Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления
Кафедра Интеллектуальных информационных технологий

ОТЧЁТ

по ознакомительной практике

Выполнил: Р. В. Липский

Студент группы
121701
Проверил: В. В. Голенков

Минск 2022

Содержание

Введение

Цель:

Закрепить практические навыки формализации информации в интеллектуальных системах с использованием семантических сетей.

Задачи:

- Построение формализованных фрагментов теории интеллектуальных компьютерных систем и технологий их разработки;
- Построение формальной семантической спецификации библиографических источников, соответствующих указанным выше фрагментам;
- Оформление конкретных предложений по развитию текущей версии Стандарта интеллектуальных компьютерных систем и технологий их разработки

1 Постановка задачи

Часть 2 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"

⇒ *библиографическая ссылка**:

- *Стандарт OSTIS*
- *Материалы конференций OSTIS*
- *Журнал "Онтология проектирования"*
- *Справочник по Искусственному интеллекту в трех томах*
- *Энциклопедический словарь по информатике для начинающих*
- *Толковый словарь по Искусственному интеллекту*
- *Статья "Электронные обучающие системы с использованием интеллектуальных технологий"*

  ⇒ *URL**:

  [http://raai.org/library/tolk/aivoc.html]

⇒ *аттестационные вопросы**:

⟨
- *Вопрос 29 по Части 2 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"*
- *Вопрос 30 по Части 2 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"*

⟩

Вопрос 29 по Части 2 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"

:= [Интерфейс интеллектуальной компьютерной системы, основанный на смысловом представлении знаний как интеллектуальный решатель интерфейсных задач. Понятие интерфейсной задачи. Рецепторно-эффекторный уровень взаимодействия интеллектуальной компьютерной системы с внешней средой. Мультирецепторная (мультисенсорная) интеграция и рецепторно-эффекторная координация. Языковое взаимодействие интеллектуальной компьютерной системы с внешними субъектами (обмен сообщениями). Интерфейсная мультимодальность, проблема понимания. Пользовательский интерфейс.]

⇒ *библиографическая ссылка**:

- *Стандарт OSTIS*
- *Microsoft.WindowsDA-2011el*

  := [Microsoft,. Windows desktop applications - guidelines. — 2011. https://docs.microsoft.com/ru-RU/windows/apps/design/controls/.]
- *Maybury.M.IntellUserInterfIntro-1998art*

  := [Maybury, Mark. Intelligent user interfaces: An introduction / Mark Maybury, W. Wahlster // Readings in Intelligent User Interfaces. — 1998. — P. 1–13.]
- *Ehlert.P.IntellUserInterfIAS-2003art*

  := [Ehlert, Patrick. Intelligent user interfaces: Introduction and survey. — 2003.]

Вопрос 30 по Части 2 Учебной дисциплины "Представление и обработка информации в интеллектуальных системах"

:= [Библиотека многократно используемых компонентов интеллектуальных компьютерных систем, основанных на смысловом представлении знаний.]

⇒ *библиографическая ссылка**:

- *Библиотека многократно используемых компонентов OSTIS*
  $\in$ *раздел стандарта OSTIS*
- *Pfister.C.ComponentSoftware-1997art*

  $:=$ [C., Phister. Component Software / Phister C. — 1997]
- *Szyperski.C.CompSoftBOOP-2002book*

  $:=$ [Szyperski, C. Component Software: Beyond Object-oriented Programming / C. Szyperski, D. Gruntz, S. Murer. ACM Press Series. — ACM Press, 2002.]
- *Khan.A.PerspecStudyISFCBD-2015art*

  $:=$ [Khan, Asif. A perspective study of intelligent system for component based development / Asif Khan, Md Mottahir Alam, Mohammad Shariq // International Journal of Computer Applications. — 2015. — Vol. 117. — P. 11–17.]
- *USGenServAdm.2.101.Definitions-reg*

  $:=$ [Government, U. S. Federal acquisition regulation.]

2 Текущее состояние Стандарта

2.1 Текущее состояние раздела «Библиотека многократно используемых компонентов OSTIS»

Библиотека многократно используемых компонентов OSTIS

:= [Библиотека OSTIS]

⇒ *примечание**:

[Библиотека многократно используемых компонентов включает в себя как сами используемые компоненты, так и средства их спецификации, а также средства поиска компонентов по различным критериям, и, таким образом, представляют собой целую подсистему со своей базой знаний и решателем задач.]

⇒ *декомпозиция**:

{
- *Библиотека платформ интерпретации sc-моделей компьютерных систем*
- *Библиотека многократно используемых компонентов баз знаний ostis-систем*
- *Библиотека многократно используемых внутренних агентов ostis-систем*
  ⇒ *декомпозиция**:
  {
  - *Библиотека sc-агентов информационного поиска*
  - *Библиотека sc-агентов погружения интегрируемого знания в базу знаний*
  - *Библиотека sc-агентов выравнивания онтологии интегрируемого знания с основной онтологией текущего состояния базы знаний*
  - *Библиотека sc-агентов планирования решения явно сформулированных задач*
  - *Библиотека sc-агентов логического вывода*
  - *Библиотека sc-агентов обнаружения и удаления информационного мусора*
  - *Библиотека координирующих sc-агентов*
  - *Библиотека sc-моделей языков программирования выского уровня и соответствующих им интерпретаторов*

  }
- *Библиотека многократно используемых методов, интерпретируемых ostis-системами*
  ⇒ *подсистема**:
  Библиотека многократно используемых программ обработки sc-текстов
  ⇒ *подсистема**:
  Библиотека многократно используемых scp-программ
- *Библиотека многократно используемых компонентов интерфейсов ostis-систем*
  ⇒ *декомпозиция**:
  {
  - *Библиотека описаний внешних языков*
  - *Библиотека редакторов внешних информационных конструкций*
  - *Библиотека трансляторов в sc-память*
  - *Библиотека трансляторов из sc-памяти во внешнее представление*
  - *Библиотека визуализаторов*
  - *Библиотека сред общения*
  - *Библиотека элементов управления пользовательским интерфейсом*

  }

- *Библиотека многократно используемых встраиваемых ostis-систем*

}

2.2 Текущее состояние раздела «Предметная область и онтология интерфейсов ostis-систем»

Предметная область интерфейсов ostis-систем

∈ *предметная область*

∋ *максимальный класс объектов исследования′*:
пользовательский интефейс

∋ *класс объектов исследования′*:

- *командный пользовательский интерфейс*
- *графический пользовательский интерфейс*
- *WIMP-интерфейс*
- *SILK-интерфейс*
- *естественно-языковой интерфейс*
- *речевой интерфейс*
- *пользовательский интерфейс ostis-системы*
- *компонент пользовательского интерфейса*
- *атомарный компонент пользовательского интерфейса*
- *неатомарный компонент пользовательского интерфейса*
- *визуальная часть пользовательского интерфейса ostis-системы*
- *компонент пользовательского интерфейса для представления*
- *компонент вывода*
- *компонент выполнения*
- *параграф*
- *декоративный компонент пользовательского интерфейса*
- *контейнер*
- *меню*
- *строка меню*
- *панель инструментов*
- *панель вкладок*
- *окно*
- *модальное окно*
- *немодальное окно*
- *интерактивный компонент пользовательского интерфейса*
- *флаговая кнопка*
- *радиокнопка*
- *переключатель*
- *кнопка-счетчик*
- *полоса прокрутки*
- *кнопка*

...

компонент пользовательского интерфейса

:= [user interface component]

⊃ *компонент пользовательского интерфейса для отображения*

:= [presentation user interface component]

⊃ *компонент вывода*

:= [output]

⊃ *компонент вывода изображения*

:= [image-output]

⊃ *компонент вывода графической информации*

:= [graphical-output]

⊃ *диаграмма*

:= [chart]

⊃ *карта*

:= [map]

⊃ *индикатор выполнения*

:= [progress-bar]

⊃ *компонент вывода видео*

:= [video-output]

⊃ *компонент вывода звука*

:= [sound-output]

⊃ *компонент вывода текста*

:= [text-output]

⊃ *заголовок*

:= [headline]

⊃ *параграф*

:= [paragraph]

⊃ *сообщение*

:= [message]

⊃ *декоративный компонент пользовательского интерфейса*

:= [decorative user interface component]

⊃ *разделитель*

:= [separator]

⊃ *пустое пространство*

:= [blank-space]

⊃ *контейнер*

:= [container]

⊃ *меню*

:= [menu]

⊃ *строка меню*

:= [menu-bar]

⊃ *панель инструментов*

:= [tool-bar]

⊃ *строка состояния*

:= [status-bar]

⊃ *таблично-строковый контейнер*

:= [table-row-container]

⊃ *списковый контейнер*

:= [list-container]

⊃ *таблично-клеточный контейнер*

:= [table-cell-container]

⊃ *древовидный контейнер*

:= [tree-container]

⊃ *панель вкладок*

:= [tab-pane]

⊃ *панель вращения*

:= [spin-pane]

⊃ *узловой контейнер*

:= [tree-node-container]

⊃ *панель прокрутки*

:= [scroll-pane]

⊃ *окно*

:= [window]

⊃ *модальное окно*

:= [modal-window]

⊃ *немодальное окно*

:= [non-modal-window]

⊃ *интерактивный компонент пользовательского интерфейса*

:= [interactive user interface component]

⊃ *компонент ввода данных*

:= [data-input-component]

⊃ *компонент ввода данных с прямой ответной реакцией*

:= [data-input-component-with-direct-feedback]

⊃ *компонент ввода текста с прямой ответной реакцией*

:= [text-input-component-with-direct-feedback]

⊃ *многострочное текстовое поле*

:= [multi-line-text-field]

⊃ *однострочное текстовое поле*

:= [single-line-text-field]

⊃ *ползунок*

:= [slider]

⊃ *область рисования*

:= [drawing-area]

⊃ *компонент выбора*

:= [selection-component]

⊃ *компонент выбора нескольких значений*

:= [selection-component-multiple-values]

⊃ *компонент выбора одного значения*

:= [selection-component-single-values]

⊃ *компонент выбора данных*

:= [selectable-data-representation]

⊃ *флаговая кнопка*

:= [check-box]

⊃ *радиокнопка*

:= [radio-button]

⊃ *переключатель*

:= [toggle-button]

⊃ *выбираемый элемент*

:= [selectable-item]

⊃ *компонент ввода данных без прямой ответной реакции*

:= [data-input-component-without-direct-feedback]

⊃ *кнопка-счётчик*

:= [spin-button]

⊃ *компонент речевого ввода*

:= [speech-input]

⊃ *компонент ввода движений*

:= [motion-input]

⊃ *компонент для представления и взаимодействия с пользователем*

:= [presentation-manipulation-component]

⊃ *активирующий компонент*

:= [activating-component]

⊃ *компонент непрерывной манипуляции*

:= [continuous-manipulation-component]

⊃ *полоса прокрутки*

:= [scrollbar]

⊃ *компонент редактирования размера*

:= [resizer]

⊃ *компонент запроса действий*

:= [operation-trigger-component]

⊃ *компонент выбора команд*

:= [command-selection-component]

⊃ *кнопка*

:= [button]

⊃ *пункт меню*

:= [menu-item]

⊃ *компонент ввода команд*

:= [command-input-component]

компонент пользовательского интерфейса для представления

⇒ *пояснение**:

[*компонент пользовательского интерфейса для представления* – компонент пользовательского интерфейса, не подразумевающий взаимодействия с пользователем.]

компонент вывода

⇒ *пояснение**:

[*компонент вывода* – компонент пользовательского интерфейса, предназначенный для представления информации.]

индикатор выполнения

⇒ *пояснение**:

[*индикатор выполнения* – компонент пользовательского интерфейса, предназначенный для отображения процента выполнения какой-либо задачи.]

параграф

⇒ *пояснение**:

[*параграф* – компонент пользовательского интерфейса, предназначенный для отображения блоков текста. Он отделяется от других блоков пустой строкой или первой строкой с отступом.]

декоративный компонент пользовательского интерфейса

⇒ *пояснение**:

[*декоративный компонент пользовательского интерфейса* – компонент пользовательского интерфейса, предназначенный для стилизации интерфейса.]

контейнер

⇒ *пояснение**:

[*контейнер* – компонент пользовательского интерфейса, задача которого состоит в размещении набора компонентов, включённых в его состав.]

меню

⇒ *пояснение**:

[*меню* – компонент пользовательского интерфейса, содержащий несколько вариантов для выбора пользователем.]

строка меню

⇒ *пояснение**:

[*строка меню* – горизонтальная полоса , содержащая ярлыки меню. Строка меню предоставляет пользователю место в окне, где можно найти большинство основных функций программы.]

панель инструментов

⇒ *пояснение**:

[*панель инструментов* – компонент пользовательского интерфейса, на котором размещаются элементы ввода или вывода данных.]

панель вкладок

⇒ *пояснение**:

[*панель вкладок* – контейнер, который может содержать несколько вкладок (секций) внутри, которые могут быть отображены, нажав на вкладке с названием в верхней части панели. Одновременно отображается только одна вкладка.]

окно

⇒ *пояснение**:

[*окно* – обособленная область экрана, содержащая различные элементы пользовательского интерфейса. Окна могут располагаться поверх друг друга.]

модальное окно

⇒ *пояснение**:

[*модальное окно* – окно, которое блокирует работу пользователя с системой до тех пор, пока пользователь окно не закроет.]

немодальное окно

⇒ *пояснение**:

[*немодальное окно* – окно, которое позволяет выполнять переключение между данным окном и другим окном без необходимости закрытия окна.]

интерактивный компонент пользовательского интерфейса

⇒ *пояснение**:

[*интерактивный компонент пользовательского интерфейса* – компонент пользовательского интерфейса, с помощью которого осуществляется взаимодействие с пользователем.]

флаговая кнопка

⇒ *пояснение**:

[*флаговая кнопка* – компонент пользовательского интерфейса, позволяющий пользователю управлять параметром с двумя состояниями — включено и отключено.]

радиокнопка

⇒ *пояснение**:

[*радиокнопка* – компонент пользовательского интерфейса, который позволяет пользователю выбрать одну опцию из предопределенного набора.]

переключатель

⇒ *пояснение**:

[*переключатель* – компонент пользовательского интерфейса, который позволяет пользователю переключаться между двумя состояниями.]

кнопка-счетчик

⇒ *пояснение**:

[*кнопка-счетчик* – компонент пользовательского интерфейса, как правило, ориентированный вертикально, с помощью которого пользователь может изменить значение в прилегающем текстовом поле, в результате чего значение в текстовом поле увеличивается или уменьшается.]

полоса прокрутки

⇒ *пояснение**:

[*полоса прокрутки* – компонент пользовательского интерфейса, который используется для отображения компонентов пользовательского интерфейса, больших по размеру, чем используемый для их отображения контейнер.]

кнопка

⇒ *пояснение**:

[*кнопка* – компонент пользовательского интерфейса, при нажатии на который происходит программно связанное с этим нажатием действие либо событие.]

3 Предложения к включению в Стандарт

3.1 Предложения к включению в раздел «Библиотека многократно используемых компонентов OSTIS»

Библиотека многократно используемых компонентов OSTIS

...

⇒ *направления развития**:

{ • *уточнение требований к библиотеке многократно используемых компонентов*

• *уточнение технологии разработки многократно используемых компонентов*

}

⇒ *технология**:

компонентое программное обеспечение

компонентное программное обеспечение

⇒ *определение**:

[...компонентное программное обеспечение является конструкцией из компонентов, часть из которых могут быть серийными компонентами, а другие — изготовленными на заказ.]

= *Quote.Phister.C.ComponentSoftware-1997art.p4*

⊂ *COTS-программное обеспечение*

⇒ *проблемы текущего состояния**:

{ • [Сложно предугадать, каким оьразом поведут себя компоненты в различных условиях и окружениях, так как обычно COTS-программное обеспечение поставляется в виде чёрного ящика с ограниченным доступом.]

= *Quote1.Khan.A.PerspecStudyISFCBD-2015art.p11*

• [Также сложно удовлетворить требования пользователя при использовании компонентно-ориентированной архитектуры, обычно необходим процесс подробной настройки компонента.]

= *Quote2.Khan.A.PerspecStudyISFCBD-2015art.p11*

• [Для разработки приложения из компонинтов или улучшения поведения компонентов, необходимо создавать "обёртки"и "склеивать"компоненты, полскольку большая часть COTS-программного обеспечения лишена технологии plug and play и разработчику приходить вручную создавать "обёртки"для интеграции компонентов. Мало того, "обёртки"необходимо поддерживать, пока поддерживается сама система.]

= *Quote.Khan.A.PerspecStudyISFCBD-2015art.p11-12*

• [Компоненты упаковываются и доставляются во многих различных формах (например, библиотеки функций, COTS-приложения, фреймворки).]

= *Quote1.Khan.A.PerspecStudyISFCBD-2015art.p12*

• [Зачастую процесс интеграции компонентов страдает от негибкости и нехватки схем оценки компонентов. Это проблема зачастую сопровождается отстутствием стандартов интероперабельности между компонентами и адекватной поддержки от производителя.]

= *Quote2.Khan.A.PerspecStudyISFCBD-2015art.p12*

}

⇒ *требования**:

{ • [Фундаментальное требование к компонентному ПО состоит в том, чтобы компоненты могли разрабатываться и продаваться независимо друг от друга, и, таким образом, могли комбинироваться потребителем.]

= *Quote2.Phister.C.ComponentSoftware-1997art.p3*

• [

□ стандарт, который допускает динамическую загрузку компонентов (динамически загружаемые библиотеки, соглашения вызовов)

□ стандартный программный интерфейс

□ механизм защиты, который предотвращает нелегальные изменения состояний одних компонентов другими

□ способ разделения данных между компонентами без их копирования взад-вперед и без явных преобразований в линейные потоки байт

]

= *Quote2.Phister.C.ComponentSoftware-1997art.p15*

}

⇒ *достоинства**:

{ • [При разбиении сложной проблемы на более простые, которые могут быть решены независимо друг от друга, проблема становится более управляемой.]

= *Quote.Phister.C.ComponentSoftware-1997art.p4-5*

• [Компоненты могут разрабатываться параллельно, если заранее ясно определено, что каждый компонент должен предоставлять, так что программист может немедленно использовать интерфейс компонента, который разрабатывается другим сотрудником — даже если его реализация еще не существует.]

= *Quote1.Phister.C.ComponentSoftware-1997art.p5*

• [В программном проекте может случиться так, что кто-то вспомнит, что какая-либо часть текущей задачи уже была решена в предыдущем проекте. Если это частичное решение было оформлено как независимый компонент, оно может быть повторно использовано в новом проекте. При этом произойдет экономия времени и средств.]

= *Quote2.Phister.C.ComponentSoftware-1997art.p5*

• [Они [покупатели] могут сэкономить время за счет покупки компонентов на рынке, и тем самым снижают вероятность того, что программные решения уже устареют к моменту своей готовности. Компонентное ПО позволяет добавлять со временем новую функциональность, присоединяя новые компоненты к уже существующим решениям. Таким образов, решение может расширяться, чтобы соответствовать новым нуждам.]

= *Quote3.Phister.C.ComponentSoftware-1997art.p5*

}

⇒ *недостатки**:

{ • [Компонентная реализация — все еще трудная инженерная проблема. Проектирование интерфейсов компонент даже еще более вызывающе. Это требует хорошо образованных и опытных инженеров. Разработка действительно многоразовых высококачественных компонентов не может быть совершена «одним выстрелом»; она требует итеративного долговременного совершенствования, и потому является дорогостоящей.]

= *Quote.Phister.C.ComponentSoftware-1997art.p7*

}

⇔ *следует отличать**:

- *объектно-ориентированное программное обеспечение*
- *модульное программное обеспечение*

⇒ *пояснение**:

- [Программный компонент – то, что непосредственно разворачивается – как изолируемая часть системы – в компонентно-ориентированном подходе. Вопреки частым утверждениям, объекты почти никогда не продаются, не покупаются и не разворачиваются.]
  = *Quote.Szyperski.C.CompSoftBOOP-2002book.p10*
- [...при разработке компонента также может быть использована совсем иная технология, такая как чистые функции или язык ассемблера, которые не имеют ничего общего с объектно-ориентированностью изнутри.]
  = *Quote1.Szyperski.C.CompSoftBOOP-2002book.p11*
- [Определение [объекта] не включает в себя независимости или компонуемости.]
  = *Quote2.Szyperski.C.CompSoftBOOP-2002book.p11*

∋ *многократно используемый компонент*

∋ *объектная модель*

COTS-программное обеспечение

⇒ *пояснение**:

[Commercially available off-the-shelf (COTS) означает любой предмет продажи (включая материалы для сборки), которые:

□ Являются коммерческим продуктом

□ Продаются в значительных количествах на коммерческом рынке

□ Предлагаются Правительству [заказчику] контрактом любого уровня, без модификаций, в той же форме, в которой он продаётся на коммерческом рынке

]

= *Quote.USGenServAdm.2.101.Definitions-reg*

объектная модель

⇒ *определение**:

[Объектная модель определяет необходимые правила совместимости компонентов на уровне двоичного [машинного] кода, так что компоненты могут взаимодействовать на конкретной машине, даже если они разрабатывались независимо.]

= *Quote.Phister.C.ComponentSoftware-1997art.p18*

∋ *язык определения интерфейсов*

∋ *репозиторий интерфейсов*

∋ *формат динамически компонуемых библиотек*

∋ *репозиторий реализации*

∋ *компонующий загрузчик*

∋ *механизм для проверки версий загружаемого кода*

∋ *механизм для создания экземпляров загруженного компонента*

∋ *соглашения по вызову методов компонента*

∋ *механизм для навигации между полиморфными типами компонента*

∋ *формат сообщений*

∋ *механизм уничтожения объектов*

= *sc-агент обнаружения и удаления информационного мусора*

язык определения интерфейсов

:= [IDL]

:= [interface description language]

⇒ *пояснение**:

[Компонент может пользоваться услугами другого компонента. Для этого разработчику надо всего лишь знать интерфейс используемого компонента. Возможно, для этого интерфейса пока еще даже нет реальной реализации. Интерфейс должен быть некоторым образом описан. Текстовое описание интерфейса записывается на языке описания интерфейса (IDL, interface description language). В идеале, он должен быть подмножеством того языка, который использует разработчик. Однако в целом объектная модель независима от языка, и конечно же, IDL может быть подмножеством только лишь очень похожих языков программирования.]

= *Quote.Phister.C.ComponentSoftware-1997art.p18-19*

репозиторий интерфейсов

⇒ *пояснение**:

[IDL-описание предоставляет разработчику информацию об интерфейсе объекта. Если возможно, компилятор должен использовать такую информацию для проверки того, корректно ли используется интерфейс. В этих целях часто существует двоичный формат для описание интерфейсов, в дополнение к текстовому IDL-формату. Доступный набор таких двоичных описаний называется репозиторием интерфейсов. Он может состоять просто из набора так называемых символьных файлов, или он может храниться в базе данных какого-либо рода.]

= *Quote1.Phister.C.ComponentSoftware-1997art.p19*

формат динамически компонуемых библиотек

⇒ *пояснение**:

[Когда компонент компилируется, компилятор должен создать файл, который содержит сгенерированный код. Формат такого файла должен подходить для загрузки во время выполнения, то есть, он должен быть динамически загружаемой библиотекой.]

= *Quote2.Phister.C.ComponentSoftware-1997art.p19*

репозиторий реализации

⇒ *пояснение**:

[Во время выполнения, когда компоненту первый раз потребовались услуги другого компонента, тот компонент загружается. Загрузчик сначала находит кодовый файл для компонента. Расположение кодового файла может быть очевидным, например, имя компонента может напрямую определять путь к его кодовому файлу в файловой системе. Другой вариант - текущее положение кодового файла может запрашиваться из конфигурационной базы данных. Косвенный подход более гибок, но более требователен в плане системного администрирования. Набор кодовых файлов или соответствующая база данных называются репозиторием реализации.]

= *Quote3.Phister.C.ComponentSoftware-1997art.p19*

механизм для проверки версий загружаемого кода

⇒ *пояснение**:

[Когда загрузчик успешно нашел компонент и загрузил его код в память, он может проверить, действительно ли загруженный код реализует запрошенный интерфейс, и совместимы ли версии загруженного компонента и его клиента. Такая проверка — насущная необходимость, поскольку нельзя допустить, чтобы кофликт версий привел некоторое время спустя к краху, и при этом пользователь мог бы только догадываться о причине проблемы. Некоторые объектные модели не предоставляют адекватного механизма контроля версий и перекладывают бремя проверки соответствия частично и полностью на клиентский компонент.]

= *Quote4.Phister.C.ComponentSoftware-1997art.p19*

механизм для создания экземпляров загруженного компонента

⇒ *пояснение**:

[Как только код компонента загружен и проверен, могут быть созданы экземпляры его классов. Для этих целей объектная модель должна обладать механизмом размещения объектов. Некоторые объектные модели предлагают косвенный подход к размещению, чтобы дать дополнительную гибкость. В частности, объект может создаваться другим объектом, так называемой фабрикой. Объект-фабрика может сам решать, как размещать и как инициализировать объекты.]

= *Quote5.Phister.C.ComponentSoftware-1997art.p19*

соглашения по вызову методов компонента

⇒ *пояснение**:

[После того, как объект был создан и стал доступен для других, можно вызывать его методы. Вызов метода — это вызов процедуры, который выполняется не напрямую, а через объект, так что различные объекты могут приводить к вызову различного кода. Обычно объект содержит указатель на таблицу указателей на процедуры. Каждый элемент этой таблицы соответствует одному методу объекта. Поэтому вызов метода - всего лишь непрямой вызов процедуры через таблицу методов объекта. Обычно для этих вызовов используются соглашения вызовов базовой операционной системы.]

= *Quote6.Phister.C.ComponentSoftware-1997art.p19*

механизм для навигации между полиморфными типами компонента

⇒ *пояснение**:

[Полиморфизм — одно из главных качеств любой объектно-ориентированной или компонентно-ориентированной системы. Некотрый интерфейс, поддерживаемый объектом, может быть известен во время компиляции. Он называется его статическим типом. Но объект может приобретать дополнительные возможности, то есть, расширять свой интерфейс, во время выполнения. В зависимости от динамического типа объекта эти возможности могут отличаться. Это называется полиморфизмом («много форм»). Объектная модель должна обеспечить средства, с помощью которых можно получить доступ к таким дополнительным возможностям, если и только если они доступны. Объектно-ориентированный язык программирования в этих целях задает языковые конструкции, такие как расширение типа (также известное как субтипизация или наследование интерфейса), проверки типа или охрана типа (то есть, безопасные преобразования между типами). Объектная модель может предоставлять похожую функциональность. Без полифморфизма система не может быть расширяемой.]

= *Quote1.Phister.C.ComponentSoftware-1997art.p20*

механизм уничтожения объектов

⇒ *пояснение**:

[Если объект больше не используется, то есть, на него больше нет внешних ссылок, он должен быть уничтожен, чтобы освободилась занимаемая им память. В компонентных системах на объект, который компонент предоставляет вовне, ссылаются какие-либо другие объекты, о которых сам объект ничего не знает. Поэтому должны существовать правила, которые указывают, кто и когда должен уничтожить данный объект. Например, это может сделать другой объект, у которого осталась последняя ссылка на данный. Для определения того момента, когда ссылка на объект стала последней, необходим механизм трассировки ссылок, например, счетчик ссылок в каждом объекте, который подсчитывает количество текущих ссылок на него. Когда количество становится равным нулю, память объекта может быть освобождена. Для того, чтобы такие правила работали, критически важно их правильное использование всеми компонентами. В замкнутых приложениях неправильная работа с памятью является самым главным источником ошибок; но вы, по крайней мере, знаете, кто виноват в произошедшем сбое. В открытых компонентных системах один неправильно работающий компонент может привести к краху всю систему, при этом сложно определить виновного производителя. Поэтому управление памятью в компонентной программной среде является принципиально более важным вопросом, чем в монолитном ПО. В идеале правила и механизмы, вводимые объектной моделью, должны быть достаточно простыми, полными и ясными, чтобы их можно было автоматизировать, то есть, освободить разработчика от ручной работы, предоставив автоматический сборщик мусора. Автоматический сборщик мусора в открытых системах - это не роскошь, а необходимость.]

= *Quote2.Phister.C.ComponentSoftware-1997art.p20*

формат сообщений

⇒ *пояснение**:

[Объектная модель, которая поддерживает распределенные объекты, должна определять формат сообщений, который описывает потоки байт, порождаемые удаленным вызовом методов. Того, кто вызывает метод, называют клиентом, того, чей метод вызывают, — сервером. По самому общему сценарию серверный объект и клиент выполняются на разных машинах. С точки зрения разработчика клиент может напрямую вызывать методы серверного объекта. В реальности клиент взаимодействует всего лишь с прокси-объектом, который является локальным представителем настоящего объекта, работающего на удаленной серверной машине. В большинстве реализаций прокси-объект, также как и его аналог на стороне сервера, могут генерироваться автоматически из интерфейса объекта.]

= *Quote3.Phister.C.ComponentSoftware-1997art.p20*

многократно используемый компонент

⇒ *определение**:

[Компонент — это элемент конструкции с определенным, зафиксированным в спецификации, интерфейсом и явными зависимостями от контекста. Компоненты могут распространяться независимо друг от друга и компоноваться третьей стороной.]

= *Quote.Phister.C.ComponentSoftware-1997art.p3*

⇒ *требования**:

{
- *независимость многократно используемого компонента*
- *компонуемость многократно используемого компонента*
- *скрытое внутреннее состояние многократно используемого компонента*
- *уникальность многкратно используемого компонента*

}

⇒ *декомпозиция**:

{ • *интерфейс многократно используемого компонента*
  • *реализация многократно используемого компонента*
}

независимость многократно используемного компонента
⇒ *пояснение**:
[Для того, чтобы компонент можно было развернуть независимо, он должен быть отделён от своего окружения и других компонентов.]
= *Quote1.Szyperski.C.CompSoftBOOP-2002book.p36*

компонуемость многократно используемого компонента
⇒ *пояснение**:
[...он [компонент] должен поставляться с чёткой спецификацией того, что он требует и предоставляет. Другими словами, компонент должен скрывать свою имплементацию и взаимодействовать с окружением при помощи чётко определённых интерфейсов.]
= *Quote2.Szyperski.C.CompSoftBOOP-2002book.p36*

скрытое внутреннее состояние многократно используемного компонента
⇒ *пояснение**:
[...компонент не должен иметь наблюдаемого (извне) состояния.]
= *Quote3.Szyperski.C.CompSoftBOOP-2002book.p36*

уникальность многократно используемого компонента
⇒ *пояснение**:
[...не имеет смысла иметь несколько копий [компонента] в одной и той же операционной системе, поскольку они в любом случае будут фактически неотличимы. Другими словами, в любом процессе (или ином контексте), не должно быть более одной копии конкретного компонента.]
= *Quote4.Szyperski.C.CompSoftBOOP-2002book.p36*

интерфейс многократно используемого компонента
⇒ *определение**:
[Интерфейс определяет стандарт, которому производители компонентов обязаны следовать и которого пользователи вправе ожидать.]
= *Quote.Phister.C.ComponentSoftware-1997art.p15*
⇒ *пояснение**:
[...каждому компоненту нужно взаимодействовать с другими компонентами исключительно через их интерфейсы; это разновидность контракта, иначе говоря, закон для программиста.]
= *Quote1.Phister.C.ComponentSoftware-1997art.p17*
⇔ *следует разделять**:
реализация многократно используемого компонента
⇒ *требования**:
{ • *ясность интерфейса многократно используемого компонента*
  • *существенность интерфейса многократно используемого компонента*
}

ясность интерфейса многократно используемного компонента
⇒ *пояснение**:

[В то время как синтаксис интерфейса может быть задан очень легко, ясная семантика трудноуловима. Обычно имеется простой неформальный текст, который описывает, что из себя представляет некоторый интерфейс. Формальные и полуформальные методы специфицирования могут помочь сделать интерфейсы менее противоречивыми.]

= *Quote.Phister.C.ComponentSoftware-1997art.p16*

существенность интерфейса многократно используемого компонента

⇒ *пояснение**:

[Интерфейс, который задает слишком много несущественных деталей, заставит программистов использовать их и допускать их истинность. Становится невозможным изменить эти детали позже, даже если это действительно потребуется.]

= *Quote2.Phister.C.ComponentSoftware-1997art.p17*

3.2 Предложения к включению в раздел «Предметная область и онтология интерфейсов ostis-систем»

пользовательский интерфейс

⊃ *графический пользовтельский интерфейс*
- ⊃ *WIMP-интерфейс*
  - ⊃ *IUI-интерфейс*
    - ⊃ *мультимодальный интерфейс*
    - ⊃ *естественно-языковой интерфейс*
      - ⊃ *речевой интерфейс*
    - ⊃ *мимический интерфейс*
    - ⊃ *жестовый интерфейс*

IUI-интерфейс

:= [intelligent user interface]

:= [IUI]

⇒ *пояснение**:

[Интеллектуальные пользовательские интерфейсы (IUI) — это человеко-машинные интерфейсы, целью которых является повышение эффективности, результативности и естественности взаимодействия человека и машины путем представления, рассуждений и действий на основе моделей пользователя, предметной области, задачи, дискурса и медиа (например, , графика, естественный язык, жест). Как следствие, эта междисциплинарная область опирается на исследования и лежит на пересечении взаимодействия человека с компьютером, эргономики, когнитивной науки и искусственного интеллекта и его подобластей (например, компьютерное зрение, обработка речи и языка, представление знаний и рассуждения, машинное обучение/обнаружение знаний, планирование и агентное моделирование, пользовательское и дискурсивное моделирование).**]**

= *Quote1.Maybury.M.IntellUserInterfIntro-1998art.p2*

⇒ *цели**:

[

- □ **Создание персонализированных систем**
  Не существует двух людей с одинаковыми привычками, предпочтениями, методами работы и окружением. Интеллектуальный интерфейс, который учитывает все эти различия может создать персонализованный метод взаимодействия. Интерфейс знает пользователя и может использовать это знание для общения с пользователем.
- □ **Перегрузка информацией и проблемы фильтрации**
  Найти правильную информацию на компьютере или в сети может быть сложнее, чем найти нитку в стоге сена. Интеллектуальные интерфейсы могут снизить перегрузку информацией, которая возникает при поиске в больших базах данных или сложных системах. При помощи фильтрации неважной информации интерфейс может снизить когнитивную нагрузку на пользователя. К тому же, IUI может предложить новые и полезные источники, о которых пользователь ранее не знал.
- □ **Помощь в пользовании новым и сложным ПО**
  Работа с компьютерными системами может быть сложна, особенно, если это первый опыт пользователя в работе с конкретным ПО. Пока пользователь пытается

разобраться с новой программой, новые версии ПО и прочие обновления могут добавить новый функционал. Многие пользователи не успевают за всеми этими обновлениями. Интеллектуальные вспомогательные системы могут обнаруживать и исправлять недопонимания пользователя, объяснять новые концепции, снабжать пользователя информацией для упрощения работы.

- **Делегирование задач от пользователя**
  IUI, во время действия пользователя, может понимать и узнавать намерение пользователя и даже забрать себе часть работы пользователя, позволяя ему сфокусироваться на более важных задачах.
- **Прочие формы взаимодействия**
  На данный момент, наиболее распространенные средства взаимодействия – клавиатура и мышь. IUI также рассматривает иные способы взаимодействия компьютера и человека (например, речь или жесты). Альтернативные формы взаимодействия могут помочь людям с ограниченными возможностями проще пользоваться компьютером.

]
= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p2*
⇒ *требования**:
[

- Адаптивный пользовательский интерфейс должен разрабатываться параллельно с приложением. Это необходимо, так как разработчику постоянно приходится сосредотачиваться на частях системы, которые необходимо адаптировать.
- Не мешайте взаимодействию пользователя. Пользователь всегда должен иметь возможность игнорировать упреждающие действия IUI. Предлагайте, а не действуйте.
- Работайте в режиме реального времени. Большая часть преимуществ IUI заключается в том, что он действует, пока пользователь занят работой с системой.
- Воспользуйтесь временем, который пользователь тратит на раздумия. Когда пользователь думает о том, что вводить дальше, IUI может использовать доступное время для своих целей, таким образом он не рискует замедлить взаимодействие пользователя с системой.
- Следите за тем, что делает пользователь. Воспользуйтесь преимуществом «бесплатной» информации, скрытой в действиях пользователя.
- Позвольте пользователю выбирать свой личный стиль взаимодействия. Разным пользователям нравятся разные стили интерфейса, и некоторые методы могут отвлекать или сбивать с толку некоторых пользователей.

]
= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p8*
⇒ *недостатки**:
[

- Адаптивная система непредсказуема и менее прозрачна, чем классический интерфейс. Если система может адаптировать свой ответ и не дает один и тот же ответ дважды на одни и те же входные данные, то система становится непредсказуемой. Это затруднит понимание пользователем системы, лишив его воз-

можности дважды выполнить успешное действие.

- □ Пользователи больше не контролируют ситуацию. IUI может принимать решения за пользователя, тем самым выводя пользователя за пределы контура управления.
- □ IUI часто ошибаются. Многие системы IUI используют следы и ошибки для определения намерений или предпочтений пользователя. Поэтому пользователям необходимо давать обратную связь системе или даже устранять ошибки, допущенные системой.
- □ Имитация интеллекта и адаптивности увеличивает риск того, что пользователь будет думать, что компьютер может делать то, на что он не способен, что создает ложные ожидания. В частности, с антропоморфными агентами пользователи могут полагать, что они могут взаимодействовать с IUI так же, как с другим человеком.
- □ Кто ответственный? Если система может принимать решения самостоятельно, то кто несет ответственность за действия: программист системы, пользователь или сама система?
- □ А как насчет конфиденциальности и упорства пользователя? Что происходит с профилем пользователя, созданным и поддерживаемым IUI? Можете ли вы гарантировать, что он безопасен и не будет использован не по назначению?

]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003.p9-10*

⇒ *направления развития**:

[

- □ **Когнитивная теория и основы эмпирической науки.**
  Необходимо лучше понять уникальные лингвистические и рабочие характеристики естественных модальностей общения (речь, жесты, взгляд и мимика), а также то, как эти модальности можно комбинировать наилучшим образом.
- □ **Новые концепции мультимодального интерфейса.**
  Текущие исследования мультимодальных интерфейсов сосредоточены в основном на обработке естественного языка и жестах пера. Должны быть изучены дополнительные входные данные, такие как движения тела и выражение лица.
- □ **Мультимодальный язык и обработка диалогов.**
  Необходима общая теория разговорного взаимодействия, которая имеет дело с репрезентацией намерений для неречевых модальностей.
- □ **Методы обработки ошибок.**
  Мягкая обработка ошибок по-прежнему является проблемой в мультимодальных интерфейсах, и взаимное устранение неоднозначности между сигналами может быть улучшено. Также следует изучить влияние третьего или более модальностей на частоту ошибок, а также производительность при сложные (мобильные) условия.
- □ **Адаптивные мультимодальные архитектуры.** Мультимодальные архитектуры вряд ли адаптивны, и адаптация к конкретному пользователю или среде может повысить узнаваемость обработки ввода, а также обеспечить большую гибкость и простоту использования для пользователя.

□ **Повсеместные многопользовательские вычисления на нескольких устройствах.**
В будущем мобильные вычисления станут более важными, поэтому нам необходимо рассмотреть роль мультимодальных интерфейсов в этой области. Кроме того, когда несколько пользователей взаимодействуют друг с другом, например, через Интернет, интерфейсы должны учитывать множественный ввод с удаленных устройств.

□ **Мультимодальная исследовательская инфраструктура.**
Должны быть разработаны вспомогательные инструменты для исследования мультимодальных интерфейсов. Это включает; полуавтоматические методы моделирования для сбора эмпирических данных и прототипирования новых систем, автоматизированные инструменты для сбора и анализа мультимодальных корпусов, новые метрики для оценки мультимодальных систем и программные инструменты, которые поддерживают быстрое создание мультимодальных приложений следующего поколения.

]
= *Quote1.Ehlert.P.IntellUserInterfIAS-2003.p36*

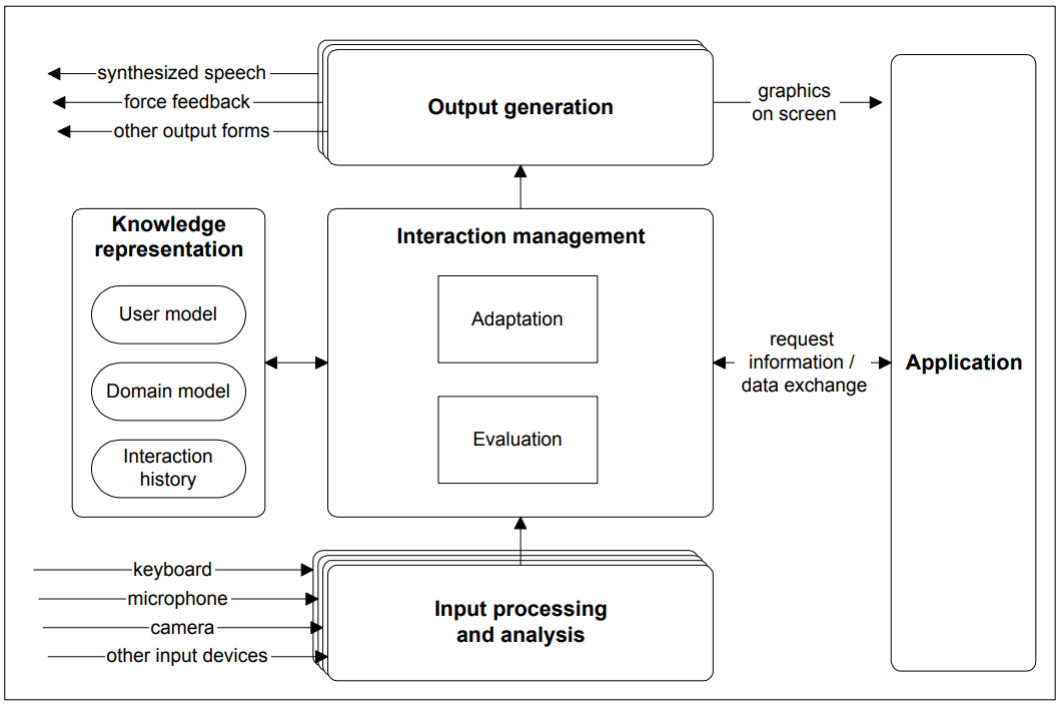
⇔ *следует отличать**:
WIMP-интерфейс
⇒ *пояснение**:
[«Интеллект» в IUI – это то, что отличает его от традиционных интерфейсов... он [IUI] включает в себя механизмы, которые выполняют автоматизированный анализ мультимедиа, дизайн и управление взаимодействием... Традиционные пользовательские интерфейсы различают только три модели: представление, диалог и приложение. Уточнения, помимо этих трех моделей, которые можно найти в IUI, включают явные модели пользователя, дискурса и предметной области, анализ входных данных и генерацию выходных данных, а также механизмы управления взаимодействием, такие как объединение и интерпретация неточных, двусмысленных и/или неточных входных данных, контроль ход диалога или адаптация вывода презентации к текущей ситуации. Исследования показали, что можно адаптировать многие из фундаментальных концепций, разработанных на сегодняшний день в компьютерной лингвистике и теории дискурса, таким образом, чтобы они стали полезными и для мультимедийных пользовательских интерфейсов. В частности, семантические и прагматические понятия, такие как коммуникативные акты, когерентность, фокус, ссылка, модель дискурса, модель пользователя, импликатура, анафора, риторические отношения и двусмысленность объема, приобретают расширенное значение в контексте мультимодальной коммуникации... искусственный интеллект многое, чтобы внести свой вклад в пользовательские интерфейсы, включая использование представлений знаний для инструментов разработки интерфейсов на основе моделей. Архитектура интеллектуальных пользовательских интерфейсов, применение генерации планов и распознавания в управлении диалогами, применение временных и пространственных рассуждений для координации медиа, использование пользовательских моделей для настройки взаимодействия и т. д.]
= *Quote1.Maybury.M.IntellUserInterfIntro-1998art.p3-4*

⇔ *следует отличать**:
интеллектуальная система
⇒ *пояснение**:
[Часто совершаемая ошибка — путать IUI с интеллектуальной системой. Система,

демонстрирующая некоторую форму интеллекта, не обязательно является интеллектуальным интерфейсом. Существует много интеллектуальных систем с очень простыми неинтеллектуальными интерфейсами, и тот факт, что система имеет интеллектуальный интерфейс, ничего не говорит об интеллекте базовой системы.]

⇒ *архитектура**:

[



]

⇒ *источник**:
Ehlert.P.IntellUserInterfIAS-2003art

∋ *интеллектуальные технологии ввода*
∋ *моделирование пользователя*
∋ *модель пользователя*
∋ *адаптация к пользователю*
∋ *генерация пояснений*

моделирование пользователя

⇒ *пояснение**:

[Моделирование пользоваетля охватывает методы, которые позволяют системе поддерживать или делать выводы о пользователе на основе полученного ввода]

= *Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p4*

генерация пояснений

⇒ *пояснение**:

[Генерация пояснений охватывает все методы, которые позволяют системе объяснять свои результаты пользователю, например, речевой вывод, агенты интеллектуального интерфейса, тактильную обратную связь в среде виртуальной реальности.]

= *Quote4.Ehlert.P.IntellUserInterfIAS-2003art.p4*

модель пользователя

⇒ *пояснение**:

[Для достижения персонализации IUI часто включают представление пользователя. Эти пользовательские модели регистрируют данные о поведении, знаниях и способностях

пользователя. Новые знания о пользователе могут быть получены на основе истории ввода и взаимодействия пользователя с системой.]

= *Quote5.Ehlert.P.IntellUserInterfIAS-2003art.p4*

адаптация к пользователю

⇒ *пояснение**:

[Адаптация к пользователю включает в себя все методы, которые позволяют адаптировать общение между человеком и машиной к разным пользователям и разным ситуациям, например, машинное обучение или понимание контекста.]

= *Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p4*

интеллектуальные технологии ввода

⇒ *пояснение**:

[Интеллектуальная технология ввода использует инновационные методы для получения ввода от пользователя. Эти методы включают в себя естественный язык, отслеживание и распознавание жестов, выражение лица распознавание и отслеживание взгляда среди прочего;]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p4*

⊃ *речевой ввод*

⊃ *естественно-языковой ввод*

⊃ *жестовый ввод*

⊃ *мимический ввод*

мимический ввод

⊃ *распознование лиц*

- ⊃ *распознование выражений лиц*
- ⊃ *отслеживание взгляда*
- ⊃ *чтение по губам*

отслеживание взгляда

⇒ *пояснение**:

[Люди используют свои глаза с очень небольшим сознательным усилием. Большую часть времени мы автоматически смотрим на объект, над которым работаем. Чтобы четко видеть объект, необходимо переместить глаза так, чтобы объект оказался в центральной ямке, небольшой области в центре сетчатки. Ямка покрывает примерно один градус зрительной дуги. Из-за этого положение глаз человека обеспечивает довольно хорошую индикацию (с точностью до одного градуса ширины центральной ямки) точки фокусировки внимания человека на дисплее.]

= *Quote3.Ehlert.P.IntellUserInterfIAS-2003.p18*

чтение по губам

⇒ *пояснение**:

[Проблема с распознаванием речи на основе акустического сигнала заключается в том, что оно очень плохо работает при большом количестве шума. Причина этого в том, что очень трудно отличить голос говорящего от других звуков. Идея чтения по губам заключается в том, что компьютер пытается определить, что кто-то говорит, просто глядя на движения губ человека, как глухой человек. Анализируя видеоизображения рта и используя геометрические признаки, такие как ширина и высота губ, можно определить наиболее вероятный звук (фонему).]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003.p19*

жестовый ввод

⇒ *пояснение**:

[Люди часто используют жесты во время разговора, часто подсознательно. Наиболее часто используемые жесты связаны с речью, например, указывая на объект, о котором идет речь. Этот вид жестикуляции называется жестикуляцией. Жесты, функционирующие независимо от речи, называются автономными жестами, например язык жестов.]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p17*

⇒ *ввод**:

жест

жест

⇒ *декомпозиция**:

{ • *символический жест*
• *дейктический жест*
• *знаковый жест*
• *пантомимический жест*
}

символический жест

⇒ *пояснение**:

[Символические жесты имеют (единственное) вербальное и часто культурно-зависимое значение, например, знак «ОК» или язык жестов для глухих.]

= *Quote2.Ehlert.P.IntellUserInterfIAS-2003.p17*

дейктический жест

⇒ *пояснение**:

[Дейктические жесты делаются путем указания или движения, чтобы привлечь внимание к какому-либо объекту или событию.]

= *Quote3.Ehlert.P.IntellUserInterfIAS-2003.p17*

знаковый жест

⇒ *пояснение**:

[Знаковые жесты — это жесты, которые отображают информацию о размере, форме или ориентации объектов, пространственных отношениях и действиях, например, использование рук для обозначения размера пойманной рыбы).]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003.p18*

пантомимический жест

⇒ *пояснение**:

[Пантомимические жесты состоят из манипулирования невидимым воображаемым объектом или инструментом, например, сжатие кулака и движение, обозначающее молоток).]

= *Quote2.Ehlert.P.IntellUserInterfIAS-2003.p18*

естественно-языковой ввод

:= [естественно-языковая система]

⇒ *декомпозиция**:

{ • *распознавание речи*

- *понимание языка*
- *управление диалогом*
- *запрос к базе данных*
- *генерация ответа*
- *речевой вывод*

}

распознование речи

⇒ *пояснение**:

[Распознавание речи; преобразование входного речевого высказывания в строку из слов.]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p13*

понимание языка

⇒ *пояснение**:

[Понимание языка; анализ строки слов (насколько это возможно) для извлечения смыслового представления распознанного высказывания.]

= *Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p13*

управление диалогом

⇒ *пояснение**:

[Управление диалогом; управление взаимодействием или диалогом между системой и пользователем, что включает в себя координацию других компонентов системы.]

= *Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p13*

запрос к базе данных

⇒ *пояснение**:

[Запрос к базе данных; получение информации, запрошенной пользователем.]

= *Quote4.Ehlert.P.IntellUserInterfIAS-2003art.p13*

генерация ответа

⇒ *пояснение**:

[Генерация ответа; спецификация текста, который должен быть выходным сообщением системы.]

= *Quote5.Ehlert.P.IntellUserInterfIAS-2003art.p13*

речевой вывод

⇒ *пояснение**:

[Речевой вывод; фактическая генерация выходного сообщения с использованием синтеза речи или предварительно записанных фраз.]

= *Quote6.Ehlert.P.IntellUserInterfIAS-2003art.p13*

мультимодальный интерфейс

⇒ *пояснение**:

[Идея мультимодальных интерфейсов заключается в использовании нескольких входных каналов при взаимодействии человека с компьютером. Вместо того, чтобы вводить только речь или текст, система может использовать распознавание изображений, чтобы смотреть на лицо или жесты пользователя. Таким образом, информация из одного режима взаимодействия может дополнять информацию, полученную из другого режима.

Мультимодальные интерфейсы пытаются интегрировать речь, письменный текст, язык тела, жесты, движения глаз или губ и другие формы общения, чтобы лучше понимать пользователя и общаться более эффективно. Конечно, выбор используемых модальностей в мультимодальных интерфейсах во многом зависит от применения системы. Когда пользователи могут выбирать из нескольких способов взаимодействия с системой, система может стать доступной для более широкого круга пользователей, например для людей с ограниченными возможностями. Кроме того, мультимодальные пользовательские интерфейсы гораздо более надежны, чем обычные интерфейсы, по крайней мере, в теории. Обработка входных данных из одной модальности может быть упрощена за счет использования информации из другой, и пользователь может выбрать модальность, которая наименее подвержена ошибкам с учетом обстоятельств.]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p20*

⊃ *слияние информации*

слияние информации

⇒ *пояснение**:

[Основным узким местом мультимодальных интерфейсов является объединение информации, полученной от используемых модальностей, в режиме реального времени. Все мультимодальные интерфейсы нуждаются в той или иной форме координации ввода или системы слияния для синхронизации связанного ввода.]

= *Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p20*

⇒ *декомпозиция**:

{ • *слияние на уровне особенностей*

⇒ *пояснение**:

[Слияние на уровне функций уместно при объединении двух связанных модальностей, таких как обработка речи и движения губ. Преимуществом слияния на уровне функций является улучшенная скорость распознавания благодаря взаимодополняемости обоих входных каналов (взаимное устранение неоднозначности). Недостаток такой тесной интеграции обработки ввода заключается в том, что система должна переобучаться при изменении одной из модальностей ввода. Кроме того, обучение системы с несколькими модальностями с одновременным вводом данных намного сложнее, чем обучение одной модальности за раз.]

= *Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p20*

• *слияние на уровне семантики*

⇒ *пояснение**:

[Слияние на семантическом уровне намного проще. У него нет преимущества прямой дополнительной информации, но систему слияния на семантическом уровне гораздо проще создавать и расширять. Например, система может использовать несколько готовых методов распознавания, а новые модальности могут быть легко добавлены позже. Интеграция на семантическом уровне обычно выполняется либо путем объединения существующих данных, либо путем поиска отсутствующих данных. Последняя называется интеграцией на основе фреймов, когда системы пытаются заполнить недостающие слоты данных.]

= *Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p20-21*

}

/* Далее идёт описание ранее неописанных компонентов интерфейса. */

компонент ввода данных

...
⊃ *поле автозаполнения*
:= [autosuggest-field]
⊃ *строка навигатора*
:= [breadcrumb-bar]
:= [панель навигации]
⊃ *управляющий элемент выбора даты в календаре*
:= [calendar-date-field]
⊃ *средство выбора цвета*
:= [color-palette]
:= [палитра]
⊃ *поле со списком*
:= [list-field]
⊃ *ссылки на содержимое в текстовых элементах управления*
:= [content-link]
⊃ *управляющий элемент выбора даты*
:= [date-choice]
⊃ *форма*
:= [form]
⊃ *элемент управления рукописным вводом*
:= [handwriting-element]

поле автозаполнения
⇒ *пояснение**:
[Используйте AutoSuggestBox, чтобы предоставить список предложений, из которых пользователь по мере ввода текста может выбрать нужное.**]**
= *Quote1.Microsoft.WindowsDA-2011el*

строка навигатора
⇒ *определение**:
[Панель навигации предоставляет прямой путь к страницам или папкам в текущее расположение. Он часто используется для ситуаций, когда путь навигации пользователя (в файловой системе или системе меню) должен быть постоянно видимым, и пользователю может потребоваться вернуться к предыдущему расположению.**]**
= *Quote2.Microsoft.WindowsDA-2011el*

управляющий элемент выбора даты в календаре
⇒ *определение**:
[Управляющий элемент выбора даты в календаре — это раскрывающийся элемент управления, оптимизированный для выбора отдельной даты в представлении календаря, когда важна контекстная информация, например день недели или заполненность календаря. Вы можете изменить календарь таким образом, чтобы обеспечить дополнительный контекст или ограничить доступные даты.**]**
= *Quote3.Microsoft.WindowsDA-2011el*

средство выбора цвета
⇒ *определение**:

[Палитра используется для просмотра и выбора цвета. По умолчанию он позволяет пользователю перемещаться по цветам в цветовом спектре или указывать цвет в текстовых полях.]

= *Quote5.Microsoft.WindowsDA-2011el*

поле со списком

⇒ *определение**:

[Поле со списком (также известное как раскрывающийся список) позволяет представить список элементов, из которых пользователь может выбирать. Сначала поле со списком представлено в компактном состоянии, а затем развертывается для отображения списка элементов, доступных для выбора.]

= *Quote6.Microsoft.WindowsDA-2011el*

ссылки на содержимое в текстовых элементах управления

⇒ *определение**:

[Ссылки на содержимое позволяют вставлять в текстовые элементы управления форматированные данные. Благодаря этому пользователь может находить и использовать больше информации о людях и местах, не покидая вашего приложения.]

= *Quote10.Microsoft.WindowsDA-2011el*

управляющий элемент выбора даты

⇒ *определение**:

[Элемент выбора даты — это стандартизованный способ, позволяющий пользователям выбирать локализованное значение даты с помощью сенсорного ввода, мыши или клавиатуры.]

= *Quote11.Microsoft.WindowsDA-2011el*

форма

⇒ *определение**:

[Форма — это группа элементов управления, которые собирают данные пользователей и отправляют их. Формы обычно используются для страниц параметров, опросов, создания учетных записей и многого другого.]

= *Quote15.Microsoft.WindowsDA-2011el*

элемент управления рукописным вводом

⇒ *определение**:

[Элемент управления рукописным вводом позволяет включить в приложении базовые функции рукописного ввода]

= *Quote17.Microsoft.WindowsDA-2011el*

интерактивный компонент пользовательского интерфейса

⊃ *контекстное меню*

:= [раскрывающееся меню]

:= [context-menu]

⊃ *панель команд*

:= [command-panel]

⊃ *карточка контакта*

:= [contact-card]

⊃ *диалоговые элементы управления*

:= [dialogue-element]

⊃ *расширитель*

:= [expander]

⊃ *представление пролистывания*

:= [scroller]

⊃ *всплывающе элементы*

:= [pop-up]

⊃ *представления списка и сетки*

:= [grid]

⊃ *гиперссылка*

:= [hyperlink]

⊃ *отображение карт*

:= [map-element]

контекстное меню

⇒ *определение**:

[Всплывающие элементы меню используются в сценариях меню и контекстного меню для отображения списка команд или параметров, запрашиваемых пользователем.**]**

= *Quote18.Microsoft.WindowsDA-2011el*

панель команд

⇒ *определение**:

[Панели команд предоставляют пользователям удобный доступ к самым распространенным задачам приложения. Панели команд могут предоставлять доступ к командам приложения или страниц, работая с любым шаблоном навигации.**]**

= *Quote7.Microsoft.WindowsDA-2011el*

карточка контакта

⇒ *определение**:

[Карта контакта отображает контактные данные, такие как имя, номер телефона и адрес контакта. Карточка контакта также позволяет пользователю редактировать контактные данные. Можно выбрать, какую карточку следует отобразить: компактную карточку контакта или полную карточку контакта, которая содержит дополнительные сведения.**]**

= *Quote8.Microsoft.WindowsDA-2011el*

диаологовые элемента управления

⇒ *определение**:

[Диалоговые элементы управления — это модальные наложения пользовательского интерфейса, которые предоставляют контекстную информацию о приложении. Они блокируют взаимодействие с окном приложения, пока пользователь явно их не закроет. Они часто требуют от пользователя совершения каких-либо действий.**]**

= *Quote9.Microsoft.WindowsDA-2011el*

расширитель

⇒ *определение**:

[Элемент управления "Расширитель"позволяет отображать или скрывать менее важное содержимое, связанное с частью основного содержимого, которое всегда отображается. Элементы, содержащиеся в заголовке , всегда видны. Пользователь может развернуть и

свернуть область содержимого , в которой отображается дополнительное содержимое, взаимодействуя с заголовком. Когда область содержимого развернута, она отправляет другие элементы пользовательского интерфейса из пути; он не накладывает другой пользовательский интерфейс. Может увеличиваться вверх или вниз.]

= *Quote12.Microsoft.WindowsDA-2011el*

представление пролистывания

⇒ *определение**:

[Используйте представление пролистывания для просмотра изображений или других элементов в коллекции, например фотографий в альбоме или элементов на странице описания продукта.]

= *Quote13.Microsoft.WindowsDA-2011el*

всплывающий элемент

⇒ *определение**:

[Всплывающий элемент — это контейнер с возможностью исчезновения, который отображает в качестве содержимого произвольный пользовательский интерфейс. Всплывающие элементы могут содержать другие вложенные всплывающие элементы или контекстные меню.]

= *Quote14.Microsoft.WindowsDA-2011el*

гиперссылка

⇒ *определение**:

[Гиперссылки используются для перехода в другую часть приложения, в другое приложение либо по указанному универсальному коду ресурса в отдельном приложении браузера.]

= *Quote16.Microsoft.WindowsDA-2011el*

отображение карт

⇒ *определение**:

[Карту можно показывать во всплывающем окне, называемом карточкой места, или в полнофункциональном элементе управления с картой.]

= *Quote19.Microsoft.WindowsDA-2011el*

4 Формальная семантическая спецификация библиографических источников

Phister.C.ComponentSoftware-1997art

⇒ *тип**:
статья

⇒ *ключевой знак**:
- *комопнент*
- *компоненто-ориентированное программирование*

⇒ *аннотация**:
[Появление компонентного программного обеспечения может быть самым важным событием в индустрии программного обеспечения с момента появления языков программирования высокого уровня. Компонентное программное обеспечение сочетает в себе преимущества заказного и стандартного программного обеспечения. Это позволяет создавать решения, более приспособолены к эволюции, т. е. которые лучше масштабируются, которые легче обслуживать, которые можно расширять с течением времени и постепенно модернизировать.]

⇒ *цитата**:
Quote1.Phister.C.ComponentSoftware-1997art.p3
= [A component is a unit of composition with a contractually specified interface and explicit context dependencies only. Components can be deployed independently of each other and are subject to composition by third parties.]

⇒ *цитата**:
Quote2.Phister.C.ComponentSoftware-1997art.p3
= [It is a fundamental requirement of component software that components may be developed and sold independently of each other, and yet can be combined by the customer.]

⇒ *цитата**:
Quote.Phister.C.ComponentSoftware-1997art.p4
= [...component software is a composition of components, some of which may be standard components and others may be custom components.]

⇒ *цитата**:
Quote.Phister.C.ComponentSoftware-1997art.p4-5
= [By splitting a complex problem into simpler problems which can be solved independently, the problem becomes more manageable.]

⇒ *цитата**:
Quote1.Phister.C.ComponentSoftware-1997art.p5
= [Components can be developed in parallel if it is clearly defined in advance what each component should provide, so that a programmer can immediately use the interface of a component currently being developed by another team member - even though its implementation doesn't exist yet.]

⇒ *цитата**:
Quote2.Phister.C.ComponentSoftware-1997art.p5
= [It may happen in a software project that someone remembers that some part of the current problem had already been solved for an earlier project. If this partial solution has the form of a separate component, it can be reused in the new project, thus saving time and cost.]

⇒ *ɥumama**:
Quote3.Phister.C.ComponentSoftware-1997art.p5
= [They [buyers] get customized solutions quicker. They can save time by buying components on the market, and thereby it becomes less likely that the solutions are obsolete by the time they are ready for use. Component software allows to add new functionality over time, by adding new components to an already existing solution. In this way, a solution can be extended to handle new needs over time.]

⇒ *ɥumama**:
Quote.Phister.C.ComponentSoftware-1997art.p7
= [Component implementation is still a difficult engineering problem. The design of component interfaces is even more challenging. It takes well educated and experienced engineers to do it. Developing a truly reusable high-quality component cannot be achieved "in one shot"; it requires iterative improvement over a long time, and therefore is expensive.]

⇒ *ɥumama**:
Quote1.Phister.C.ComponentSoftware-1997art.p15
= [The interface defines a standard for what component vendors have to provide and what component users can expect.]

⇒ *ɥumama**:
Quote2.Phister.C.ComponentSoftware-1997art.p15
= [

- □ a standard that enables dynamic loading of components (dynamic link libraries, calling conventions)
- □ a standard programming interface
- □ a protection mechanism which prevents a component from illegally modifying the state of other components
- □ a way to share data between components without copying them back and forth, and without explicit conversions to or from linear byte streams

]

⇒ *ɥumama**:
Quote.Phister.C.ComponentSoftware-1997art.p16
= [While the syntax of an interface can be defined easily, clear semantics are elusive. Usually there is just an informal text describing what the interface means. Formal and semi-formal specification methods can help to make interfaces less ambiguous.]

⇒ *ɥumama**:
Quote1.Phister.C.ComponentSoftware-1997art.p17
= [...every component is required to interact with other components through their interfaces exclusively; this is a kind of universal contract, i.e., a law.]

⇒ *ɥumama**:
Quote2.Phister.C.ComponentSoftware-1997art.p17
= [An interface which defines too many inessential details will cause programmers to use them and to rely on their availability. This makes it impossible to change these details later, even though it may become strongly desirable to do so. Giving too many details is especially enticing if there already exists some complex but undocumented code, which is to be turned into a component.]

⇒ *ɥumama**:

Quote.Phister.C.ComponentSoftware-1997art.p18

= [An object model defines the necessary rules to make components compatible on a binary level, such that components can interact on a particular machine even if they have been developed independently.]

⇒ *ɥumama**:

Quote.Phister.C.ComponentSoftware-1997art.p18-19

= [A component may use the services of another component. To achieve this, a developer only needs to know the used component's interface. Possibly, there does not even exist an actual implementation for this interface yet. An interface needs to be described in some way. A textual description of an interface is written in an interface description language (IDL). Ideally, this is just a subset of the programming language that the developer uses. However, a general object model is language-independent, and of course an IDL can only be a genuine subset of very similar programming languages.]

⇒ *ɥumama**:

Quote1.Phister.C.ComponentSoftware-1997art.p19

= [An IDL description provides information about an object's interface to the developer. If possible, a compiler should use the same information to check whether the interface is used correctly. For this purpose, there often exists a binary format for interface descriptions in addition to the textual IDL format. The available collection of such binary descriptions is called the interface repository. It may just consist of a collection of so-called symbol files, or it may be stored in some kind of database.]

⇒ *ɥumama**:

Quote2.Phister.C.ComponentSoftware-1997art.p19

= [When a component is compiled, the compiler needs to create a file containing the generated code. The format of such a file must be suitable for run-time loading, i.e., it must be a dynamic link library.]

⇒ *ɥumama**:

Quote3.Phister.C.ComponentSoftware-1997art.p19

= [At run-time, when a component for the first time needs the services of another component, this component is loaded. The loader first locates the code file of the component. Locating the code file may be straight-forward, e.g., the name of a component may directly determine the file system path to its code file. Alternatively, a configuration database may be consulted to determine the correct location of a suitable code file. This indirect approach is more flexible, but is more demanding in terms of system administration. The collection of code files or the corresponding database is called an implementation repository.]

⇒ *ɥumama**:

Quote4.Phister.C.ComponentSoftware-1997art.p19

= [When the loader has successfully located a component and loaded its code into memory, it can check whether the loaded code really implements the interface that was requested, and whether the versions of the loaded component and its client are compatible. This check is of fundamental importance, because it is not acceptable that a version conflict leads to a crash some time later, leaving the user without clue as to the source of the problem. Some object models provide no adequate versioning mechanism and shift the burden of consistency checking partially or completely to the client component.]

⇒ *ɥumama**:

Quote5.Phister.C.ComponentSoftware-1997art.p19

= [Once a component's code has been loaded and checked, instances of its classes can be created. For this purpose, the object model needs to define an allocation mechanism

for objects. Some object models suggest an indirect approach to allocation, in order to gain additional flexibility. In particular, an object may be created by another object, a so-called factory object. A factory object may decide on its own how to allocate or how to initialize objects.]

⇒ *цumama**:

Quote6.Phister.C.ComponentSoftware-1997art.p19

= [When an object thus finally has been created and made accessible to others, its methods can be called. A method call is a procedure call performed indirectly via an object, so that different objects can lead to different code being called. Typically, an object contains a pointer to a table of procedure pointers. Each element of the table corresponds to one method of the object. A method call then simply becomes an indirect procedure call via the object's method table. Usually the calling conventions of the underlying operating system are used for these procedure calls.]

⇒ *цumama**:

Quote1.Phister.C.ComponentSoftware-1997art.p20

= [Polymorphism is one of the fundamental properties of any object-oriented or component-oriented system. In a program, some interface supported by an object may be known at compile-time. This is called its static type. But an object may reveal additional capabilities, i.e., an extended interface, at run-time. Depending on the object's dynamic type, these capabilities may differ. This is called polymorphism ("many shapes"). An object model needs to provide a means to gain access to these optional capabilities if and only if they are available. An object-oriented programming language defines language constructs such as type extension (also known as subtyping or interface inheritance), type tests, or type guards (i.e., safe type casts) for this purpose. The object model must provide similar functionality. Without polymorphism, a system would not be extensible.]

⇒ *цumama**:

Quote2.Phister.C.ComponentSoftware-1997art.p20

= [When an object is no longer used, i.e., when it is no longer being referenced from the outside, it must be deallocated to free the memory that it occupies. In a component world, the objects that a component makes accessible to the outside may become referenced by any number of other objects that it doesn't even know about. There must be rules that establish who must deallocate a given object, and when. For example, the object that releases the last remaining reference to another object could deallocate it. To determine whether an object owns the last reference to another one, a mechanism is needed that helps tracking references, e.g., a reference count in each object which counts the number of currently existing references to it. When the count goes down to zero, the object's memory can be freed. Correct usage by all components is critical for such rules to work. In a closed application, incorrect memory management is one of the most expensive sources of errors; but at least you know who to blame if the application crashes. In an open component world, one malfunctioning component can cause others to crash, which makes it difficult to pinpoint the culpable vendor. Thus memory management in a component software environment is a fundamentally more critical issue than for monolithic software. Ideally, the rules and mechanisms defined by an object model should be sufficiently simple, complete and clear that they can be automated; i.e., it should be possible to relieve the developer from manual deallocation, by providing an automatic garbage collector. Automatic garbage collection in an open world is no luxury, it is a necessity.]

⇒ *цumama**:

Quote3.Phister.C.ComponentSoftware-1997art.p20

= [An object model that supports distributed objects must define a message format, which

describes the byte streams produced by a remote method call. The caller of a method is called the client, the callee is called the server. In the most general scenario, the server object is implemented on another machine than the client. From a developer's perspective, the client can directly call a server object's methods. In reality, the client only interacts with a proxy object, which is a local representation of the true object implementation on the remote server machine. In most implementations, a proxy, as well as its server-side counterpart, can be generated automatically out of the object's interface.]

Szyperski.C.CompSoftBOOP-2002book

⇒ *тип**:
книга

⇒ *ключевой знак**:
- *компонент*
- *компонентное проектирование*
- *компоненто-ориентированное программирование*

⇒ *аннотация**:
[Книга посвящена посвящена компонентному проектированию, как способу повторно использовать уже разработанное программное обеспечение.]

⇒ *цитата**:
Quote.Szyperski.C.CompSoftBOOP-2002book.p10

= [A software component is what is actually deployed – as an isolatable part of a system – in a component-based approach. Contrary to frequent claims, objects are almost never sold, bought, or deployed.]

⇒ *сравнение**:
{
- *компонентное программное обеспечение*
- *объектно-ориентированное программное обеспечение*

}

⇒ *цитата**:
Quote1.Szyperski.C.CompSoftBOOP-2002book.p11

= [...a component could just as well use some totally different implementation technology, such as pure functions or assembly language, and look not at all object-oriented from the inside.]

⇒ *сравнение**:
{
- *компонентное программное обеспечение*
- *объектно-ориентированное программное обеспечение*

}

⇒ *цитата**:
Quote2.Szyperski.C.CompSoftBOOP-2002book.p11

= [The definition [of object] does not include notions of independence or late composition.]

⇒ *сравнение**:
{
- *компонентное программное обеспечение*
- *объектно-ориентированное программное обеспечение*

}

⇒ *цитата**:
Quote1.Szyperski.C.CompSoftBOOP-2002book.p36

= [For a component to be independently deployable, it needs to be well separated from its environment and other components.]

⇒ *цитата**:
Quote2.Szyperski.C.CompSoftBOOP-2002book.p36

= [...it [component] needs to come with clear specifications of what it requires and provides. In other words, a component needs to encapsulate its implementation and interact with its environment by means of well-defined interfaces.]

⇒ *цитата**:
Quote3.Szyperski.C.CompSoftBOOP-2002book.p36
= [...a component should not have any (externally) observable state.]

⇒ *цитата**:
Quote4.Szyperski.C.CompSoftBOOP-2002book.p36
= [...it makes little sense to have multiple copies [of a component] in the same operating system process as these would be mutually indistinguishable anyway. In other words, in any given process (or other loading context), there will be at most one copy of a particular component.]

USGenServAdm.2.101.Definitions-reg

⇒ *тип**:
закон

⇒ *цитата**:
Quote.USGenServAdm.2.101.Definitions-reg
= [Commercially available off-the-shelf (COTS) item means any item of supply (including construction material) that is

- □ A commercial product (as defined in paragraph (1) of the definition of "commercial product" in this section);
- □ Sold in substantial quantities in the commercial marketplace; and
- □ Offered to the Government [customer], under a contract or subcontract at any tier, without modification, in the same form in which it is sold in the commercial marketplace;

]

Khan.A.PerspecStudyISFCBD-2015art

⇒ *тип**:
статья

⇒ *аннотация**:
[Nowadays, researchers envisage an Intelligent ComponentOriented Software Development methodology which is an amalgam of the two approaches resulting in more flexible, reusable and customizable agent components. This helps in pushing forward the development timelines and quality expectations to newer heights. In this paper we mainly analyzed various states of art intelligent component-oriented software development techniques and studied the research gap in the component selection processes. Recommendations for future research direction for Intelligent Component-Oriented Software Development are also highlighted in this paper.]

⇒ *ключевой знак**:
- *мультиагентные системы*
- *компоненто-ориентированное проектирование*
- *агенто-ориентированное проектирование*
- *самоадаптирующиеся системы*

⇒ *цитата**:
Quote1.Khan.A.PerspecStudyISFCBD-2015art-p.11
= [It is very difficult to guess how the components behave under different conditions and

environments as mostly COTS software comes up as a black box with limited access.]

⇒ *цитата**:
Quote2.Khan.A.PerspecStudyISFCBD-2015art-p.11
= [It is also difficult to map user requirement to the component based architecture and generally there is a need for a process which fully customized the component as per the customer requirement.]

⇒ *цитата**:
Quote.Khan.A.PerspecStudyISFCBD-2015art-p.11-12
= [In order to developed application from components or tailor components to a new situation, efforts are required to build wrappers and the glue between components, since most of the COTS software lacks in plug and play technique and developer has to build wrappers for component integration. Further, wrappers need to be maintained as the system evolves.]

⇒ *цитата**:
Quote1.Khan.A.PerspecStudyISFCBD-2015art-p.12
= [Components are packaged and delivered in many different forms (example: function libraries, off-the shelf applications and frameworks).]

⇒ *цитата**:
Quote2.Khan.A.PerspecStudyISFCBD-2015art-p.12
= [Most component integration processes suffer from inflexibility by a lack of component evaluation schemes. This problem is often compounded by a lack of interoperability standards between component frameworks and adequate vendor support.]

Maybury.M.IntellUserInterfIntro-1998art

⇒ *тип**:
статья

⇒ *ключевой знак**:
- *пользовательский интерфейс*
- *интеллектуальный пользовательский интерфейс*

⇒ *аннотация**:
[В этом введении описывается потребность в интеллектуальных пользовательских интерфейсах (IUI), определяется основная терминология, используемая в этой области. После описания теоретических основ интеллектуальных пользовательских интерфейсов в этом вводном разделе описываются современное состояние дел и обобщает структуру и содержание этой коллекции, в которой рассматриваются некоторые остающиеся фундаментальные проблемы в этой области.]

⇒ *цитата**:
Quote1.Maybury.M.IntellUserInterfIntro-1998art.p2
= [Intelligent user interfaces (IUIs) are human-machine interfaces that aim to improve the efficiency, effectiveness, and naturalness of human-machine interaction by representing, reasoning, and acting on models of the user, domain, task, discourse, and media (e.g., graphics, natural language, gesture). As a consequence, this interdisciplinary area draws upon research in and lies at the intersection of human-computer interaction, ergonomics, cognitive science, and artificial intelligence and its subareas (e.g., vision, speech and language processing, knowledge representation and reasoning, machine learning/knowledge discovery, planning and agentmodeling, user and discourse modeling).]

⇒ *цитата**:
Quote1.Maybury.M.IntellUserInterfIntro-1998art.p3-4

= [The "intelligence"in IUIs that distinguishes them from traditional interfaces... it [IUI] includes mechanisms that perform automated media analysis, design, and interaction management... Traditional user interfaces distinguish only three models: presentation, dialog, and application. Refinements beyond these three models that are found in IUIs include explicit models of the user, discourse and domain, input analysis and output generation, and mechanisms to manage the interaction, such as fusing and interpreting imprecise, ambiguous, and/or inaccurate input, controlling the dialog progression, or tailoring presentation output to the current situation. Research so far has shown that it is possible to adapt many of the fundamental concepts developed to date in computational linguistics and discourse theory in such a way that they become useful for multimedia user interfaces as well. In particular, semantic and pragmatic concepts like communicative acts, coherence, focus, reference, discourse model, user model, implicature, anaphora, rhetorical relations, and scope ambiguity take on an extended meaning in the context of multimodal communication... artificial intelligence has much to contribute to user interfaces, including the use of knowledge representations for model-based interface development tools. Architecture of Intelligent User Interfaces application of plan generation and recognition in dialog management, the application of temporal and spatial reasoning to media coordination, the use of user models to tailor interaction, and so on.]

Ehlert.P.IntellUserInterfIAS-2003art

⇒ *тип**:
статья

⇒ *ключевой знак**:
- *интеллектуальный пользовательский интерфейс*
- *IUI*
- *intelligent user interface*
- *адаптивный интерфейс*
- *взаимодействие человек-компьютер*
- *модель пользователя*
- *распознавание плана*
- *мультимодальное взаимодействие*
- *интеллектуальные агенты*
- *software usability*
- *искусственный интеллект*

⇒ *аннотация**:
[Этот отчет является частью исследования литературы по интеллектуальным пользовательским интерфейсам. Цель этого отчета состоит в том, чтобы представить область, объяснить концепции и предоставить глобальный обзор существующих приложений (исследований) с интеллектуальным пользовательским интерфейсом. Этот отчет предназначен для ученых-компьютерщиков или опытных пользователей компьютеров. Чтобы полностью понять отчет, некоторые базовые знания об искусственном интеллекте могут быть полезны.]

⇒ *цитата**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p2
= [

□ **Creating personalized systems**
No two persons are the same and people have different habits, preferences, and working methods and environment. An intelligent interface that takes these

differences into account can provide a personalized method of interaction. The interface knows the user and can use that knowledge in its communication with the user.

- **Information overflow or filtering problems**
  Finding the right information on your computer or on the Internet can be like looking for a needle in a haystack. Intelligent interfaces can reduce the information overload that often results from finding information in large databases or complex systems. By filtering out irrelevant information, the interface can reduce the cognitive load on the user. In addition, the IUI can propose new and useful information sources not known to the user.

- **Providing help on using new and complex programs**
  Computer systems can be very complicated to work with when you first start to use them. As you struggle to get to know and understand a new program, new software versions or updates may appear that include new functionality. Many computer users fail to keep up with these developments. Intelligent help systems can detect and correct user misconceptions, explain new concepts, and provide information to simplify tasks.

- **Taking over tasks from the user**
  An IUI can also look at what you are doing, understand and recognize your intent, and take over some of your tasks completely, allowing you to focus on other things.

- **Other forms of interaction**
  Currently, the most common interaction devices are the keyboard and the mouse. IUIresearch looks at other forms of human-computer interaction (e.g. speech or gestures). By providing multiple forms of interaction, people with a disability will be able to use computers more easily.

]

⇒ *цитата**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p4
= [Intelligent input technology uses innovative techniques to get input from a user. These techniques include natural language, gesture tracking and recognition, facial expression recognition, and gaze tracking among others;]

⇒ *цитата**:
Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p4
= [User modeling covers techniques that allow a system to maintain or infer knowledge about a user based on the received input]

⇒ *цитата**:
Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p4
= [User adaptivity includes all techniques that allow the communication between human and machine to be adapted to different users and different situations for example, machine learning or context awareness]

⇒ *цитата**:
Quote4.Ehlert.P.IntellUserInterfIAS-2003art.p4
= [Explanation generation covers all techniques that allow a system to explain its results to a user for example, speech output, intelligent interface agents, tactile feedback in a virtual reality environment.]

⇒ *цитата**:
Quote5.Ehlert.P.IntellUserInterfIAS-2003art.p4

= [To achieve personalization, IUIs often include a representation of a user. These user models log data about the userís behavior, knowledge, and abilities. New knowledge about the user can be inferred based on the input and interaction history of the user with the system.]

⇒ *µumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003.p5

= [An often-made mistake is to confuse an IUI with an intelligent system. A system exhibiting some form of intelligence is not necessarily an intelligent interface. There are many intelligent systems with very simple non-intelligent interfaces and the fact that a system has an intelligent interface does not say anything about the intelligence of the underlying system]

⇒ *µumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003.p8

= [

- □ An adaptive user interface must be developed in parallel with the application. This is necessary since the designer continuously needs to focus on the system parts that need to be adapted.
- □ Do not disturb the user's interaction. It should always be possible for the user to ignore the proactive actions of the IUI. Suggest rather than act.
- □ Operate in real-time. Much of the benefit of an IUI comes from acting while the user is busy working with the system.
- □ Take advantage of the user's think time. When the user is thinking about what input to provide next, the IUI can take advantage of the available processing time, so it does not risk slowing down the userís interaction with the system.
- □ Watch what the user is doing. Take advantage of ìfreeî information implicit in user actions.
- □ Allow the user to choose his personal interaction style. Different users like different interface styles and some techniques may be distracting or confusing to some users.

]

⇒ *µumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003.p9-10

= [

- □ An adaptive system is unpredictable and less transparent than a DM interface. If a system can adapt its response and does not give the same response twice given the same input, then the system becomes unpredictable. This will hinder the userís comprehension of the system, making it impossible for him to do a successful action twice.
- □ Users are not in control anymore. An IUI can make decisions for the user, thus placing the user outside the control loop.
- □ IUIs often make mistakes. Many IUI systems use trail and error to determine a userís intent or preferences. Therefore users need to give feedback to the system or even resolve the mistakes made by the system.
- □ Simulated intelligence and adaptivity increases the risk of the user thinking that computer can do things that it cannot, thus creating false expectations. Especially

with anthropomorphic agents users may believe they can interact with the IUI just as with another person.

- □ Who is responsible? If a system can make decisions on its own, who is responsible for the actions: the programmer of the system, the user, or the system itself?
- □ What about the privacy and thrust of the user? What happens to the user profile that is created and maintained by an IUI? Can you guarantee that it is safe and will not be misused?

]

⇒ *цumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Speech recognition; conversion of an input speech utterance into a string of words.]

⇒ *цumama**:
Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Language understanding; analysis of the string of words (as much as possible) to extract a meaning representation for the recognized utterance.]

⇒ *цumama**:
Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Dialogue management; controlling the interaction or dialogue between the system and the user, which includes coordination of other components of the system.]

⇒ *цumama**:
Quote4.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Database query; retrieving the information requested by the user.]

⇒ *цumama**:
Quote5.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Response generation; specification of the text that is to be the output message of the system.]

⇒ *цumama**:
Quote6.Ehlert.P.IntellUserInterfIAS-2003art.p13
= [Speech output; actual generation of the output message using text-to-speech synthesis or prerecorded sentences.]

⇒ *цumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p17
= [People use gestures a lot when talking, often subconsciously. The most often-used gestures are those associated with speech, for example pointing to an object that one is referring to. This kind of gesturing is called gesticulation. Gestures that function independent of speech are called autonomous gestures, for example sign language.]

⇒ *цumama**:
Quote2.Ehlert.P.IntellUserInterfIAS-2003.p17
= [Symbolic gestures have a (single) verbal and often cultural dependant meaning, for example the OK sign, or sign language for deaf people.]

⇒ *цumama**:
Quote3.Ehlert.P.IntellUserInterfIAS-2003.p17
= [Deictic gestures are made by pointing or motioning to direct attention to some object or event.]

⇒ *цumama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003.p18
= [Iconic gestures are gestures that display information about the size, shape or orientation

of objects, spatial relations, and actions, for example using hands to indicate the size of fish that one caught).]

⇒ *μmama**:
Quote2.Ehlert.P.IntellUserInterfIAS-2003.p18
= [Pantomimic gestures consist of manipulating an invisible imaginary object or tool, for example making a fist and moving to indicate a hammer).]

⇒ *μmama**:
Quote3.Ehlert.P.IntellUserInterfIAS-2003.p18
= [People use their eyes with very little conscious effort. Most of the time, we automatically look at the object we are working on. To see an object clearly, it is necessary to move your eyes so that the object appears on the fovea, a small area at the center of the retina. The fovea covers approximately one degree of visual arc. Because of this, a personís eye position provides a rather good indication (to within the one-degree width of the fovea) of a personís focus point of attention on a display.]

⇒ *μmama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003.p19
= [A problem with speech recognition based on an acoustic signal, is that it functions very badly when there is a lot of noise. The reason for this is that it is very hard to distinguish the speakerís voice from other sounds. With lip reading the idea is that the computer tries to determine what someone is saying just by looking at the personís lip movements, like a deaf person. By analyzing video images of the mouth and using geometric features, such as the width and height of the lips, the most probable sound (phoneme) can be determined.]

⇒ *μmama**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p20
= [With multimodal interfaces, the idea is to use multiple input channels in human-computer interaction. Instead of only speech or text as input, the system can use image recognition to look at a userís face or gestures. This way, information from one mode of interaction can complement the information received from another mode. Multimodal interfaces try to integrate speech, written text, body language, gestures, eye or lip movements and other forms of communication in order to better understand the user and to communicate more effectively. Of course, the choice of the used modalities in multimodal interfaces depends very much on the application of the system. When users can choose from multiple modalities to interact with a system, the system has the potential to be accessible to a broader range of users, for example people with disabilities. Also, multimodal user interfaces are much more robust than normal interfaces, at least in theory. Processing input from one modality can be simplified by using information from another and the user can choose the modality that is the least error prone given the circumstances.]

⇒ *μmama**:
Quote2.Ehlert.P.IntellUserInterfIAS-2003art.p20
= [The main bottleneck in multimodal interfaces is to combine the information gathered from the used modalities in real time. All multimodal interfaces need some form of input coordination or fusion system to synchronize related input.]

⇒ *μmama**:
Quote3.Ehlert.P.IntellUserInterfIAS-2003art.p20
= [Feature-level fusion is appropriate when combining two related modalities, such as speech processing and lip movements. The advantage of feature-level fusion is the improved recognition rate due to the complementary nature of both input channels (mutual disambiguation). A drawback of this tight integration of input processing is

that the system has to be retrained when the one of the input modalities is changed. In addition, training a system with multiple modalities using simultaneous input is much more difficult than training one modality at the time.]

⇒ *цитата**:
Quote1.Ehlert.P.IntellUserInterfIAS-2003art.p20-21
= [Semantic-level fusion is much easier. It does not have the benefit of direct complementary information, but a semantic-level fusion system is much easier to create and extend. Such as system can use multiple off-the-shelf recognition techniques and new modalities can easily be added later. Semantic-level integration is usually done either through unification of existing data or by looking for missing data. The latter is called frame-based integration, where the systems tries to fill missing data slots.]

⇒ *цитата**:
Quote1.Ehlert.P.IntellUSerInterfIAS-2003.p36
= [

- □ Cognitive theory and empirical science underpinnings. A better understanding is required of the unique linguistic and performance characteristics of natural communication modalities (speech, gesture, gaze and facial expressions) and of how these modalities can be combined best.
- □ New multimodal interface concepts. Current research on multimodal interfaces has focused mainly on natural language processing, pen-based gestures. Additional input such as body motion and facial expressions should be studied.
- □ Multimodal language and dialogue processing. A general theory of conversational interaction is needed that deals with intent representation for non-speech modalities.
- □ Error handling techniques. Graceful error handling is still a problem in multimodal interfaces and mutual disambiguation between signals can be improved. Also the impact of a third or more modalities on the error rate should be studied, as well as performance under difficult (mobile) environments.
- □ Adaptive multimodal architectures. Multimodal architectures are hardly adaptive and adapting to a specific user or environment can increase the recognition of input processing, as well as provide more flexibility and ease of use for the user.
- □ Multi-device multi-user ubiquitous computing. In the future, mobile computing will become more important, so we need to look at the role of multimodal interfaces in that area. In addition, when multiple users are interacting together, for example via the Internet, interfaces need to take multiple input from remote devices into account.
- □ Multimodal research infrastructure. Supporting tools for multimodal interface research should be developed. This includes; semi-automatic simulation methods for empirical data collection and prototyping of new systems, automated tools for collecting and analyzing multimodal corpora, novel metrics for evaluating multimodal systems, and software tools that support the rapid creation of next-generation multimodal applications.

]

Microsoft.WindowsDA-2011el

⇒ *тип**:
электронный ресурс
⇒ *ключевой знак**:

- *интерфейс*
- *компоненты интерфейса*

⇒ *аннотация**:

[Руководство по проектированию и примеры кода пользовательского интерфейса для создания процессов взаимодействия с приложениями для Windows.]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/apps/design]

⇒ *цитата**:

Quote1.Microsoft.WindowsDA-2011el

= [Use an AutoSuggestBox to provide a list of suggestions for a user to select from as they type.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/auto-suggest-box]

⇒ *цитата**:

Quote2.Microsoft.WindowsDA-2011el

= [A BreadcrumbBar provides the direct path of pages or folders to the current location. It is often used for situations where the user's navigation trail (in a file system or menu system) needs to be persistently visible and the user may need to go back to a previous location.]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/apps/design/controls/breadcrumbbar]

⇒ *цитата**:

Quote3.Microsoft.WindowsDA-2011el

= [The calendar date picker is a drop down control that's optimized for picking a single date from a calendar view where contextual information like the day of the week or fullness of the calendar is important. You can modify the calendar to provide additional context or to limit available dates.]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/apps/design/controls/calendar-date-picker]

⇒ *цитата**:

Quote4.Microsoft.WindowsDA-2011el

= [A calendar view lets a user view and interact with a calendar that they can navigate by month, year, or decade. A user can select a single date or a range of dates. It doesn't have a picker surface and the calendar is always visible.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/calendar-view]

⇒ *цитата**:

Quote5.Microsoft.WindowsDA-2011el

= [A color picker is used to browse through and select colors.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/color-picker]

⇒ *цитата**:

Quote6.Microsoft.WindowsDA-2011el

= [Use a combo box (also known as a drop-down list) to present a list of items that a user can select from. A combo box starts in a compact state and expands to show a list of selectable items. A list box is similar to a combo box, but is not collapsible/does not have a compact state.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/combo-box]

⇒ *цumama**:
Quote7.Microsoft.WindowsDA-2011el
= [Command bars provide users with easy access to your app's most common tasks. Command bars can provide access to app-level or page-specific commands and can be used with any navigation pattern.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/command-bar]

⇒ *цumama**:
Quote8.Microsoft.WindowsDA-2011el
= [The contact card displays contact information, such as the name, phone number, and address, for a Contact. The contact card also lets the user edit contact info.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/contact-card]

⇒ *цumama**:
Quote9.Microsoft.WindowsDA-2011el
= [Dialog controls are modal UI overlays that provide contextual app information. They block interactions with the app window until being explicitly dismissed. They often request some kind of action from the user.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/dialogs-and-flyouts/dialogs]

⇒ *цumama**:
Quote10.Microsoft.WindowsDA-2011el
= [Content links provide a way to embed rich data in your text controls, which lets a user find and use more information about a person or place without leaving the context of your app.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/content-links]

⇒ *цumama**:
Quote11.Microsoft.WindowsDA-2011el
= [The date picker gives you a standardized way to let users pick a localized date value using touch, mouse, or keyboard input.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/date-picker]

⇒ *цumama**:
Quote12.Microsoft.WindowsDA-2011el
= [The Expander control lets you show or hide less important content that's related to a piece of primary content that's always visible.]
⇒ *url**:
[https://docs.microsoft.com/en-us/windows/apps/design/controls/expander]

⇒ *цumama**:
Quote13.Microsoft.WindowsDA-2011el
= [Use a flip view for browsing images or other items in a collection, such as photos in an album or items in a product details page, one item at a time.]
⇒ *url**:
[https://docs.microsoft.com/en-US/windows/apps/design/controls/flipview]

⇒ *цumama**:

Quote14.Microsoft.WindowsDA-2011el

= [A flyout is a light dismiss container that can show arbitrary UI as its content. Flyouts can contain other flyouts or context menus to create a nested experience.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/dialogs-and-flyouts/flyouts]

⇒ *цumama**:

Quote15.Microsoft.WindowsDA-2011el

= [A form is a group of controls that collect and submit data from users. Forms are typically used for settings pages, surveys, creating accounts, and much more.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/forms]

⇒ *цumama**:

Quote16.Microsoft.WindowsDA-2011el

= [Hyperlinks navigate the user to another part of the app, to another app, or launch a specific uniform resource identifier (URI) using a separate browser app.]

⇒ *url**:

[https://docs.microsoft.com/en-us/windows/apps/design/controls/hyperlinks]

⇒ *цumama**:

Quote17.Microsoft.WindowsDA-2011el

= [Use the InkCanvas when you need to enable basic inking features in your app]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/apps/design/controls/inking-controls]

⇒ *цumama**:

Quote18.Microsoft.WindowsDA-2011el

= [Menu flyouts are used in menu and context menu scenarios to display a list of commands or options when requested by the user.]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/apps/design/controls/menus]

⇒ *цumama**:

Quote19.Microsoft.WindowsDA-2011el

= [You can show a map in light dismissable window called a map placecard or in a full featured map control.]

⇒ *url**:

[https://docs.microsoft.com/en-US/windows/uwp/maps-and-location/display-maps]

5 Предложения по развитию текущей версии Стандарта интеллектуальных компьтерных систем и технологий их разработки

Предлагаю дополнить разделы «Библиотека многократно используемых компонентов» и «Предметная область и онтология интерфейсов ostis-систем» Стандарта информацией, представленной в разделах 3.1 «Предложения к включению в раздел "Библиотека многократно используемых компонентов"» и 3.2 «Предложения к включению в раздел "Предметная область и онтология интерфейсов ostis-систем"» соответственно, а также формальной семантической спецификацией библиографических источников, описанной в разделе 4 данного отчёта.

Предложения к включению в раздел «Описание языка графического представления информационных конструкций в ostis-системах»

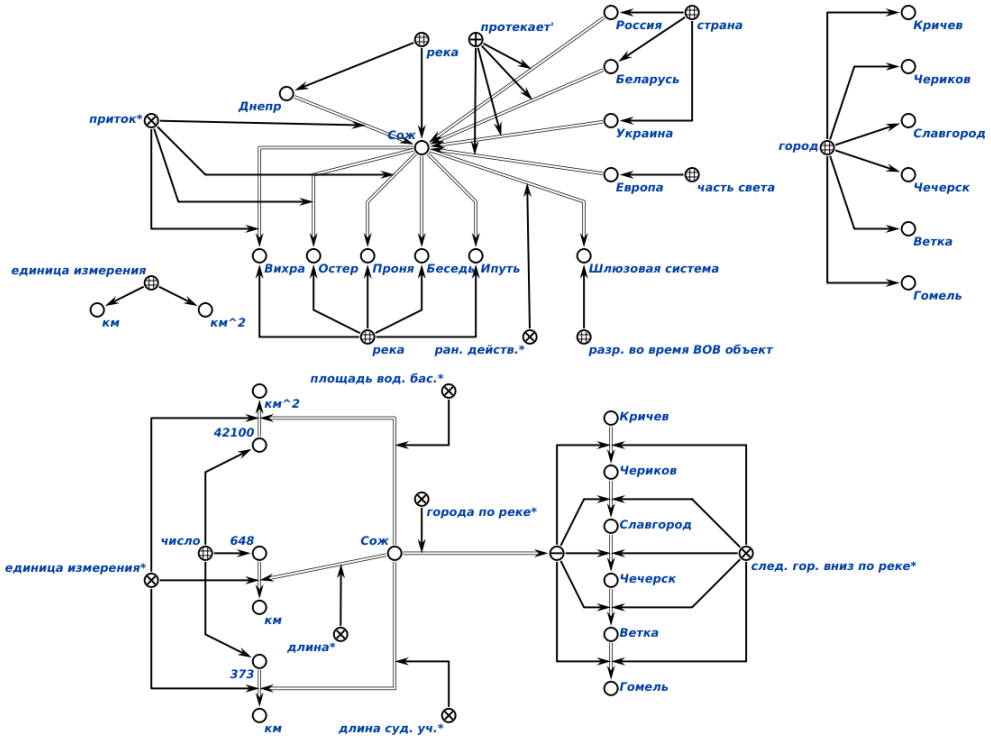
∋ *Пример формализации на SCg №1*

⇒ *исходный текст**:

[Сож — река в Европе, протекает по территории России, Беларуси и частично по границе с Украиной. Левый приток Днепра. Длина реки — 648 км (из них 493 км по Беларуси), площадь её водосборного бассейна — 42 100 км^2. Основные притоки: Вихра, Остёр, Проня, Беседь, Ипуть. Длина судоходного участка реки — 373 км. Ранее на Соже действовала шлюзованная система, разрушенная во время Великой Отечественной войны.На реке стоят следующие города: (вниз по течению): Кричев, Чериков, Славгород, Чечерск, Ветка, Гомель.]

⇒ *SCg**:

[



]

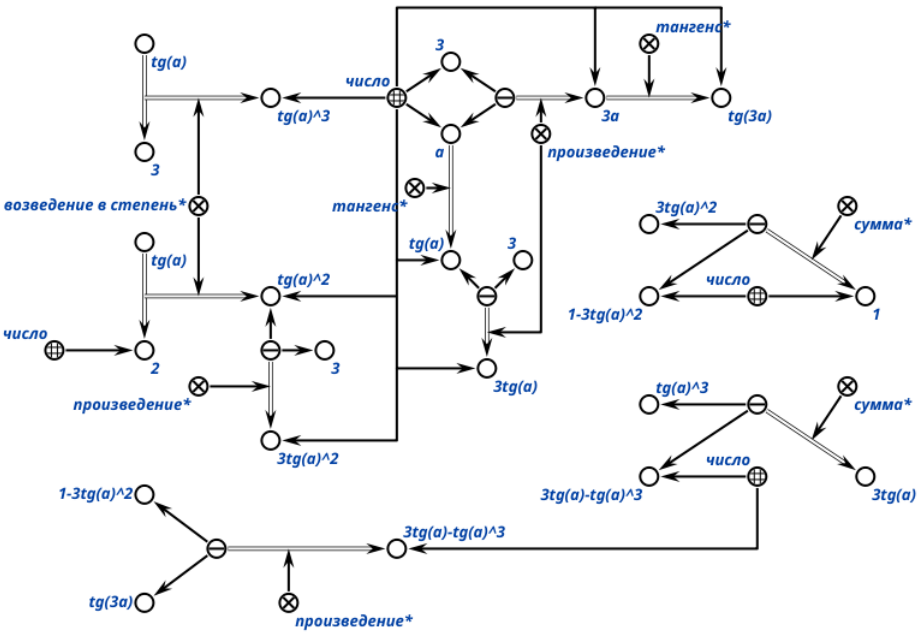
∋ *Пример формализации на SCg №2*

⇒ *исходный текст**:

[$tg(3\alpha) = \frac{3tg\alpha - tg^3\alpha}{1-3tg^2\alpha}$]

⇒ *SCg**:

[



]

Заключение

Основываясь на различных источниках были построены формализованные фрагменты теории интеллектуальных систем и технологий, соответствующие разделам Стандарта «Библиотека многократно используемых компонентов OSTIS», а также «Предметная область и онтология интерфейсов ostis-систем» с использованием языка внешнего представления баз знаний SCn (Semantic Code natural). Проведена работа по уточнению и дополнению тем, пока что не раскрытых в Стандарте OSTIS, определению и пояснению сущностей, составлена формальная семантическая спецификация библиографических источников. Также к отчёту прилагаются примеры формализации естественно-языковых текстов при помощи языка SCg (Semantic Code Graphics).

Список использованных источников

[1] Голенков, В. В. Открытая технология онтологического проектирования, производства и эксплуатации семантически совместимых гибридных интеллектуальных компьютерных систем / В. В. Голенков, Н. А. Гулякина, Д. В. Шункевич. — Бестпринт, 2021.

[2] Ehlert, Patrick. Intelligent user interfaces: Introduction and survey / Patrick Ehlert. — 2003.

[3] Government, U. S. Federal acquisition regulation. — https://www.acquisition.gov/far/2.101#FAR_2_101. — [Online; accessed 7-June-2022].

[4] Khan, Asif. A perspective study of intelligent system for component based development / Asif Khan, Md Mottahir Alam, Mohammad Shariq // International Journal of Computer Applications. — 2015. — Vol. 117. — P. 11–17.

[5] Maybury, Mark. Intelligent user interfaces: An introduction / Mark Maybury, W. Wahlster // Readings in Intelligent User Interfaces. — 1998. — P. 1–13.

[6] Microsoft,. Windows desktop applications - guidelines. — https://docs.microsoft.com/en-US/windows/apps/design. — 2011. — [Online; accessed 3-June-2022].

[7] Phister, C. Component software / C. Phister. — 1997.

[8] Szyperski, C. Component Software: Beyond Object-oriented Programming / C. Szyperski, D. Gruntz, S. Murer. ACM Press Series. — ACM Press, 2002.