УДК 004.056.5
https://doi.org/10.31854/2307-1303-2025-13-4-1-14
EDN: SFRRWR

Моделирование интерфейса информационной системы и инструкций по работе с ней с учетом девиации поведения пользователя

Моисеенко Г. Ю.

Министерство обороны Российской Федерации,
Москва, 119160, Российская Федерация

Постановка задачи. *Непреднамеренное нарушение пользователем инструкций по работе с информационной системой, приводящее к информационным угрозам (неумышленный инсайдинг), является серьезной проблемой в сфере информационной безопасности. Основная причина нарушений заключается в том, что вследствие определенного психоэмоционального состояния пользователя возникает девиация его поведения и он может ошибаться как в выборе, так и в работе с элементами интерфейса системы: например, вводить конфиденциальные данные в «открытые» поля.* ***Цель*** *работы состоит в описании программного средства моделирования, разработанного на базе авторской модели интерфейса системы и инструкций.* ***Методы исследования:*** *компьютерное моделирование, программная инженерия, эксперимент.* ***Результат:*** *помимо самого факта создания программного средства доказана его работоспособность в части моделирования интерфейса в информационной системе и инструкций по работе с ней, а также продемонстрирована наглядность получаемого графического отображения.* ***Практическая значимость*** *заключается в том, что данное средство позволяет реализовать метод противодействия девиации поведения пользователя путем решения оптимизационной задачи по уточнению инструкций в части спецификации описания элементов интерфейса; при этом данная задача является многокритериальной, поскольку увеличение содержания инструкции ведет к обратному эффекту – усложнению ее восприятия человеком.*

Ключевые слова: *неумышленный инсайдинг, девиация поведения, моделирование, программное средство, эксперимент*

Введение

Одной из причин нарушения информационной безопасности организаций ИТ-сектора является наличие внутренних нарушителей – так называемых инсайдеров, уже находящихся внутри периметра безопасности при выполнении своих должностных обязанностей. Несмотря на то, что данная проблема известна достаточно давно, тем не менее более глубокого изучения требует ее отдельное направление – *неумышленный инсайдинг*, одна из основных причин которого заключается в непреднамеренном совершении сотрудником действий, ведущих

Библиографическая ссылка на статью:
Моисеенко Г. Ю. Моделирование интерфейса информационной системы и инструкций по работе с ней с учетом девиации поведения пользователя // Информационные технологии и телекоммуникации. 2025. Т. 13. № 4. С. 1–14. DOI: 10.31854/2307-1303-2025-13-4-1-14. EDN: SFRRWR

Reference for citation:
Moiseenko G. Modeling the Interface of an Information System and Instructions for Working with It, Taking into Account the Deviation of User Behavior // Telecom IT. 2025. Vol. 13. Iss. 4. PP. 1–14. (in Russian). DOI: 10.31854/2307-1303-2025-13-4-1-14. EDN: SFRRWR

к информационным угрозам [1]. Поскольку значительное количество действий в организации производится с информационной системой, неумышленный инсайдинг опасен при работе именно с ней.

Одной из первопричин неумышленного инсайдера является девиативное поведение сотрудников системы из-за их человеческих особенностей и состояния, например, утомленности или стресса [2], что в результате увеличивает вероятность ошибки при вводе данных в элементы интерфейса, близкие к описанию того, которое требует инструкция. Так, например, в состоянии спешки пользователь может перепутать поля для ввода логина и пароля, а при получении ошибки последний попадет в открытый лог или даже выведется на экран (являясь с точки зрения системы логином).

Одним из путей противодействия инсайдерам такого рода является корректировка самих инструкций по работе с информационной системой (в сторону усиления их однозначности), снижающая тем самым непреднамеренное отклонение от них. Для получения полноценного метода противодействия требуется создание механизма моделирования интерфейса информационной системы и инструкций по работе с ней, который при этом позволял бы учитывать и девиацию поведения пользователя [3]. В результате задача противодействия путем корректировки инструкций может быть сведена к оптимизационной, позволяющей находить рациональный баланс между точностью спецификации инструкции (вернее, элементов интерфейса) и сложностью ее восприятия человеком (чрезмерное разрастание содержания инструкции приведет к обратному эффекту – росту количества и даже «качества» ошибок) [4]. Описание созданной программной реализации такого средства моделирования (в формате спецификации программного интерфейса для использования), а также базовый пример его работы приводятся ниже.

Модель интерфейса

Взаимодействие пользователя с интерфейсом происходит согласно соответствующей инструкции по работе с информационной системой. При этом все многообразие таких систем ограничено наиболее востребованными в целом ряде сфер, а именно – продуцирующими, согласно которым от пользователя требуется ввод определенного набора параметров через графические формы, в результате чего информационная система предоставляет некоторый информационный продукт [5]. При этом логика «прохождения» интерфейса может отличаться от линейной, что обеспечивается проверкой условий, влияющих на последовательность появления графических форм.

Авторская модель интерфейса информационной системы и инструкций по работе с ней (публикация планируется к выходу в 2025 г.) имеет аналитический вид и основана на следующих базовых положениях.

Положение 1. Инструкция по работе с информационной системой состоит из последовательности шагов, определяющих некоторый элемент интерфейса и необходимые действия над ним [6]. При этом, поскольку топология большинства интерфейсов представляет собой дерево (или граф) логики появления его форм, то и линейных инструкций может быть несколько. Выбор же каждой следующей

формы интерфейса (в случае их вариативности) определяется совпадением одного из введенных пользователем значений в форму с заданным условием; например, при заказе билета установка флаговой кнопки провоза багажа дополнительно приведет к появлению соответствующей формы.

Положение 2. Безошибочная работа пользователя по инструкции предполагает точный поиск нужного элемента интерфейса и выполнение над ним заданного действия.

Положение 3. Информационный продукт создается системой на основе данных, введенных пользователем через элементы интерфейса.

Положение 4. Девиация (т. е. отклонение) поведения пользователя может привести к выбору неверных элементов интерфейса и, следовательно, к неточному выполнению инструкции.

Положение 5. Следствием девиации может быть неверный ввод данных (нарушение целостности), часть из которых ошибочно попадет в открытый доступ (нарушение конфиденциальности); также инструкция может быть не завершена из-за выполнения иной логики работы с интерфейсом (нарушение доступности).

Положение 6. Вероятность каждого из нарушений информационной безопасности [7] может быть оценена исходя из конкретного интерфейса, инструкции и уровня девиации поведения определенного пользователя (как постоянного, так и меняющегося в процессе работы – например, в зависимости от сложности описания конкретного шага инструкции).

Положение 7. Пользователь выбирает элемент интерфейса, ориентируясь на его визуальные и функциональные свойства или признаки (близость таких характеристик, как текст [8], форма [9], цвет [10] и пр.), а девиация расширяет множество выбора схожих элементов.

Положение 8. Алгоритм поиска элементов для пользователя с девиативным поведением находит все элементы интерфейса, чья близость к эталону (схожесть) по разным признакам не превышает заданных порогов.

Положение 9. Противодействие девиации возможно путем добавления в описание шагов инструкции уточняющих деталей о признаках элемента, что сужает возможность (снижает вероятность) совершения ошибки пользователем.

Положение 10. Чрезмерное добавление уточнений в инструкцию усложняет ее восприятие, что само по себе может негативно сказаться на работе пользователя.

Положение 11. Исходя из двух предыдущих положений, задача противодействия девиации сводится к оптимизационной – требуется найти такой набор уточнений в инструкции, который минимизирует как вероятность ошибок (и как следствие, нарушений), так и уровень сложности инструкции – т. е. имеется многокритериальное условие.

Положение 12. Решение оптимизационной задачи возможно различными способами, однако из-за «комбинаторного взрыва» наиболее подходящими могут оказаться эвристические.

Положение 13. Иным способом противодействия девиативному поведению является не корректировка инструкций, а модернизация собственно интерфейса для устранения «слабых мест», провоцирующих ошибки пользователя (ко-

торые, в том числе, приводят к реализации комбинированных атак [11]). Впрочем, способ, как правило, применим только перед или в процессе проектирования информационной системы, поскольку готовый интерфейс установленной системы вряд ли подлежит модификации.

Программная реализация

Реализация средства моделирования (интерфейса и инструкций), позволяющая учитывать девиацию поведения пользователя, разработана на языке программирования Python и представляет собой набор классов с собственным функциональным назначением. Создание экземпляров классов, настройка и построение связей между ними позволяет в программной памяти создать граф логики переходов между формами интерфейса. Ниже приводится описание всех классов, а также их полей и методов, в формате спецификации программного интерфейса (т. е. без детализации внутренней реализации), что позволяет использовать данное средство моделирования в качестве внешнего модуля при создании соответствующих алгоритмов, способов и архитектур систем противодействия неумышленному инсайдингу.

Класс "ElemType(Enum)" служит для перечислений (что здесь и далее указывается ключевым словом "Enum" в скобках) и предназначен для хранения возможных типов элемента интерфейса, которые могут быть следующими: "Undefined" – неопределено, "TextBox" – текстовое поле, "ComboBox" – выпадающий список, "Spinner" – «спиннер», "Button" – обычная кнопка, "Radio" – радиокнопка, "CheckBox" – флаговая кнопка.

Класс "Element" предназначен для хранения информации о графическом элементе формы интерфейса; его поля и методы приводятся далее.

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса Element и предназначен для инкрементированного назначения уникальных идентификаторов элементам; поле позволяет хранить значение централизованно и в единственном варианте (в классе).

Поле "Id" содержит текущий уникальный идентификатор элемента интерфейса.

Поле "Name" содержит имя элемента интерфейса, которое может быть проинтерпретировано как его заголовок.

Поле "Type" содержит тип элемента интерфейса и принимает одно из значений перечисления в классе ElemType(Enum).

Поле "IsConfident" содержит бинарный признак того, что информация в нем является конфиденциальной.

Поле "Properties" содержит кортеж значений свойств (признаков) элемента. Так, например, в нем могут располагаться текстовая метка, форма и цвет элемента.

Метод "__init__(self, name, type, is_confident, peoperties = None)" является конструктором класса и инициализирует следующие его поля: имя (Name), тип (Type), признак конфиденциальности (is_confident) и список свойств элемента

(properties). Указание свойств является опциональным, что здесь и далее указывается с помощью конструкции языка программирования "= None". Также, если свойства не указаны, то для них меткой-признаком элемента считается его имя.

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах элемента формы, примером которой является следующая строка:

```
Element_1: 'Login' { type = TextBox, confident = 0 }, properties = ['Login']
```

Класс "Form" предназначен для хранения информации о графической форме интерфейса; его поля и методы приводятся далее.

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса Form и предназначен для инкрементированного назначения уникальных идентификаторов формам.

Поле "Id" содержит текущий уникальный идентификатор формы интерфейса.

Поле "Name" содержит имя формы интерфейса, которое может быть проинтерпретировано как ее заголовок.

Поле "Elements" содержит список всех элементов класса Element, расположенных на форме.

Метод "__init__(self, name)". Метод является конструктором класса и инициализирует его поле имени (Name).

Метод "AddElement(self, element)" добавляет новый элемент (element) в список элементов формы (Elements).

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах формы, примером которой является следующая строка:

```
Form_1: 'Enter' { elements = ['Element_1', 'Element_2', 'Element_3'] }
```

Класс "Condition" предназначен для хранения информации об условии перехода между формами интерфейса согласно логике работы с ним; его поля и методы приводятся далее.

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса Condition и предназначено для инкрементированного назначения уникальных идентификаторов условиям переходов между формами.

Поле "Id" содержит текущий уникальный идентификатор элемента интерфейса.

Поле "Element" содержит элемент формы, данные в котором проверяются (согласно условию) при определении следующей формы для перехода.

Поле "Value" содержит значение, которое сопоставляется с данными в элементе при определении следующей формы для перехода.

Метод "__init__(self, element, value)" является конструктором класса и инициализирует следующие его поля: элемент (Element), значение (Value).

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах условия перехода между формами, примером которой является следующая строка:

```
Condition_3: Element_6 == True
```

Класс "Link" предназначен для хранения информации о переходах между формами интерфейса согласно логике работы с ним – т. е. представляет собой однонаправленную связь «Откуда → Куда»; его поля и методы приводятся далее.

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса и предназначено для инкрементированного назначения уникальных идентификаторов связей между формами.

Поле "Id" содержит текущий уникальный идентификатор перехода между формами интерфейса.

Поле "FormFrom" содержит ссылку на предыдущую форму интерфейса в переходе между их парой, т. е. элемент связи «Откуда».

Поле "FormTo" содержит ссылку на следующую форму интерфейса в переходе между их парой, т. е. элемент связи «Куда».

Поле "Condition" содержит ссылку на условие (опционально), согласно которому выбирается данный переход между формами.

Метод "__init__(self, form_from, form_to, condition = None)" является конструктором класса и инициализирует следующие поля класса: ссылка на предыдущую (form_from) и следующую (form_to) формы, условие для перехода (condition); указание условия является опциональным.

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах переходов между формами, примером которой является следующая строка:

```
Link_1: Form_1 -> Form_2
```

Класс «Interface» предназначен для хранения информации обо всем интерфейсе, содержащем формы, их элементы, связи и условия; его поля и методы приводятся далее.

Поле "RootForm" содержит ссылку на главную (стартовую) форму интерфейса.

Поле "Forms" содержит список всех форм класса Form, присутствующих в интерфейсе.

Поле "Links" содержит список всех переходов между формами класса, присутствующих в интерфейсе.

Метод "__init__(self)" является конструктором класса и производит ряд служебных действий.

Метод "SetRootForm(self, form)" устанавливает главную форму интерфейса (form присваивается к RootForm).

Метод "AddForm(self, form)" добавляет новую форму (form) в список форм интерфейса (Forms).

Метод "LinkForms(self, form_from, form_to, condition = None) " создает переход между формами («form_from → form_to» с опциональным условием condition) и добавляет его в список переходов интерфейса (Links); указание условия является опциональным.

Метод "BuildDescription(self)" создает полное текстовое описание всех объектов интерфейса (с их свойствами): форм, элементов, переходов. Описание строится путем вызова метода "__str__" для объектов соответствующих классов.

DOI: 10.31854/2307-1303-2025-13-4-1-14
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_4/1-14.pdf

Метод "BuildDot(self)" создает описание модели интерфейса в виде графа формата DOT, подходящего для визуализации; примером описания является следующий (приведены основные конструкции описания, остальные сокращены с помощью «...»):

```
digraph {
  subgraph cluster_1 {
    label="#1: Enter" shape=box _Form_Anchor_1 [label="" shape=point]
    Element_1 [label="#1: Login [TextBox]" color=black penwidth=1 shape=box]
    ...
  }
  subgraph cluster_2 {
    ...
  }
  _Form_Anchor_1 -> _Form_Anchor_2 [lhead=cluster_2 ltail=cluster_1 name=""
style=solid]
    ...
}
```

Затем по данному описанию генерируется графическое изображение модели интерфейса (с применением классического средства GraphViz).

Класс "ActionType(Enum)" служит для перечисления и хранения возможных типов действий над элементом интерфейса, которые могут быть следующими: Undefined – не определено, Enter – ввод данных, Select – выбор данных, Click – нажатие на кнопку.

Класс "Step" предназначен для хранения информации о каждом шаге инструкции по работе с информационной системой; его поля и методы приводятся далее.

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса Step и предназначено для инкрементированного назначения уникальных идентификаторов шагов инструкций.

Поле "Id" содержит текущий уникальный идентификатор шага интерфейса.

Поле "Element" содержит ссылку на элемент формы, над которым производятся действия согласно шагу интерфейса.

Поле "Action" содержит тип действия над элементом интерфейса и принимает одно из значений перечисления в классе ActionType.

Поле "Value" содержит значение (опционально), которое необходимо ввести в элемент интерфейса согласно действиям шага.

Метод "__init__(self, element, action, value = None)" является конструктором класса и инициализирует следующие его поля: элемент (Element), действие (Action), значение (Value); указание значения является опциональным.

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах шага инструкции, примером которой является следующая строка:

```
Step_1: Element_1 <- Enter
```

Класс "Instruction" предназначен для хранения информации об инструкции по работе с информационной системой; его поля и методы приводятся далее.

DOI: 10.31854/2307-1303-2025-13-4-1-14
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_4/1-14.pdf

Статическое поле "Id_Counter" содержит глобальный счетчик экземпляров класса Instruction и предназначено для инкрементированного назначения уникальных идентификаторов инструкций.

Поле "Id" содержит текущий уникальный идентификатор элемента инструкции.

Поле "Name" содержит имя инструкции, которое может быть проинтерпретировано как ее название или назначение.

Поле "Steps" содержит список всех шагов класса Step, присутствующих в инструкции.

Метод "__init__(self, name)" является конструктором класса и инициализирует его поле имени (Name).

Метод "AddStep(self, step)" добавляет новый шаг (step) в список шагов инструкции (Steps).

Метод "__str__(self)" предназначен для генерации отладочной информации о свойствах инструкции, примером которой является следующая строка:

```
Instruction_1: 'Buy ticket without baggage' { steps = ['Step_1: Element_1 <-
  Enter', 'Step_2: Element_2 <- Enter', 'Step_3: Element_3 <- Click' …] }
```

Метод "BuildDescription(self)" создает полное текстовое описание шагов инструкции (с их свойствами), которое строится путем вызова метода *"__str__"* для объектов соответствующих классов.

Девиации поведения пользователя в рамках созданного средства моделирования учитываются с помощью свойств элементов (поле "Properties" класса "Element"), поскольку позволяет оценить близость описания элемента в инструкции и его визуальное (или иное) представление на форме.

Пример моделирования интерфейса

Приведем пример моделирования интерфейса информационной системы и инструкций по работе с ней. Предположим, что информационная система предназначена для заказа пассажиром билета до места назначения через личный кабинет; при этом опционально он может приобрести место для багажа. Таким образом, в интерфейсе системы присутствуют четыре следующие формы:

1) вход в личный кабинет (название "Enter");

2) ввод данных о поездке, включая необходимость провоза багажа (название "Ticket");

3) ввод данных о багаже (название "Baggage");

4) оплата поездки (название "Payment").

Интерфейс начинается с первой формы, содержащей два текстовых поля (тип "TextBox") с именем пользователя (заголовок "Login") и его паролем (заголовок "Password").

На второй форме присутствуют два текстовых поля с пунктом назначения (заголовок "Location") и датой / временем отправления (заголовок "Data and time"), а также флаговая кнопка (тип "CheckBox") для выбора необходимости провоза багажа (заголовок "Has baggage?").

DOI: 10.31854/2307-1303-2025-13-4-1-14
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_4/1-14.pdf

Третья форма отображается опционально при наличии у пассажира багажа и содержит текстовое поле с его количеством (заголовок “Quantity”).

Завершает интерфейс четвертая форма для оплаты поездки с текстовым полем для ввода данных банковской карты (заголовок “Card number”).

Кроме того, последним элементом каждой из форм является обычная кнопка (тип “Button”) для завершения действий и перехода к следующей форме (надпись “Ok”).

Для моделирования интерфейса требуется создание объектов соответствующих классов, назначение им свойств и вызов соответствующих методов. Непосредственный программный код на языке Python по использованию вышеописанного средства моделирования приведен в Листинге 1.

Листинг 1. Программный код моделирования интерфейса информационной системы (пример)

```
iface = Interface()

elem_1_1 = Element('Login', ElemType.TextBox, False)
elem_1_2 = Element('Password', ElemType.TextBox, True)
elem_1_3 = Element('Ok', ElemType.Button, False)
form_1 = Form('Enter')
form_1.AddElement(elem_1_1)
form_1.AddElement(elem_1_2)
form_1.AddElement(elem_1_3)

elem_2_1 = Element('Location', ElemType.TextBox, False)
elem_2_2 = Element('Date and time', ElemType.TextBox, False)
elem_2_3 = Element('Has baggage?', ElemType.CheckBox, False)
elem_2_4 = Element('Ok', ElemType.Button, False)
form_2 = Form('Ticket')
form_2.AddElement(elem_2_1)
form_2.AddElement(elem_2_2)
form_2.AddElement(elem_2_3)
form_2.AddElement(elem_2_4)

elem_3_1 = Element('Quantity', ElemType.TextBox, False)
elem_3_2 = Element('Ok', ElemType.Button, False)
form_3 = Form('Baggage')
form_3.AddElement(elem_3_1)
form_3.AddElement(elem_3_2)

elem_4_1 = Element('Card number', ElemType.TextBox, False)
elem_4_2 = Element('Ok', ElemType.Button, False)
form_4 = Form('Payment')
form_4.AddElement(elem_4_1)
form_4.AddElement(elem_4_2)

iface.AddForm(form_1)
iface.AddForm(form_2)
iface.AddForm(form_3)
iface.AddForm(form_4)
iface.SetRootForm(form_1)

iface.LinkForms(form_1, form_2)
iface.LinkForms(form_2, form_3, Condition(elem_2_3, True))
iface.LinkForms(form_2, form_4, Condition(elem_2_3, False))
iface.LinkForms(form_3, form_4)
```

Аналогичное программное моделирование двух инструкций (заказ без багажа и с ним) по работе с информационной системой в ее интерфейсе приведено в Листинге 2.

Листинг 2. Программный код моделирования инструкций по работе с информационной системой (пример)

```
instructions = []

instruction_1 = Instruction('Buy ticket without baggage')
instruction_1.AddStep(Step(elem_1_1, ActionType.Enter))
instruction_1.AddStep(Step(elem_1_2, ActionType.Enter))
instruction_1.AddStep(Step(elem_1_3, ActionType.Click))
instruction_1.AddStep(Step(elem_2_1, ActionType.Enter))
instruction_1.AddStep(Step(elem_2_2, ActionType.Enter))
instruction_1.AddStep(Step(elem_2_3, ActionType.Select, False))
instruction_1.AddStep(Step(elem_2_4, ActionType.Click))
instruction_1.AddStep(Step(elem_4_1, ActionType.Enter))
instruction_1.AddStep(Step(elem_4_2, ActionType.Click))
instructions.append(instruction_1)

instruction_2 = Instruction('Buy ticket with baggage')
instruction_2.AddStep(Step(elem_1_1, ActionType.Enter))
instruction_2.AddStep(Step(elem_1_2, ActionType.Enter))
instruction_2.AddStep(Step(elem_1_3, ActionType.Click))
instruction_2.AddStep(Step(elem_2_1, ActionType.Enter))
instruction_2.AddStep(Step(elem_2_2, ActionType.Enter))
instruction_2.AddStep(Step(elem_2_3, ActionType.Select, True))
instruction_2.AddStep(Step(elem_2_4, ActionType.Click))
instruction_2.AddStep(Step(elem_3_1, ActionType.Enter))
instruction_2.AddStep(Step(elem_3_2, ActionType.Click))
instruction_2.AddStep(Step(elem_4_1, ActionType.Enter))
instruction_2.AddStep(Step(elem_4_2, ActionType.Click))
instructions.append(instruction_2)
```

Соответственно, графическое изображение модели интерфейса, подходящее для выполнения инструкций и генерируемое с помощью метода “BuildDot()” у класса Interface приведено на рисунке 1; красным контуром помечено поле для конфиденциальных данных, а жирным – поле для проверки условия при выборе следующей формы.

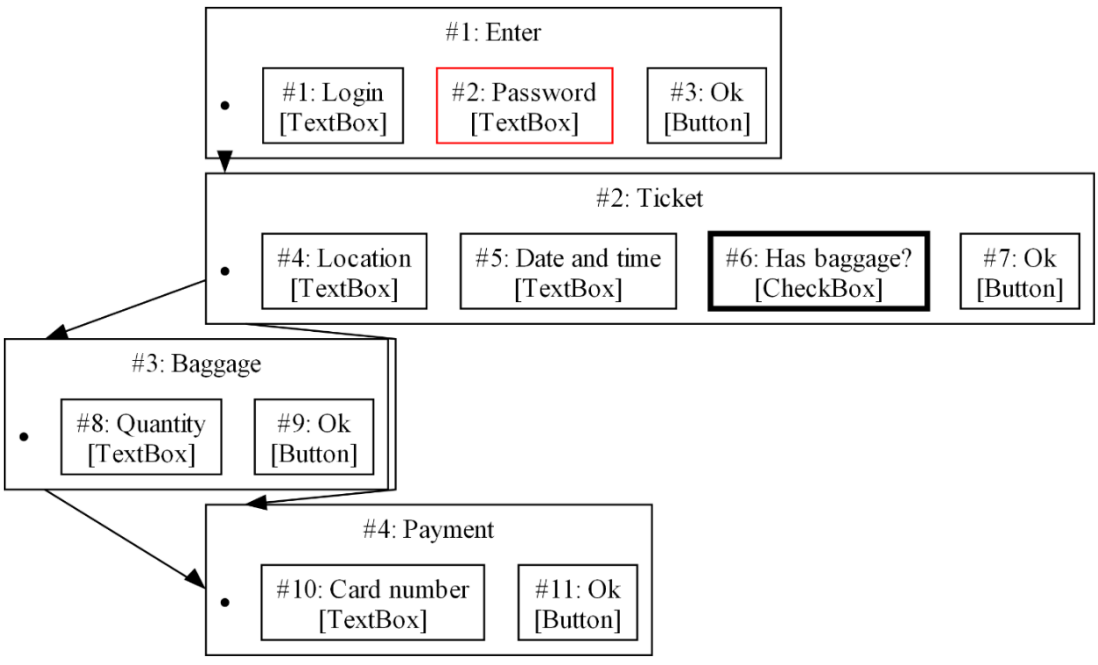

Рис. 1. Графическое изображение модели интерфейса для покупки билета (с опциональным багажом)

DOI: 10.31854/2307-1303-2025-13-4-1-14
URL: https://www.sut.ru/doci/nauka/1AEA/ITT/2025_4/1-14.pdf

Представление интерфейса является интуитивно понятным, соответствует назначению информационной системы, классам средства моделирования и не требует дополнительного пояснения. Для упрощения представления значения поля “Properties” (класс “Element”) в графическом изображении интерфейса не отображаются.

Заключение

В работе описана программная реализация средства моделирования интерфейса информационной системы и инструкций по работе с ней, основанная на авторской аналитической модели. Полученная реализация позволяет моделировать взаимодействие пользователя с информационной системой продуцирующего типа; также, в модели интерфейса учитывается девиация поведения пользователя, приводящая к нарушениям конфиденциальности, целостности и доступности информации. Средство моделирования является полностью работоспособным, что демонстрирует проведенный эксперимент. Планируется продолжение исследования в направлении создания метода повышения устойчивости инструкций к девиации поведения пользователя путем внесения в них уточняющих формулировок и снижения тем самым вероятности ошибок выполнения.

Литература

1. Буйневич М. В., Моисеенко Г. Ю. Нарушение регламента при работе с информационной системой как угроза безопасности информационным ресурсам // Региональная информатика и информационная безопасность: сборник трудов Санкт-Петербургской международной конференции и Санкт-Петербургской межрегиональной конференции (Санкт-Петербург, 23–25 октября 2024 г.). СПб., 2024. С. 78–79. EDN: JRRYNA

2. Ковтунович М. Г., Маркачев К. Е. Информационный стресс // Психологическая наука и образование. 2008. Т. 13. № 5. С. 83–91. EDN: JXDPBX

3. Моисеенко Г. Ю. Обзор способов формализации должностных регламентов деятельности (согласно отечественным исследованиям) // Национальная безопасность и стратегическое планирование. 2024. № 4 (48). С. 35–42. DOI: 10.37468/2307-1400-2024-4-35-42. EDN: EVKBGL

4. Буйневич М. В., Моисеенко Г. Ю. Повышение «устойчивости» регламентов деятельности как способ противодействия неумышленному инсайдингу // Вопросы кибербезопасности. 2024. № 6 (64). С. 108–116. DOI: 10.21681/2311-3456-2024-6-108-116. EDN: HRNCWF

5. Царегородцев А. В., Романовский С. В., Волков С. Д., Самойлов В. Е. Управление рисками информационной безопасности цифровых продуктов финансовой экосистемы организации // Моделирование, оптимизация и информационные технологии. 2020. Т. 8. № 4 (31). DOI: 10.26102/2310-6018/2020.31.4.038. EDN: SKZBBF

6. Курта П. А. Взаимодействие пользователя с информационной системой. Часть 1. Схема взаимодействия и классификация недостатков // Известия СПбГЭТУ ЛЭТИ. 2020. № 8–9. С. 35–45. EDN: VLVMXL

7. Абдуллин Т. И., Баев В. Д., Буйневич М. В., Бурзунов Д. Д., Васильева И. Н. и др. Цифровые технологии и проблемы информационной безопасности. СПб.: СПГЭУ, 2021. 163 с. EDN: NXZPBQ

8. Буйневич М. В., Израилов К. Е. Авторская метрика оценки близости программ: приложение для поиска уязвимостей с помощью генетической деэволюции // Программные продукты и системы. 2025. Т. 38. № 1. С. 89–99. DOI: 10.15827/0236-235X.149.089-099. EDN: RAPDHK

9. Вострых А. В. Алгоритм оценки эффективности визуальной эстетики интерфейсов специализированных программных продуктов, используемых экстренными службами // Национальная безопасность и стратегическое планирование. 2024. № 3 (47). С. 77–89. DOI: 10.37468/2307-1400-2024-3-77-89. EDN: BEEHGJ

10. Данилова М. В., Моллон Д. Д. Цветовые категории и цветоразличение // Экспериментальная психология. 2010. Т. 3. № 3. С. 39–56. EDN: MWKCAD

11. Буйневич М. В., Моисеенко Г. Ю. Комбинирование разнородных деструктивных воздействий на информационную систему и противодействие атакам (на примере инсайдерской деятельности и DDoS-атаки) // Информационные технологии и телекоммуникации. 2023. Т. 11. № 3. С. 27–36. DOI: 10.31854/2307-1303-2023-11-3-27-36. EDN: LWQWNX

Статья поступила 25 ноября 2025 г.
Одобрена после рецензирования 19 декабря 2025 г.
Принята к публикации 20 декабря 2025 г.

Информация об авторе

Моисеенко Григорий Юрьевич – руководитель направления Министерства обороны Российской Федерации. E-mail: mogreq@mail.ru

https://doi.org/10.31854/2307-1303-2025-13-4-1-14
EDN: SFRRWR

Modeling the Interface of an Information System and Instructions for Working with It, Taking into Account the Deviation of User Behavior

G. Moiseenko

Ministry of Defense of the Russian Federation,
Moscow, 119160, Russian Federation

Problem statement. *Unintentional violations by a user of instructions for working with an information system, leading to information security threats (unintentional insider incidents), are a serious issue in the field of information security. The main cause of such violations is that, due to a certain psycho-emotional state of the user, a deviation in behavior occurs, and the user may make mistakes both in choosing and in working with system interface elements: for example, entering confidential data into "open" fields.* ***The aim*** *of this work is to describe a software tool for modeling, developed based on the author's system interface model and instructions.* ***Research methods:*** *computer modeling, software engineering, experiment.* ***The result:*** *in addition to the very fact of creating a software tool, its operability has been proven in terms of modeling the interface in an information system and instructions for working with it, as well as the visibility of the resulting graphical representation.* ***The practical significance*** *lies in the fact that this tool allows you to implement a method to counteract the deviation of user behavior by solving the optimization problem of clarifying instructions in terms of the specification of the description of interface elements; at the same time, this task is multi-criteria, since increasing the content of instructions leads to the opposite effect – complicating its perception by humans.*

Key words: *unintentional insider, behavior deviation, modeling, software tool, experiment*

References

1. Buinevich M., Moiseenko G. Threat to information resources security by violating the rules of work with the information system // Proceedings of the St. Petersburg International Conference and the St. Petersburg Interregional Conference "Regional Informatics and Information Security" (Saint Petersburg, October 23–25, 2024). St. Petersburg, 2024. PP. 78–79. EDN: JRRYNA

2. Kovtunovich M. G., Markachev K. E. Information stress // Psychological Science and Education. 2008. Vol. 13. Iss. 5. PP. 83–91. EDN: JXDPBX

3. Moiseenko G. Review of methods of formalizing job descriptions (according to domestic research) // National Security and Strategic Planning. 2024. Iss. 4 (48). PP. 35–42. DOI: 10.37468/2307-1400-2024-4-35-42. EDN: EVKBGL

4. Buinevich M. V., Moiseenko G. Yu. The instructions "resistant" increasing as a way to counter unintentional insiding // Voprosy Kiberbezopasnosti. 2024. Iss. 6 (64). PP. 108–116. DOI: 10.21681/2311-3456-2024-6-108-116. EDN: HRNCWF

5. Tsaregorodtsev A. V., Romanovskiy S. V., Volkov S. D., Samoylov V. E. Digital products' information security risk management in the organization financial ecosystem // Modeling, Optimization and Information Technology. 2020. Vol. 8. Iss 4 (31). DOI: 10.26102/2310-6018/2020.31.4.038. EDN: SKZBBF

6. Kurta P. A. Interaction of the user with the information system. Part 1. Scheme of interaction and classification of disadvantages // News of ETU. 2020. Iss. 8–9. PP. 35–45. EDN: VLVMXL

7. Abdullin T. I., Baev V. D., Buinevich M. V., Burzunov D. D., Vasilieva I. N., et al. Digital Technologies and Information Security Issues. St. Petersburg: Saint Petersburg State Economic University Publ., 2021. 163 p. EDN: NXZPBQ

8. Buynevich M. V., Izrailov K. E. Author's metric for assessing proximity of programs: application for vulnerability search using genetic de-evolution // Software & Systems. Vol. 38. Iss. 1. PP. 89–99. DOI: 10.15827/0236-235X.149.089-099. EDN: RAPDHK

9. Vostryh A. Algorithm for assessing the efficiency of visual aesthetics of interfaces of specialized software products used by emergency services // National Security and Strategic Planning. 2024. Iss. 3 (47). PP. 77–89. DOI: 10.37468/2307-1400-2024-3-77-89. EDN: BEEHGJ

10. Danilova M. V., Mollon J. D. Color discrimination and color categories // Experimental Psychology (Russia). 2010. Vol. 3. Iss. 3. PP. 39–56. EDN: MWKCAD

11. Buinevich M., Moiseenko G. Combining of heterogeneous destructive impact on the information system and countering attacks (on Example by Insider Activity and DDoS-attack) // Telecom IT. 2023. Vol. 11. Iss. 3. PP. 27–36. (in Russian) DOI: 10.31854/2307-1303-2023-11-3-27-36. EDN: LWQWNX

Information about Author

Moiseenko Grigory – Head of direction, Ministry of Defense of the Russian Federation. E-mail: mogreq@mail.ru