УДК 004.41
DOI: 10.26102/2310-6018/2024.45.2.040

Разработка элементов технологии переноса геоинформационной системы Integro на Linux на основе системного подхода

А.С. Шумихин✉

Всероссийский научно-исследовательский геологический нефтяной институт, Москва, Российская Федерация

Резюме. Статья посвящена выбору технологического подхода к задаче переноса Windows-приложения, разработанного с использованием некроссплатформенной библиотеки компонентов пользовательского интерфейса и имеющего плагинную архитектуру, на операционную систему Linux. Описывается подход, который может применяться в случаях, когда гибкость и низкие накладные расходы более важны, чем возможность использования готового решения. В работе использованы методы системного анализа. Рассмотрены существующие варианты готовых решений и их элементов. Итоговое решение состоит в использовании разработки, управляемой моделями, для разделения компонентов, специфичных для платформы и независимых от нее, хорошо определенными программными интерфейсами. Разработанный вариант технологии порождения исходного кода из декларативного описания модели объектно-ориентированных интерфейсов позволяет организовать взаимодействие объектов, разделенных границей модулей, компиляторов и библиотек времени выполнения. С использованием стека технологий XML обеспечены валидация, автодополнение и преобразование декларативного описания модели в исходный код на языке C++. Представление интерфейсов основано на таблицах виртуальных методов, каждый из которых является функцией в стиле C. В качестве ссылки на интерфейс объекта используется структура, содержащая указатель на объект и указатель на таблицу виртуальных методов. Для каждого интерфейса генерируются определения функций, описания структуры таблицы виртуальных методов и ссылки на интерфейс, обертки для ссылок и базовые классы реализаций на C++. Технология успешно использована при разработке геоинформационной системы INTEGRO.

Ключевые слова: плагинная архитектура, объектно-ориентированное программирование, двоичный интерфейс приложений, C++, INTEGRO.

Для цитирования: Шумихин А.С. Разработка элементов технологии переноса геоинформационной системы INTEGRO на Linux на основе системного подхода. *Моделирование, оптимизация и информационные технологии.* 2024;12(2). URL: https://moitvivt.ru/ru/journal/pdf?id=1584 DOI: 10.26102/2310-6018/2024.45.2.040

Developing elements of technology to port Integro geoinformation system to Linux based on the system approach

A.S. Shumikhin✉

All-Russian research geological oil institute, Moscow, the Russian Federation

Abstract: The article discusses choosing a technological approach to porting a Windows desktop application that utilizes a non-cross-platform user interface component library, and that implements a plugin architecture, to Linux. The approach described can be used in cases when flexibility and low overhead is preferred over a ready-made solution. The work has been done based on systems analysis. A collection of existing options and their elements is examined. The resulting solution consists in using model-driven software development to separate platform-specific components from cross-platform ones

by means of well-defined programming interfaces. The suggested version of a technology by which source code is generated from a declarative description of an object-oriented interface model provides interoperability between objects, residing in different modules and separated by a compiler or a runtime library boundary. The XML technology stack is used to implement validation, code completion and transformation of model descriptions into C++ source code. Interfaces are represented by virtual method tables. Each method is a C-style function. A reference to an interface is a structure containing a pointer to a virtual method table, and a pointer to an object instance. For each interface there is a number of declarations and definition generated: a set of function declarations, a virtual method table declaration, an interface reference structure declaration, wrappers for interface references and implementation base classes in C++. The technology is successfully applied in the development of INTEGRO geographic information system.

Keywords: plug-in architecture, object-oriented programming, application binary interface, C++, INTEGRO.

For citation: Shumikhin A.S. Developing elements of technology to port Integro geoinformation system to Linux based on the system approach. *Modeling, Optimization and Information Technology*. 2024;12(2). URL: https://moitvivt.ru/ru/journal/pdf?id=1584 DOI: 10.26102/2310-6018/2024.45.2.040 (In Russ.).

Введение

Отвечая на растущую необходимость в обеспечении технологической независимости страны в области программного обеспечения, сотрудники отделения Геоинформатики «ВНИИГеосистем» ФГБУ «ВНИГНИ» подготовили бета-версию геоинформационной системы (ГИС) INTEGRO для операционных систем семейства Linux, широко представленного дистрибутивами отечественной разработки. ГИС INTEGRO разрабатывается с 2007 года, а с 2010 года применяется в десятках геологоразведочных предприятий. Она обладает функционалом, направленным на обеспечение потребностей геологоразведочных работ, и сопоставима в этом с системой ArcGis разработки американской компании Esri, повсеместно применявшейся до недавнего времени в геологоразведочных и нефтегазодобывающих предприятиях [1]. В настоящее время она позиционируется как импортозамещающее геоинформационное обеспечение для решения задач недропользования [2, 3].

Созданию Linux-версии ГИС INTEGRO предшествовал предварительный этап научных исследований, связанных с выбором критериев эффективности программной среды и анализом по этим критериям альтернативных вариантов ее реализации, продолжительный этап внедрения обновленной кроссплатформенной плагинной архитектуры и поддерживающей ее технологии [4], а затем – работ по преобразованию ГИС INTEGRO в мультиплатформенный комплекс [5]. В исследовательской части использована методика системного анализа, изложенная Квейдом [6] и другими авторами, состоящая из следующих шагов:

1. Постановка задачи.
2. Выбор альтернативных путей решения задачи.
3. Исследование ресурсов, расходуемых на решение задачи.
4. Составление модели.
5. Выбор критериев оценки.
6. Сравнение альтернатив и принятие решения.

Постановка задачи и исследования на основе системного подхода

Первоначально ГИС INTEGRO разрабатывалась как Windows-приложение в среде разработки Embarcadero C++ Builder из состава пакета Embarcadero RAD Studio.

Для создания пользовательского интерфейса использовалась библиотека визуальных компонентов Visual Control Library (VCL). В своих первых версиях система уже поддерживала расширение функциональности при помощи подключаемых модулей (плагинов). Плагин INTEGRO мог предоставлять дополнительные функции обработки данных и соответствующие им элементы пользовательского интерфейса. Однако создавать такие модули необходимо было с использованием тех же библиотек и средств разработки, какие применялись и для базовой системы. В то же время, задача импортозамещения требовала обеспечить максимальную независимость приложения от проприетарных продуктов иностранного производства.

С учетом изложенного выше, были зафиксированы следующие требования к внедряемым изменениям:

1. Расширяемость.

Следовало создать возможность использовать для создания подключаемых модулей более широкий спектр языков, компиляторов и интегрированных сред разработки, в том числе свободно распространяемых. Так, рассматривалась возможность встраивания интерпретатора Python, и проводились соответствующие эксперименты [7]. Это позволило бы упростить привлечение новых специалистов к разработке расширений.

2. Переносимость.

Приложение и плагины к нему должны были быть легко портируемы на другие платформы, в первую очередь, на операционную систему Linux, поскольку именно она способна составить доступную альтернативу операционной системе Windows.

3. Бесшовное внедрение изменений.

Требовалось, чтобы вносимые изменения не приводили к нарушению работы приложения, отдельных его модулей или функций. В таком случае вновь создаваемые расширения могли бы разрабатываться по новой технологии, в то время как разработанные ранее модули продолжали бы функционировать до их замены. Таким образом, исключалась необходимость поддержки двух различных версий приложения одновременно.

Выполнению указанных требований мешали следующие обстоятельства:

1. Проблема совместимости двоичного интерфейса приложений.

Двоичный интерфейс приложений (Application Binary Interface, ABI), то есть набор соглашений о представлении данных в памяти и о порядке вызова функций, определенных на языке программирования высокого уровня, может различаться при использовании различных языков программирования, компиляторов, параметров компиляции и библиотек времени выполнения. В достаточной мере стандартизирован только ABI для языка программирования C [1]. Язык C, однако, не предоставляет непосредственной поддержки для парадигмы объектно-ориентированного программирования. В то же время именно эта парадигма наилучшим образом зарекомендовала себя при разработке приложений с развитым графическим пользовательским интерфейсом и сложной структурой обрабатываемых данных. Именно в связи с этим в качестве основного языка разработки ГИС INTEGRO выбран C++, а двоичный интерфейс между модулями обеспечивается, главным образом, привязкой объектов C++ к языку C. Объем стереотипного кода, который требуется для такой привязки, осложняет разработку исходного кода подключаемых модулей и необходимых для них программных интерфейсов. В случае разработки модулей на других языках программирования требуется дополнительно разрабатывать код для привязки объектов к этим языкам. Таким образом, проблема совместимости двоичного интерфейса

[1] Sutter H. Defining a Portable C++ ABI. 2014. URL: https://isocpp.org/files/papers/n4028.pdf (дата обращения: 08.04.2024).

приложений ограничивала расширяемость приложения ГИС INTEGRO.

2. Зависимость от средств разработки.

Как следствие сложностей, связанных с привязкой объектно-ориентированных интерфейсов к языку C, в первом поколении плагинов INTEGRO для подключения дополнительных элементов пользовательского интерфейса на двоичном уровне в некоторых случаях применялись непосредственно компоненты VCL. Такое решение ограничивало выбор средств разработки, применяемых для плагинов, компиляторами и библиотеками Embarcadero C++ Builder и Delphi, а также средой разработки RAD Studio. Кроме того, использование библиотеки VCL, которая поддерживает разработку приложений только для операционной системы Windows[2], противоречило требованию переносимости. К сожалению, постепенная ее замена библиотеки VCL на другой функциональный аналог была невозможна ввиду низкой функциональной совместимости (intropерability) VCL с другими библиотеками. Единовременная же замена VCL привела бы к появлению альтернативной версии приложения, то есть, было бы нарушено требование к бесшовности внедрения изменений.

Чтобы преодолеть указанные препятствия к выполнению изложенных выше требований, были поставлены следующие цели:

1. Выбрать архитектурное решение и технологию разработки, удовлетворяющие этим требованиям.
2. В соответствии с выбранными архитектурой и технологией модифицировать исходный код основного приложения.
3. Перейти к использованию выбранной технологии при разработке новых плагинов.
4. По новой технологии разработать плагины взамен существующих.
5. Портировать приложение и плагины на операционную систему Linux.

Критерием достижения поставленных целей было определено выполнение следующих условий:

6. Функционирование в составе приложения плагинов, разработанных с помощью иных средств разработки, нежели те, которые применяются в базовой системе.
7. Функционирование приложения под управлением операционной системы Linux.
8. Совместимость версий плагинов к приложению для Windows и Linux на уровне исходного кода.
9. Сохранение прежней функциональности системы по мере внедрения выбранных архитектурных и технологических решений.

Поиск решения

Для поиска приемлемого решения была составлена матрица сравнения существующих технологий, которые могли бы быть использованы для построения плагинной системы INTEGRO. Список не претендует на полноту, но отражает альтернативы, которые были изучены при разработке INTEGRO. Рассматривались следующие признаки:

1. Наличие разрешительной лицензии.
2. Совместимость двоичного интерфейса приложений (ABI).
3. Наличие привязок к языкам, отличным от C++.
4. Кроссплатформенность.
5. Непосредственная поддержка объектно-ориентированной парадигмы.
6. Использование прямых вызовов функций, минимизирующее потери

[2] RAD Studio devs. RAD Studio 12 Topics. GUI Application Frameworks. URL: https://docwiki.embarcadero.com/RADStudio/Athens/en/GUI_Application (дата обращения: 08.04.2024)

производительности, связанные с обеспечением нужного уровня абстракции.

7. Неинвазивность, в данном случае отсутствие необходимости изменять код C++-объекта для обеспечения его совместимости с данной архитектурой.

В число признаков, отражаемых в таблице, не был включен порог вхождения, который невозможно было оценить объективно.

Список был дополнен (тогда еще гипотетической) собственной минималистичной технологией, удовлетворяющей всем необходимым критериям. Для ее разработки планировалось позаимствовать отдельные идеи из рассмотренных подходов. Сравнение технологий взаимодействия через границу модулей представлено в Таблице 1.

Таблица 1 – Сравнение технологий взаимодействия через границу модулей
Table 1 – Comparison of technologies to support interoperability over module boundary

	Qt[3]	Poco OSP[4]	SOF[5]	nOSGI[6]	CTK[7]	Celix C++[8]	Celix C	COM/Active [8]	XPCOM[9]	CORBA [9]	Своя разработка
Год завершения разработки			2011	2013							
Свободная лицензия	+	+	?	+	+	+	+	+	+	+	
ABI-совместимость	-	-	-	-	-	-	+	+	+	+	+
Привязки	-	-	+	-	-	-	-	+	+	+	+
Кроссплатформенность	+	+	+	+	+	+	+	-	+	+	+
Поддержка ООП	+	+	+	+	+	+	-	+	+	+	+
Прямые вызовы	+	+	+	+	+	+	+	+	-	+/-	+
Неинвазивность	-	+	+	+	+	+	+	-	-	-[9]	+

Как можно видеть из приведенной таблицы, готовое решение, полностью удовлетворяющее всем необходимым критериям, найдено не было. В связи с этим следовало рассмотреть отдельные подходы, использованные в этих и других известных решениях, чтобы оценить их пригодность для разработки собственной технологии.

[3] QT developers. About Qt. URL: https://wiki.qt.io/About_Qt (дата обращения: 08.04.2024).
[4] POCOpro C++ Frameworks. URL: https://docs.pocoproject.org/pro/00100-OSPOverview.html (дата обращения: 08.04.2024).
[5] SOF - An OSGI-like modularization framework for C++. URL: https://www.codeproject.com/Articles/28426/SOF-An-OSGI-like-modularization-framework-for-C (дата обращения: 08.04.2024).
[6] Sascha Zelzer. OSGi and C++. URL: http://blog.cppmicroservices.org/2012/03/29/osgi-and-c++/ (дата обращения: 08.04.2024).
[7] CTK Plugin Framework: Introduction URL: https://commontk.org/index.php/Documentation/CTK_Plugin_Framework:_Introduction (дата обращения: 08.04.2024).
[8] Apache Celix Introduction URL: https://celix.apache.org/docs/2.4.0/celix/documents/README.html (дата обращения: 08.04.2024).
[9] XPCOM Part 1: An introduction to XPCOM URL: https://web.archive.org/web/20130805144401/http://www.ibm.com/developerworks/webservices/library/co-xpcom/index.html (дата обращения: 08.04.2024).

Итак:

1. Совместимость объектно-ориентированного ABI, когда она имеется, обеспечивается одним из двух способов: с помощью маршаллинга, то есть, преобразования в массив байт (XPCOM, CORBA), либо прямыми вызовами функций в стиле C с использованием хорошо стандартизированного соглашения о вызовах. Очевидно, что маршаллинг вызывает дополнительные накладные расходы на каждый вызов функции, поэтому при обращении к объектам, находящимся в том же адресном пространстве, его использование нежелательно.

2. В тех решениях, в которых имеются привязки интерфейсов к различным языкам программирования (XPCOM, CORBA, COM/ActiveX), используется порождение исходного кода по декларативному описанию, заданному на специально разработанном языке описания интерфейсов (IDL). Каждая из названных технологий предлагает свой вариант языка IDL.

3. Кроссплатформенность достигается путем следования стандарту языков программирования C и C++ и использованием (либо созданием) кроссплатформенных инструментов разработки.

4. Неинвазивностью среди рассмотренных технологий обладают только те, в которых применяется несовместимый двоичный интерфейс C++, что нельзя было назвать приемлемым подходом. Существует, однако, решение, подсказанное языком Rust [10] . Это так называемые динамические трейт-объекты (dynamic trait objects), представляющие собой адаптеры [10] интерфейсов, передаваемые по значению. В сущности, трейт-объект представляет собой структуру, содержащую ссылку на адаптируемый объект и ссылку на таблицу виртуальных методов. Передача таких объектов по значению позволяет использовать их аналогично ссылкам на интерфейсы.

Таким образом, вырисовываются очертания приемлемого решения: создается модель интерфейсов в виде описания на некотором декларативном языке [11]; по заданной модели порождается код на языке C, обеспечивающий совместимый двоичный интерфейс приложений, в котором объектно-ориентированная парадигма программирования поддерживается за счет трейт-объектов; по той же модели порождаются привязки интерфейса к языку C++ и, при необходимости, к другим языкам программирования. Наибольшую сложность в этом подходе представляет собой создание спецификации языка описания интерфейсов и транслятора для него. Кроме того, для поддержки разработки моделей желательно иметь средства автодополнения и валидации кода на таком языке. Языки описания интерфейсов (IDL) в COM, XPCOM и CORBA имеют синтаксис, подобный языку C++. Для разработки компиляторов с таких языков применяются программы генерации лексических и синтаксических анализаторов (парсеров), такие как lex и yacc [12], позволяющие представить описание модели в виде дерева синтаксического разбора (Abstract Syntax Tree, AST). Затем на основе дерева разбора генерируется исходный код на целевом языке. Отдельно создаются расширения для интегрированных сред разработки, поддерживающие автодополнение и валидацию кода на IDL. Такое решение, однако, видится избыточно сложным и недостаточно гибким на фоне генерации кода с использованием стека технологий XML (Extensible Markup Language).

Технология XML для генерации кода применялась, в частности, разработчиками библиотеки XCB (X protocol C-language Binding, реализация X-протокола для языка C). Преимущество использования XML состоит в том, что оно не требует разработки специального парсера: древовидная структура описания модели создается парсером

[10] The Rust Reference. Trait objects. URL: https://doc.rust-lang.org/reference/types/trait-object.html (дата обращения: 08.04.2024)

XML-документа. Для целей валидации и автодополнения кода XML-документа разрабатывается описание XML-схемы (XML Schema Definition). Описание схемы XML-документа также представляет собой XML-документ, который описывает структуру других XML-документов [11]. Генерация кода из XML-описания может осуществляться с применением шаблонов XSLT – еще одного вида XML-документов, описывающих правила преобразования XML-документа в другой XML-документ или текст [12]. Альтернативно код может генерироваться программой на любом языке программирования, для которого имеются библиотеки с функциями чтения XML, например, на языке Python. Располагая возможностью редактировать описание модели, его схему и шаблоны для преобразования в исходный код на выбранном языке, разработчик получает максимально гибкий инструмент для моделирования архитектуры приложения.

Примечание. Для языка XML существуют альтернативы с более лаконичным синтаксисом – YAML, JSON, но, судя по объему выдачи Google, они менее популярны. При необходимости не составит труда конвертировать описание интерфейсов в YAML, JSON, или обратно.

Реализация

Конкретная схема описания интерфейсов разрабатывалась с учетом возможностей языка C, на котором следовало реализовать связующую прослойку кода, ответственную за двоичную совместимость модулей. Описывая эту прослойку, будем называть:

– интерфейсом класса объектов – тип таблицы виртуальных методов;

– интерфейсом типа объектов – экземпляр таблицы виртуальных методов;

– интерфейсом объекта – структуру, содержащую указатели на этот объект и на таблицу виртуальных методов.

Таблица виртуальных функций класса представляет собой структуру, поля которой содержат указатели на методы, каждый из которых является функцией и принимает в качестве одного из параметров указатель на объект данного класса.

Функция может иметь возвращаемое значение и другие параметры, для каждого из которых должен быть задан тип. Тип структуры, ее поля и параметры функций имеют идентификаторы (имена).

Соответствующее описание на языке XML может выглядеть следующим образом:

```
<interface name="InterfaceName0">
  <method name="MethodName0" type="ReturnType0">
    <parameter name="ParameterName0" type="ParameterType0">
    </parameter>
  </method>
</interface>
```

Из такого описания может быть сгенерирован код в стиле C (для упрощения приводится в синтакисисе C++):

```
// объявление таблицы виртуальных методов:
struct InterfaceName0_methods;
```

[11] W3C. XML Schema Part 0: Primer Second Edition. 2004. URL: https://www.w3.org/TR/xmlschema-0/ (дата обращения: 08.04.2024).

[12] Overview of XSLT. URL: https://www.data2type.de/en/xml-xslt-xslfo/xslt/ (дата обращения: 08.04.2024).

```
// объявление типа метода:
typedef ReturnType0 InterfaceName0_MethodName0_method(void * self, ParameterType0 ParameterName0);

// определение таблицы виртуальных методов:
struct InterfaceName0_methods
  {
  InterfaceName0_MethodName0_method * MethodName0;
  };

// определение трейт-объекта («ссылки» на интерфейс):
struct InterfaceName0_trait
  {
  InterfaceName0_methods * methods;
  void * instance;
  };

// определение функции-обертки для вызова метода:
ReturnType0 InterfaceName0_MethodName0(InterfaceName0_trait self, ParameterType0 ParameterName0);
  {
  return self.methods.MethodName0(self.instance, ParameterName0);
  };
```

Структура InterfaceName0_trait используется в качестве «ссылки» на интерфейс объекта. Для вызова метода этого интерфейса служит функция InterfaceName0_MethodName0().

Как можно видеть, здесь тип указателя на экземпляр «стерт». При написании исходного кода вручную стирание типа и его последующее восстановление было бы небезопасной операцией, поскольку оно подвержено ошибкам.

В случае генерации кода ошибка может быть исключена. Однако в описании модели необходимо указать, для какого типа должен быть сгенерирован код реализации интерфейса, например:

```
<implementation interface="InterfaceName0" type="Implementation0">
</implementation>
```

В таком случае сгенерированный код может быть следующим:

```
//объявление функции, реализующей метод:
InterfaceName0_MethodName0_impl(Implementation0 * self, ParameterType0 ParameterName0);

//объявление функции-адаптера метода для таблицы виртуальных методов:
ReturnType0 InterfaceName0_MethodName0_method(void * self, ParameterType0 ParameterName0)
{
return InterfaceName0_MethodName0_impl((Implementation0 *)self, ParameterName0);
};

//объявление функции-конструктора "ссылки" на интерфейс:
InterfaceName0_trait InterfaceName0_to_Implementation0(Implementation0 * instance)
{
static InterfaceName0_methods methods {InterfaceName0_MethodName0_method};
struct InterfaceName0_trait result={&methods, instance};
return result;
};
```

Здесь функция InterfaceName0_to_Implementation0() создает «ссылку» на интерфейс InterfaceName0 к объекту Implementation0. Тип Implementation0 должен быть определен заранее, а метод MethodName0() интерфейса InterfaceName0 необходимо определить в виде функции InterfaceName0_MethodName0_impl(), объявленной объявление которой содержится в сгенерированном коде.

При использовании этих функций тип указателя на экземпляр восстанавливается безопасно, поскольку таблица виртуальных методов трейт-объекта может содержать только такие указатели на функции, которые соответствуют стертому типу указателя на экземпляр.

Реально используемая в INTEGRO структура XML для описания модели и генерируемый на ее основе код несколько сложнее, чем в приведенных примерах. В них реализованы такие возможности как «нулевые» трейт-объекты, возвращаемые значения по умолчанию, преобразование типов трейт-объектов. Чтобы исключить нарушение правила одного определения[13], предусмотрено версионирование интерфейсов. Кроме того, генерируется код в стиле C++ как для вызова методов объектов (обертки), так и для базовых классов объектов, реализующих интерфейсы модели (заготовки реализации). Описание набора интерфейсов, интерфейса, метода и его параметра может содержать строку документации. Строки документации являются неотъемлемой частью описания интерфейсов. Предполагается, что документация должна содержать описание условий и порядка взаимодействия интерфейсов, а изменение этих описаний означает изменение интерфейса.

Заключение

С использованием описанного подхода представлен набор интерфейсов для расширения функциональности приложения INTEGRO и разработан ряд плагинов. Часть плагинов разработана средствами Microsoft Visual Studio. Основное приложение и плагины портированы на операционную систему Linux, при этом сборка плагинов под Windows и Linux производится из единой кодовой базы. Условная компиляция в исходном коде плагинов для этого не используется. В процессе портирования библиотека компонентов пользовательского интерфейса VCL была заменена фреймворком Qt. При этом за счет использования паттерна «мост» [10] приложение продолжало сохранять функциональность, реализованную средствами VCL, в течение всего времени, пока разрабатывалась ее замена на Qt. Таким образом, критерии достижения поставленных целей были полностью выполнены.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Варламов А.И., Гогоненков Г.Н. Состояние и проблемы импортозамещения в области геофизических работ на нефть и газ. *Геоинформатика*. 2018;(3):3–7.
Varlamov A.I., Gogonenkov G.N. Geophysical exploration for oil and gas: current status and problems of import substitution. *Geoinformatika.* 2018;(3):3–7. (In Russ.).
2. Черемисина Е.Н., Финкельштейн М.Я., Деев К.В., Большаков Е.М. ГИС INTEGRO. Состояние и перспективы развития в условиях импортозамещения. *Геология нефти и газа.* 2021;(3):31–40. https://doi.org/10.31087/0016-7894-2021-3-31-40
Cheremisina E.N., Finkel'shtein M.Ya., Deev K.V., Bol'shakov E.M. GIS INTEGRO. Status and prospects for development in the context of import substitution. *Geologiya*

[13] ISO/IEC (2003). ISO/IEC 14882:2003(E): Programming Languages – C++ §3.2 One definition rule [basic.def.odr] para. 3

nefti i gaza = Russian Oil and Gas Geology. 2021;(3):31–40. (In Russ.). https://doi.org/10.31087/0016-7894-2021-3-31-40

3. Черемисина Е.Н., Финкельштейн М.Я., Любимова А.В. ГИС INTEGRO – импортозамещающий программно-технологический комплекс для решения геолого-геофизических задач. *Геоинформатика.* 2018;(3):8–17.
   Cheremisina Ye.N., Finkelstein M.Ya., Lyubimova A.V. GIS INTEGRO – import substitution software for geological and geophysical tasks. *Geoinformatika.* 2018;(3):8–17. (In Russ.).
4. Шумихин А.С. Особенности архитектуры ГИС INTEGRO. *Геоинформатика.* 2018;(3):68–75.
   Shoumikhin A.S. Architectural features of GIS INTEGRO software. *Geoinformatika.* 2018;(3):68–75. (In Russ.).
5. Черемисина Е.Н., Финкельштейн М.Я., Деев К.В., Мурадян А.В. Ближайшие перспективы развития геоинформационного комплекса INTEGRO. *Геоинформатика*. 2021;(1):5–10.
   Cheremisina Ye.N., Finkelstein M.Ya., Deyev K.V., Muradyan A.V. The nearest prospects of the geoinformation complex INTEGRO development. *Geoinformatika.* 2021;(1):5–10. (In Russ.).
6. Квейд Э. *Анализ сложных систем.* Москва: Издательство «Советское радио»; 1969. 520 с.
   Quade E.S. *Analysis for Military Decisions.* Rand McNally; 1964. 382 p.
7. Деев К.В. Перспективы развития ГИС INTEGRO. *Геоинформатика*. 2020;(1):3–7.
   Deyev K. Perspective ways of the GIS INTEGRO development. *Geoinformatika.* 2020;(1):3–7. (In Russ.).
8. Oberg R.J. *Understanding & Programming COM+: A Practical Guide to Windows 2000 DNA.* Upper Saddle River, NJ: Pretence Hall PTR; 2000. 656 p.
9. Цимбал А. *Технология CORBA.* Санкт-Петербург: Питер; 2001. 624 с.
   Tsimbal A. *Tekhnologiya CORBA.* Saint Petersburg: Piter; 2001. 624 p. (In Russ.).
10. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Addison-Wesley, Pearson Education; 1994. 416 p.
11. Frankel D.S. *Model Driven Architecture™: Applying MDA™ to Enterprise Computing.* Indianapolis: Wiley Publishing; 2003. 354 p.
12. Levine J.R., Mason T., Brown D. *lex & yacc.* Sebastopol, CA: O'Reilly Media; 1992. 388 p.

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Шумихин Александр Сергеевич, старший научный сотрудник, Всероссийский научно-исследовательский геологический нефтяной институт, Москва, Российская Федерация.
e-mail: shmkhn@geosys.ru

Aleksandr S. Shumikhin, Senior Researcher, All-Russian research geological oil institute, Moscow, the Russian Federation.

Статья поступила в редакцию 07.06.2024; одобрена после рецензирования 14.06.2024; принята к публикации 21.06.2024.

The article was submitted 07.06.2024; approved after reviewing 14.06.2024; accepted for publication 21.06.2024.