(19) Евразийское патентное ведомство

(11) 046340 (13) B1

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ЕВРАЗИЙСКОМУ ПАТЕНТУ

(45) Дата публикации и выдачи патента
2024.03.01

(21) Номер заявки
202390965

(22) Дата подачи заявки
2020.11.10

(51) Int. Cl. ***G06F 8/20*** (2018.01)
G06F 8/30 (2018.01)
G06F 9/44 (2018.01)

(54) СИСТЕМА И СПОСОБ СОЗДАНИЯ И ИСПОЛНЕНИЯ ВЫСОКОМАСШТАБИРУЕМЫХ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ

(31) **2020136739**
(32) **2020.11.09**
(33) **RU**
(43) **2023.07.07**
(86) **PCT/RU2020/000595**
(87) **WO 2022/098253 2022.05.12**

(71)(73) Заявитель и патентовладелец:
ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ "КРАФТ СИСТЕМЗ" (RU)

(72) Изобретатель:
Голубев Дмитрий Алексеевич (RU)

(74) Представитель:
Котлов Д.В. (RU)

(56) US-A1-20130104100
US-B2-10324690
US-B2-7735062

(57) Изобретение относится к области разработки и исполнения программных веб-приложений. Заявлена группа объектов, а именно облачная система и способ для разработки и исполнения программных веб-приложений. Облачная система состоит из микросервисов, взаимодействующих между собой посредством программного интерфейса. Микросервисы представляют собой сервис метамодели; сервис управления метаданными; сервис кросс-компилятора метаязыка, выполненный с возможностью перевода исходного кода метаязыка, представляющего описание бизнес-логики объекта приложения, в код на языке Erlang, а также компиляции полученного кода на языке Erlang в бинарный код для виртуальной машины Erlang, который может быть загружен и исполнен сервисом исполнения приложений; сервис управления сессиями; сервис исполнителя приложений; сервис интерфейса с внешними данными, выполненный с возможностью работы с внешними базами данных и веб-сервисами; сервис внешнего API системы. Сервисы метамодели и управления сессиями выполнены с возможностью дополнения по меньшей мере одним сервисом по схеме "ведущий-ведомый" для распределения нагрузки при запросах. Технический результат заключается в повышении производительности и осуществлении горизонтального масштабирования при разработке и исполнении приложений, а также в упрощении описания бизнес-логики приложений за счет использования метаязыка.

046340 B1

046340 B1

Область техники

Настоящее техническое решение относится к разработке и исполнению программных веб-приложений. Более конкретно, изобретение относится к способам создания высоко масштабируемых облачных приложений, разработанных с использованием программных платформ.

Уровень техники

Из источника информации US 10324690 B2, опубликованного 18.06.2019, известно решение, которое описывает систему и способ для автоматического создания корпоративных программных приложений с минимальным уровнем ручного кодирования. Предпочтительный вариант осуществления предоставляет инструмент графического дизайна, который моделирует приложение с использованием Unified Model Language (UML), проверяет модель UML и автоматически генерирует развертываемое приложение. Предпочтительный вариант осуществления также предоставляет структуру библиотек, из которой может быть построено целевое приложение.

Из источника информации US 7735062 B2, опубликованного 08.06.2010, известно решение, которое описывает систему и способ разработки программных приложений. Заявленные система и способ позволяют создавать визуальные модели приложений, сохранять версии программных приложений в централизованном репозитории, автоматически генерировать и развертывать программные приложения, определять зависимости между программными приложениями, а также автоматизируют разработку нескольких зависимых программных приложений несколькими разработчиками.

Решения, известные из уровня техники, имеют ряд недостатков, а именно использование XML формата для хранения описания моделей, которое может вызывать дополнительные сложности сематического анализа связей с другими моделями и элементами, так как они могут храниться в раздельных XML фрагментах в реляционной базе данных. Известные из уровня техники решения имеют слабую масштабируемость. Также решения, известные из уровня техники, для разработки и исполнения приложений предполагают наличие определенной готовой ИТ инфраструктуры у клиента для работы сгенерированного приложения, что не подходит для использования в режиме SaaS (Software As A Service), а более подходит для разработки и использования в режиме "on-premises" (на территории клиента).

Сущность изобретения

Технической задачей является создание системы и способа для быстрой разработки высоко масштабируемых облачных приложений, которые можно использовать по модели SaaS. Для решения технической задачи были созданы облачная система и способ разработки и исполнения программных веб-приложений, охарактеризованные в независимых пунктах формулы. Дополнительные варианты реализации настоящего изобретения представлены в зависимых пунктах изобретения.

Технический результат заключается в повышении производительности и осуществлении горизонтального масштабирования при разработке и исполнении приложений, а также в упрощении описания бизнес-логики приложений за счет использования метаязыка. Дополнительно технический результат заключается в реализации назначения.

Заявленный результат достигается за счет осуществления облачной системы для разработки и исполнения программных веб-приложений, состоящей из микросервисов, взаимодействующих между собой посредством программного интерфейса, при этом микросервисы представляют собой

сервис метамодели, включающий репозиторий метаданных, и сервис интерфейса с метамоделью, при этом сервис метамоделей выполнен с возможностью дополнения по меньшей мере одним сервисом по схеме "ведущий-ведомый" для распределения нагрузки при запросах;

сервис управления метаданными, содержащий веб-интерфейс для работы с метаданными, взаимодействующий с сервисом метамоделей для получения и изменения метаданных, выполненный с возможностью предоставления удобных средств работы с метаданными, составляющими конфигурацию модулей приложений, реализации визуального редактора для построения интерфейсных форм приложения, предоставления визуального редактора исходного кода метаязыка, реализации механизмов работы с версиями описанных на метаязыке метаданных, предоставления средств перевода метаданных на другие языки для локализации приложений;

сервис кросс-компилятора метаязыка, выполненный с возможностью перевода исходного кода метаязыка, представляющего описание бизнес-логики объекта приложения, в код на языке Erlang, компиляции полученного кода на языке Erlang в бинарный код для виртуальной машины Erlang, который может быть загружен и исполнен сервисом исполнения приложений;

сервис управления сессиями, выполненный с возможностью авторизации пользователей, создания сессии, осуществления поиска сервиса исполнителя с необходимыми метаданными, инициации сервиса исполнителя нужными метаданными, создания процесса исполнителя и веб-интерфейса, дополнения по меньшей мере одним сервисом по схеме "ведущий-ведомый" для распределения нагрузки при запросах;

сервис исполнителя приложений, выполненный с возможностью загрузки и выполнения приложений, сохраненных в системном репозитории в виде набора метаобъектов сложной структуры, включая описание бизнес-логики в виде скомпилированного бинарного кода, в рамках конкретной рабочей сессии пользователя, дополнения по меньшей мере одним сервисом для распределения нагрузки при запросах;

сервис интерфейса с внешними данными, выполненный с возможностью работы с внешними база-

ми данных и веб-сервисами;

сервис внешнего API системы.

В частном варианте реализации предлагаемой системы кросс-компилятор метаязыка содержит лексер, парсер и генератор кода Erlang.

В другом частном варианте реализации предлагаемой системы сервис исполнения приложений содержит ядро исполнителя, кэш метаданных, веб-интерфейс и процессы исполнения, динамически создаваемые по запросу.

В другом частном варианте реализации предлагаемой системы сервис интерфейса с внешними данными выполнен с возможностью работы с внешними базами данных через коннекторы, учитывающие специфику соответствующего источника данных.

В другом частном варианте реализации предлагаемой системы сервис интерфейса с внешними данными выполнен с возможностью обеспечения унифицированного API работы сданными или веб-сервисами.

Заявленный результат также достигается за счет осуществления способа разработки и исполнения программных веб-приложений, содержащий этапы, на которых

описывают объекты приложений в виде метаданных, хранящихся в репозитории метаданных, посредством сервиса управления метаданными и сервиса метамодели с использованием метаязыка компилирующего типа, при этом описание объектов приложений основано на описании классов метамодели;

описанные метаданные, представленные в виде набора метаобъектов, содержащих свойства, вложенные объекты, а также исходный код метаязыка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, при этом осуществляют сравнение исходного кода метаязыка с грамматикой языка и выдают список лексем распознанного языка, осуществляют синтаксический разбор полученного списка лексем, получают синтаксическое дерево разбора лексем, осуществляют семантический анализ синтаксического дерева разбора лексем, получают преобразованный исходный код метаязыка на языке Erlang, осуществляют компилирование полученного кода на языке Erlang в исполняемый бинарный код для виртуальной машины Erlang, записывают полученный бинарный код в системный репозиторий;

после запуска полученного приложения посредством сервиса управления сессиями создают сессию и осуществляют поиск сервиса исполнителя с необходимыми метаданными, инициацию сервиса исполнителя нужными метаданными, создание процесса исполнителя и веб-интерфейс;

осуществляют запуск и выполнение бинарного кода из системного репозитория посредством сервиса исполнителя, при этом исполнение бинарного кода осуществляется непосредственно на виртуальной машине Erlang.

В частном варианте реализации предлагаемой системы метамодель содержит классы, субклассы, атрибуты, поведение, типы и субтипы.

В другом частном варианте реализации предлагаемой системы описание классов объектов включает добавление новых классов и наследование существующих классов.

Описание чертежей

Реализация изобретения будет описана в дальнейшем в соответствии с прилагаемыми чертежами, которые представлены для пояснения сути изобретения и никоим образом не ограничивают область изобретения. К заявке прилагаются следующие фигуры.

Фиг. 1 иллюстрирует обобщенную архитектуру предлагаемой облачной системы. Позиция 101 - программная платформа, 102 - платформа разработки приложений, 103 - платформа исполнения приложений, 104 - реляционная СУБД, 105 - репозиторий метаданных, 106 - метаинформация, 107 - блок описания интерфейса, 108 - блок описания бизнес-логики, 109 - блок описания данных.

Фиг. 2 иллюстрирует архитектуру предлагаемой облачной системы. Позиция 110 - пользователь системы, 111 - сервис управления метаданными, 112 - веб-интерфейс дизайнера, 113 - сервис кросс-компилятора, 114 - балансировщик, 115 - сервис управления сессиями, 116 - менеджер сессий основной, 117 - таблица сессий основная, 118 - менеджер сессий дополнительный, 119 - таблица сессий дополнительная, 120 - сервис метамодели основной, 121 - сервис метамодели дополнительный, 122 - метамодель основная, 123 - метамодель дополнительная, 124 - база данных модели основная, 125 - база данных модели дополнительная, 126 - сервис провайдера данных, 127 - провайдер данных, 28 - коннектор к базе данных, 129 - внешняя база данных, 130 - сервис веб-провайдера, 131 - веб-провайдер, 132 - веб-коннектор, 133 - внешний веб-сервис, 134 - сервис исполнителя, 135 - ядро исполнителя, 136 - кэш метаданных, 137 - процесс веб-интерфейса исполнителя, 138 - процесс исполнителя.

Фиг. 3 иллюстрирует схему сервиса исполнителя. Позиция 139 - сервис исполнителя, 140 - ядро исполнителя, 141 - основной супервизор, 142 - внешний API, 143 - основной исполнитель, 144 - супервизор исполнителей, 145 - супервизор веб-интерфейса, 146 - кэш метаданных, 147 - процесс исполнителя, 148 - процесс веб-интерфейса.

Фиг. 4 иллюстрирует обобщенную структуру метаданных. Позиция 149 - метамодель, 150 - метаданные, 151 - конфигурация, 152 - описание классов, 153 - описание атрибутов классов, 154 - описание поведения классов, 155 - описание типов, 156 - описание субтипов, 157 - описание модулей, 158 - описание метаобъектов, 159 - описание свойств объектов, 160 - описание бизнес-логики объектов, 161 - версии

метаобъектов, 162 - системные аккаунты, 163 - схемы метаданных, 164 - роли, 165 - пользователи, 166 - привилегии.

Фиг. 5 иллюстрирует структуру описания класса метамодели. Позиция 167 - основной класс, 168 - базовый основной класс, 169 - дочерний основной класс, 170 - субкласс, 171 - атрибуты класса, 172 - типы значений, 173 - субтипы атрибутов, 174 - списки значений, 175 - события класса, 176 - методы класса.

Фиг. 6 иллюстрирует структуру описания объекта метаданных. Позиция 177 - основной класс, 178 - метаобъект, 179 - атрибуты класса, 180 - события класса, 181 - методы класса, 182 - субкласс 1, 183 - субкласс 2, 184 - атрибут класса A1, 185 - событие класса E1, 186 - реализация метода класса, 187 - свойство и значение объекта для атрибута A1, 189 - реализация события объекта E1, 190 - реализация процедуры объекта P1.

Фиг. 7 иллюстрирует схему версионирования объекта метаданных. Позиция 191 - метаобъект, 192 - неизменяемые атрибуты объекта, 193 - версия 1 данных объекта, 194 - версия 2 данных объекта, 195 - значение свойства объекта версии 1, 196 - реализация события объекта версии 1, 197 - реализация процедуры объекта версии 1, 198 - значение свойства объекта версии 2, 199 - реализация события объекта версии 2, 200 - реализация процедуры объекта версии 2.

Фиг. 8 иллюстрирует схему разграничения метаинформации. Позиция 201 - аккаунт, 202 - схема 1, 203 - схема 2, 204 - модуль, 205 - объект, 206 - связи между объектами, 207 - метамодель, 208 - описание данных, 209 - субаккаунты, 210 - роли, 211 - пользователи, 212 - привилегии.

Фиг. 9 иллюстрирует процесс компиляции исходного кода метаязыка системы. Позиция 213 - компилятор метаязыка, 214 - компилятор Erlang, 215 - исходный код метаязыка, 216 - исходный код на Erlang, 217 - бинарный модуль в формате BEAM.

Фиг. 10 иллюстрирует алгоритм работы компилятора метаязыка. Позиция 218 - исходный код метаязыка, 219 - лексер, 220 - парсер, 221 - генератор кода Erlang, 222 - список лексем, 223 - синтаксическое дерево, 224 - исходный код на Erlang, 225 - ошибка лексического анализа, 226 - ошибка синтаксического разбора, 227 - ошибка компиляции.

Фиг. 11 иллюстрирует схему работы парсера метаязыка. Позиция 228 - парсер, 229' - список лексем, 229 - синтаксический разбор, 230 - проверка дубликатов переменных, 231 - проверка обращений к переменным, 232 - проверка обращений к встроенным функциям, 233 - проверка обращений к метаданным, 234 - синтаксическое дерево, 235 - контекстно-свободная грамматика, 236 - таблица имен (переменных), 237 - таблица встроенных функций, 238 - репозиторий метаданных.

Фиг. 12 иллюстрирует пример синтаксического дерева. Позиция 239 - переменная N, 240 - оператор присваивания, 241 - оператор вычитания, 242 - функция sin(), 243 - число 100, 244 - оператор умножения, 245 - число 10, 246 - оператор сложения, 247 - функция abs(), 248 - число -1, 249 - число 3.

Фиг. 13 иллюстрирует схему работы генератора кода Erlang. Позиция 250 - генератор кода, 251 - синтаксическое дерево, 252 - обход синтаксического дерева, 253 - компиляция объявлений переменных, 254 - компиляция операторов и выражений, 255 - проверка аргументов функций, 256 - генерация кода Erlang, 257 - исходный код на Erlang, 258 - семантическая проверка, 259 - таблица встроенных функций, 260 - шаблоны кода Erlang.

Фиг. 14 иллюстрирует структуру модуля Erlang. Позиция 261 - модуль Erlang, 262 - предварительные декларации, 263 - основная функция модуля, 264 - объявление модуля, 265 - объявление параметров процедуры, 266 - функции работы с переменными, 267 - функции работы с массивами, 268 - функция обработки RETURN, 269 - регистрация параметров, 270 - инициализация параметров, 271 - регистрация переменных, 272 - основной код процедуры, 273 - вызов функции RETURN.

Фиг. 15 иллюстрирует алгоритм создания сессии пользователя. Позиция 274 - запрос на создание сессии, 275 - менеджер сессий, 276 - запрос на выделение исполнителя, 277 - проверка на наличие исполнителя со схемой, 278 - проверка на наличие исполнителя с модулем, 279 - проверка на достижение максимального предела по количеству пользователей, 281 - проверка на достижение максимального предела по емкости кэша, 280 - инстанциация нового сервиса исполнителя, 282 - сервис исполнителя, 283 - инициализация схемы, 284 - загрузка модуля из метаданных, 285 - создание процесса исполнителя, 286 - создание процесса веб-интерфейса.

Фиг. 16 иллюстрирует алгоритм выполнения процедуры бизнес-логики. Позиция 287 - процесс веб-интерфейса, 288 - инициация выполнения процедуры, 289 - процесс исполнителя, 290 - проверка кода процедуры в процессе, 291 - передача процедуры на выполнение, 292 - Erlang OTP, 293 - вызов функций ядра системы, 294 - основной исполнитель, 295 - вызов встроенных функций метаязыка, 296 - загрузка кода в кэш метаданных, 297 - проверка кода процедуры в кэше, 298 - загрузка кода из кэша в процесс исполнителя, 299 - бинарный код процедуры, 300 - системный репозиторий, 301 - сервис модели, 302 - бинарный код процедуры, 303 - кэш метаданных.

Фиг. 17 иллюстрирует общую схему вычислительного устройства.

Детальное описание изобретения

В приведенном ниже подробном описании реализации изобретения приведены многочисленные детали реализации, призванные обеспечить отчетливое понимание настоящего изобретения. Однако квалифицированному в предметной области специалисту будет очевидно, каким образом можно использо-

вать настоящее изобретение как с данными деталями реализации, так и без них. В других случаях хорошо известные методы, процедуры и компоненты не были описаны подробно, чтобы не затруднять излишне понимание особенностей настоящего изобретения.

Кроме того, из приведенного изложения будет ясно, что изобретение не ограничивается приведенной реализацией. Многочисленные возможные модификации, изменения, вариации и замены, сохраняющие суть и форму настоящего изобретения, будут очевидными для квалифицированных в предметной области специалистов.

Предлагаемые способ и облачная система разработки и исполнения программных веб-приложений основаны на представлении программных приложений в виде метамоделей, хранящихся в системном репозитории, и выполняются на вычислительном устройстве.

Облачная система для разработки и исполнения программных веб-приложений состоит из микросервисов, взаимодействующих между собой посредством программного интерфейса API. Предлагаемая облачная система обеспечивает высокую производительность и возможность горизонтального масштабирования при одновременной работе большого количества пользователей.

На фиг. 1 проиллюстрирована обобщенная архитектура предлагаемой системы разработки и исполнения программных веб-приложений.

Архитектура предполагает разделение программных приложений на две большие части: программное ядро (платформу) 101 и прикладную часть (метаинформацию) 106, хранимую в репозитории 105. Метаинформация хранится в системном репозитории, организованном в реляционной системе управления базами данных (СУБД) 104. Программное ядро (платформа) 101 состоит из двух больших подсистем: платформы разработки приложений 102 и платформы исполнения приложений 103.

Платформа разработки приложений 102 отвечает за предоставление инструментов разработчику для описания модели приложений в формате платформы. Пример таких инструментов разработки: навигатор метаданных системы, редактор атрибутов метаобъектов, редактор форм интерфейса объектов, редактор процедур бизнес-логики.

Платформа исполнения приложений 103 предоставляет механизмы и ресурсы для запуска приложений по описанию в метаданных. Платформа исполнения использует метаинформацию о приложениях, хранимую в репозитории метаданных, для загрузки и выполнения приложений, включая генерацию интерфейса приложений, взаимодействие с пользователем и выполнение процедур бизнес-логики.

Описание приложений в метаинформации 106 состоит из трех основных частей: описание интерфейса бизнес-объектов 107, описание бизнес-логики объектов 108, описание данных 109. Интерфейс бизнес-объектов приложений описывается набором атрибутов и визуальных форм объектов. Бизнес-логика объектов описывается с помощью встроенного метаязыка системы, компилируемого для исполнения в бинарные модули. Описание данных приложения определяется провайдером данных и описывается набором источников данных, которые могут включать в себя различные СУБД или веб-сервисы.

На фиг. 2 проиллюстрирована подробная архитектура предлагаемой системы.

Основой для работы предлагаемой системы является сервис метамодели 120, который обеспечивает хранение и доступ к метаданным в системном репозитории. Сервис состоит из реляционной базы данных 124, включающей системный репозиторий, и собственно метамодели 122, которая обеспечивает все операции по работе с системным репозиторием и предоставляет внешний интерфейс API к репозиторию, способный обрабатывать множество параллельных асинхронных запросов. Использование API позволяет абстрагироваться от структуры данных внутри репозитория (в реляционной базе данных) и обеспечить независимость от способа хранения метаданных.

Основными функциями сервиса метамодели по работе с системным репозиторием являются

получение метаинформации по произвольным запросам к репозиторию;

ввод новой метаинформации;

изменение существующей метаинформации;

удаление метаинформации.

Метамодель обеспечивает целостность метаданных при любых операциях с метаинформацией.

Для повышения производительности системы при большом количестве запросов к репозиторию, а также для улучшения отказоустойчивости в системе предусмотрено построение конфигурации метамодели по схеме "ведущий-ведомый", при которой основной сервис ("ведущий") дополняется несколькими "ведомыми" и между всеми ними возможно распределение нагрузки при запросах. На фиг. 1 изображен "ведомый" сервис метамодели 121, который имеет также свою собственную реляционную базу данных 125, в которой находится копия системного репозитория. Базы данных основного сервиса и "ведомого" сервиса синхронизируются (реплицируются) в автоматическом режиме, при этом операции изменения метаинформации возможны только в основном сервисе, а запросы на чтение метаинформации возможны во всех сервисах. В случае если основной сервис метамодели по какой-то причине отказывает, на его место встает один из "ведомых" и становится основным, а все остальные сервисы начинают синхронизироваться с ним. В данной схеме количество "ведомых" сервисов метамодели практически неограниченно и определяется требованиями масштабирования системы.

Сервис управления метаданными 111 предназначен для обеспечения пользователя системы всеми

необходимыми ресурсами и инструментами для разработки приложений. Основной сервиса является веб-интерфейс дизайнера приложений, который предоставляет удобный интерфейс работы с метаданными для разработчика.

Веб-интерфейс дизайнера взаимодействует напрямую с сервисом метамодели через внешний интерфейс API для получения метаданных и для выполнения изменений в метаданных.

Основные функции сервиса следующие:

управление бизнес-аккаунтами и субаккаунтами, а также доступом к ним;

управление пользователями системы, ролями пользователей, списком привилегий;

управление схемами метаданных и доступом к ним;

управление метаинформацией (набором классов и их описанием, включая атрибуты и поведение);

удобная навигация по приложениям (модулям) и объектам метаданных в выбранной схеме;

предоставление редакторов объектов метаданных.

Веб-интерфейс дизайнера также реализует визуальный редактор для построения интерфейсных форм объектов, а также редактор исходного кода процедур и событий бизнес-логики.

В сервисе реализован механизм работы с версиями объектов метаданных, а также есть возможность перевода метаданных бизнес-объектов на другие языки для локализации приложений. Веб-интерфейс дизайнера также взаимодействует с сервисом кросс-компилятора 113 для перевода исходного кода процедур метаязыка в исполняемый бинарный код.

Сервис управления метаданными обеспечивает параллельную работу большого количества пользователей одновременно, выделяя для каждой сессии пользователя отдельный процесс.

Реализация сервиса управления метаданными предполагает также возможность горизонтального масштабирования при помощи запуска дополнительных сервисов, между которыми будет осуществляться распределение сессий с помощью балансировщика нагрузки 114.

Сервис кросс-компилятора 113 служит для перевода исходного кода процедур и событий бизнес-логики приложений (метаязыка) в исполняемые бинарные модули, которые могут быть загружены и исполнены сервисом исполнителя приложений и является частью системы разработки приложений вместе с сервисом управления метаданными и используется только из веб-интерфейса дизайнера. Сервис кросс-компилятора предоставляет внешний API для вызова своих методов и может обрабатывать большое количество асинхронных запросов одновременно. При необходимости горизонтального масштабирования для обслуживания еще большего количества запросов возможен запуск нескольких сервисов компилятора, запросы к которым распределяются автоматически с помощью балансировщика нагрузки 114.

Сервис кросс-компилятора 113 содержит лексер, парсер и генератор кода Erlang. На вход сервису компилятора поступает исходный текст процедур метаязыка системы, а также дополнительная информация, необходимая для компиляции, например модуль, объект. Компилятор обеспечивает полный грамматический разбор и синтаксический анализ конструкций входного языка, а также выдачу ошибок при компиляции. В процессе компиляции сервис взаимодействует с сервисом метамодели 120 для получения дополнительной метаинформации об объектах метаданных, которые используются в исходном коде процедуры, например, чтобы валидировать список и типы аргументов вызываемой процедуры приложения или объекта. В случае если сервис метамодели недоступен для запроса из компилятора, процесс компиляции не может быть завершен. При успешном завершении компиляции компилятор выдает бинарный код скомпилированной процедуры, совместимый с виртуальной машиной Erlang, который записывается сервисом управления метаданными в системный репозиторий.

Сервис управления сессиями 115 содержит менеджер сессий 116 и локальную таблицу сессий 117. Основная функция менеджера сессий - аутентификация пользователя системы и авторизация его для использования ресурсов системы. Аутентификация и авторизация выполняется для любого пользователя, который будет использовать подсистему разработки или исполнения приложений. Т.е. прежде чем пользователь сможет запустить, например, веб-интерфейс дизайнера, сервис управления сессиями должен идентифицировать пользователя по его учетным данным (например, с помощью логина и пароля). Для аутентификации сервис управления сессиями делает запрос в сервис метамодели, где хранятся все учетные записи пользователей. Если аутентификация успешна, сервис должен авторизовать пользователя для запуска веб-интерфейса дизайнера с параметрами, запрошенными пользователем, например бизнес-аккаунт, схема метаданных.

В случае если авторизация прошла успешно, сервис создает сессию для пользователя и передает сессию в веб-интерфейс дизайнера, который далее обрабатывает все входящие запросы. В случае запуска пользователем бизнес-приложения из системного репозитория сервис управления сессиями после создания сессии выделяет ресурсы исполнителя приложений и передает сессию исполнителю. Процесс выделения ресурсов исполнителя состоит из следующих этапов:

поиск сервиса исполнителя с необходимыми метаданными;

инициация дополнительного сервиса исполнителя если необходимо;

инициация сервиса исполнителя нужными метаданными;

создание процесса исполнителя и веб-интерфейса дизайнера.

После создания сессии информация о ней записывается в локальную таблицу сессий менеджера

сессий. После того как пользователь завершает работу с приложением, информация о сессии удаляется из таблицы сессий сервиса, а выделенные процессы исполнителя и веб-интерфейса дизайнера прекращают свою работу.

Для повышения производительности сервиса управления сессиями при большом количестве запросов, а также для улучшения отказоустойчивости в системе предусмотрено построение конфигурации сервиса по схеме "ведущий-ведомый", при которой основной сервис ("ведущий") дополняется несколькими "ведомыми" и между всеми ними возможно распределение нагрузки при запросах. На диаграмме изображен "ведомый" менеджер сессий 118, который имеет также свою собственную локальную таблицу сессий 119, в которой находится копия таблицы сессий основного менеджера сессий. Таблицы сессий "ведущего" и "ведомого" менеджера сессий синхронизируются в автоматическом режиме. В случае если "ведущий" менеджер сессий по какой-то причине отказывает, на его место встает один из "ведомых" и становится основным, а все остальные менеджеры начинают синхронизироваться с ним. Балансировщик нагрузки 114 служит для автоматического распределения нагрузки между сервисами управления сессиями, а также между сервисами метамодели.

Сервис исполнителя 134 является основой подсистемы исполнения приложений и обеспечивает запуск и выполнение бизнес-приложений, описание которых хранится в системном репозитории.

Основные компоненты сервиса исполнителя: ядро исполнителя 135, кэш метаданных 136, процесс веб-интерфейса 137 и процесс исполнителя 138.

Ядро исполнителя 140 (фиг. 3) состоит из основного супервизора 141 (фиг. 3), модуля внешнего API 142, основного исполнителя 143 вместе со встроенными функциями метаязыка, супервизора процессов исполнителей 144, супервизора процессов веб-интерфейса 145. Внешний API 142 предоставляет набор методов для управления и использования сервиса исполнения, включая методы для инициализации сервиса, загрузки метаданных, получения информации о сервисе и его состоянии, выполнения процедур бизнес-логики. Модуль основного исполнителя 143 реализует основные методы работы сервиса, а также содержит реализацию встроенных функций метаязыка системы. Супервизор исполнителей 144 отвечает за инстанциацию и управление процессами исполнителей 147. Супервизор веб-интерфейса 145 отвечает за инстанциацию и управление процессами веб-интерфейса 148.

Кэш метаданных 146 (фиг. 3) - это программно-управляемые структуры в памяти процесса сервиса исполнения, предназначенные для локального хранения метаинформации, необходимой для запуска и выполнения бизнес-приложений сервисом исполнения. Ядро исполнителя 140 реализует внешний API для сервиса, а также отвечает за взаимодействие с сервисом метамодели и сервисом управления сессиями. Также ядро исполнителя управляет внутренними процессами сервиса.

Кэш метаданных 146 (фиг. 3) хранит метаинформацию, связанную со схемой метаданных (например, описания провайдеров и источников данных), а также полную информацию о бизнес-объектах приложений (модулей), содержащихся в схеме. Кэш может хранить метаинформацию, принадлежащую только одной схеме метаданных. Использование локального кэша во время работы исполнителя исключает необходимость обращения за метаданными к сервису метамодели, что значительно увеличивает производительность и уменьшает интенсивность коммуникаций внутри системы.

Процесс веб-интерфейса 137 (фиг. 2) и процесс исполнителя 138 предназначены для обслуживания отдельной сессии пользователя, который работает с бизнес-приложением в системе.

Процесс исполнителя 147 (фиг. 3) порождается супервизором исполнителей 144 в ответ на запрос к сервису исполнителя со стороны сервиса управления сессиями. Процесс веб-интерфейса 148 создается супервизором веб-интерфейса 145 при создании новой сессии пользователя во время запуска бизнес-приложения на выполнение. Для каждой сессии пользователя всегда создается отдельная пара процессов исполнения и веб-интерфейса, за счет чего достигается полная изоляция выполнения бизнес-приложений для разных сессий. Сервис веб-интерфейса 148 взаимодействует со своим парным сервисом исполнителя 147 во время работы, например, при определенных действиях пользователя приложения сервис веб-интерфейса может инициировать исполнение процедуры бизнес-логики с помощью процесса исполнителя. Процесс исполнителя 147 также тесно взаимодействует с кэшем метаданных, например, когда необходимо извлечь информацию о метаобъекте во время выполнения бизнес-процедуры. При этом разные процессы исполнителя могут одновременно использовать один и тот же набор данных в кэше сервиса исполнителя, если это сессии одного и того же приложения. Таким образом, один сервис исполнителя может обслуживать большое количество сессий пользователя, если они относятся к одному и тому же набору приложений схемы метаданных.

Сервис исполнителя полностью автономен и после инициализации практически независим от других сервисов системы, что позволяет выполнять практически неограниченное горизонтальное масштабирование системы исполнения приложений для обслуживания очень большого количества одновременных сессий пользователей. Масштабирование достигается посредством автоинстанциации дополнительных сервисов исполнителей, необходимых для обслуживания новых пользователей, в зависимости от текущей загрузки системы.

Для работы приложения с внешними данными предназначены сервисы провайдеров внешних данных 126 (фиг. 2) и веб-сервисов 130. Основу сервиса провайдера составляет соответствующий провайдер

(данных 127 или веб-провайдер 131), а также коннекторы к источникам данных 128 или коннекторы к веб-сервисам 132. Каждый коннектор учитывает специфику соответствующего источника данных 129 или веб-сервиса 133 и обеспечивает работу с ними на низком уровне протокола передачи данных или API сервиса, выполняя передачу данных или команд из/в источник данных или веб-сервис. Провайдер данных или веб-сервиса обеспечивает унифицированный API работы с данными или сервисами для бизнес-приложений системы, абстрагируясь от специфики конкретных источников данных или веб-сервисов. Конфигурирование провайдеров данных веб-сервисов выполняется с помощью сервиса управления метаданными, конфигурация провайдеров хранится в метаданных в системном репозитории и может быть в любой момент скорректирована разработчиком приложения. Провайдеры взаимодействуют с процессами исполнителя сервиса исполнителя приложений, принимая от них команды на получение или изменение данных или передавая полученные данные.

Далее будет расписан способ разработки и исполнения программных веб-приложений.

Первым этапом работы предлагаемого способа является описание произвольных бизнес-объектов приложения в метаданных, хранимых в системном репозитории (метамодели) в реляционной базе данных. Описание осуществляется посредством сервиса управления метаданными и сервиса метамодели с использованием процедурного метаязыка компилирующего типа с поддержкой объектов для реализации сложной бизнес-логики приложений, при этом описание объектов приложений основано на описании классов метамодели.

Структура данных репозитория спроектирована таким образом, чтобы обеспечить максимальную гибкость и отсутствие практических ограничений на возможные способы описания бизнес-объектов, включая набор их свойств (атрибутов), поведения (методов и событий), структуры и связей. Для решения данной задачи введен уровень абстракции модели в виде классов объектов (протообъектов), которые определяют все необходимые для описания реальных бизнес-объектов структуры и данные.

Фиг. 4 описывает обобщенную структуру репозитория метаданных предлагаемой системы. Основные части репозитория: метамодель 149, метаданные 150 и конфигурация 151.

Метамодель 149 служит для описания состава и структуры протообъектов метаданных. Данный подход обеспечивает максимальную гибкость и отсутствие практических ограничений на возможные способы описания бизнес-объектов, позволяя расширять базовый набор классов метамодели при необходимости. Описание класса 152 включает в себя набор атрибутов 153 и их типов 155 (а также субтипов 156 при необходимости); поведение 154 - набор методов и событий класса; структуру класса, в случае если класс является сложным и содержит вложенные классы.

Метаданные 150 содержат описания бизнес-объектов 158, являющихся частью приложений (модулей) 157. Описание каждого объекта 158 строится на основе описания его класса 152 в метамодели 150. Набор свойств 159 и их типов объекта однозначно определяется набором атрибутов класса объекта. Значениями свойств объектов могут быть как скалярные величины простых типов, так и ссылки на другие объекты или их свойства. Набор процедур бизнес-логики 160 объекта определяется на основании поведения (методов и событий), описанных в классе. Данные объекта могут версионироваться, т.е. могут сохраняться различные варианты значений свойств объекта, созданные в разные моменты времени.

Конфигурация 151 содержит структуры, необходимые для организации всей метаинформации и обеспечения доступа к ней со стороны пользователей системы. Вся метаинформация и, в частности, конфигурация разделена по аккаунтам 162. Аккаунт представляет собой группировку информации по признаку доступа и объединяет роли, пользователей и их привилегии, а также включает в себя схемы со всеми метаданными внутри. Схема 163 - это еще один уровень разделения метаданных по семантическому признаку. Схема определяет, какая метамодель используется внутри данной схемы и, соответственно, как описываются метаобъекты в данной схеме.

Фиг. 5 иллюстрирует структуру описания класса метамодели. Классы могут быть одного из двух видов: основной класс 167 или субкласс 170. Основной класс может использоваться для описания бизнес-объектов в метаданных. Субклассы используются для формирования структуры сложных (составных или вложенных) классов и не могут самостоятельно использоваться для описания бизнес-объектов, а только в составе основных классов.

Любой класс описывается набором атрибутов 171, каждый из которых имеет определенный тип 172, а также возможно субтип 173. Тип значения атрибута может быть простым скалярным типом (например, числовым, текстовым) или являться ссылкой на элемент списка значений 174, другой основной класс 168 или атрибут основного класса. Набор атрибутов однозначно определяет, какие свойства бизнес-объекта данного класса должны быть описаны и сохранены в метаданных.

Кроме атрибутов, у класса могут быть описаны методы 176 и события 175. Совокупность методов и событий класса определяет его поведение. Метод класса - это именованная предопределенная процедура, которая определяет операции, возможные для объекта данного класса. Пример методов класса: создать новый элемент, записать данные, удалить элемент. Реализация метода возложена на сам класс, т.е. процедуры методов должны быть заранее определены для каждого класса. Процедуры методов вызываются на выполнение в результате взаимодействия пользователя с приложением либо программным способом (из других процедур бизнес-логики). Событие класса - это именованная процедура класса, которая опре-

деляется для бизнес-объекта в метаданных. Для каждого класса имеется свой набор событий, который однозначно определяет, какие события могут быть определены для бизнес-объекта данного класса, т.е. реализация процедуры события делается в бизнес-объекте, а не в классе. Процедуры событий объектов вызываются на выполнение в результате взаимодействия пользователя с приложением при наступлении определенных условий (например, двойной щелчок левой кнопки мыши).

Для описания сложных (вложенных) классов применяется структура класса, которая является иерархическим набором ссылок на субклассы, которые составляют данный класс. Иерархия здесь означает, что каждый используемый субкласс в структуре, в свою очередь, может также иметь вложенные субклассы. В структуре класса нельзя использовать классы основного вида, допускается использование только субклассов. Структура класса определяет, из каких составных частей состоит бизнес-объект и как будет строиться представление объекта в метаданных. В качестве примера можно привести основной класс "Документ", в составе которого может находиться субкласс "Форма". Используя данную структуру, возможно представить бизнес-объект класса "Документ", как совокупность объектов форм, определяющих визуальное представление данного объекта.

Расширение метамодели возможно двумя способами: добавление новых классов или наследование из уже существующего класса. Для вновь создаваемого класса необходимо определить набор атрибутов, событий и методов, а также структуру. Как только новый класс описан, становится возможным создание бизнес-объектов данного класса. Наследование используется, когда новый класс в значительной степени повторяет уже существующий класс с небольшими изменениями. При наследовании дочерний класс получает все атрибуты, методы и события, а также структуру родительского класса, при этом возможно добавить собственные атрибуты, методы, события и дополнить структуру класса. Переопределение атрибутов и методов, а также структуры родительского класса, запрещено в дочернем классе.

Фиг. 6 иллюстрирует способ описания метаобъекта метаданных на основе определения класса метамодели. В данном примере у класса 171 определены два атрибута (179): атрибут A1 184 и атрибут A2. В соответствии сданным определением при создании описания метаобъекта 178, принадлежащего данному классу, должны быть введены значения свойств объекта 187, соответствующих атрибутам A1 и A2. При этом тип значения, например, для свойства объекта A1 187 должен соответствовать типу и субтипу атрибута класса A1 184. В метаобъекте можно (но необязательно) определить реализацию процедур для событий класса 180. При этом сигнатуры процедур объекта (наименование процедуры, тип возвращаемого значения, список и типы аргументов) должны полностью соответствовать описанию событий в классе. Например, реализация события объекта E1 189 должна соответствовать описанию события класса E1 185. В объекте можно определить реализацию только тех событий, которые были описаны в классе. Реализация процедур событий объекта происходит посредством использования объекта субкласса "событие объекта" 182, определенного в структуре класса. Как уже было описано выше, методы класса 181 должны быть реализованы непосредственно в классе. В данном примере у класса определены два метода 186: метод 1 и метод 2. Наконец, согласно определению класса в данном примере в метаобъекте возможно (но необязательно) определить сколько угодно процедур бизнес-логики. Процедуры заранее не описываются в классе. Реализация процедур объекта происходит посредством использования объекта субкласса "процедура объекта" 183, определенного в структуре класса.

Механизм хранения объектов в репозитории метаданных предполагает возможность версионирования (необязательно). Фиг. 7 описывает схему версионирования данных метаобъекта. При этом все данные метаобъекта 191 разделяются на неверсионируемую и версионируемую части.

Неверсионируемая часть объекта содержит неизменяемые атрибуты 192, например такие, как класс, ссылка на родительский объект, и используется для реализации ссылок на объекты и установления связей между объектами. Данные объекта, которые могут изменяться в результате разработки (доработки) объекта, вынесены в версионируемую часть 193, 194.

Версионируемая часть содержит значения свойств объекта 195, реализацию событий объекта 196, реализацию процедур объекта 197. Версией объекта называется совокупность версионируемых данных объекта, сохраненных в репозитории метаданных с уникальным идентификатором (номером) версии.

В случае если система работает в режиме без версионирования, то все изменения в данных объекта записываются с номером версии 0 (старые данные при этом не сохраняются). Если включен режим версионирования, то измененные данные объекта 194 записываются с новым номером версии, при этом старые данные объекта 193 также сохраняются с предыдущим номером версии.

Для того чтобы начать изменения объекта в режиме версионирования, необходимо выполнить операцию "check-out" (взятие объекта на редактирование), при этом текущая (актуальная) версия объекта блокируется для изменения другими пользователями и создается новая версия объекта, куда и вносятся изменения. Как только изменения завершены, выполняется операция "check-in", в результате чего новая версия объекта становится актуальной, а предыдущая версия разблокируется.

Каждая версия метаобъекта может принадлежать той или иной ветке. В рамках одной ветки все версии объекта сохраняются последовательно, однако в разных ветках могут сохраняться параллельно разные версии объектов. Только одна версия в рамках одной ветки может быть актуальной (это последняя версия в ветке). Кроме признака актуальной версии, версии могут быть отмечены признаком "рабо-

чей" версии. Рабочая версия - это стабильная и протестированная версия объекта, которая может быть использована для исполнения в приложении. Можно установить только одну рабочую версию объекта (она необязательно должна совпадать с актуальной).

Предлагаемые облачная система и способ могут работать в режиме SaaS. Для функционирования системы разработки и исполнения приложений в режиме одновременного ее использования многими пользователями необходимо обеспечить надежный способ разграничения метаинформации, включая метамодель, метаданные и конфигурацию. Фиг. 8 поясняет способ разграничения метаинформации в системе.

Основой разграничения является сущность, называемая "аккаунт" 201. Под аккаунтом здесь понимается неличная учетная запись зарегистрированного бизнес-пользователя системы. В рамках основного аккаунта можно создавать субаккаунты 209, которые являются подчиненными к основному аккаунту. Аккаунт должен содержать как минимум один личный профиль пользователя 211, при этом количество профилей пользователя не ограничено. Каждому профилю пользователя может быть назначена определенная роль 210 и набор привилегий 212. Важной особенностью данного способа разграничения является то, что пользователи, их роли и привилегии определяются изолированно в рамках аккаунта, т.е. являются независимыми от пользователей, ролей и привилегий другого аккаунта.

Следующим уровнем разграничения метаинформации является сущность, обозначенная как "схема" 202, 203. Схема объединяет взаимосвязанную информацию метамодели и метаданных, такую как описание классов, данных и метаобъектов. При данном подходе каждая схема содержит собственную метамодель (описание классов или протообъектов), что обеспечивает независимость от других схем даже в рамках одного аккаунта и позволяет безопасно расширять или изменять метамодель без риска непредсказуемого изменения описания бизнес-объектов другой схемы. Схема служит также в качестве "контейнера" приложений, разрабатываемых и исполняемых на платформе, и обозначаемых термином "модуль" 204. Приложение (модуль) является совокупностью бизнес-объектов 205, каждый из которых может принадлежать только одному модулю. В целях обеспечения возможности переиспользования бизнес-объектов одного модуля в других модулях задействуется механизм связей 206 между модулями посредством создания ссылок на объекты. Связи (ссылки на объекты) можно создавать только между модулями в рамках одной схемы. Схема также содержит описание источников внешних данных 208, которые нужны для работы приложений. Источники данных могут использоваться любым приложением (модулем) в рамках одной схемы. Таким образом, модули группируются по схемам по функциональному признаку (связанные бизнес-объекты) и по принципу использования общих данных.

Как уже было упомянуто ранее, разработка программных веб-приложений основана на описании произвольных бизнес-объектов приложения в метаданных посредством процедурного метаязыка компилирующего типа с поддержкой объектов для реализации сложной бизнес-логики приложений. Грамматика языка построена таким образом, что позволяет непосредственно работать с объектами метаданных приложений, а также эффективно использовать данные из внешних источников. Весь код бизнес-логики приложений разбивается на отдельные процедуры или обработчики событий и привязан к бизнес-объектам в метаданных. Это позволяет использовать контекст метаобъекта в самой процедуре, например ссылку на текущую форму данных объекта, и получить прямой доступ к данным в форме. Таким образом, с помощью средств языка можно в режиме исполнения манипулировать данными, с которыми работает пользователь приложения, и даже вмешиваться в ход обработки данных или в поведение метаобъекта.

Общая структура грамматики метаязыка представлена тремя основными блоками: синтаксические элементы языка, организация данных и обработка данных.

Синтаксические элементы языка включают синтаксические конструкции, например комментарии (//, /**/), литералы (1, 's', "s"), скобки ([],(),{}), а также ключевые слова, которые используются при написании кода на метаязыке.

Организация данных включает информацию о типах данных (any, integer, string и т.д.), структуре хранения данных и объектах. Метаязык является строго типизируемым языком. Это означает, что значения разных типов не могут использоваться в выражениях без приведения типа, а также в качестве аргументов функции могут передаваться только значения соответствующего определению функции типа.

Обработка данных происходит посредством использования арифметических операций (+, ++, -, --, *, /, ^), операторов присвоения (=, +=, -=, *=, /=), операторов ветвления (IF, THEN, ELSE, CHOOSE, CASE, RETURN), логических операций (=, <>, >, <, >=, <=, NOT, AND, OR, ANDALSO, ORELSE), операторов циклов (FOR - NEXT, DO - LOOP, CONTINUE, EXIT), обработки ошибок (THROW, TRY - CATCH, FINALLY), работы с объектами (CREATE, DESTROY, FUNCTION, EVENT, CALL), операторов SQL (CONNECT, DISCONNECT, SELECT, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK, OPEN, CLOSE, FETCH, EXECUTE), а также встроенных функций (встроенные функции, функции преобразования, строковые функции, числовые функции, функции даты-времени, функции массивов, разные функции).

Язык поддерживает вызовы других процедур метамоделей, а также рекурсивные вызовы.

Второй этап предлагаемого способа заключается в том, что описанные метаданные, представленные в виде набора метаобъектов, содержащих свойства, вложенные объекты, а также исходный код метаязыка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, который впо-

следствии компилируют в исполняемый бинарный код для виртуальной машины Erlang (BEAM), посредством сервиса кросс-компилятора.

Фиг. 9 иллюстрирует обобщенный процесс компиляции исходного кода процедур метаязыка системы в бинарные файлы BEAM. На входе компилятору поступает исходный код метаязыка 215. Компилятор метаязыка 213 анализирует его и преобразует в текст, полностью удовлетворяющий синтаксису языка Erlang и реализующий ту же самую логику. Компилятор метаязыка преобразует синтаксические конструкции и вызовы функции в соответствующие конструкции и вызовы функций Erlang, получая результирующий код на языке Erlang 216. Далее используется стандартный компилятор языка Erlang 214 для преобразования исходного кода Erlang в бинарный модуль в формате BEAM 217.

Далее со ссылкой на фиг. 10 будет описана работа кросс-компилятора метаязыка.

Как уже было указано выше, кросс-компилятор состоит из 3 основным модулей: лексер 219, парсер 220 и генератор кода Erlang 221.

На вход лексеру 219 поступает текст исходного кода метаязыка 218, лексер 219 сверяет входной поток с грамматикой языка и в результате анализа выдает список распознанных лексем языка 222.

Лексер 219 выполняет анализ входного потока исходного кода метаязыка на основе правил грамматики. Каждый вид лексемы описан в виде шаблона - регулярного выражения. Лексер 219 различает игнорируемые символы (пробел, символ завершения строки, комментарий), строковые и числовые литералы, значения даты и времени, идентификаторы, операторы сравнения, логические операторы, арифметические операторы, скобки, ключевые слова. Лексемы, которые продуцирует лексер, представлены в виде кортежей. Формат лексемы, возвращаемой лексером, следующий:

{лексема, номер строки, символ(ы)}

В случае если в процессе анализа исходного кода лексер не может распознать лексему, он выдает ошибку 225 с указанием номера строки исходного кода и процесс компиляции останавливается.

Если все лексемы из входного потока разобраны корректно, то лексер выдает список разобранных лексем и процесс компиляции продолжается. Список лексем поступает на вход парсеру 220 и он осуществляет синтаксический разбор лексем, сверяя лексемы и их порядок с определением синтаксиса языка.

Парсер 228 (фиг. 11) осуществляет синтаксический разбор списка лексем 229 из входного потока, преобразовывая их в синтаксическое дерево 234, на основе определения правил грамматики. Синтаксис конструкций метаязыка задан с помощью контекстно-свободной грамматики 235 или записи BNF (Backus-Naur Form) в виде правил грамматического вывода (продукций). Кроме проверки соответствия порядка лексем правилам грамматического вывода (синтаксису), парсер выполняет еще несколько следующих важных действий:

1) ведет таблицу переменных 236;

2) проверяет дубликаты при объявлении переменных 230;

3) проверяет что переменная декларирована перед использованием;

4) проверяет размерность массивов при инициализации во время объявления;

5) проверяет правильность обращения к переменной и элементу массива 231;

6) валидирует обращение к встроенным функциям, включая количество аргументов 232;

7) валидирует обращение к объектам метаданных приложения 233.

Если в процессе разбора обнаруживается некорректная синтаксическая конструкция, парсер 220 выдает сообщение об ошибке 226 с указанием номера строки исходного кода и процесс компиляции останавливается. Если синтаксический разбор списка лексем прошел успешно, то парсер выдает синтаксическое дерево разбора лексем 223, которое поступает на вход генератора кода Erlang 221.

Фиг. 12 иллюстрирует способ представления следующего арифметического выражения в виде синтаксического дерева после обработки парсером метаязыка:

n = (abs(-1) + 3) * 10 - sin(100),

Результатом выполнения лексического анализа и синтаксического разбора является следующее синтаксическое дерево кортежей:

```
{2,stmt_assign,
{2,var,number,local,write,n,[]},
op_eq,
{2,op_minus,
 {2,op_mul,
  {2,op_plus,
   {2,function,number,abs,[number],[{2,unary_minus,{2,integer,1}}]},
   {2,integer,3}},
  {2,integer,10}},
 {2,function,number,sin,[number],[{2,integer,100}]}}}.
```

Генератор кода 221 выполняет нисходящий обход синтаксического дерева 223, осуществляя семантический анализ и различные дополнительные проверки, например соответствие типов в выражениях.

Для каждой синтаксической конструкции генератор кода выполняет преобразование ее в код целевого языка (Erlang).

Генератор кода 250 (фиг. 13) получает на вход синтаксическое дерево кортежей 251 и на выходе генерирует исходный код на языке Erlang 257. Генератор кода осуществляет нисходящий обход 252 синтаксического дерева, выполняя семантические проверки 258 на каждом шаге. В процессе обхода синтаксического дерева генератор выполняет компиляцию объявлений переменных 253 или компиляцию операторов и выражений 254. Семантические проверки при компиляции объявлений переменных и массивов включают проверку выражений инициализации. Семантические проверки при компиляции операторов и выражений включают в себя дополнительные проверки синтаксиса, а также проверку типов выражений, соответствие типов операндов выражений. При компиляции вызовов функций 285 все передаваемые фактические аргументы также проверяются на соответствие типам аргументов функции по таблице встроенных функций 259. Генерация кода Erlang 256 производится на основании заранее заданных шаблонов кода Erlang 260.

Если в процессе обхода синтаксического дерева 223 генератор кода 221 обнаруживает ошибку (например, несоответствие типов в выражении), он выдает ошибку компиляции 227 с указанием номера строки исходного кода и процесс останавливается.

В случае если обход синтаксического дерева 223 завершился успешно, то на выходе генератора кода 221 образуется исходный код на языке Erlang 224, который далее можно использовать для компиляции в бинарный формат.

Модуль сгенерированного кода на языке Erlang 261 представлен на фиг. 14. Весь код модуля состоит из двух больших частей: предварительных деклараций 262 и основной функции модуля 263. Предварительные декларации включают в себя объявление модуля 264, объявление параметров процедуры 265, функции работы с переменными 266, функции работы с массивами 267, функция обработки оператора RETURN 268. Объявление модуля содержит имя процедуры метаязыка, которое транслируется в модуль Erlang. Объявление параметров содержит список идентификаторов и типов параметров процедуры метаязыка, которые транслируется в модуль Erlang. Функции работы с переменными - это вспомогательные функции, которые служат для получения или установки значения локальных переменных процедуры в модуле. Функции работы с массивами - это вспомогательные функции, которые служат для получения или установки значения локальных массивов процедуры в модуле, включая обращение к элементам массивов. Функция обработки оператора RETURN необходима для корректного завершения основной функции модуля и передачи возвращаемого значения процедуры, а также для установки значения параметров, переданных по ссылке.

Основная функция модуля начинается с регистрации параметров 269 процедуры в таблице переменных модуля. Далее следует вызов функции инициализации параметров 270. Функция инициализации 270 отвечает за установку значений параметров по фактическим аргументам, переданным в процедуру (в данном случае в основную функцию модуля Erlang) при ее вызове. Затем идет секция регистрации локальных переменных 271 процедуры в таблице переменных модуля. При регистрации переменные всегда инициализируются значением либо тем, которое указано при объявлении переменной, либо значением по умолчанию для конкретного типа переменной (например, для строковой переменной - это пустая строка, для числовой переменной - 0). Далее следует секция основного кода 272, который генерируется при компиляции исходного кода процедуры метаязыка. В конце функции модуля вызывается функция обработки оператора RETURN 273. Данная функция вызывается всегда независимо от того, была ли указана данная функция в исходном коде процедуры метаязыка или нет, и даже в том случае, если процедура метаязыка не возвращает значения. Функция обработки RETURN 268 устанавливает значения входных параметров, переданных по ссылке, удаляет таблицу переменных модуля и передает в вызывающую функцию возвращаемое значение процедуры (если оно есть).

Полученный исполняемый бинарный код записывают в системный репозиторий.

Третий этап работы предлагаемого способа заключается в том, что после запуска полученного приложения, посредством сервиса управления сессиями создают сессию и осуществляют поиск сервиса исполнителя с необходимыми метаданными, инициацию сервиса исполнителя нужными метаданными, создание процесса исполнителя и веб-интерфейс.

При получении запроса на создание новой сессии 274 (фиг. 15), сервис управления сессий 275 пытается выделить исполнитель для данной сессии, на основе параметров идентификатора схемы и идентификатора модуля (приложения) согласно описанному далее алгоритму. Происходит поиск работающего сервиса исполнителя с указанной схемой 276. Если сервиса исполнителя с нужной схемой нет, то инстанциируется новый сервис исполнителя 280, 282, а затем в сервисе инициализируется нужная схема 283. Если сервис с нужной схемой найден, то далее проверяется - загружен ли в данном сервисе в кэш метаданных нужный модуль 278. Если модуль загружен, то следующая проверка определяет, не превышен ли допустимый предел по количеству одновременно обслуживаемых пользователей в сервисе 279. Если предел уже достигнут, то также выполняется инстанциация нового сервиса исполнителя и инициализация схемы. Если предел по количеству пользователей не достигнут, то в данном сервисе просто создаются процессы исполнителя и веб-интерфейс для сессии, так как модуль в сервисе уже загружен. В случае

же если на предыдущем шаге при проверке модуля оказалось, что он не загружен, то необходимо проверить, не превышен ли в сервисе максимально допустимый предел по емкости кэша 281. Если предел достигнут, то необходимо инстанциировать новый сервис исполнителя. Если же нет, то достаточно загрузить модуль в кэш метаданных сервиса и далее создать процессы исполнителя и веб-интерфейс для сессии. На этом процесс создания сессии завершается.

Заключительный шаг работы предлагаемого способа заключается в том, что осуществляют запуск и выполнение бинарного кода из системного репозитория посредством сервиса исполнителя, при этом исполнение бинарного кода осуществляется непосредственно на виртуальной машине Erlang.

На фиг. 16 процесс веб-интерфейса 287 является инициатором вызова процедуры на выполнение 288, для чего он обращается с запросом в процесс исполнителя 289. Процесс исполнителя проверяет, загружен ли бинарный модуль (бинарный код) процедуры в процесс исполнителя 290. Если бинарный код не загружен, процесс исполнителя обращается в локальный кэш метаданных 303, который проверяет, загружен ли бинарный код процедуры в кэш 297. Если бинарный код не загружен в кэш, то сервис исполнителя обращается с запросом к сервису модели 301 для загрузки его из системного репозитория 300, после чего кэш метаданных может вернуть требуемый код процессу исполнителя. Как только бинарный код процедуры (бинарный модуль) загружен в процесс исполнителя, он передает его на исполнение виртуальной машине Erlang 291. С этого момента исполнение бинарного модуля целиком и полностью контролируется платформой Erlang OTP 292 и кодом бинарного модуля 299. Тем не менее в коде бинарного модуля могут быть вызовы функций ядра системы 293 или вызовы встроенных функций метаязыка 295, которые обрабатываются основным исполнителем сервиса 294.

Фиг. 17 иллюстрирует общую схему вычислительного устройства 1700, обеспечивающего обработку данных, необходимую для реализации заявленного решения.

В общем случае устройство 1700 содержит такие компоненты, как один или более процессоров 1701, по меньшей мере одну память 1702, средство хранения данных 1703, интерфейсы ввода/вывода 1704, средство В/В 1705, средства сетевого взаимодействия 1706.

Процессор 1701 устройства выполняет основные вычислительные операции, необходимые для функционирования устройства 1700 или функциональности одного или более его компонентов. Процессор 1701 исполняет необходимые машиночитаемые команды, содержащиеся в оперативной памяти 1702.

Память 1702, как правило, выполнена в виде ОЗУ и содержит необходимую программную логику, обеспечивающую требуемый функционал.

Средство хранения данных 1703 может выполняться в виде HDD, SSD дисков, рейд массива, сетевого хранилища, флэш-памяти, оптических накопителей информации (CD, DVD, MD, Blue-Ray дисков) и т.п. Средство 1703 позволяет выполнять долгосрочное хранение различного вида информации, например вышеупомянутых файлов с наборами данных пользователей, базы данных, содержащих записи измеренных для каждого пользователя временных интервалов, идентификаторов пользователей и т.п.

Интерфейсы 1704 представляют собой стандартные средства для подключения и работы с серверной частью, например USB, RS232, RJ45, LPT, COM, HDMI, PS/2, Lightning, FireWire и т.п.

Выбор интерфейсов 1704 зависит от конкретного исполнения устройства (N00), которое может представлять собой персональный компьютер, мейнфрейм, серверный кластер, тонкий клиент, смартфон, ноутбук и т.п.

В качестве средств В/В данных 1705 в любом воплощении системы, реализующей описываемый способ, должна использоваться клавиатура. Аппаратное исполнение клавиатуры может быть любым известным, например, это может быть как встроенная клавиатура, используемая на ноутбуке или нетбуке, так и обособленное устройство, подключенное к настольному компьютеру, серверу или иному компьютерному устройству. Подключение при этом может быть как проводным, при котором соединительный кабель клавиатуры подключен к порту PS/2 или USB, расположенному на системном блоке настольного компьютера, так и беспроводным, при котором клавиатура осуществляет обмен данными по каналу беспроводной связи, например радиоканалу, с базовой станцией, которая, в свою очередь, непосредственно подключена к системному блоку, например к одному из USB-портов. Помимо клавиатуры, в составе средств В/В данных также может использоваться: джойстик, дисплей (сенсорный дисплей), проектор, тачпад, манипулятор мышь, трекбол, световое перо, динамики, микрофон и т.п.

Средства сетевого взаимодействия 1706 выбираются из устройства, обеспечивающий сетевой прием и передачу данных, например Ethernet карту, WLAN/Wi-Fi модуль, Bluetooth модуль, BLE модуль, NFC модуль, IrDa, RFID модуль, GSM модем и т.п. С помощью средств 1705 обеспечивается организация обмена данными по проводному или беспроводному каналу передачи данных, например WAN, PAN, ЛВС (LAN), Интранет, Интернет, WLAN, WMAN или GSM.

Компоненты устройства 1700 сопряжены посредством общей шины передачи данных 1710.

В настоящих материалах заявки было представлено предпочтительное раскрытие осуществление заявленного технического решения, которое не должно использоваться как ограничивающее иные, частные воплощения его реализации, которые не выходят за рамки испрашиваемого объема правовой охраны и являются очевидными для специалистов в соответствующей области техники.

ФОРМУЛА ИЗОБРЕТЕНИЯ

1. Облачная система для разработки и исполнения программных веб-приложений, выполняющаяся на вычислительном устройстве, содержащем процессор и память, хранящую инструкции, исполняемые процессором, и состоящая из микросервисов, взаимодействующих между собой посредством программного интерфейса, при этом микросервисы представляют собой

сервис метамодели, включающий репозиторий метаданных, и сервис интерфейса с метамоделью, при этом сервис метамоделей выполнен с возможностью дополнения по меньшей мере одним сервисом по схеме "ведущий-ведомый" для распределения нагрузки при запросах;

сервис управления метаданными, содержащий веб-интерфейс для работы с метаданными, взаимодействующий с сервисом метамоделей для получения и изменения метаданных, выполненный с возможностью предоставления удобных средств работы с метаданными, составляющими конфигурацию модулей приложений, реализации визуального редактора для построения интерфейсных форм приложения, предоставления визуального редактора исходного кода метаязыка, реализации механизмов работы с версиями описанных на метаязыке метаданных, предоставления средств перевода метаданных на другие языки для локализации приложений;

сервис кросс-компилятора метаязыка, выполненный с возможностью перевода исходного кода метаязыка, представляющего описание бизнес-логики объекта приложения, в код на языке Erlang, компиляции полученного кода на языке Erlang в бинарный код для виртуальной машины Erlang, который может быть загружен и исполнен сервисом исполнения приложений;

сервис управления сессиями, выполненный с возможностью авторизации пользователей, создания сессии, осуществления поиска сервиса исполнителя с необходимыми метаданными, инициации сервиса исполнителя нужными метаданными, создания процесса исполнителя и веб-интерфейса, дополнения по меньшей мере одним сервисом по схеме "ведущий-ведомый" для распределения нагрузки при запросах;

сервис исполнителя приложений, выполненный с возможностью загрузки и выполнения приложений, сохраненных в системном репозитории в виде набора метаобъектов сложной структуры, включая описание бизнес-логики в виде скомпилированного бинарного кода, в рамках конкретной рабочей сессии пользователя, дополнения по меньшей мере одним сервисом для распределения нагрузки при запросах;

сервис интерфейса с внешними данными, выполненный с возможностью работы с внешними базами данных и веб-сервисами;

сервис внешнего API системы.

2. Система по п.1, отличающаяся тем, что кросс-компилятор метаязыка содержит лексер, парсер и генератор кода Erlang.

3. Система по п.1, отличающаяся тем, что сервис исполнителя приложений содержит ядро исполнителя, кэш метаданных, веб-интерфейс и процессы исполнения, динамически создаваемые по запросу.

4. Система по п.1, отличающаяся тем, что сервис интерфейса с внешними данными выполнен с возможностью работы с внешними базами данных через коннекторы, учитывающие специфику соответствующего источника данных.

5. Система по п.1, отличающаяся тем, что сервис интерфейса с внешними данными выполнен с возможностью обеспечения унифицированного API работы с данными или веб-сервисами.

6. Компьютерно-реализуемый способ разработки и исполнения программных веб-приложений, выполняющийся на вычислительном устройстве, которое содержит процессор и память, хранящую инструкции, исполняемые процессором и содержащие этапы, на которых

описывают объекты приложений в виде метаданных, хранящихся в репозитории метаданных, посредством сервиса управления метаданными и сервиса метамодели с использованием метаязыка компилирующего типа, при этом описание объектов приложений основано на описании классов метамодели;

описанные метаданные, представленные в виде набора метаобъектов, содержащих свойства, вложенные объекты, а также исходный код метаязыка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, при этом осуществляют сравнение исходного кода метаязыка с грамматикой языка и выдают список лексем распознанного языка, осуществляют синтаксический разбор полученного списка лексем, получают синтаксическое дерево разбора лексем, осуществляют семантический анализ синтаксического дерева разбора лексем, получают преобразованный исходный код метаязыка на языке Erlang, осуществляют компилирование полученного кода на языке Erlang в исполняемый бинарный код для виртуальной машины Erlang, записывают полученный исполняемый бинарный код в системный репозиторий;

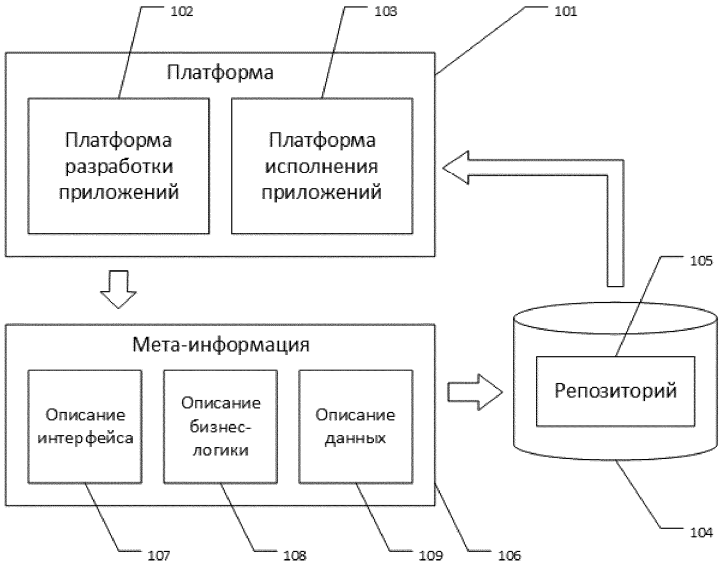
после запуска полученного приложения посредством сервиса управления сессиями создают сессию и осуществляют поиск сервиса исполнителя с необходимыми метаданными, инициацию сервиса исполнителя нужными метаданными, создание процесса исполнителя и веб-интерфейс;

осуществляют запуск и выполнение бинарного кода из системного репозитория посредством сервиса исполнителя, при этом исполнение бинарного кода осуществляется непосредственно на виртуальной машине Erlang.

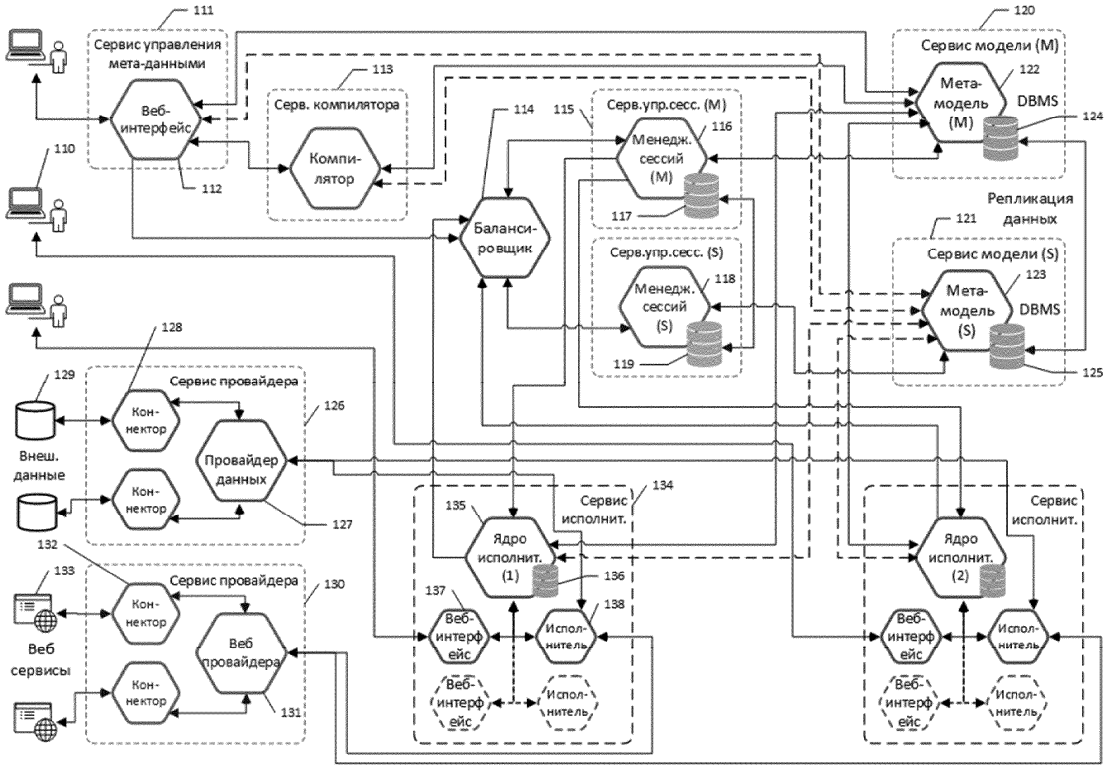
7. Способ по п.6, отличающийся тем, что метамодель содержит классы, субклассы, атрибуты, пове-

дение, типы и субтипы.

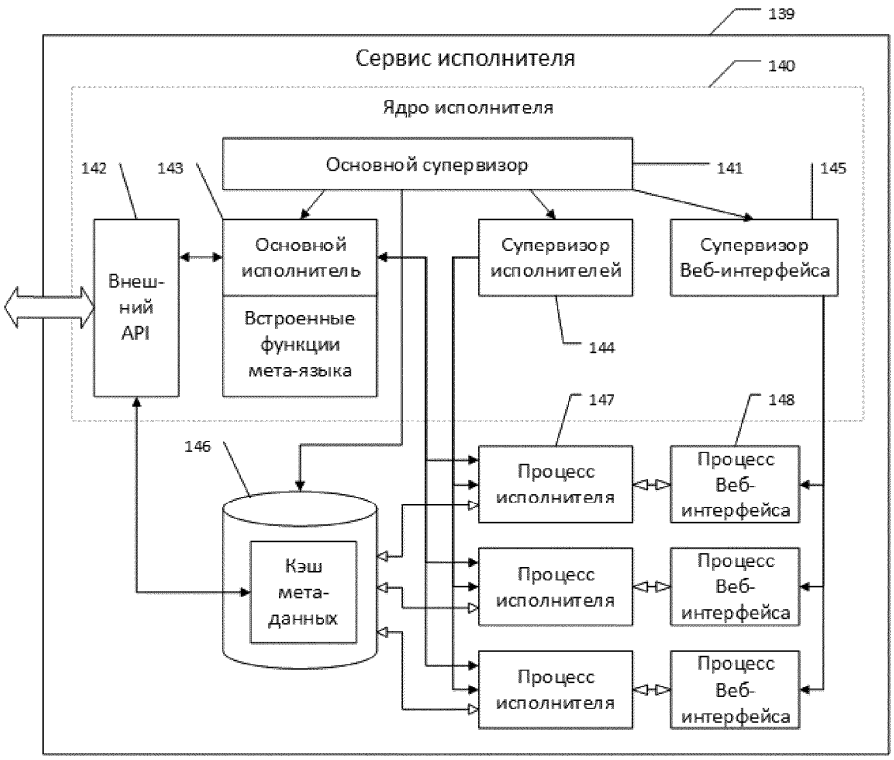
8. Способ по п.6, отличающийся тем, что описание классов объектов включает добавление новых классов и наследование существующих классов.



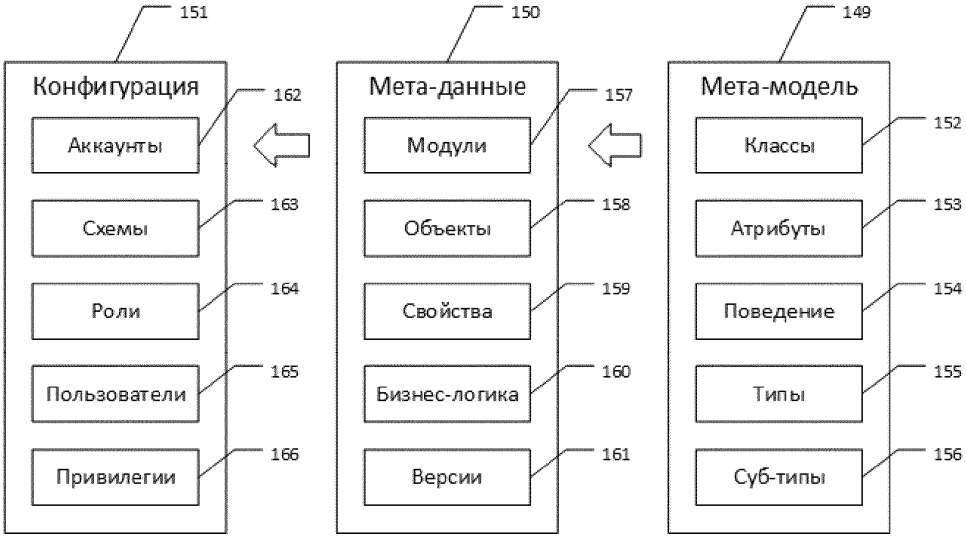
Фиг. 1



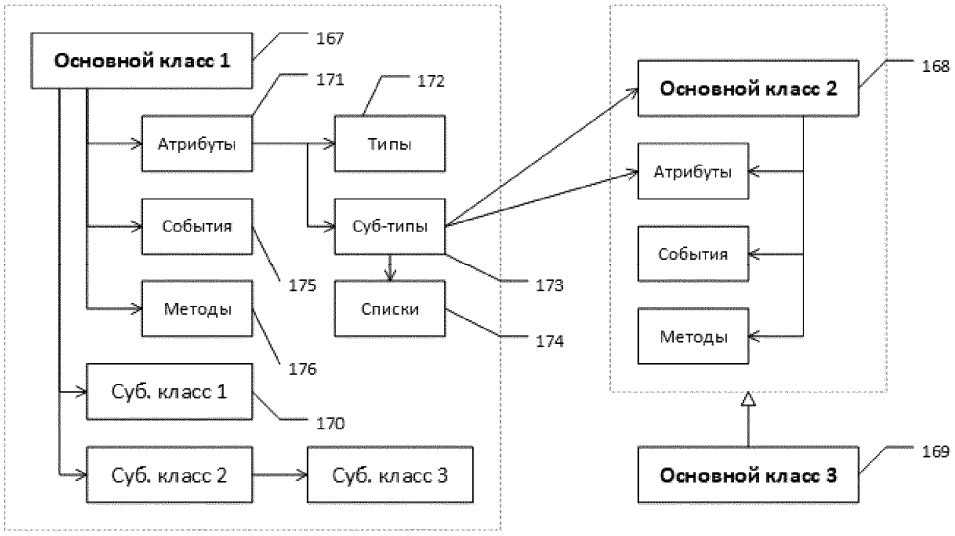
Фиг. 2


139
Сервис исполнителя
140
Ядро исполнителя
142
143
Основной супервизор
141
145
Внеш-
ний
API
Основной
исполнитель
Встроенные
функции
мета-языка
Супервизор
исполнителей
144
Супервизор
Веб-интерфейса
147
148
146
Процесс
исполнителя
Процесс
Веб-
интерфейса
Кэш
мета-
данных
Процесс
исполнителя
Процесс
Веб-
интерфейса
Процесс
исполнителя
Процесс
Веб-
интерфейса

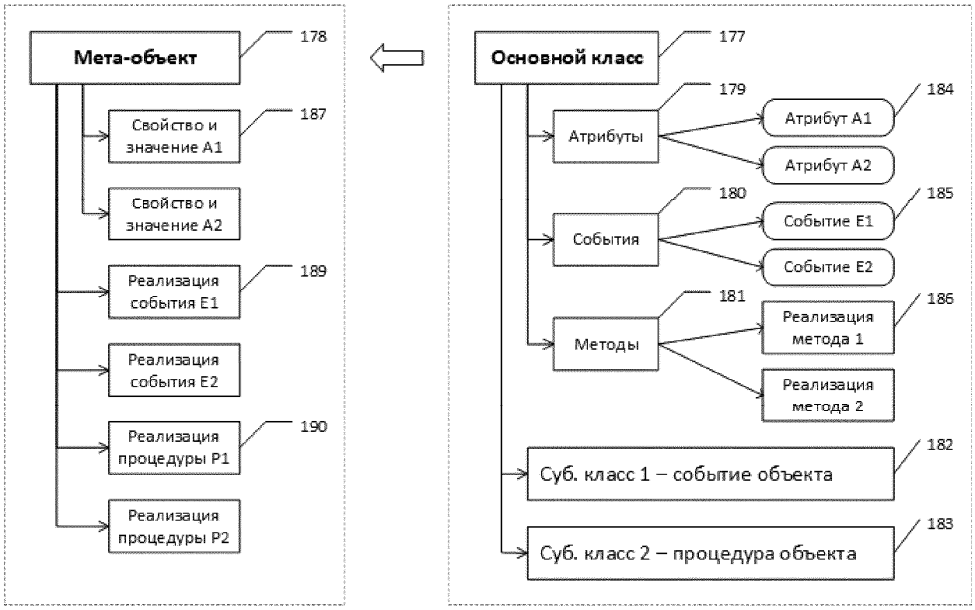
Фиг. 3


151
Конфигурация
162
Аккаунты
163
Схемы
164
Роли
165
Пользователи
166
Привилегии
150
Мета-данные
157
Модули
158
Объекты
159
Свойства
160
Бизнес-логика
161
Версии
149
Мета-модель
152
Классы
153
Атрибуты
154
Поведение
155
Типы
156
Суб-типы

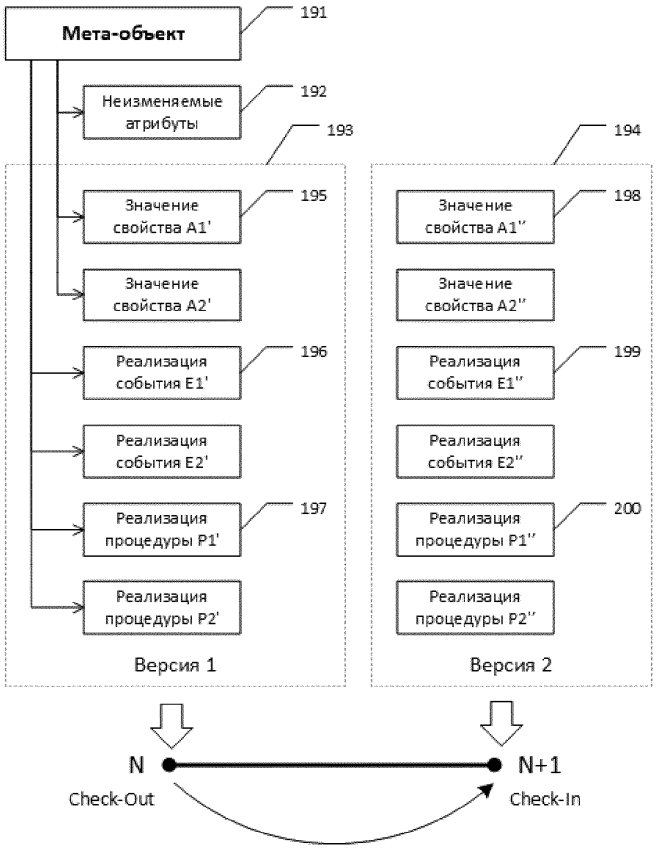
Фиг. 4


Основной класс 1
167
171
172
Атрибуты
Типы
События
Суб-типы
175
173
Методы
Списки
174
176
Суб. класс 1
170
Суб. класс 2
Суб. класс 3
Основной класс 2
168
Атрибуты
События
Методы
Основной класс 3
169

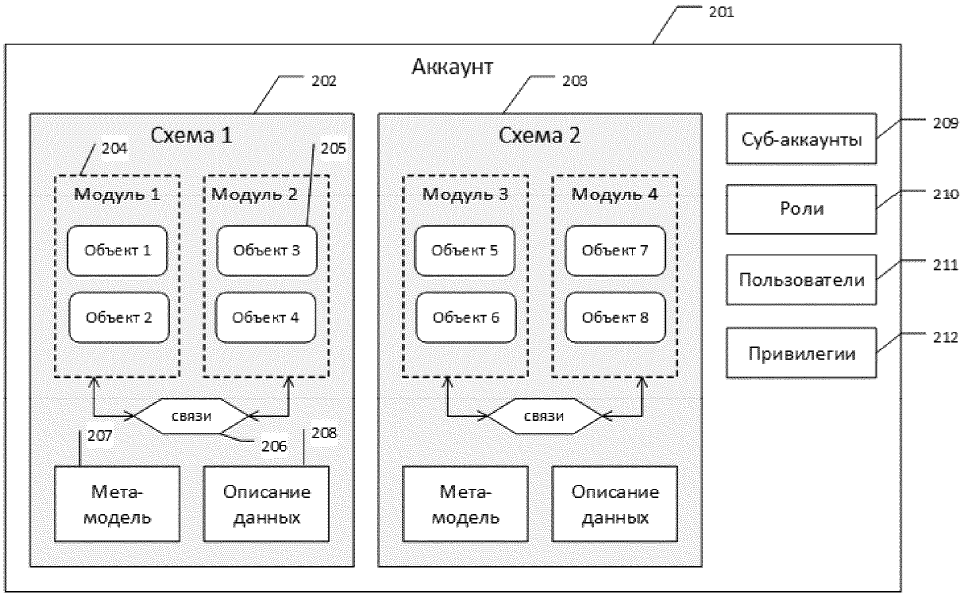
Фиг. 5


Мета-объект
178
Свойство и значение A1
187
Свойство и значение A2
Реализация события E1
189
Реализация события E2
Реализация процедуры P1
190
Реализация процедуры P2
Основной класс
177
Атрибуты
179
184
Атрибут A1
Атрибут A2
События
180
185
Событие E1
Событие E2
Методы
181
186
Реализация метода 1
Реализация метода 2
Суб. класс 1 – событие объекта
182
Суб. класс 2 – процедура объекта
183

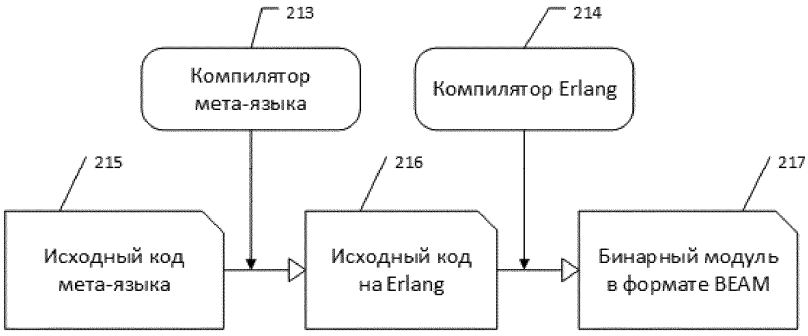
Фиг. 6


Мета-объект
191
Неизменяемые атрибуты
192
193
194
Значение свойства A1'
195
Значение свойства A2'
Реализация события E1'
196
Реализация события E2'
Реализация процедуры P1'
197
Реализация процедуры P2'
Версия 1
Значение свойства A1''
198
Значение свойства A2''
Реализация события E1''
199
Реализация события E2''
Реализация процедуры P1''
200
Реализация процедуры P2''
Версия 2
N
N+1
Check-Out
Check-In

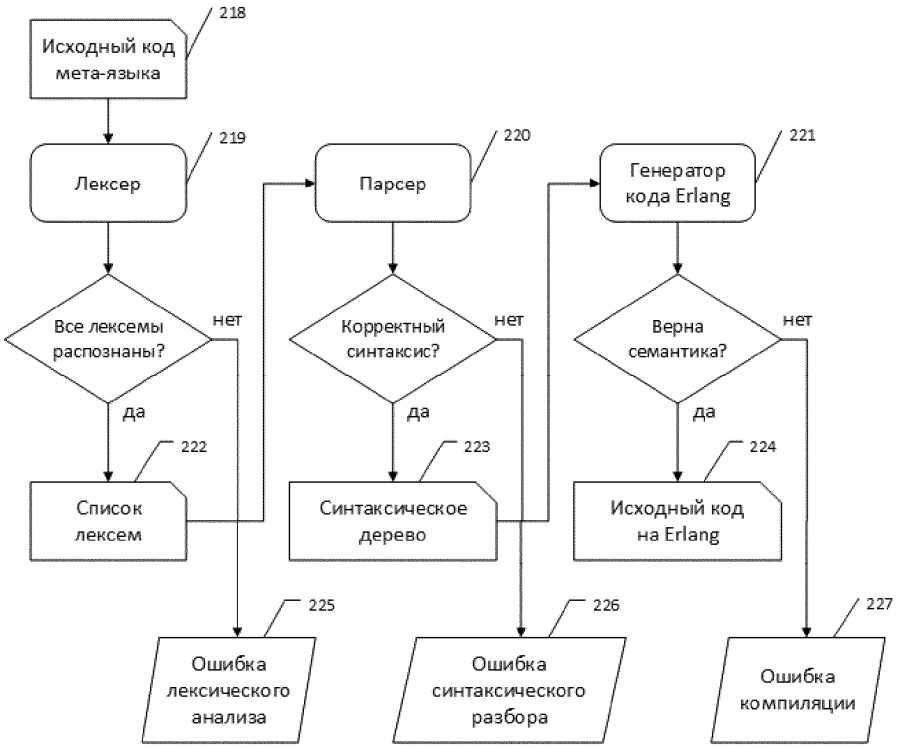
Фиг. 7


201
Аккаунт
202
203
Схема 1
204
205
Модуль 1
Модуль 2
Объект 1
Объект 2
Объект 3
Объект 4
связи
207
206
208
Мета-модель
Описание данных
Схема 2
Модуль 3
Модуль 4
Объект 5
Объект 6
Объект 7
Объект 8
связи
Мета-модель
Описание данных
Суб-аккаунты
209
Роли
210
Пользователи
211
Привилегии
212

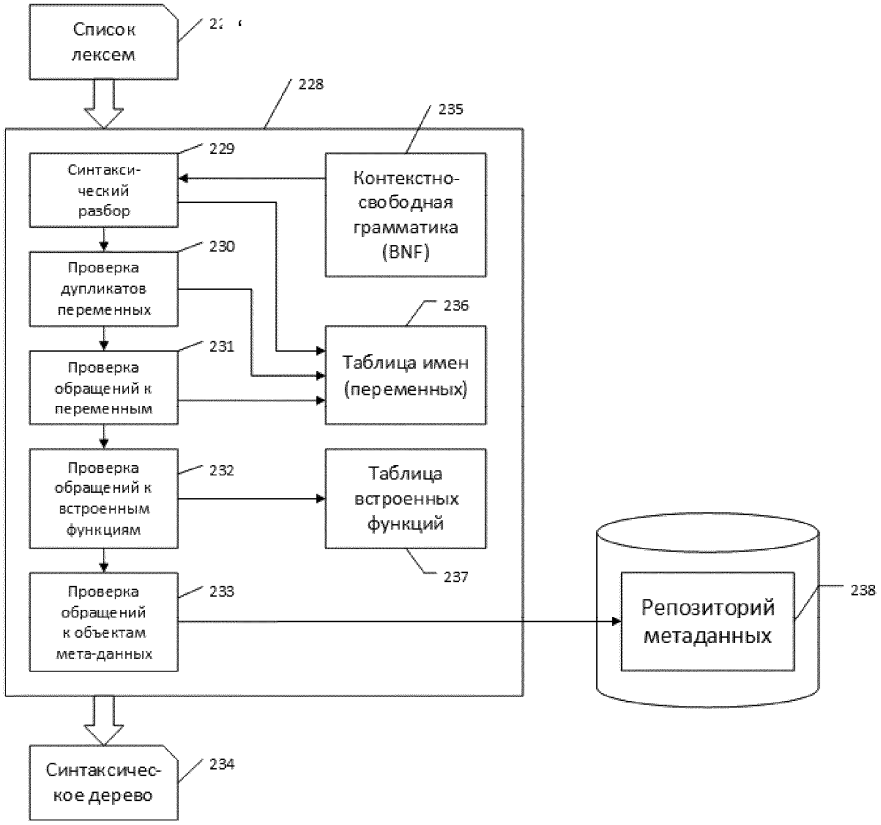
Фиг. 8


213
Компилятор мета-языка
214
Компилятор Erlang
215
Исходный код мета-языка
216
Исходный код на Erlang
217
Бинарный модуль в формате BEAM

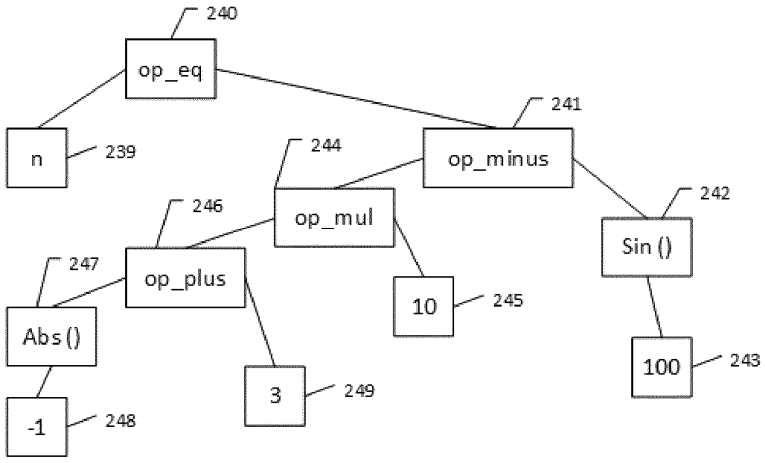
Фиг. 9


218
Исходный код мета-языка
219
Лексер
220
Парсер
221
Генератор кода Erlang
Все лексемы распознаны?
нет
да
Корректный синтаксис?
нет
да
Верна семантика?
нет
да
222
Список лексем
223
Синтаксическое дерево
224
Исходный код на Erlang
225
Ошибка лексического анализа
226
Ошибка синтаксического разбора
227
Ошибка компиляции

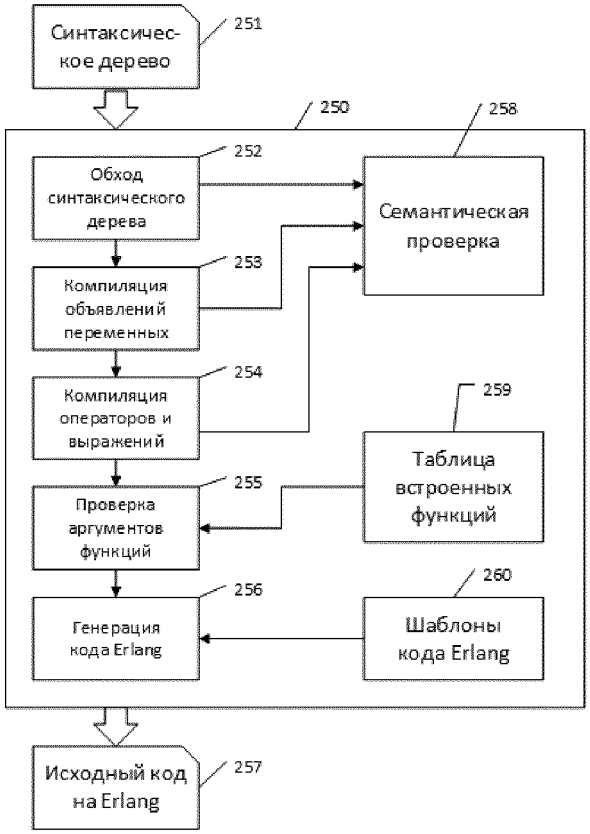
Фиг. 10


Список лексем
228
235
229
Синтакси-ческий разбор
Контекстно-свободная грамматика (BNF)
230
Проверка дупликатов переменных
236
231
Проверка обращений к переменным
Таблица имен (переменных)
Проверка обращений к встроенным функциям
232
Таблица встроенных функций
237
Проверка обращений к объектам мета-данных
233
Репозиторий метаданных
238
Синтаксичес-кое дерево
234

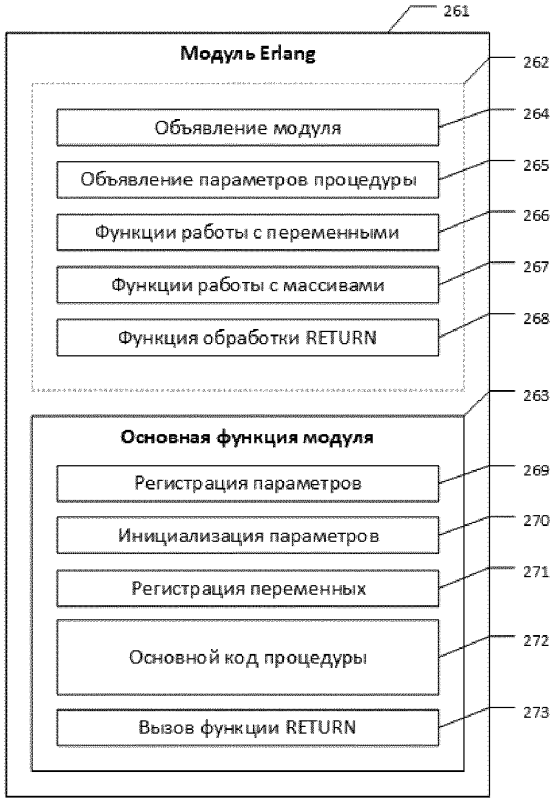
Фиг. 11


240
op_eq
241
n
239
244
op_minus
246
op_mul
242
Sin ()
247
op_plus
10
245
Abs ()
100
243
3
249
-1
248

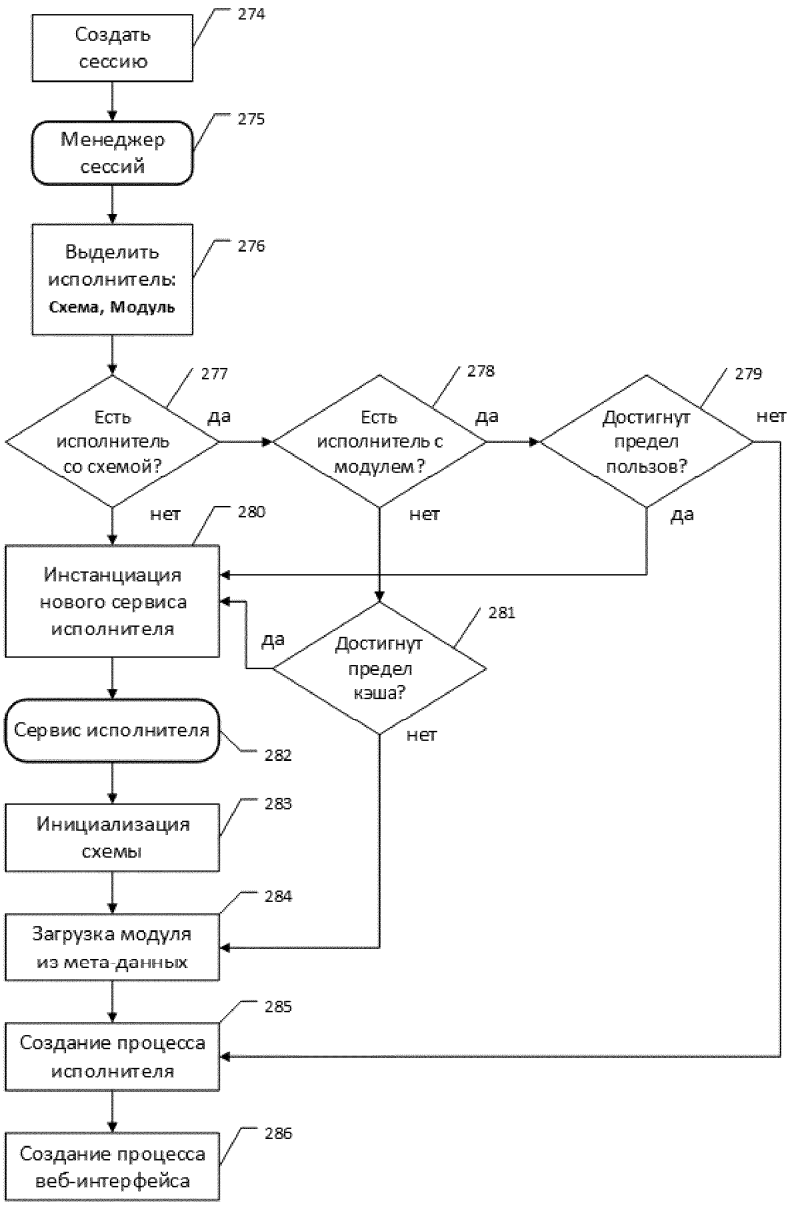
Фиг. 12


Синтаксичес-
кое дерево
251
250
258
252
Обход
синтаксического
дерева
Семантическая
проверка
253
Компиляция
объявлений
переменных
254
Компиляция
операторов и
выражений
259
Таблица
встроенных
функций
255
Проверка
аргументов
функций
260
256
Генерация
кода Erlang
Шаблоны
кода Erlang
Исходный код
на Erlang
257

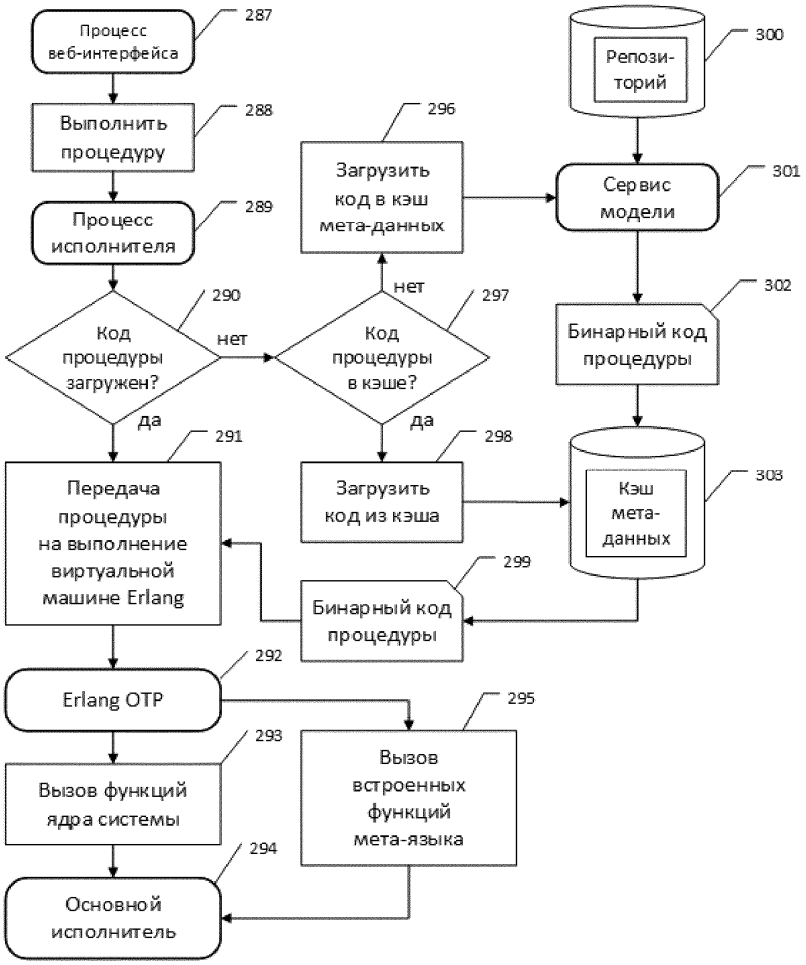
Фиг. 13


261
Модуль Erlang
262
Объявление модуля
264
Объявление параметров процедуры
265
Функции работы с переменными
266
Функции работы с массивами
267
Функция обработки RETURN
268
263
Основная функция модуля
Регистрация параметров
269
Инициализация параметров
270
Регистрация переменных
271
Основной код процедуры
272
Вызов функции RETURN
273

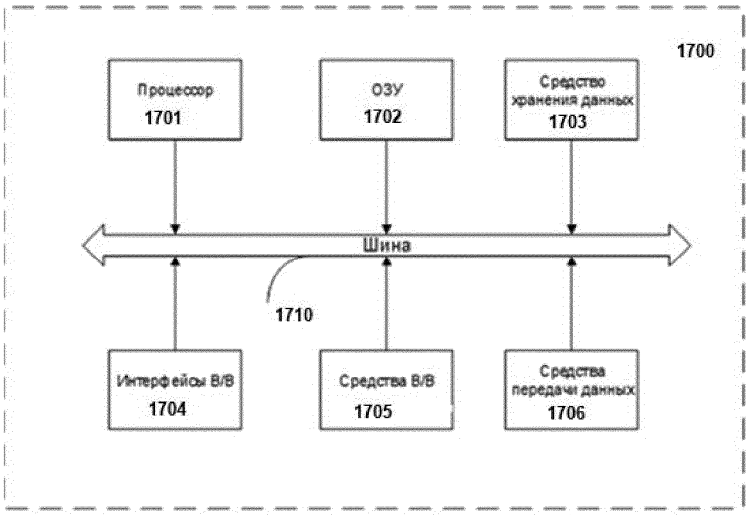
Фиг. 14


Создать сессию
274
Менеджер сессий
275
Выделить исполнитель: Схема, Модуль
276
Есть исполнитель со схемой?
277
да
нет
Есть исполнитель с модулем?
278
да
нет
Достигнут предел пользов?
279
нет
да
Инстанциация нового сервиса исполнителя
280
да
Достигнут предел кэша?
281
нет
Сервис исполнителя
282
Инициализация схемы
283
Загрузка модуля из мета-данных
284
Создание процесса исполнителя
285
Создание процесса веб-интерфейса
286

Фиг. 15


Процесс веб-интерфейса
287
Выполнить процедуру
288
Процесс исполнителя
289
Код процедуры загружен?
290
нет
да
Код процедуры в кэше?
297
нет
да
Загрузить код в кэш мета-данных
296
Репози-торий
300
Сервис модели
301
Бинарный код процедуры
302
Передача процедуры на выполнение виртуальной машине Erlang
291
Загрузить код из кэша
298
Кэш мета-данных
303
Бинарный код процедуры
299
Erlang OTP
292
Вызов функций ядра системы
293
Вызов встроенных функций мета-языка
295
Основной исполнитель
294

Фиг. 16


1700
Процессор
1701
ОЗУ
1702
Средство хранения данных
1703
Шина
1710
Интерфейсы В/В
1704
Средства В/В
1705
Средства передачи данных
1706

Фиг. 17