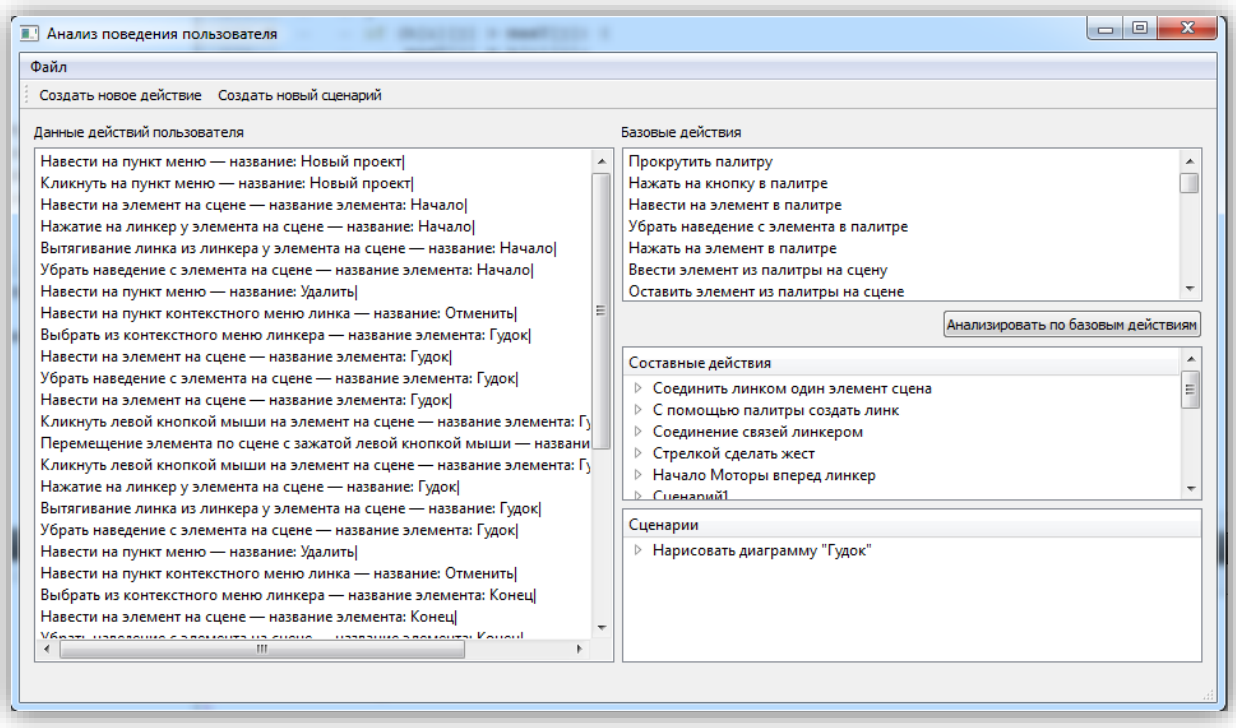


Рисунок 14. Интерфейс инструмента анализа поведения пользователя

[4] Инструментарий Qt, URL: http://qt-project.org/

Базовые действия, составные действия и сценарии

В приложении определен набор базовых действий. Информация о них хранятся в xml-формате: название действия, свойства и, если требуется, перечисляются возможные значения для свойств. Например, для базового действия «Нажать на элемент в палитре левой кнопкой мыши» свойство «Название элемента» задается набором названий всех элементов из палитры. Базовые действия – основные структурные элементы составных действий.

Составные действия – это последовательность из базовых и составных действий, которые могут быть логически объединены в одно действие. Например, можно создать составное действие «Перенос элемента из палитры на сцену» с помощью последовательности базовых действий, определенных в главе 1. При этом у базовых действий можно задать некоторые дополнительные свойства. Первое свойство – является ли действие обязательным или нет. Например, в действии «Поиск элемента в палитре» базовое действие «Прокрутить палитру» необязательно, зависит от положения полосы прокрутки и от того, где в палитре расположен нужный элемент. Второе свойство – повторяющееся действие или нет. Например, можно создать действие «Провести жест мышью» и указать, что это действие может повторяться несколько раз для создания элемента [11]. Третье свойство – длительность выполнения базового действия. Для каждого действия можно задать диапазон длительности: значение «от» и значение «до» в миллисекундах. Составные действия можно сохранить в xml-файл, и они сразу загружаются в приложение.

Сценарии – это набор из составных действий, то есть для действий, включенных в этот набор, неважен порядок выполнения. В остальном процесс задания нового сценария аналогичен процессу создания составных действий. Сценарии также хранятся в xml-формате.

Загрузка лога поведения пользователя

В основном окне приложения можно выбрать файл и загрузить действия пользователя. В процессе загрузки каждая строка лога распознается как базовое действие с заданными значениями свойств и загружается в приложение. Для действия из лога указана длительность его выполнения. У свойства «длительность» базового действия в системе помимо полей «от» и «до» есть поле «точное значение», которое и заполняется при распознавании.

Анализ лога поведения пользователя по базовым действиям

Все базовые действия разбиты по категориям в зависимости от выбранного критерия. В данном случае выбран первый предложенный критерий – блок интерфейса, с которым взаимодействует пользователь. Таким образом, имеется четыре категории: Палитра, Сцена, Редактор свойств и

Меню. Для анализа по базовым действиям генерируется таблица, в которой для каждого типа базового действия указывается, сколько раз оно встретилось в логе и какова суммарная длительность выполнения этих действий. Также генерируются графики распределения количества действий по категориям, и затраченное время на выполнение действий по категориям с помощью инструментария для создания графиков R[5].

Поиск составного действия в логе поведения пользователя

Одна из важных задач в разработке инструмента – определить, как сопоставить формальное описание и реальные действия пользователя. Формальное описание покрывает только необходимые действия для достижения цели, но пользователь может совершить и другие действия во время выполнения задачи – это могут быть ошибки (например, создание лишнего элемента на диаграмме) или нейтральные действия, которые не отдаляют, но и не приближают пользователя к цели (например, перемещение элемента по сцене). При этом в конечном итоге цель может быть достигнута. Описать всевозможные отклонения в виде дополнительной последовательности невозможно, потому что вариантов таких отклонений может быть бесконечно много. Таким образом, алгоритм сопоставления должен позволять «лишние» действия в логе пользователя, однако действия формального описания должны присутствовать обязательно, причем в заданном порядке.

Если пользователь начал выполнять задачу, ошибся, начал выполнять ее снова и успешно завершил, важно, чтобы алгоритм сопоставления нашел обнаружил первую часть совершаемого действия именно во втором случае. То есть нужно искать локально наиболее похожие на участки из формального описания действия.

Мы рассмотрели существующие алгоритмы сравнения строк и в итоге остановились на алгоритме Смита-Ватермана [12] – алгоритм динамического программирования для локального выравнивания последовательностей, который широко используется в биоинформатике. Алгоритм определяет относительные позиции строк символов локально схожих участков (рисунок 15).

[5] Язык R, URL: http://www.r-project.org/

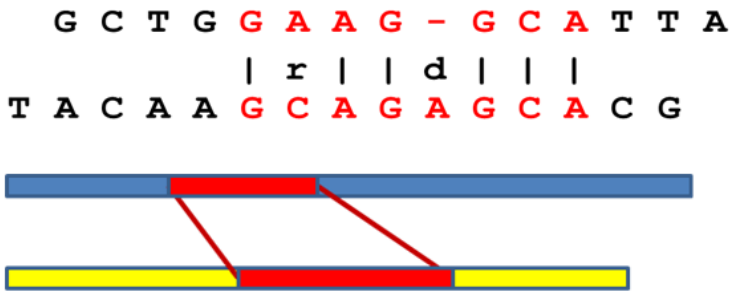


Рисунок 15. Результат локального выравнивания последовательностей с помощью алгоритма Смита-Ватермана

В алгоритме строится матрица H размера (m + 1) × (n + 1), где m – длина первой строки, а n – второй. Первый вспомогательный столбец и строка матрицы заполняются нулями, остальные элементы по следующему правилу:

$$H(i,j) = max\begin{Bmatrix} 0 \\ H(i-1,j-1) + s(a_i, b_j) \\ max_{k\geq 1}\{H(i-k,j) + W_k\} \\ max_{l\geq 1}\{H(i,j-l) + W_l\} \end{Bmatrix}, 1 \leq i \leq m, 1 \leq j \leq n$$

$a, b -$ входные строки,
$m, n -$ соответсвующие длины,
$s(a, b) -$ функция схожести строк,
$H(i, j) -$ максимальный счет сравнения суффиксов $[1..i]$ и $[1..j]$
$W_i -$ схема штрафов за несовпадения.

Локальное выравнивание работает в предположении, что последовательности совпадают в некоторых участках, но есть и различные фрагменты (рисунок 16).

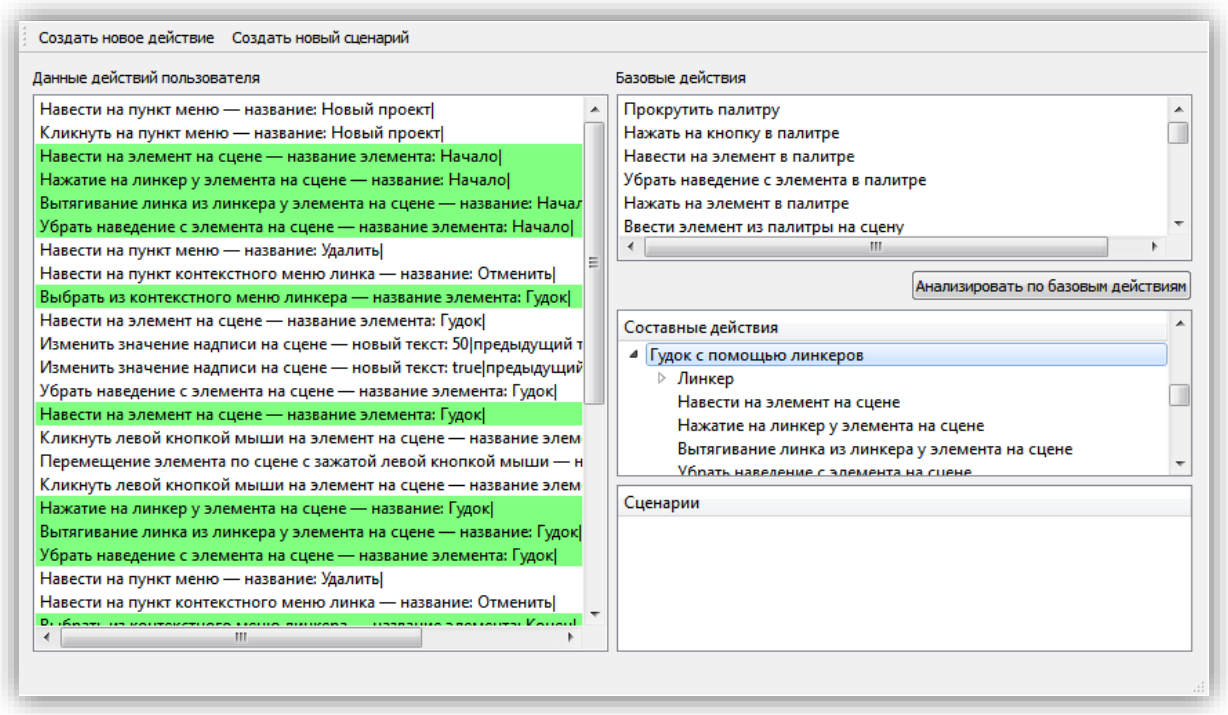


Рисунок 16. Результат сопоставления составного действия «Создание диаграммы с помощью линкеров»

Для сравнения лога поведения пользователя и составного действия мы строим матрицу, где по строкам – действия из лога, а по столбцам – действия правила. Далее с помощью функции сравнения, функции штрафов за несовпадения мы заполняем матрицу алгоритма и находим наиболее подходящее в зависимости от выбранной функции штрафов сопоставления. Мы рассматривали простейший случай и назначали одинаковый штраф всем «лишним» действиям. В дальнейшем алгоритм можно настраивать в зависимости от задач исследователя, текущий вариант алгоритма удовлетворяет требованиям, указанным ранее.

Поиск сценария в логе поведения пользователя

Сценарий – это набор действий, а не строгая последовательность, поэтому для сопоставления реальных действий пользователя и действий сценария нужно перебирать все варианты упорядочения, чего хотелось бы избежать. Для решения этой проблемы в модуле сбора информации о действиях пользователя в QReal:Robots мы добавили сбор необходимым нам откликов системы, соответствующим нашим основным действиям сценария, например, создался элемент «Гудок». Так предварительно перед сопоставлением мы конструируем сценарий в нужном порядке. Такой подход полностью решает проблему, если пользователь выполнил сценарий успешно, однако решает ее частично, если некоторых основных действий из сценария нет среди откликов системы. В этом случае возможны варианты: либо пользователь не пытался выполнить эти действия, либо он частично их

выполнил, но не достиг результата. В настоящий момент мы строим сценарий для поиска только из тех действий, которые есть в откликах системы, а остальные фрагменты анализируются отдельно.

Глава 6. Апробация

Для апробации разработанного инструмента были проведены эксперименты, цель которых – выяснить, какой способ создания элементов и соединения их связями окажется наиболее эффективным. Мы рассмотрели одну из основных групп пользователей QReal:Robots – студенты и молодые специалисты технического направления. Для того, чтобы обеспечить репрезентативность выборки в данной группе пользователей, мы выбрали 2 независимых критерия [13]: пол (мужской/женский) и опыт программирования в QReal:Robots (начинающий пользователь/опытный пользователь). В экспериментах приняли участие 24 человека, равномерно распределенные по выбранным критериям. Участникам предлагалось нарисовать одну и ту же диаграмму по изображению разными способами: с использованием палитры, с использованием линкеров, с использованием мышиных жестов (рисунок 17).

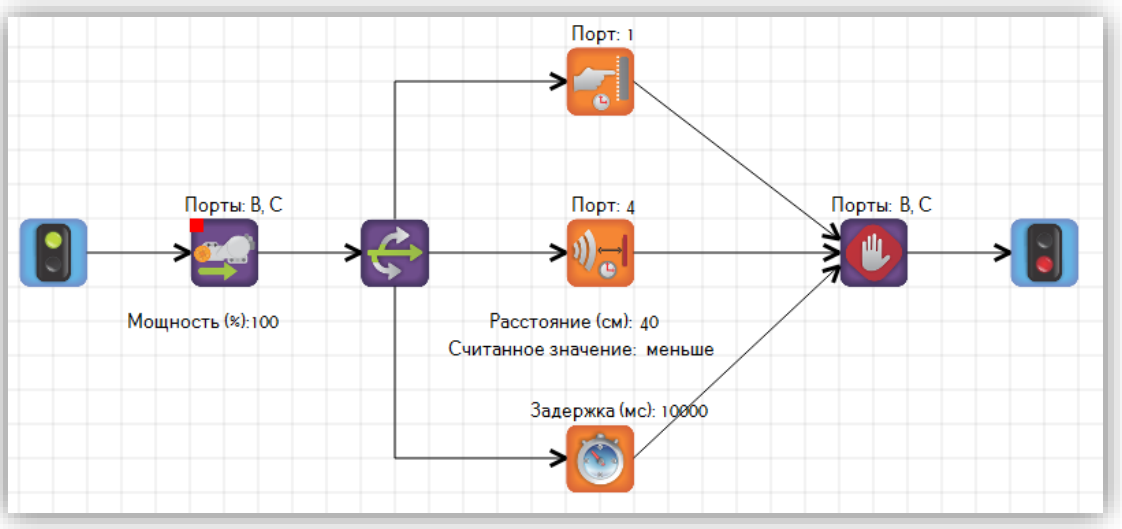


Рисунок 17. Диаграмма в QReal:Robots для проведения экспериментов

После проведения экспериментов мы загрузили лог каждого участника тестирования в приложение, автоматически определили действия создания элемента и соединения их связями в последовательности действий пользователя и собрали следующую информацию: время, затраченное на создание элемента (без учета его поиска), время, затраченное на работу с каждым блоком интерфейса, время на соединение связями блоков на сцене. Дополнительно мы собрали информацию об эффективности выполнения действия – в скольких случаях выполнения (в процентах) необходимой последовательности действий пользователь получил ожидаемый результат. Например, если пользователь перенес элемент из палитры на сцену, то элемент появился на сцене. Также мы отдельно измерили точность выполнения

действия – сколько обязательных действий (в процентах) из формального описания пользователь выполнил для достижения цели.

Результаты: создание элемента

В результате анализа по составному действию «Создание элемента» без учета поиска мы получили, что наименьшее время занимает создание элемента с помощью палитры. С помощью линкеров и жестов мышью элементы создаются приблизительно одинаковое время (таблица 3). Отметим, что при создании элементов с помощью линкеров создается также и линия соединения, но в этом случае мы рассматриваем это как побочный эффект и будем учитывать при сравнении времени создания связей между элементами. При создании элемента с помощью мышиных жестов измерение эффективности выполнения показало, что когда пользователь нарисовал жест и ожидает элемент, то в примерно 15% случаях элемент не создается. Когда использовались линкеры при создании элемента, пользователь совершил в среднем порядка 30% необязательных действий. Отметим, что мы не измеряли точность выполнения для создания элемента с помощью жестов мышью, поскольку этот способ требует более детальной работы с позицией курсора мыши.

СПОСОБ СОЗДАНИЯ / **КРИТЕРИЙ**	**СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ ПАЛИТРЫ**	**СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ ЛИНКЕРОВ**	**СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ МЫШИНЫХ ЖЕСТОВ**
СРЕДНЕЕ ВРЕМЯ (МС)	1974,96	3865,3	3378,77
95% CONFIDENCE (МС)	129,3	347,02	309
ЭФФЕКТИВНОСТЬ ВЫПОЛНЕНИЯ (%)	100	100	84,8
ТОЧНОСТЬ ВЫПОЛНЕНИЯ (%)	100	67,3	---

Таблица 3. Результаты сравнения способов создания элемента (без учета поиска)

Однако после анализа времени выполнения сценариев мы выяснили, что в целом на создание диаграммы с помощью палитры тратится больше времени, чем с помощью других способов (таблица 4). Можно предположить, что при использовании палитры некоторые задачи пользователя требуют дополнительных временных затрат.

СПОСОБ СОЗДАНИЯ / КРИТЕРИЙ	СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ ПАЛИТРЫ	СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ ЛИНКЕРОВ	СОЗДАНИЕ ЭЛЕМЕНТА С ПОМОЩЬЮ МЫШИНЫХ ЖЕСТОВ
ОБЩЕЕ СРЕДНЕЕ ВРЕМЯ (С)	213,3	155,12	189,17
95% CONFIDENCE (С)	15,62	21,42	18,2

Таблица 4. Общее время выполнения сценария

По результатам анализа времени, затраченного на работу с каждым блоком при создании диаграммы с помощью палитры мы получили, пользователь проводит около 40% времени в работе с палитрой, а здесь только 20% времени занимает непосредственно создание элемента (рисунок 18). Таким образом, мы определили, что поиск в палитре нужного элемента занимает значительную часть времени создания диаграммы и этот участок взаимодействия стоит оптимизировать.

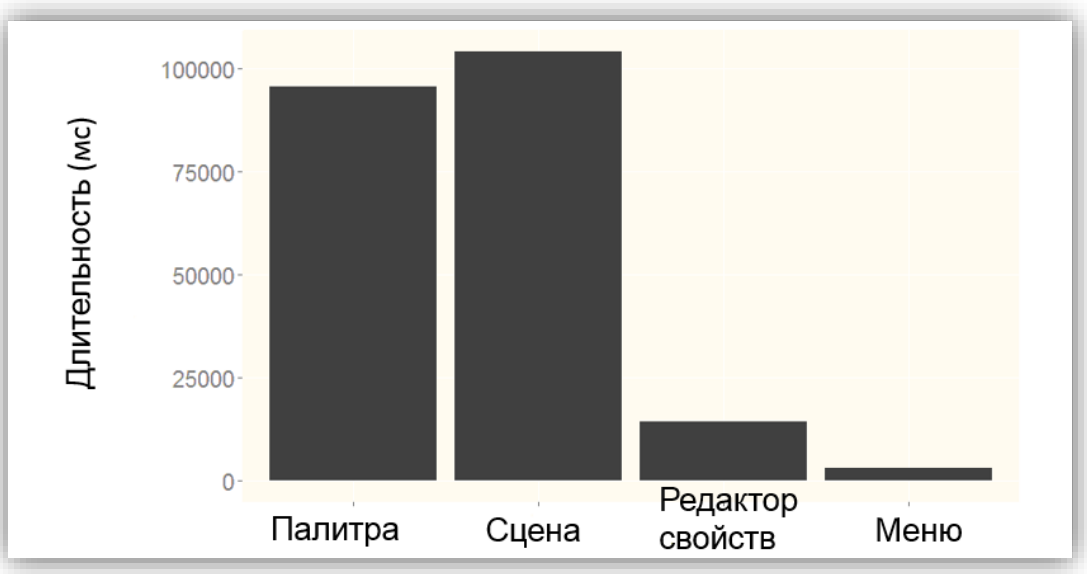


Рисунок 18. График распределения по времени, затраченного на работу с каждым блоком интерфейса при создании диаграммы с помощью палитры

Результаты: соединение элементов связями

Рассмотрим соединение элементов связями (таблица 5). Если соединять связями с помощью элемента из палитры, то сначала надо его создать, а затем присоединить начало связи к одному элементу, и конец связи – к другому. Эти три действия могут быть совмещены, если элемент «Линия соединения» бросить на сцену сразу между блоками и связь соединит нужные блоки автоматически. Также иногда начало связи может присоединиться к ближайшему блоку, а конец связи остаться неприсоединенным. Результаты

экспериментов показали, что в около 90% случаев происходит именно последнее, то есть наиболее вероятный сценарий – вытащить связь из палитры на первый блок, а затем присоединить конец связи ко второму. Мы отдельно рассчитали создание линии соединения и присоединение связи к одному из блоков и определили среднее время во всевозможных случаях (без учета поиска связи в палитре).

При создании блоков с помощью линкеров в большинстве случаев связь создается как побочный эффект. Однако возникают случаи, когда в блок ведут несколько линий соединения, поэтому из линкера можно вытянуть связь и на существующий элемент.

Создание связей с помощью жестов оказалось наиболее быстрым способом. Однако эффективность мышиных жестов оказалась ниже, чем у других.

СПОСОБ СОЕДИНЕНИЯ / **КРИТЕРИЙ**	**СОЕДИНЕНИЕ СВЯЗЬЮ ИЗ ПАЛИТРЫ**			**СОЕДИНЕНИЕ С ПОМОЩЬЮ ЛИНКЕРОВ**	**СОЕДИНЕНИЕ МЫШИНЫМ ЖЕСТОМ**
	только создание	создание и одно соединение	создание и два соединения		
СРЕДНЕЕ ВРЕМЯ (МС)	2016,3	3563,6	5110,9	3428,4	1353,2
ЧАСТОТА (%)	8,3	90,2	1,5		
95% CONFIDENCE (МС)	350,4			561,4	207,7
ЭФФЕКТИВНОСТЬ ВЫПОЛНЕНИЯ (%)	100			100	73,3
ТОЧНОСТЬ ВЫПОЛНЕНИЯ (%)	100			53,6	100

Таблица 5. Результаты сравнения способов соединения элементов

Выводы

В результате апробации созданных инструментов мы выяснили, что создание элементов с помощью палитры (без поиска) – самый стабильный и точный способ, при этом время у него наименьшее, однако с использованием палитры поиск элемента занимает большую часть времени, следовательно, нужно отдельно изучить вопрос об оптимизации структуры палитры.

Создание связей с помощью жестов значительно выигрывает по времени по сравнению с другими способами, но не всегда срабатывает. При оптимизации этот способ сможет быть приоритетным.

Анализ поведения пользователя по сценарию

Помимо обобщенного анализа инструмент позволяет анализировать отдельно каждую последовательность действий пользователя и генерировать график его движения к цели.

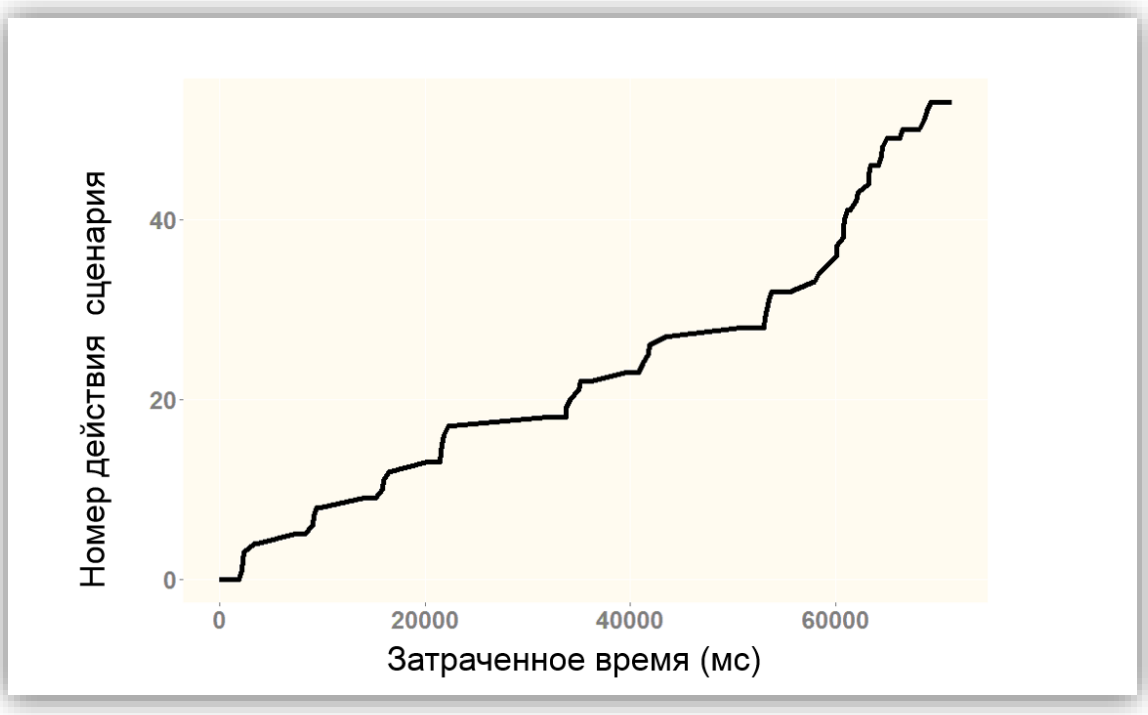


Рисунок 19. График движения пользователя к цели согласно формальному описанию

График показывает следующее: по оси Y мы располагаем номер действия из формального описания, по оси X – реальное время, затраченное пользователем к моменту выполнения этого действия. Если движение по оси Y идет вверх, значит, пользователь выполнил действие из сценария, иначе пользователь выполнил необязательное действие.

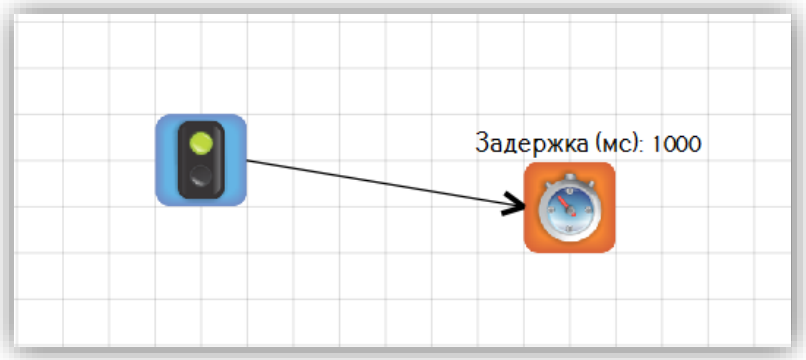


Рисунок 20. Результат создания элемента с помощью линкеров

На этом графике нам особенно важны эти участки, чтобы отдельно изучить, что в это время делал пользователь и предположить причину.

Например, рисунок 19 показывает создание диаграммы с помощью механизма линкеров опытным пользователем. Плоские участки демонстрируют наличие необязательных действий, в которых пользователю пришлось передвигать элемент в нужное место на сцене. Это связано с особенностью создания элементов с помощью линкеров – элемент появляется не прямо по направлению связи, а сдвигается вниз (рисунок 20).

Рекомендации

По результатам экспериментов мы определили ряд часто встречающихся проблем и предложили определили некоторые рекомендации по улучшению процесса взаимодействия с пользователем в QReal:Robots:

1. При создании элемента с помощью линкеров сейчас линкер появляется при наведении на элемент, только если он сейчас находится в фокусе. Линкер расположен на небольшом расстоянии от элемента, поэтому, если пользователь не попадает при нажатии в область действия линкера, то он не только не может начать вытягивать связь из него, но и пропадает фокус на активном элементе. Чтобы снова начать действие, необходимо кликнуть на элемент, затем навести на него, и только тогда снова появится линкер. Для решения этой проблемы предлагается:
    a. Увеличить область действия линкера, чтобы увеличить шанс попасть в него при нажатии кнопкой мыши.
    b. Сделать так, чтобы линкер появлялся при наведении на элемент, даже если он не в фокусе. Это позволит избавить пользователя от дополнительных действий в случае ошибки.
2. В процессе создания элемента с помощью линкера при отпускании кнопки мыши появляется контекстное меню. В нем расположены подряд следующие пункты: «удалить», «отменить», «создать новый элемент». Пользователю чаще всего нужен последний пункт, но для того, чтобы навести на него курсор мыши, приходится пройти еще пункты меню «удалить» и «отменить». Рекомендуется разместить первым пунктом контекстного меню «создать новый элемент».
3. По завершению выбора элемента при создании с помощью линкеров элемент создается не по направлению связи, а с небольшим сдвигом вниз, поэтому пользователь вынужден вручную располагать его в нужную позицию. Рекомендуется создавать элемент в позиции, максимально точно соответствующей позиции конца связи (координата по оси X нового элемента – координата по оси X позиции конца связи при отпускании кнопки мыши).
4. Создание элемента с помощью палитры оказалось наиболее стабильным и быстрым, но поиск в палитре занимает значительное время от всего затраченного времени на создание диаграммы. Мы определили, что

пользователь часто быстро прокручивает палитру вверх-вниз в поисках нужного элемента. Следовательно, пользователь стремится быстро заметить нужный элемент, но не успевает это сделать. Одновременно в палитре пользователь видит 8-9 элементов, всего в палитре на данный момент 31 элемент. Для решения этой проблемы предлагается расширить область видимости элементов палитры, то есть увеличить размер блока интерфейса, желательно, чтобы были доступны все элементы сразу.

5. По завершению соединения блоков с помощью линкеров в случае, если конец связи не наведен точно на элемент, при отпускании кнопки мыши появляется контекстное меню. При клике на сводное место на сцене контекстное меню исчезает, а связь присоединяется к нужному элементу. В этом случае появление контекстного меню излишне, необходимо отслеживать ситуации, когда связь может присоединиться к блоку без непосредственно наведения на него и в этом случае не показывать контекстное меню.
6. Создание связей с помощью жестов мышью оказался самым быстрым способом соединения элементов. Однако в нем есть недостатки: если пользователь проводит несколько связей между элементами быстрее, чем сработает таймаут распознавания жеста (по умолчанию 1 секунда), то ни одна из связей не создастся. Для создания же блоков с помощью жестов таймаут не вызывает сложностей. Для устранения этой проблемы предлагается определять таймаут для жестов отдельно и обеспечивать их отложенное распознавание.

Заключение

В результате дипломной работы были выполнены следующие задачи:

- определены базовые действия пользователя в системе, проведена их классификация;
- реализовано протоколирование выбранных действий пользователя во время рабочей сессии в системе QReal:Robots;
- построена модель описания пользовательских сценариев;
- реализован инструмент для задания формальных сценариев и их автоматизированного анализа;
- проведены эксперименты с использованием разработанного инструмента, по результатам которых были выделены основные рекомендации по улучшению процесса работы пользователя в QReal:Robots.

Таким образом, используя формальные описания поведения пользователя, мы с помощью разработанного инструмента автоматически определили основные проблемные участки интерфейса и, проанализировав эти фрагменты отдельно, вывели рекомендации по улучшению взаимодействия.

Список литературы

1. Abowd D., Formal Aspects of Human-Computer Interaction // Oxford University Computing Laboratory, 1991. P. 1-237.
2. Stone D., Jarrett C., Woodroffe M., Minocha S., User Interface Design and Evaluation // Morgan Kaufmann, 2005.
3. Beckert B., Beuster G., A Method for Formalizing, Analyzing, and Verifying Secure User Interfaces // Formal Methods and Software Engineering Lecture Notes in Computer Science, Volume 4260, 2006. P. 55-73.
4. Burnett M., Summers B., Some Real-World Uses of Visual Programming Systems // Postscript, 1994. P. 1-11.
5. John B., Kieras D., The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast // Journal ACM Transactions on Computer-Human Interaction (TOCHI), Volume 3 Issue 4, 1996. P. 320-351.
6. Rauterberg M., How to Measure Cognitive Complexity in Human-Computer Interaction // Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research, 1996. P. 815-820.
7. John B., Gray W., GOMS analysis for parallel activities // Proceeding CHI'94 Conference Companion on Human Factors in Computing Systems, 1994. P. 395-396.
8. Rosis F., Pizzutilo S., Carolis B., Formal Description and Evaluation of User-Adapted Interfaces // Int. J. Human-Computer Studies, 1998. P. 95-120.
9. Rukienas R., Curzon P., Blandford A., Modelling Rational User Behaviour as Games between an Angel and a Demon // Software Engineering and Formal Methods, 2008. P. 355-364.
10. Терехов А.Н., Литвинов Ю.В., Брыксин Т.А., Среда обучения информатике и робототехнике QReal:Robots // Девятая независимая научно-практическая конференция «Разработка ПО 2013» (CEE SEC(R)-2013), 2013.
11. Осечкина М.С., Брыксин Т.А., Литвинов Ю.В. и др., Поддержка жестов мышью в мета-CASE-системах // Системное программирование. Вып. 5. СПб.: Изд-во СПбГУ. 2010. С. 52-75.
12. Smith T., Waterman M., Identification of Common Molecular Subsequences // Journal of Molecular Biology 147, 1981, P. 195-197.
13. Tullis T., Albert B., Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics, 2008.
14. Campos J., Doherty G., and Harrison M., Including User Behaviour as Model Checking Analysis // Tech. Rep. DI-CCTC-09-17, University of Minho, Braga, 2009. P. 1-15.
15. Shum S., Motta E., Formalization, User Strategy and Interaction Design: Users' Behaviour with Discource Tagging Semantics // 2007.

16. Lund A., Strother L., Rogers W., The human factors and ergonomics society perspective // Proceeding CHI'05 Extended Abstracts on Human Factors in Computing Systems, 2005. P. 1091-1092.
17. Streufert S., Swezey R., Aspects of Cognitive Complexity Theory and Research as Applied to a Managerial Decision Making Simulation // Research Institute for the Behavioral and Social Sciences, 1985. P. 85-96.
18. Ruksenas R., Curzon P., Back J., Blandford A., Formal Modelling of Cognitive Interpretation // Interactive Systems. Design, Specification, and Verification Lecture Notes in Computer Science, Volume 4323, 2007. P. 123-136.