(12) МЕЖДУНАРОДНАЯ ЗАЯВКА, ОПУБЛИКОВАННАЯ В СООТВЕТСТВИИ С ДОГОВОРОМ О ПАТЕНТНОЙ КООПЕРАЦИИ (PCT)

(19) Всемирная Организация Интеллектуальной Собственности
Международное бюро

(43) Дата международной публикации
12 мая 2022 (12.05.2022)

WIPO | PCT

(10) Номер международной публикации
WO 2022/098253 A1

(51) Международная патентная классификация:
G06F 8/20 (2018.01) *G06F 9/44* (2018.01)
G06F 8/30 (2018.01)

(21) Номер международной заявки: PCT/RU2020/000595

(22) Дата международной подачи:
10 ноября 2020 (10.11.2020)

(25) Язык подачи: Русский

(26) Язык публикации: Русский

(30) Данные о приоритете:
2020136739 09 ноября 2020 (09.11.2020) RU

(71) Заявитель: **ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ "КРАФТ СИСТЕМЗ" (LIMITED LIABILITY COMPANY "CRAFT SYSTEMS")** [RU/RU]; Территория инновационного центра "Сколково", ул. Нобеля, 7, этаж 4, пом. 20, р.м. 7 Москва, 121205, Moscow (RU).

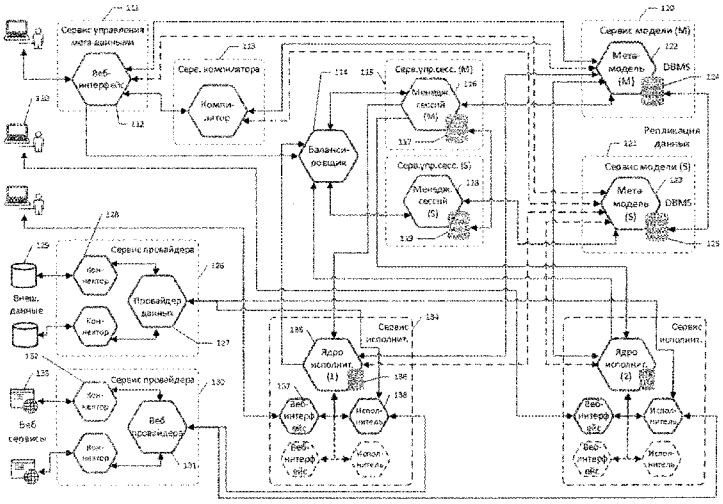
(72) Изобретатель: **ГОЛУБЕВ, Дмитрий Алексеевич (GOLUBEV, Dmitry Alexeevich)**; ул. Медицинская, 11, кв. 57 г. Нижний Новгород, 603104, g. Nizhnij Novgorod (RU).

(74) Агент: **КОТЛОВ, Дмитрий Владимирович (KOTLOV, Dmitry Vladimirovich)**; ООО "ЦИС "Сколково", территория инновационного центра "Сколково", 4, офис 402.1 Москва, 143026, Moscow (RU).

(81) Указанные государства *(если не указано иначе, для каждого вида национальной охраны)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO,

(54) **Title:** SYSTEM AND METHOD FOR CREATING AND EXECUTING HIGHLY SCALED CLOUD APPLICATIONS

(54) **Название изобретения:** СИСТЕМА И СПОСОБ СОЗДАНИЯ И ИСПОЛНЕНИЯ ВЫСОКО МАСШТАБИРУЕМЫХ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ



Фиг 2

111	Metadata management service
112	Web interface
113	Compiler service; Compiler
114	Balancer
115	Session management service (M)
116	Session management (M)
118	Session management service (S); Session management (S)
120	Model service (M)
Репликация данных	Data replication
121	Model service (S)
122	Metamodel (M)
123	Metamodel (S)
126	Provider service
127	Data provider
128	Connector
129	External data
130	Provider service
131	Web provider
132	Connector
133	Web services
134	Execution service
135	Execution core (1)
138	Web interface
138	Executor

(57) **Abstract:** The invention relates to the field of developing and executing web-based software applications. Claimed is a group of inventions, and more particularly a cloud system and a method for developing and executing web-based software applications. The cloud system consists of microservices that interact with one another by means of a software interface. Said microservices include: a metamodel service; a metadata management service; a meta-language cross compiler service capable of translating the source code of a meta-language that describes the business logic of the object of an application into an Erlang code and compiling the resulting Erlang code into a binary code for an Erlang virtual machine, which can be loaded and executed by an application execution service; a session management service; an application execution service; an external data interface service capable of working with external databases

WO 2022/098253 A1

[продолжение на следующей странице]

DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Указанные государства** *(если не указано иначе, для каждого вида региональной охраны)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), евразийский (AM, AZ, BY, KG, KZ, RU, TJ, TM), европейский патент (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Опубликована:

— *с отчётом о международном поиске (статья 21.3)*

and web services; and an external API service. The metamodel service and the session management service are designed so that they can be supplemented by at least one service on a master-slave basis in order to distribute query-related load. The technical result is that of improving efficiency, providing horizontal scaling during the development and execution of applications and also simplifying the description of the business logic of applications by the use of a meta-language.

(57) Реферат: 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 микро-сервисов, взаимодействующих между Собой посредством программного интерфейса. Микро-сервисы представляют собой: сервис мета-модели; сервис управления мета-данными; сервис кросс - компилятора мета-языка, выполненный с возможностью перевода исходного кода мета-языка, 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; сервис исполнителя приложений; сервис интерфейса с внешними данными, выполненный с возможностью работы с внешними базами данных и веб-сервисами; сервис внешнего API системы. Сервисы мета-модели 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 мета-языка.

СИСТЕМА И СПОСОБ СОЗДАНИЯ И ИСПОЛНЕНИЯ ВЫСОКО МАСШТАБИРУЕМЫХ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ

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 10, 324, 690 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 7,735,062 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 мета-языка. Дополнительно технический результат заключается в реализации назначения.

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

сервис мета-модели, включающий репозиторий мета-данных, и сервис интерфейса с мета-моделью, при этом сервис мета-моделей выполнен с возможностью дополнения по меньшей мере одним сервисом по схеме «ведущий - ведомый» для распределения нагрузки при запросах;

сервис управления мета-данными, содержащий веб-интерфейс для работы с мета-данными, взаимодействующий с сервисом мета-моделей для получения и изменения мета-данных, выполненный с возможностью предоставления удобных средств работы с мета-данными, составляющими конфигурацию модулей приложений; реализация визуального редактора для построения интерфейсных форм приложения; предоставление визуального редактора исходного кода мета-языка; реализация механизмов работы с версиями описанных на мета-языке мета-данных, предоставление средств перевода мета-данных на другие языки для локализации приложений;

сервис кросс - компилятора мета-языка, выполненный с возможностью перевода исходного кода мета-языка, 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 мета-данными, инициации сервиса исполнителя нужными мета-данными, 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 мета-объектов 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.

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

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

В другом частном варианте реализации предлагаемой системы, сервис интерфейса с внешними данными, выполненный 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 выполненный с возможностью обеспечения унифицированного API работы с данными 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 мета-данных, хранящихся в репозитории мета-данных, посредством сервиса управления мета-данными и сервиса мета-модели, с использованием мета-языка компилирующего типа, при этом описание объектов приложений основано на описании классов мета-модели;

описанные мета-данные, представленные в виде набора мета-объектов, содержащих свойства, вложенные объекты, а также исходный код мета-языка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, при

этом осуществляют сравнение исходного кода мета-языка 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 мета языка 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 мета-данными, инициацию сервиса исполнителя нужными мета-данными, 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, мета-модель содержит классы, суб-классы, атрибуты, поведение, типы и суб-типы.

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 чертежи:

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

Фиг. 2 иллюстрирует архитектуру предлагаемой облачной системы. Поз. 110 – пользователь системы, поз. 111 – сервис управления мета-данными, поз. 112 – веб-интерфейс Дизайнера, поз. 113 – сервис кросс-компилятора, поз. 114 – балансировщик, поз. 115 – сервис управления сессиями, поз. 116 – менеджер сессий основной, поз. 117 –

таблица сессий основная, поз. 118 – менеджер сессий дополнительный, поз. 119 – таблица сессий дополнительная, поз. 120 – сервис мета-модели основной, поз. 121 – сервис мета-модели дополнительный, поз. 122 – мета-модель основная, поз. 123 – мета-модель дополнительная, поз. 124 – база данных модели основная, поз. 125 – база данных модели дополнительная, поз. 126 – сервис провайдера данных, поз. 127 – провайдер данных, поз. 128 – коннектор к базе данных, поз. 129 – внешняя база данных, поз. 130 – сервис веб-провайдера, поз. 131 – веб-провайдер, поз. 132 – веб-коннектор, поз. 133 – внешний веб-сервис, поз. 134 – сервис исполнителя, поз. 135 – ядро исполнителя, поз. 136 – кэш мета-данных, поз. 137 – процесс веб-интерфейса исполнителя, поз. 138 – процесс исполнителя.

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

Фиг. 4 иллюстрирует обобщенную структуру мета-данных. Поз. 149 – мета-модель, поз. 150 – мета-данные, поз. 151 – конфигурация, поз. 152 – описание классов, поз. 153 – описание атрибутов классов, поз. 154 – описание поведения классов, поз. 155 – описание типов, поз. 156 – описание суб-типов, поз. 157 – описание модулей, поз. 158 – описание мета-объектов, поз. 159 – описание свойств объектов, поз. 160 – описание бизнес-логики объектов, поз. 161 – версии мета-объектов, поз. 162 – системные аккаунты, поз. 163 – схемы мета-данных, поз. 164 – роли, поз. 165 – пользователи, поз. 166 – привилегии.

Фиг. 5 иллюстрирует структуру описания класса мета-модели. Поз. 167 – основной класс, поз. 168 – базовый основной класс, поз. 169 – дочерний основной класс, поз. 170 – суб. класс, поз. 171 – атрибуты класса, поз. 172 – типы значений, поз. 173 – суб-типы атрибутов, поз. 174 – списки значений, поз. 175 – события класса, поз. 176 – методы класса.

Фиг. 6 иллюстрирует структуру описания объекта мета-данных. Поз. 177 – основной класс, поз. 178 – мета-объект, поз. 179– атрибуты класса, поз. 180 – события класса, поз. 181 – методы класса, поз. 182 – суб-класс 1, поз. 183 – суб-класс 2, поз. 184 – атрибут класса A1, поз. 185 – событие класса E1, поз. 186 – реализация метода класса, поз. 187 – свойство и значение объекта для атрибута A1, поз. 189 – реализация события объекта E1, поз. 190 – реализация процедуры объекта P1.

Фиг. 7 иллюстрирует схему версионирования объекта мета-данных. Поз. 191 – мета-объект, поз. 192 – неизменяемые атрибуты объекта, поз. 193 – версия 1 данных объекта, поз. 194 – версия 2 данных объекта, поз. 195 – значение свойства объекта

версии 1, поз. 196 – реализация события объекта версии 1, поз. 197 – реализация процедуры объекта версии 1, поз. 198 – значение свойства объекта версии 2, поз. 199 – реализация события объекта версии 2, поз. 200 – реализация процедуры объекта версии 2.

Фиг. 8 иллюстрирует схему разграничения мета-информации. Поз. 201 – аккаунт, поз. 202 – схема 1, поз. 203 – схема 2, поз. 204 – модуль, поз. 205 – объект, поз. 206 – связи между объектами, поз. 207 – мета-модель, поз. 208 – описание данных, поз. 209 – суб-аккаунты, поз. 210 – роли, поз. 211 – пользователи, поз. 212 – привилегии.

Фиг. 9 иллюстрирует процесс компиляции исходного кода мета-языка системы. Поз. 213 – компилятор мета-языка, поз. 214 – компилятор Erlang, поз. 215 – исходный код мета-языка, поз. 216 – исходный код на Erlang, поз. 217 – бинарный модуль в формате BEAM.

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

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

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

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

Фиг. 14 иллюстрирует структуру модуля Erlang. Поз. 261 – модуль Erlang, поз. 262 – предварительные декларации, поз. 263 – основная функция модуля, поз. 264 – объявление модуля, поз. 265– объявление параметров процедуры, поз. 266 – функции

работы с переменными, поз. 267 – функции работы с массивами, поз. 268 – функция обработки RETURN, поз. 269 – регистрация параметров, поз. 270 – инициализация параметров, поз. 271 – регистрация переменных, поз. 272 – основной код процедуры, поз. 273 – вызов функции RETURN.

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

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

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 использовать 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 мета-моделей, хранящихся в системном репозитории и выполняются на вычислительном устройстве.

Облачная система для разработки и исполнения программных веб-приложений состоит из микро-сервисов, 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 (мета-информацию) 106, хранимую в репозитории 105. Мета-информация 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 мета-данных системы, редактор атрибутов мета-объектов, редактор форм интерфейса объектов, редактор процедур бизнес-логики.

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

Описание приложений в мета-информации 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 мета-языка 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 мета-модели 120, который обеспечивает хранение и доступ к мета-данным в системном репозитории. Сервис состоит из реляционной базы данных 124, включающей системный репозиторий, и собственно мета-модели 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 мета-данных.

Основными функциями сервиса мета-модели по работе с системным репозиторием являются: получение мета-информации по произвольным запросам к репозиторию; ввод новой мета-информации; изменение существующей мета-информации; удаление мета-информации.

Мета-модель обеспечивает целостность мета-данных при любых операциях с мета-информацией.

Для повышения производительности системы, при большом количестве запросов к репозиторию, а также для улучшения отказоустойчивости, в системе предусмотрено построение конфигурации мета-модели 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 мета-модели 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 мета-информации возможны только в основном сервисе, а запросы на чтение мета-информации возможны во всех сервисах. В случае, если основной сервис мета-модели 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 мета-модели практически неограниченно и определяется требованиями масштабирования системы.

Сервис управления мета-данными 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 мета-данными для разработчика.

Веб – интерфейс Дизайнера взаимодействует напрямую с сервисом мета-модели через внешний интерфейс API для получения мета-данных и для выполнения изменений в мета-данных.

Основные функции сервиса: управление бизнес-аккаунтами и суб-аккаунтами, а также доступом к ним; управление пользователями системы, ролями пользователей, списком привилегий; управление схемами мета-данных и доступом к ним; управление мета-информацией (набором классов и их описанием, включая атрибуты и поведение); удобная навигация по приложениям (модулям) и объектам мета-данных в выбранной схеме; предоставление редакторов объектов мета-данных. Веб – интерфейс Дизайнера также реализует визуальный редактор для построения интерфейсных форм объектов; редактор исходного кода процедур и событий бизнес-логики.

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

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

Реализация сервиса управления мета-данными 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 (мета-языка) в исполняемые бинарные модули, которые могут быть загружены и исполнены сервисом исполнителя приложений и является частью системы разработки приложений вместе с сервисом управления мета-данными 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 мета-языка системы, а также дополнительная информация, необходимая для компиляции, например: модуль,

объект. Компилятор обеспечивает полный грамматический разбор и синтаксический анализ конструкций входного языка и выдачу ошибок при компиляции. В процессе компиляции сервис взаимодействует с сервисом мета-модели 120 для получения дополнительной мета-информации об объектах мета-данных, которые используются в исходном коде процедуры, например чтобы валидировать список и типы аргументов вызываемой процедуры приложения или объекта. В случае, если сервис мета-модели 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 мета-данными 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. То есть, 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 мета-модели, 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 мета-данных.

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 мета-данными, инициация дополнительного сервиса исполнителя если необходимо, инициация сервиса исполнителя нужными мета-данными, 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 мета-модели.

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 мета-данных 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 мета-языка, супервизор процессов исполнителей 144, супервизор процессов веб-интерфейса 145. Внешний API 142 предоставляет набор методов для управления и использования сервиса исполнения, включая методы для инициализации сервиса, загрузки мета-данных, 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 мета-языка 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 мета-данных 146 (Фиг.3) – это программно-управляемые структуры в памяти процесса сервиса исполнения, предназначенные для локального хранения мета-информации, необходимой для запуска и выполнения бизнес-приложений сервисом исполнения. Ядро исполнителя 140 реализует внешний API для сервиса, а также отвечает за взаимодействие с сервисом мета-модели и сервисом управления сессиями. Также, ядро исполнителя управляет внутренними процессами сервиса.

Кэш мета-данных 146 (Фиг.3) хранит мета-информацию, связанную со схемой мета-данных (например, описания провайдеров и источников данных), а также полную информацию о бизнес-объектах приложений (модулей), содержащихся в схеме. Кэш может хранить мета-информацию, принадлежащую только одной схеме мета-данных. Использование локального кэша во время работа исполнителя исключает необходимость обращения за мета-данными к сервису мета-модели, 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 мета-данных, например, когда необходимо извлечь информацию о мета-объекте 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 мета-данных.

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 авто-инстанциации 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 мета-данными, конфигурация провайдеров хранится в мета-данных 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 мета-данных, хранимых в системном репозитории (мета-модели) в реляционной базе данных. Описание осуществляется посредством сервиса управления мета-данными и сервиса мета-модели с использованием процедурного мета-языка компилирующего типа, с поддержкой объектов для реализации сложной бизнес-логики приложений, при этом описание объектов приложений основано на описании классов мета-модели.

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 (прото-объектов), которые определяют все необходимые для описания реальных бизнес-объектов структуры и данные.

Фиг. 4 описывает обобщенную структуру репозитория мета-данных предлагаемой системы. Основные части репозитория: мета-модель 149, мета-данные 150 и конфигурация 151.

Мета-модель 149 служит для описания состава и структуры прото-объектов мета-данных. Данный подход обеспечивает максимальную гибкость и отсутствие практических ограничений на возможные способы описания бизнес-объектов, позволяя расширять базовый набор классов мета-модели при необходимости. Описание класса 152 включает

в себя набор атрибутов 153 и их типов 155 (а также суб-типов 156, при необходимости); поведение 154 – набор методов и событий класса; структуру класса – в случае если класс является сложным и содержит вложенные классы.

Мета-данные 150 содержат описания бизнес-объектов 158, являющихся частью приложений (модулей) 157. Описание каждого объекта 158 строится на основе описания его класса 152 в мета-модели 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, то есть могут сохраняться различные варианты значений свойств объекта, созданные в разные моменты времени.

Конфигурация 151 содержит структуры, необходимые для организации всей мета-информации и обеспечения доступа к ней со стороны пользователей системы. Вся мета-информация 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 мета-данными внутри. Схема 163 – это еще один уровень разделения метаданных по семантическому признаку. Схема определяет – какая мета-модель используется внутри данной схемы и, соответственно, как описываются мета-объекты в данной схеме.

Фиг.5 иллюстрирует структуру описания класса мета-модели. Классы могут быть одного из двух видов: основной класс 167 или суб-класс 170. Основной класс может использоваться для описания бизнес-объектов в мета-данных. Суб-классы 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 суб-тип 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 мета-данных.

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, то есть 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 определяется для бизнес-объекта в мета-данных. Для каждого класса имеется свой набор событий, который однозначно определяет – какие события могут быть определены для бизнес-объекта данного класса, то есть 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 суб-классы, которые составляют данный класс. Иерархия здесь означает, что каждый используемый суб-класс в структуре в свою очередь может также иметь вложенные суб-классы. В структуре класса нельзя использовать классы основного вида, допускается использование только суб-классов. Структура класса определяет – из каких составных частей состоит бизнес-объект и как будет строиться представление объекта в мета-данных. В качестве примера можно привести основной класс «Документ», в составе которого может находиться суб-класс «Форма». Используя данную структуру, возможно представить бизнес-объект класса «Документ», как совокупность объектов форм, определяющих визуальное представление данного объекта.

Расширение мета-модели 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 мета-объекта мета-данных на основе определения класса мета-модели. В данном примере у класса 171 определены два атрибута (179): атрибут A1 (184) и атрибут A2. В соответствии с данным определением, при создании описания мета - объекта 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 суб-типу атрибута класса A1 (184). В мета-объекте 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 суб-класса «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 мета-объекте возможно (но необязательно) определить сколько угодно процедур бизнес-логики. Процедуры заранее не описываются в классе. Реализация процедур объекта происходит посредством использования объекта суб-класса «процедура объекта» 183, определенного в структуре класса.

Механизм хранения объектов в репозитории мета-данных предполагает возможность версионирования (необязательно). Фиг. 7 описывает схему версионирования данных мета-объекта. При этом, все данные мета-объекта 191 разделяются на неверсионируемую и версионируемую часть.

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 мета-данных 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 мета-объекта 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 “рабочей” 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 мета-информации, включая мета-модель, мета-данные и конфигурацию. Фиг. 8 поясняет способ разграничения мета-информации 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 суб-аккаунты 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, то есть являются независимыми от пользователей, ролей и привилегий другого аккаунта.

Следующим уровнем разграничения мета-информации является сущность, обозначенная как “схема” (202, 203). Схема объединяет взаимосвязанную информацию мета-модели и мета-данных, такую как описание классов, данных и мета-объектов. При данном подходе каждая схема содержит собственную мета-модель (описание классов или прото-объектов), что обеспечивает независимость от других схем даже в рамках одного аккаунта и позволяет безопасно расширять или изменять мета-модель 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 мета-данных, посредством процедурного мета-языка компилирующего типа, с поддержкой объектов для реализации сложной бизнес-логики приложений. Грамматика языка построена таким образом, что позволяет непосредственно работать с объектами мета-данных 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 мета-данных. Это позволяет использовать контекст мета-объекта 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 мета-объекта.

Общая структура грамматики мета-языка 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 мета-языке.

Организация данных включает информацию о типах данных (any, integer, string и т.д.), структуре хранения данных и объектах. Мета-язык 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 мета-моделей, а также рекурсивные вызовы.

Второй этап предлагаемого способа заключается в том, что описанные мета-данные, представленные в виде набора мета-объектов, содержащих свойства, вложенные объекты, а также исходный код мета-языка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, который впоследствии компилируют в исполняемый бинарный код для виртуальной машины Erlang (BEAM), посредством сервиса кросс-компилятора.

Фиг.9 иллюстрирует обобщенный процесс компиляции исходного кода процедур мета-языка системы в бинарные файлы BEAM. На входе компилятору поступает исходный код мета-языка 215. Компилятор мета-языка 213 анализирует его и преобразует в текст, полностью удовлетворяющий синтаксису языка Erlang и реализующий ту же самую логику. Компилятор мета-языка 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 мета языка.

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 мета-языка 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 мета-языка 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:

{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 мета-языка 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 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 мета-данных 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 мета-языка:

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 мета-языка, которое транслируется в модуль Erlang. Объявление параметров содержит список идентификаторов и типов параметров процедуры мета-языка, 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 мета-языка. 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 мета-языка или нет, и даже в том случае, если процедура мета-языка 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 мета-данными, инициацию сервиса исполнителя нужными мета-данными, 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 мета-данных 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 мета-данных 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 мета-данных 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 мета-данных 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 мета-языка 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: 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

1. Облачная система для разработки и исполнения программных веб-приложений, состоящая из микро-сервисов, взаимодействующих между собой посредством программного интерфейса, при этом микро-сервисы представляют собой:

сервис мета-модели, включающий репозиторий мета-данных, и сервис интерфейса с мета-моделью, при этом сервис мета-моделей выполнен с возможностью дополнения по меньшей мере одним сервисом по схеме «ведущий - ведомый» для распределения нагрузки при запросах;

сервис управления мета-данными, содержащий веб-интерфейс для работы с мета-данными, взаимодействующий с сервисом мета-моделей для получения и изменения мета-данных, выполненный с возможностью предоставления удобных средств работы с мета-данными, составляющими конфигурацию модулей приложений; реализация визуального редактора для построения интерфейсных форм приложения; предоставление визуального редактора исходного кода мета-языка; реализация механизмов работы с версиями описанных на мета-языке мета-данных, предоставление средств перевода мета-данных на другие языки для локализации приложений;

сервис кросс - компилятора мета-языка, выполненный с возможностью перевода исходного кода мета-языка, 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 мета-данными, инициации сервиса исполнителя нужными мета-данными, 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 мета-объектов 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 мета-языка содержит лексер, парсер и генератор кода Erlang.

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

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

6. Способ разработки и исполнения программных веб-приложений, содержащий этапы:

описывают объекты приложений в виде мета-данных, хранящихся в репозитории мета-данных, посредством сервиса управления мета-данными и сервиса мета-модели, с использованием мета-языка компилирующего типа, при этом описание объектов приложений основано на описании классов мета-модели;

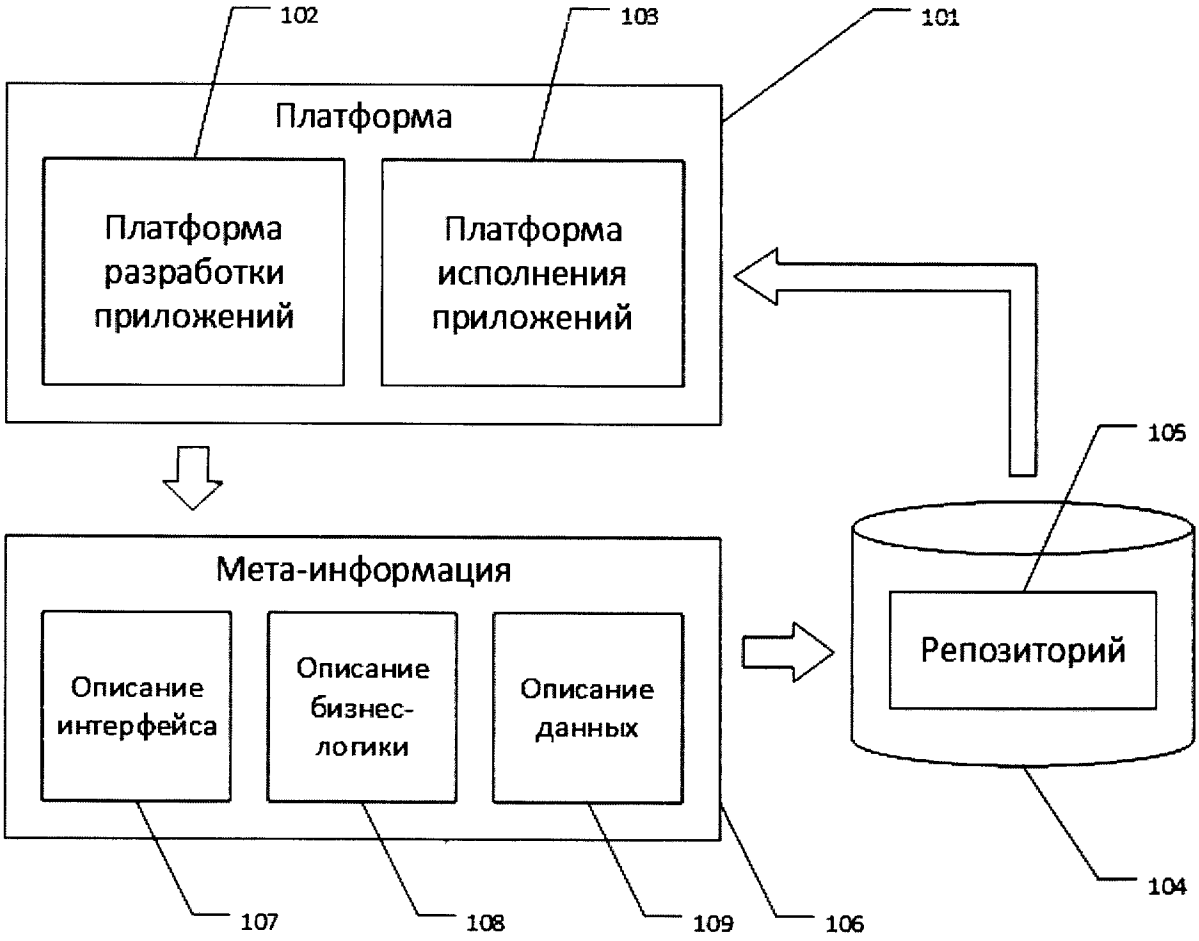
описанные мета-данные, представленные в виде набора мета-объектов, содержащих свойства, вложенные объекты, а также исходный код мета-языка, реализующий события и процедуры объекта, преобразовывают в код на языке Erlang, при этом осуществляют сравнение исходного кода мета-языка с грамматикой языка и выдают 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 мета языка 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 мета-данными, инициацию сервиса исполнителя нужными мета-данными, создание процесса исполнителя и веб-интерфейс;

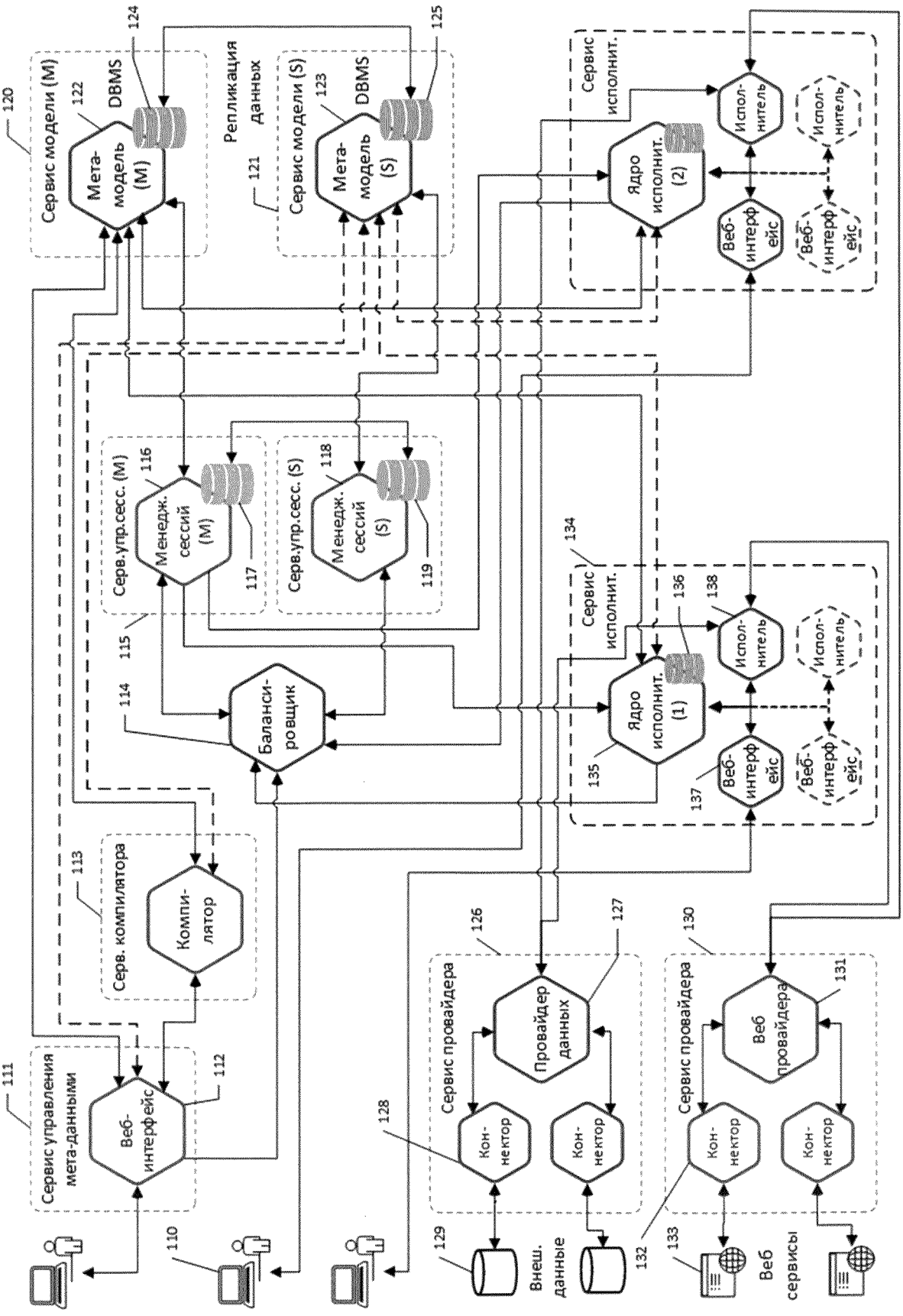
осуществляют запуск и выполнение бинарного кода из системного репозитория посредством сервиса исполнителя, при этом исполнение бинарного кода осуществляется непосредственно на виртуальной машине Erlang.

7. Способ по п.6, отличающийся тем, что мета-модель содержит классы, суб-классы, атрибуты, поведение, типы и суб-типы.

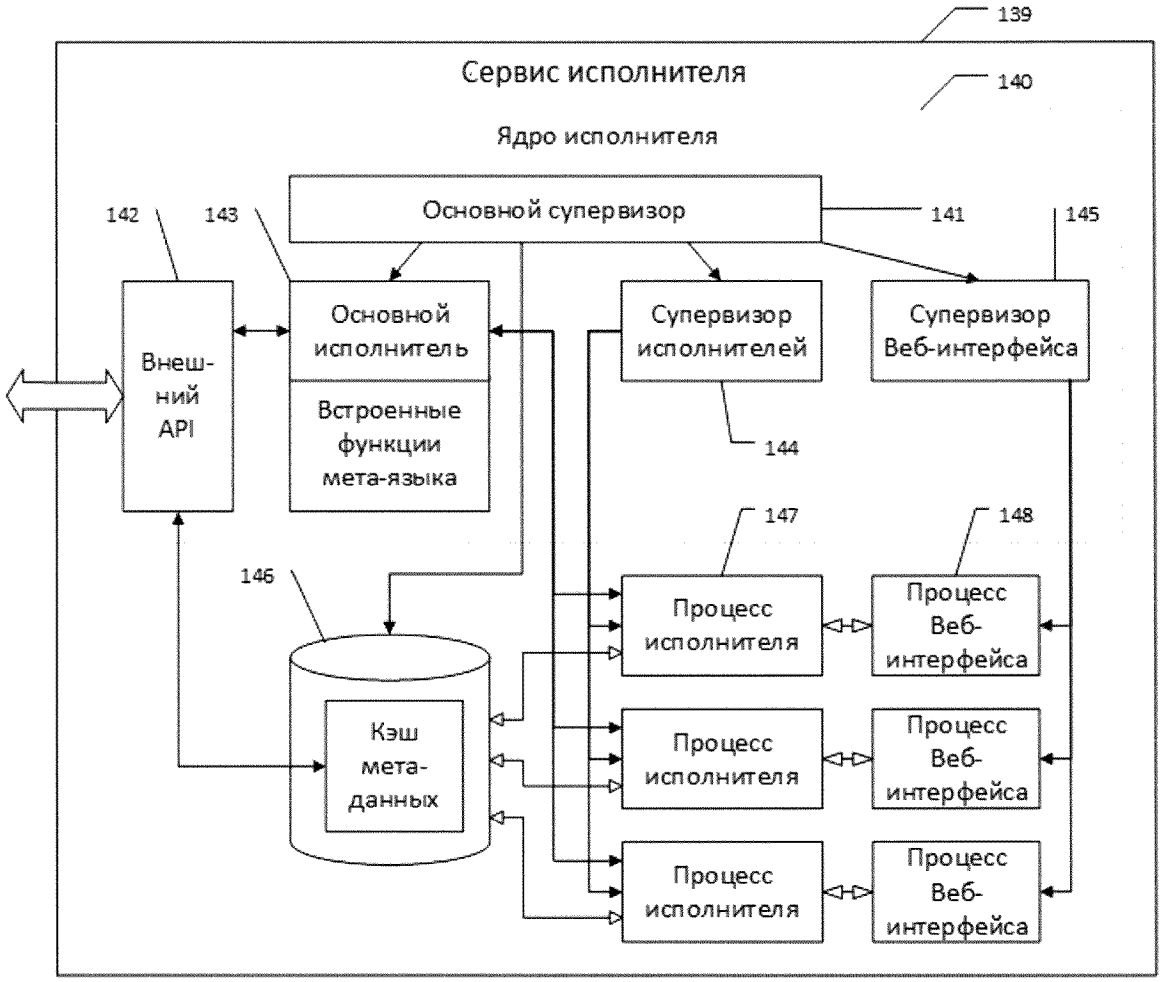
8. Способ по п.6, отличающийся тем, что описание классов объектов включает добавление новых классов и наследование существующих классов.


102
103
101
Платформа
Платформа разработки приложений
Платформа исполнения приложений
105
Мета-информация
Описание интерфейса
Описание бизнес-логики
Описание данных
Репозиторий
104
107
108
109
106

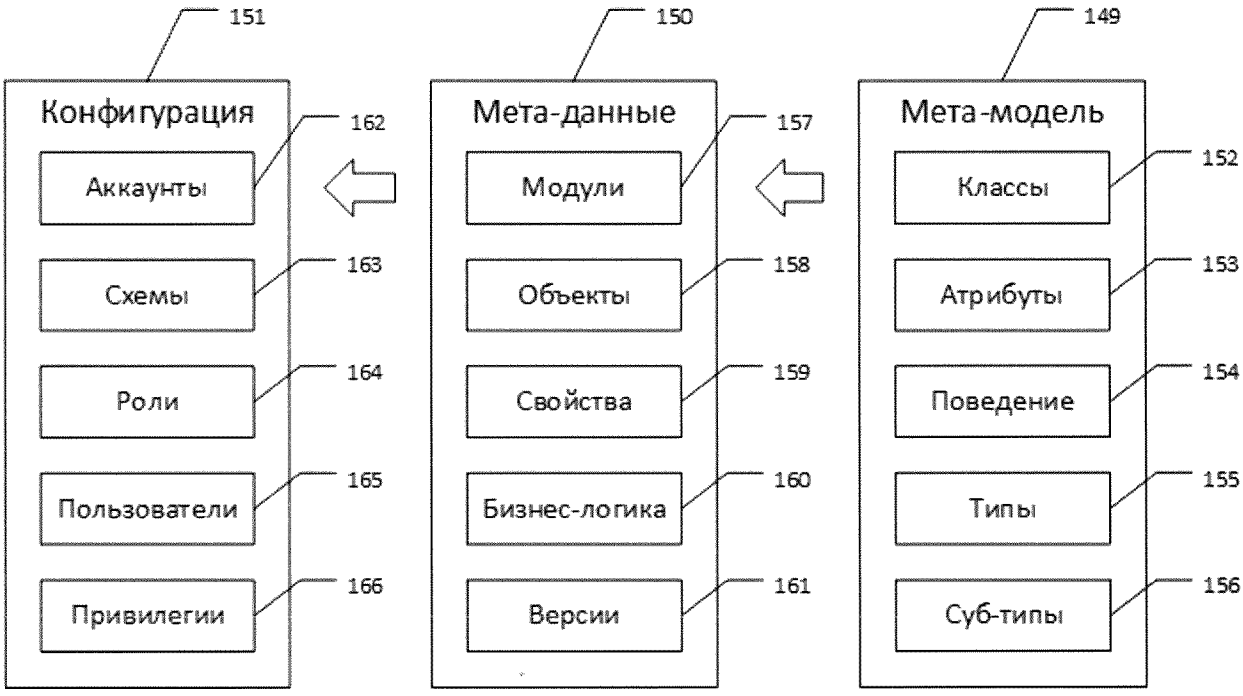
Фиг.1


120
Сервис модели (M)
122
Мета-модель (M)
DBMS
124
Репликация данных
121
Сервис модели (S)
123
Мета-модель (S)
DBMS
125
Сервис исполнит.
Ядро исполнит. (2)
Испол-нитель
Веб-интерфейс
Испол-нитель
Веб-интерфейс
Серв.упр.сесс. (M)
116
Менедж. сессий (M)
117
Серв.упр.сесс. (S)
118
Менедж. сессий (S)
119
115
114
Балансировщик
134
Сервис исполнит.
Ядро исполнит. (1)
136
138
Испол-нитель
Веб-интерфейс
Испол-нитель
Веб-интерфейс
135
137
Серв. компилятора
113
Компи-лятор
Сервис управления мета-данными
111
Веб-интерфейс
112
128
Сервис провайдера
126
Провайдер данных
127
Кон-нектор
Кон-нектор
Сервис провайдера
130
Веб провайдера
131
Кон-нектор
Кон-нектор
110
129
Внеш. данные
132
133
Веб сервисы

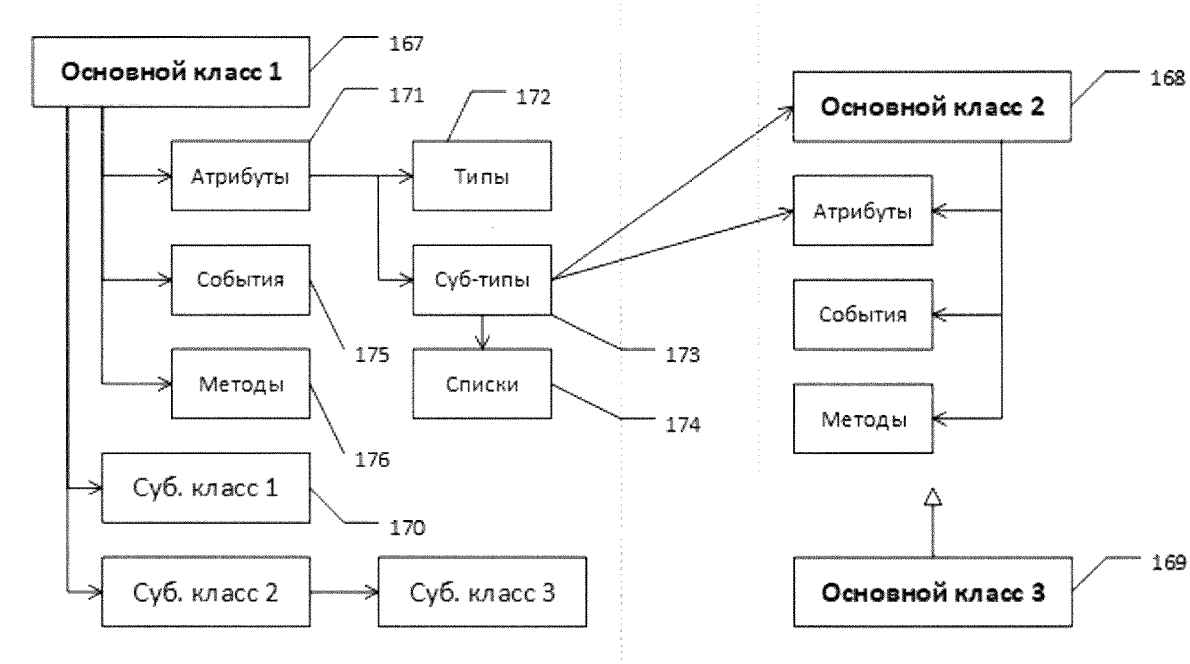
Фиг.2


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

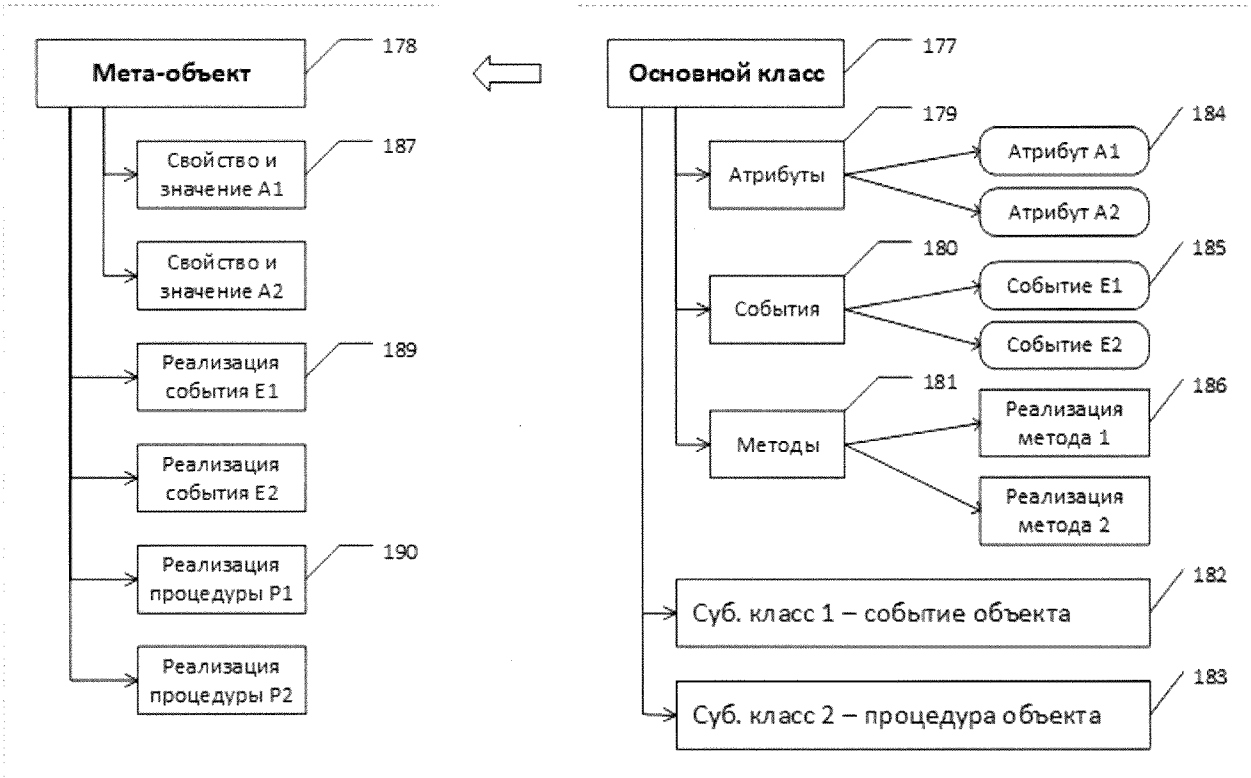
Фиг.3


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

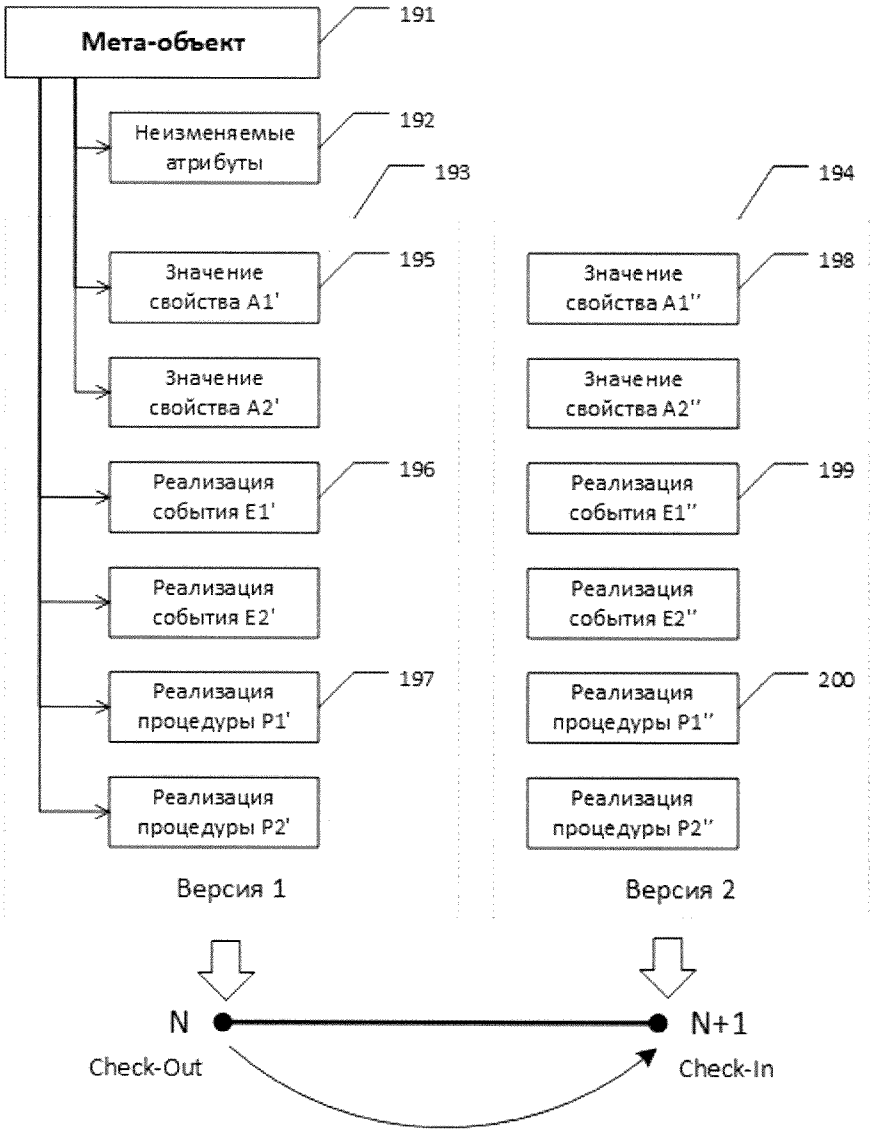
Фиг.4


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

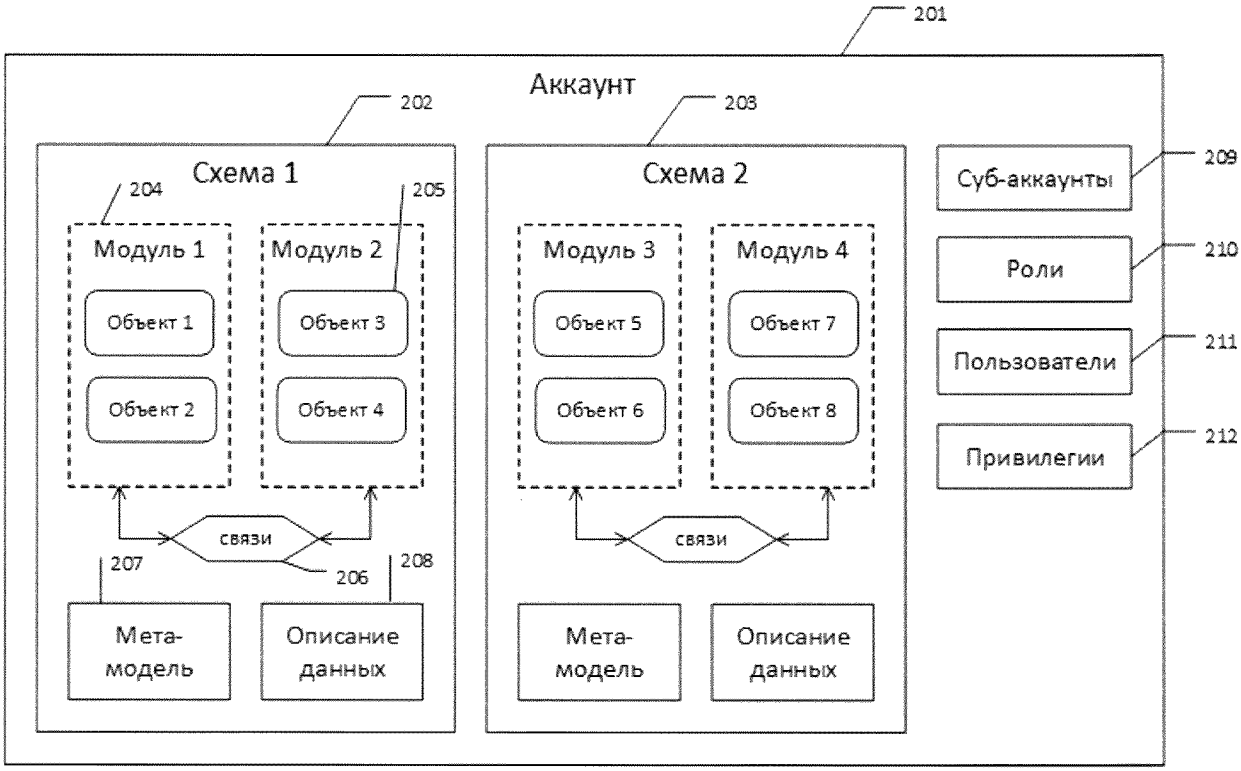
Фиг.5


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
Атрибут A1
184
Атрибут A2
События
180
Событие E1
185
Событие E2
Методы
181
Реализация метода 1
186
Реализация метода 2
Суб. класс 1 – событие объекта
182
Суб. класс 2 – процедура объекта
183

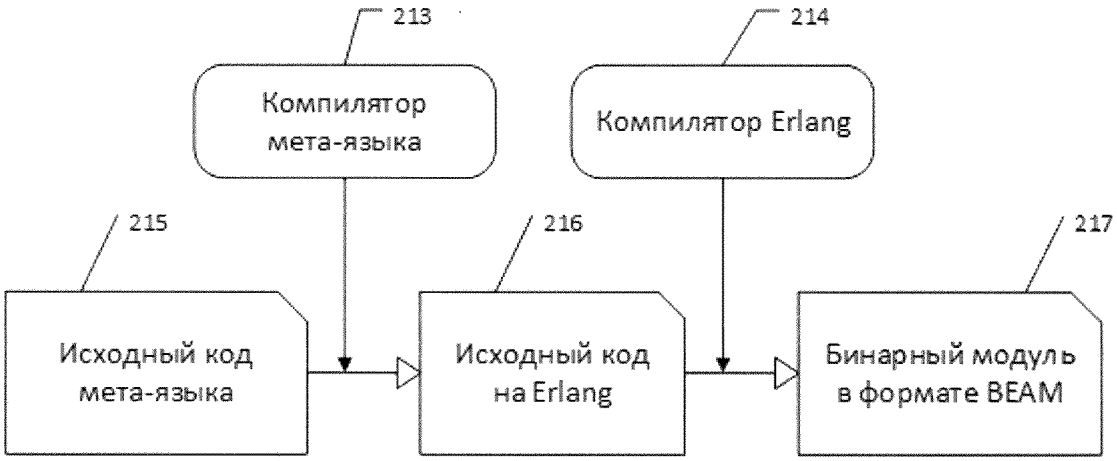
Фиг.6


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

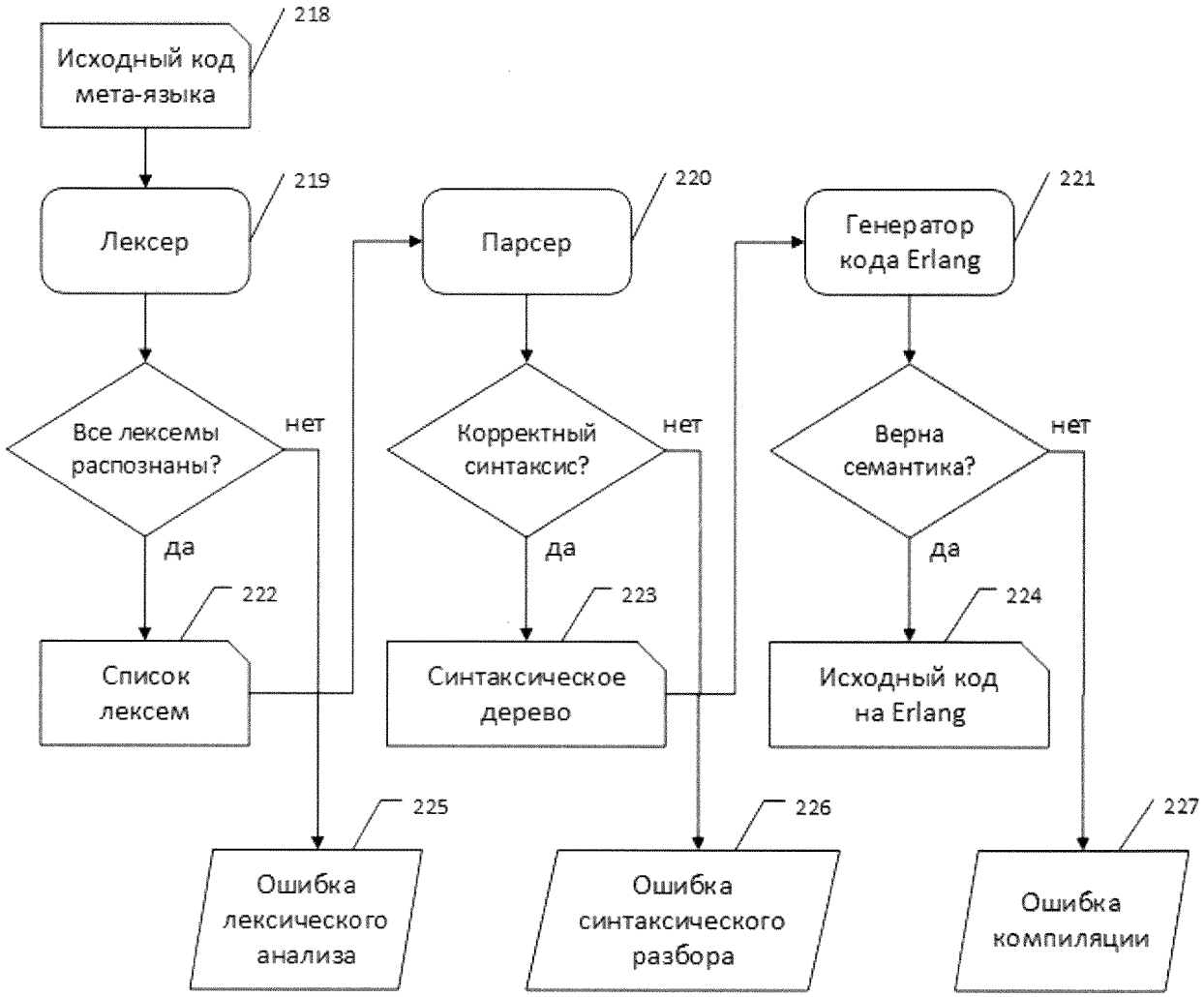
Фиг.7


201
Аккаунт
202
Схема 1
204
205
Модуль 1
Модуль 2
Объект 1
Объект 2
Объект 3
Объект 4
связи
206
207
208
Мета-модель
Описание данных
203
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

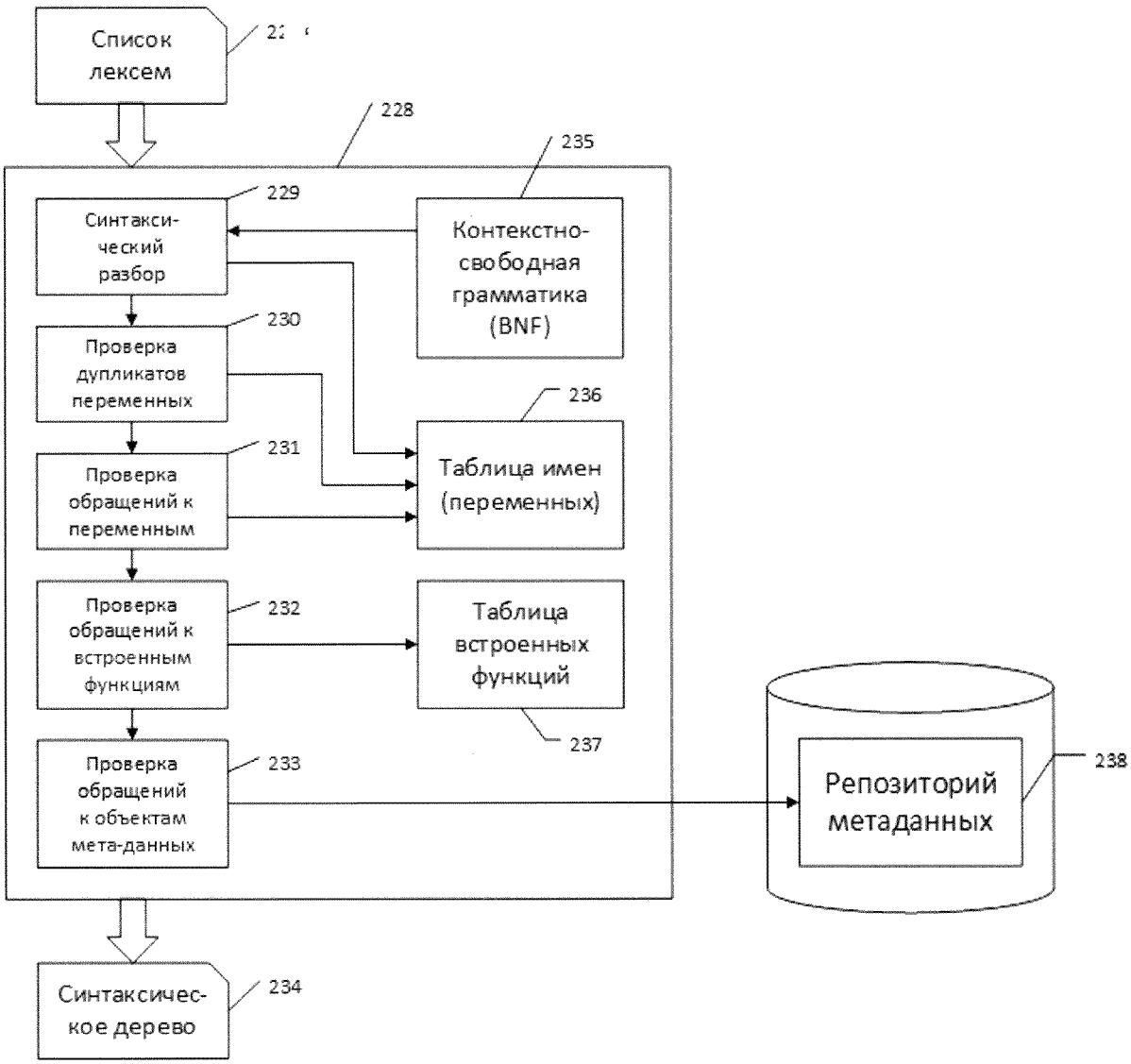
Фиг.8


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

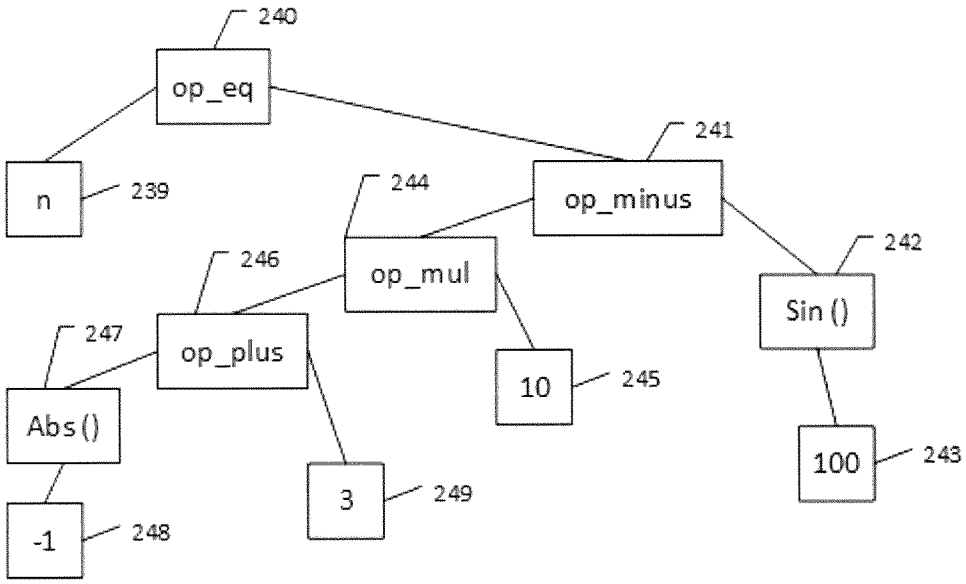
Фиг.9


Исходный код мета-языка
218
Лексер
219
Парсер
220
Генератор кода Erlang
221
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

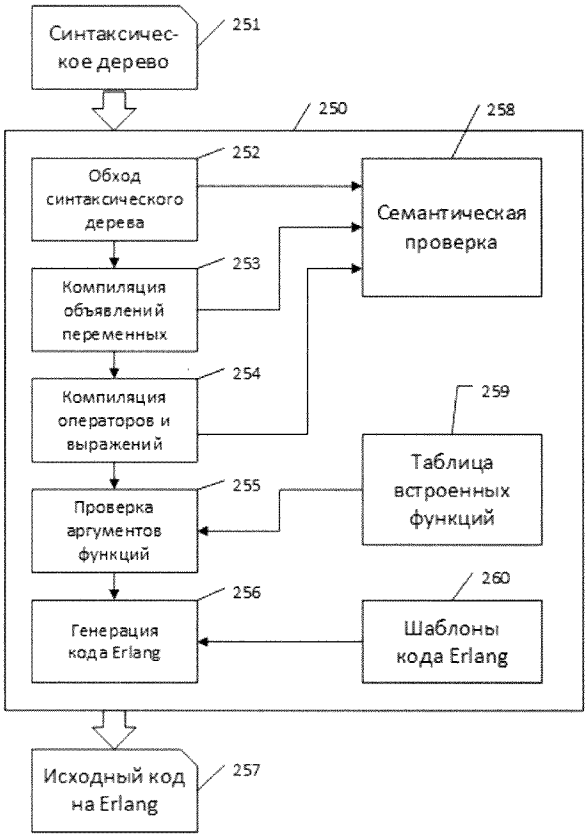
Фиг. 10


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)
232
Проверка обращений к встроенным функциям
Таблица встроенных функций
237
233
Проверка обращений к объектам мета-данных
Репозиторий метаданных
238
Синтаксичес-кое дерево
234

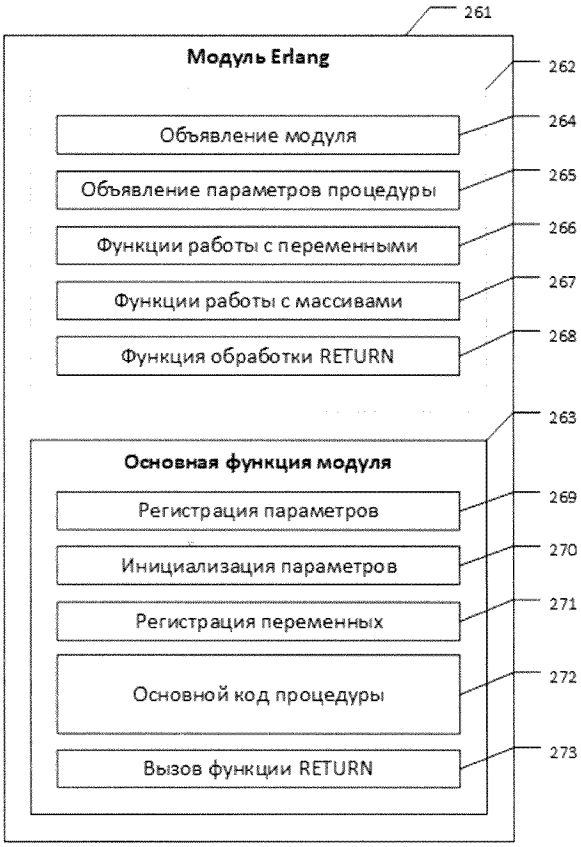
Фиг.11


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

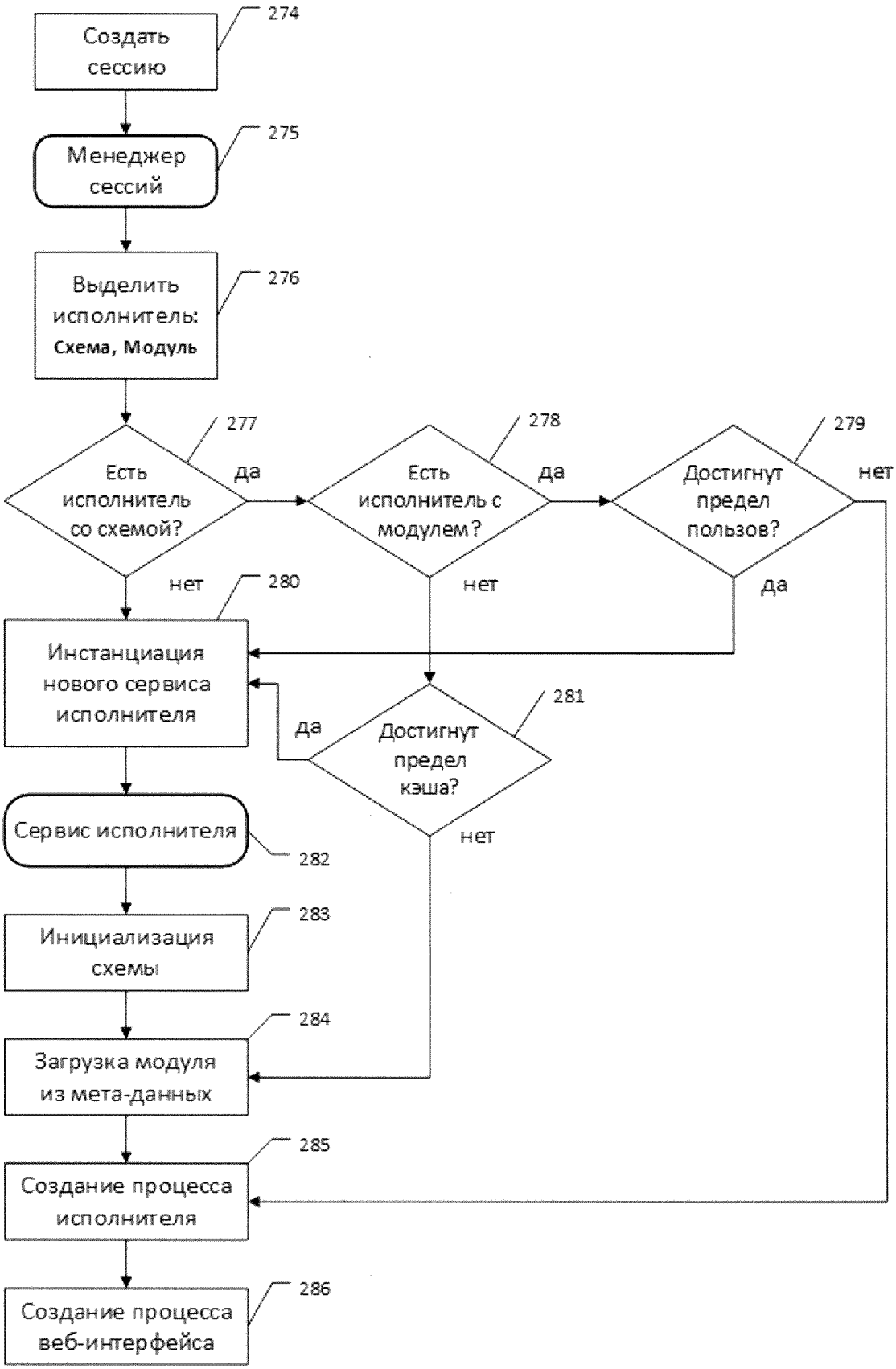
Фиг.12


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

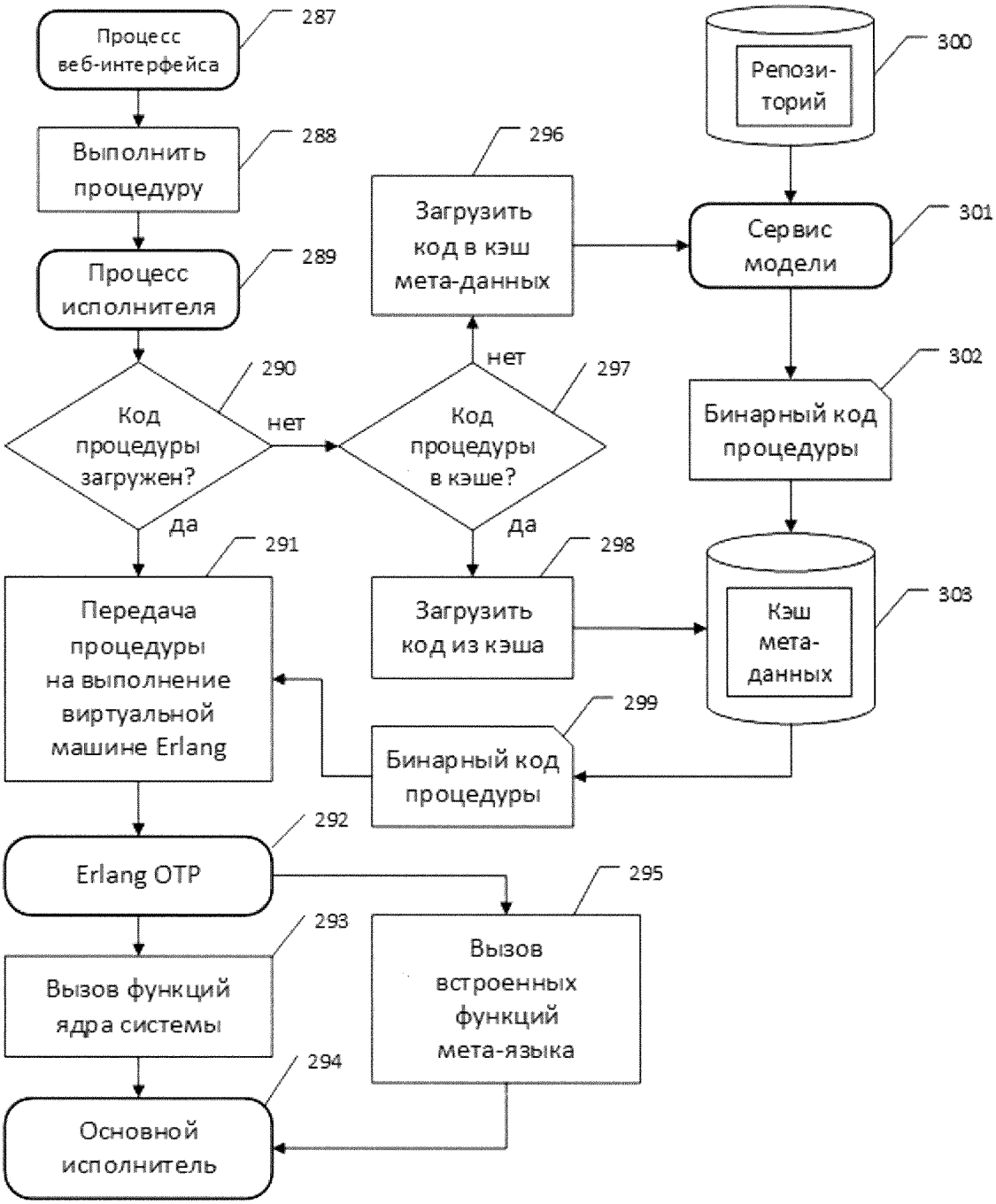
Фиг. 13


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

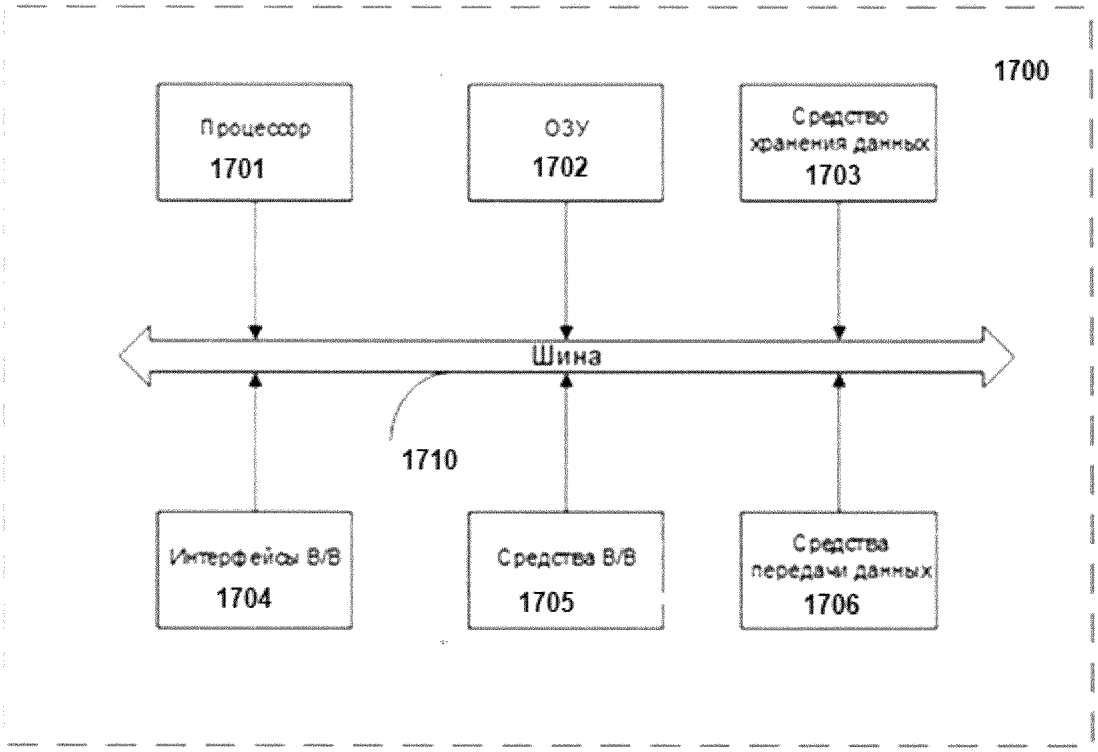
Фиг. 14


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

INTERNATIONAL SEARCH REPORT

International application No.
PCT/RU 2020/000595

A. CLASSIFICATION OF SUBJECT MATTER

G06F 8/20 (2018.01) G06F 8/30 (2018.01) G06F 9/44 (2018.01)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 8/00, 8/10, 8/20, 8/30, 8/35, 9/00, 9/06, 9/44, 7/00, 11/00, 11/36, 17/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

PatSearch (RUPTO Internal), USPTO, PAJ, Espacenet, Information Retrieval System of FIPS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2013/0104100 A1 (SAP AG) 25.04.2013, paragraphs [0015], [0025], [0039], [0045], [0069]	1-8
A	US 10324690 B2 (VERMEG SERVICES SARL) 18.06.2019	1-8
A	US 7735062 B2 (OUTSYSTEMSEM REDE, S.A.) 08.06.2010	1-8

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 July 2021 (14.07.2021)	29 July 2021 (29.07.2021)
Name and mailing address of the ISA/ RU Facsimile No.	Authorized officer Telephone No.

ОТЧЕТ О МЕЖДУНАРОДНОМ ПОИСКЕ

Номер международной заявки

PCT/RU 2020/000595

A. КЛАССИФИКАЦИЯ ПРЕДМЕТА ИЗОБРЕТЕНИЯ

G06F 8/20 *(2018.01)*
G06F 8/30 *(2018.01)*
G06F 9/44 *(2018.01)*

Согласно Международной патентной классификации МПК

B. ОБЛАСТЬ ПОИСКА

Проверенный минимум документации (система классификации с индексами классификации)

G06F 8/00, 8/10, 8/20, 8/30, 8/35, 9/00, 9/06, 9/44, 7/00, 11/00, 11/36, 17/00

Другая проверенная документация в той мере, в какой она включена в поисковые подборки

Электронная база данных, использовавшаяся при поиске (название базы и, если, возможно, используемые поисковые термины)

PatSearch (RUPTO Internal), USPTO, PAJ, Espacenet, Information Retrieval System of FIPS

C. ДОКУМЕНТЫ, СЧИТАЮЩИЕСЯ РЕЛЕВАНТНЫМИ:

Категория*	Цитируемые документы с указанием, где это возможно, релевантных частей	Относится к пункту №
A	US 2013/0104100 A1 (SAP AG) 25.04.2013, параграфы [0015], [0025], [0039], [0045], [0069]	1-8
A	US 10324690 B2 (VERMEG SERVICES SARL) 18.06.2019	1-8
A	US 7735062 B2 (OUTSYSTEMSEM REDE, S.A.) 08.06.2010	1-8

☐ последующие документы указаны в продолжении графы C.

☐ данные о патентах-аналогах указаны в приложении

* Особые категории ссылочных документов:

"A" документ, определяющий общий уровень техники и не считающийся особо релевантным

"D" документ, цитируемый заявителем в международной заявке

"E" более ранняя заявка или патент, но опубликованная на дату международной подачи или после нее

"L" документ, подвергающий сомнению притязание(я) на приоритет, или который приводится с целью установления даты публикации другого ссылочного документа, а также в других целях (как указано)

"O" документ, относящийся к устному раскрытию, использованию, экспонированию и т.д.

"P" документ, опубликованный до даты международной подачи, но после даты испрашиваемого приоритета

"T" более поздний документ, опубликованный после даты международной подачи или приоритета, но приведенный для понимания принципа или теории, на которых основывается изобретение

"X" документ, имеющий наиболее близкое отношение к предмету поиска; заявленное изобретение не обладает новизной или изобретательским уровнем, в сравнении с документом, взятым в отдельности

"Y" документ, имеющий наиболее близкое отношение к предмету поиска; заявленное изобретение не обладает изобретательским уровнем, когда документ взят в сочетании с одним или несколькими документами той же категории, такая комбинация документов очевидна для специалиста

"&" документ, являющийся патентом-аналогом

Дата действительного завершения международного поиска	Дата отправки настоящего отчета о международном поиске
14 июля 2021 (14.07.2021)	29 июля 2021 (29.07.2021)

Наименование и адрес ISA/RU:
Федеральный институт промышленной собственности,
Бережковская наб., 30-1, Москва, Г-59,
ГСП-3, Россия, 125993
Факс: (8-495) 531-63-18, (8-499) 243-33-37

Уполномоченное лицо:
Мухина Т.

Телефон № 8(495)531-65-15