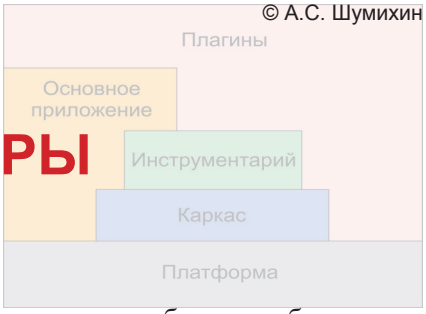
УДК 004.42

А.С. Шумихин

ОСОБЕННОСТИ АРХИТЕКТУРЫ ГИС INTEGRO



В настоящее время в отделении Геоинформатики «ВНИИГеосистем» ФГБУ «ВНИГНИ» происходит переход на обновленную технологию расширения функциональности системы при помощи подключаемых модулей – плагинов. Решение перевести ГИС INTEGRO на новую плагинную архитектуру сложилось под влиянием растущих требований к функциональности системы и к ее качеству как программного продукта.

Прежде каждый плагин INTEGRO мог предоставлять только один вид дополнительной функциональности, будь то, к примеру, инструменты для работы с окном карты или вкладки боковых панелей. Если требовалось обеспечить взаимодействие таких дополнений друг с другом, разработчик был вынужден искать собственный способ обеспечить его. Разделение плагинов по типам было связано с тем, что их программные интерфейсы зависели от различных наборов сторонних библиотек. Плагины, предоставляющие свои собственные визуальные компоненты, должны были создаваться с использованием компилятора и библиотек того же производителя (а именно – Embarcadero C++ Builder, VCL) и той же версии, что и основное приложение, в то время как для разработки других плагинов можно было использовать альтернативные инструменты.

Морально устаревал и пользовательский интерфейс приложения. Единственная панель инструментов была фиксированной, набор доступных для нее элементов управления сводился к кнопке и примитивному полю ввода, отсутствовала возможность настройки меню и панели инструментов пользователем. Ощущался недостаток способов стыковки боковых панелей. Для поддержания единообразия внешнего вида и поведения элементов пользовательского интерфейса, предоставляемых плагинами, требовались значительные организационные усилия и существенные трудозатраты. Часть этих проблем можно было решить, используя сторонние библиотеки компонентов пользовательского интерфейса, но потребность в их наличии и в опыте работы с ними усложняла бы разработку плагинов.

С накоплением все большего количества подключаемых функций и с усложнением их взаимосвязей возникла потребность преодолеть названные ограничения. Для этого необходимо было усовершенствовать как пользовательский интерфейс системы, что влекло за собой переработку соответствующей части кода плагинов, так и собственно ее плагинную архитектуру. В части пользовательского интерфейса предстояло реализовать настраиваемые меню и панели инструментов, стыкуемые боковые панели, а также расширить набор доступных видов инструментов с использованием дополнительных сторонних библиотек. Попутно был запланирован переход к полномасштабной поддержке кодировки UNICODE. Для этого, в свою очередь, требовалось перейти на более современные версии инструментов разработчика. В части плагинной архитектуры желательно было устранить зависимость интерфейсов подключаемых модулей от конкретных инструментов и библиотек и позволить разработчикам компоновать связанные функции в одном модуле. Одновременно возникло понимание, что переработка плагинной архитектуры системы позволяет заложить основу для возможного переноса приложения на отличные от Microsoft Windows платформы. При обновлении плагинной архитектуры было решено уделить особое внимание технологичности процесса разработки плагинов и использующего их кода основного приложения. Для этого предполагалось как можно более полно задействовать возможности объектно-ориентированного языка программирования и среды разработки, свести к минимуму затраты труда на написание трафаретного кода. Разумеется, на интерфейсы плагинов распространялись и общие требования по эффективности использования памяти и процессорного времени, налагаемые геоинформационной системой в связи с большими объемами обрабатываемых ею данных.

В первую очередь были рассмотрены существующие технологии в области плагинных систем. Наибольшего внимания среди них засуживает спецификация OSGi [4]. Изначально разработанная для Java-приложений, она имеет, тем не менее, несколько реализаций для C++, таких как nOSGi и SOF, и едва ли не большее количество «клонов», среди которых Poco OSP, CTK Plugin Framework, Celix (список взят из краткого обзора, подготовленного разработчиком CTK [4]). Альтернативу перечисленному составляла

самостоятельная разработка фреймворка на основе трехзвенных архитектур ActiveX, XPCOM, CORBA или независимо от них. Каждая из этих разработок обладает своими достоинствами и недостатками, обзор которых заслуживает отдельной статьи. Следует, однако, назвать основные причины, по которым они не были использованы:

- Poco OSP – коммерческая лицензия;
- SOF – использование классов C++ в качестве интерфейсов, что ограничивает выбор средств разработки одним компилятором;
- nOSGi – лицензия, не допускающая коммерческое использование;
- CTK – зависимость от фреймворка Qt;
- Celix – поддержка только C, но не C++;
- ActiveX – отсутствие переносимости;
- XPCOM, CORBA – раздутость кода, отвечающего за вызов методов.

Возможно, здесь приведены не все возможные варианты, которые были рассмотрены или которые следовало рассмотреть, но, так или иначе, была применена оригинальная разработка, о которой рассказывается ниже.

Интерфейс плагинов на двоичном уровне

Чтобы предоставить разработчику плагинов свободу в выборе компилятора, необходимо было решить проблему совместимости соглашений о представлении объектов в памяти и о вызове методов, то есть так называемого двоичного интерфейса приложений (Applcation Binary Interface, ABI). Дело в том, что единого стандарта ABI для C++ под Windows не существует. Модули C++ могут быть скомпонованы только при условии, что они скомпилированы совместимыми компиляторами с совместимыми опциями компиляции [1]. Пробел восполняют технологии COM, XPCOM, CORBA. Из них только XPCOM и CORBA обеспечивают переносимость на другие платформы, но, будучи ориентированными в основном на межпроцессное и удаленное взаимодействие, делают это ценой неоправданно высоких для одного процесса накладных расходов, связанных с передачей параметров и управлением временем жизни объектов, что выражается в снижении производительности системы. Использованный в INTEGRO подход позволяет избежать этих расходов за счет отказа от поддержки межпроцессного взаимодействия на уровне плагинных интерфейсов.

В основу объектно-ориентированных двоичных интерфейсов INTEGRO положен вызов совместимой с языком C функции в соответствии с соглашением, наиболее широко поддерживаемым компиляторами на каждой отдельно взятой платформе. Для 32-битных версий Windows это соглашение stdcall, для 64-битных – fastcall. Взаимодействие с объектом осуществляется через адаптер интерфейса. Адаптер интерфейса к объекту представляет собой структуру, содержащую указатель на таблицу таких функций и бестиповый указатель на упомянутый объект. При вызове функций из таблицы в качестве первых двух параметров передаются указатель на таблицу функций интерфейса и его указатель на объект. Реальный тип этого объекта может быть любым: можно использовать примитивный тип, составной тип, класс, область памяти. Каждый интерфейс обязательно содержит функции для получения других интерфейсов к тому же объекту. Названных структур достаточно для организации взаимодействия объектов, созданных различными компиляторами, на двоичном уровне. Однако использовать их непосредственно для разработки исходного кода неудобно и, в связи с использованием бестиповых указателей, чревато ошибками. Чтобы обойти это ограничение, используются специфичные для каждого языка прослойки автоматически генерируемого кода.

Для генерации кода, позволяющего удобно и безопасно в отношении типов обращаться к интерфейсам и реализовывать их, используется подход, называемый разработкой, управляемой моделями (Model Driven Software Engineering [2]). Вначале в соответствии со специально разработанной схемой создается модель в виде описания интерфейсов на языке XML. Затем к полученному XML-документу применяется последовательность XSLT-преобразований для получения файлов исходного кода на выбранном языке (этот выбор ограничен пока языками C или C++). XML-описание содержит идентификаторы интерфейсов, списки их методов с указанием типа возвращаемого значения и результата по умолчанию, перечни параметров каждого метода. Здесь же приводятся описания используемых нативных типов и перечисляются интерфейсы, для которых необходимо сгенерировать прослойки на каждом из языков. Для генерации кода на языке C также указываются типы, для которых необходимо реализовать адаптеры интерфейсов. Описания могут быть организованы по модульному принципу: в XML-документ можно включить указание на то, что в нем используются описания из другого XML-документа. Каждый элемент описания можно снабдить комментарием, который будет перенесен в генерируемый код. В соответствии с описанием каждого интерфейса генерируется так называемая обертка, упрощающая синтаксис вызова методов объекта, и заготовка реализации.

На языке C генерируются структуры-адаптеры интерфейсов для каждого типа, указанного с этой целью в XML-описании. Для каждого такого типа

создаются функции инициализации адаптеров, а также заготовки кода функций, реализующих их методы.

Обертки и заготовки реализаций на языке C++ представляют собой классы с соответствующими методами. В классах-обертках, наследуемых от адаптеров интерфейсов, реализуются, помимо методов интерфейса, операторы присваивания и приведения типа для обнаружения других интерфейсов объекта. В классах-заготовках реализации объявляются виртуальные методы, соответствующие методам интерфейсов, и оператор приведения к типу адаптера интерфейса.

Кроме того, генерируется код функций для получения других интерфейсов к объекту через имеющийся интерфейс. Для идентификации интерфейсов используются строковые константы. Чтобы минимизировать количество ошибок, связанных с использованием разных описаний одного и того же интерфейса в одном приложении, идентификатор интерфейса при генерации кода дополняется именем модуля, в котором объявлен этот интерфейс, и номером версии, увеличивающимся каждый раз при изменении описания данного интерфейса или других интерфейсов, от которых он прямо или косвенно зависит.

Плагинный каркас

На основе вышеописанного объектно-ориентированного интерфейса между модулями строится каркас плагинной системы INTEGRO. Описание плагинного каркаса включает в себя набор интерфейсов сервисов, используемых основным приложением INTEGRO. Под сервисом в общем смысле здесь понимается вид функциональности системы, объединенной общим назначением. В специальном смысле сервисом будет называться объект, через который можно получить доступ к такой функциональности. Центральное место среди плагинных интерфейсов INTEGRO занимает интерфейс набора расширений. Он отвечает за подключение и отключение набора расширений, а также за перечисление предоставляемых им сервисов.

Загрузку и выгрузку расширений производит объект основного приложения, называемый реестром расширений. Чтобы загрузить наборы расширений, поставляемые в виде динамических библиотек, реестр производит их поиск на основе данных конфигурационного файла и загружает их в память. Загрузив очередную плагинную библиотеку, реестр расширений создает для нее в памяти экземпляр структуры-описателя. Затем реестр импортирует из загруженной библиотеки функцию, инициализирующую адаптер набора расширений для структуры-описателя и, вызвав эту функцию, получает адаптер набора расширений. Теперь, получив запрос на перечисление сервисов, реестр расширений может переадресовывать его по очереди каждому адаптеру набора расширений. Выполнить запрос на перечисление сервисов можно практически в любой точке кода программы, тогда об этой точке можно говорить как о точке расширения плагинного каркаса.

С запросом на перечисление сервисов передается интерфейс посетителя сервисов, названного так по сходству с компонентом шаблона проектирования «Посетитель» [8]. Этот интерфейс не содержит специфических методов и служит только для получения других интерфейсов. Выполняя запрос, реализация набора расширений пытается получить через него интерфейс посетителя для того конкретного типа сервисов, который он способен предоставить. Через этот интерфейс, если он был получен, посетитель получает доступ к нужному сервису.

Способ управления временем существования объектов, предоставляющих сервисы, различается от интерфейса к интерфейсу. В плагинном каркасе INTEGRO используются три различных подхода. В простейшем случае гарантируется, что переданный методу адаптер интерфейса остается работоспособным до возврата из этого метода. Если этого недостаточно, используется передача владения объектом или совместное владение объектом. При передаче владения объектом у него существует только один владелец, отвечающий за освобождение объекта. При совместном владении таких владельцев может быть несколько. Чтобы интерфейс поддерживал один из этих двух подходов, он должен содержать методы для управления жизненным циклом объекта. После получения объекта владелец сообщает ему о принятии во владение, а по окончании использования объекта – освобождает его, вызвав соответствующий метод. Совместное владение – в связи со сравнительной сложностью его реализации и связанной с этим дополнительной вычислительной нагрузкой – применяется главным образом для крупных объектов, таких как корневые объекты сервисов. Это объекты, обращение к которым производится относительно нечасто и время жизни которых близко к времени работы приложения.

Основное приложение INTEGRO обращается к сервисам, которые позволяют расширить функциональность пользовательского интерфейса, дополнить набор классов объектной модели проекта, предоставить процедуры сохранения, загрузки и визуализации данных. Расширения INTEGRO могут определять собственные интерфейсы сервисов и включать в себя точки расширения.

Расширение пользовательского интерфейса

Сервисы расширений пользовательского интерфейса построены на приципах, сходных с много-

численными вариациями шаблона проектирования «Модель-Вид-Контроллер» (Model-Vieew-Controller, MVC [3]). Логика приложения традиционно для этого семейства шаблонов делится на три компонента: «Модель» отвечает за сохранение и обновление состояния системы и логику предметной области, «Вид» – за представление их пользователю, «Контроллер» обеспечивает интерпретацию пользовательского ввода и его преобразование для воздействия на «Модель» (рис. 1). В варианте с контроллером – посредником (Mediating controller MVC, также Model-View-Adapter, MVA [7]) «Контроллер» отвечает, кроме того, за получение данных и событий их обновления от «Модели» и за передачу их «Виду». В INTEGRO на «Контроллер» возложена еще и обязанность определять логику и структуру пользовательского интерфейса. Цель применяемого подхода – полностью отделить друг от друга и сделать независимыми реализации и интерфейсы «Модели» и «Вида», а также отсрочить решение вопроса о структуре пользовательского интерфейса до подключения плагинов.

Компоненту «Вид» названной выше триады в INTEGRO соответствуют интерфейсы, которые позволяют обновить состояние элементов управления. Интерфейсы компонента «Контроллер» служат для получения «Видом» отображаемых данных и для обработки поступающих от «Вида» событий пользовательского ввода. Они соответствуют элементам пользовательского интерфейса, таким как, например, формы, поля ввода, кнопки. Особый метод каждого из таких интерфейсов используется для внедрения зависимости «Контроллера» от «Вида» и одновременно для передачи «Виду» владения объектом «Контроллера». Объект «Контроллера» считается освобожденным и может быть удален, если отсутствует связанный с ним интерфейс «Вида». Интерфейсы «Модели» определяются при разработке плагинов и служат для получения и обновления данных «Контроллером». Как правило, они определяются в терминах предметной области или хранилища данных.

Структура пользовательского интерфейса определяется при помощи контейнеров элементов управления. Контейнер элементов управления – это (составной) элемент управления, в интерфейсе «Контроллера» которого присутствуют методы для перечисления вложенных элементов управления. Перечисление организовано по принципу шаблона проектирования «Посетитель» [8]: методу контейнера передается интерфейс «Посетителя», в котором для каждого поддерживаемого типа вложенного элемента имеется принимающий его метод. В контейнер элементов управления могут быть вложены другие контейнеры, образуя таким образом древовидную структуру.

Для расширения пользовательского интерфейса в плагине нужно реализовать сервис, который по запросу от основного приложения предоставит ему интерфейс контейнера элементов управления. Обращаясь к контейнеру, основное приложение обходит дерево вложенных в контейнер элементов управления и для каждого из них создает конкретную реализацию. Создав объект реализации, приложение передает его интерфейс соответствующему элементу, инжектируя таким образом зависимость от «Вида» в «Контроллер» и получая объект «Вида» во владение. При удалении объектов реализации инжектированная зависимость отменяется и объект «Вида» освобождается.

Когда объекты «Модели», «Вида» и «Контроллера» созданы и связаны друг с другом, «Вид» получает ввод от пользователя, определяет, какому действию над элементом пользовательского интерфейса этот ввод соответствует, и сообщает об этом действии «Контроллеру». «Контроллер» анализирует действие над элементом управления и, возможно, вносит соответствующие изменения в модель. При изменении модели «Контроллер» сообщает «Виду» о необходимости обновить состояние. При обновлении состояния «Вида» он запрашивает у «Контроллера» данные для отображения пользователю. «Контроллер» получает необходимые данные от «Модели» и передает их «Виду».

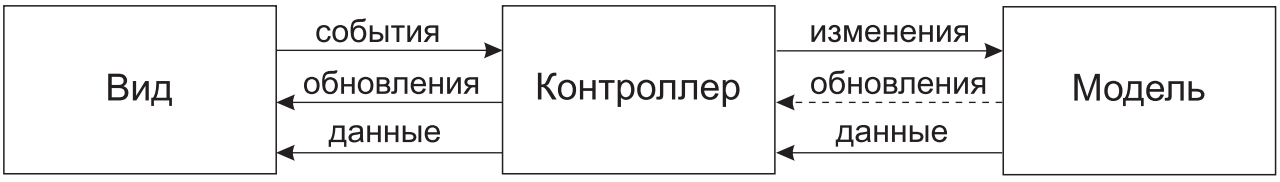


Рис. 1. Взаимодействие компонентов «Модель», «Вид» и «Контроллер» в INTEGRO

Fig. 1. Interaction of Model, View and Controller components in в INTEGRO

Пусть, например, плагину требуется отображать в панели инструментов масштаб карты и управлять им. В этом случае объект, хранящий значение масштаба в памяти, будет играть роль «Модели». Пусть этот объект поддерживает подписку на события изменения своих данных в соответствии с шаблоном проектирования «Наблюдатель» [8]. В этом случае в плагине должен быть создан объект (или их совокупность), поддерживающий интерфейс сервиса инструментов и интерфейс «Контроллера» числового поля ввода для панели инструментов. Этот объект должен содержать указатель на объект «Модели» и быть его «Наблюдателем». В объекте «Контроллера» должны быть реализованы методы для получения числа, для изменения числа и для связывания с «Видом» из интерфейса контроллера. Метод получения числа извлекает значение масштаба из объекта «Модели» и возвращает его (как будет сказано ниже, «Контроллер» возвращает это значение «Виду»). Метод изменения числа принимает число (от «Вида») и устанавливает его в качестве значения масштаба в объект «Модели». Метод связывания с «Видом» принимает адаптер интерфейса «Вида» и должен сохранить копию адаптера в памяти «Контроллера». При поступлении сообщения об изменении от «Модели» «Контроллер» должен через сохраненный адаптер «Вида» вызвать метод обновления «Вида». Основное приложение, создавая панель инструментов, обратится к загрузчику плагинов с запросом на обнаружение сервисов, передав тому «Посетитель» сервисов инструментов. Этот «Посетитель» будет передан плагину в виде обобщенного «Посетителя» сервисов. Обнаружив у полученного «Посетителя» интерфейс «Посетителя» сервиса инструментов, плагин должен передать этому «Посетителю» соответствующий интерфейс созданного в плагине объекта. Основное приложение, получив интерфейс к сервису, обратится к нему для перечисления предоставляемых инструментов, передав «Посетитель» инструментов. Объект плагина должен передать этому (новому) «Посетителю» интерфейс своего «Контроллера». Получив интерфейс «Контроллера», основное приложение создаст для панели инструментов объект, реализующий поле ввода чисел, и передаст «Контроллеру» интерфейс «Вида» для этого объекта через метод связывания. При получении ввода от пользователя объект поля ввода будет вызывать метод установки числа в интерфейсе «Контроллера», а тот, в свою очередь, устанавливать значение масштаба в «Модель». При получении вызовов метода обновления через интерфейс «Вида» компонент поля ввода будет обновлять отображаемое значение, обращаясь за ним к «Контроллеру», а тот – к «Модели». Когда приложение будет заканчивать работу и удалит панель инструментов вместе с полем ввода, объект поля ввода отсоединится от «Контроллера», передав ему «нулевой» интерфейс «Вида», и соответствующий объект в плагине можно будет удалить. При этом реализацию поля ввода в основном приложении можно изменять, не затрагивая код плагина, а реализацию «Модели» и «Контроллера» – не затрагивая код основного приложения. Изменения «Контроллера» также не затрагивают «Модель». Если «Модель» представляет собой абстракцию (интерфейс), то ее реализацию можно изменять, не изменяя код «Контроллера».

В основном приложении INTEGRO элементы управления реализуются на основе классов библиотеки визуальных компонентов (Visual Component Library, VCL) и производных от них классов сторонних библиотек (DevExpress, VirtualTrees), то есть, аналогично подходу библиотеки SWT (Standard Widget Toolkit, от Eclipse Foundation для Java [5]), используются нативные реализации элементов управления. Пользуясь отсутствием зависимости «Контроллера» от конкретной реализации «Вида», последнюю можно изменять вплоть до перевода основного приложения на другую библиотеку классов пользовательского интерфейса. Существует модификация основного приложения INTEGRO, основанная на кроссплатформенном фреймворке Qt [6]. Она используется утилитами INTEGRO.

Каркас INTEGRO предлагает набор интерфейсов для описания меню, панелей инструментов, стыкуемых панелей и диалогов, а также для расширения возможностей специфических для ГИС элементов управления, среди которых окна двумерного и трехмерного видов. В числе доступных элементов управления для диалогов и стыкуемых панелей – кнопка, поле ввода, выпадающий список, таблица, иерархический список и т. п. Их расположением на экране можно управлять с помощью визуальных контейнеров элементов управления: панелей групп, матричных панелей, панелей со вкладками. Для панелей инструментов и меню доступны кнопки (пункты меню), выпадающие меню, поля для ввода разнотипных значений.

Возможности работы с двумерными и трехмерными видами сцен (карт) INTEGRO плагин может расширить, предоставляя собственные режимы взаимодействия с окном сцены. Режим взаимодействия с окном сцены определяет стратегию (в смысле одноименного шаблона проектирования) реакции окна на ввод пользователя и визуализации этого взаимодействия. Пользователь взаимодействует с видом при помощи активных и пассивных элементов

изображения в окне вида: манипуляторов и набросков. Реакция манипулятора на события пользовательского ввода определяется окном сцены, и режиму передаются события воздействия пользователя на манипулятор. Набросок только отображается режимом в виде слоя графических объектов, наложенного на изображение сцены. В каждый момент времени пользователь может взаимодействовать только с одним, активным режимом. Плагин отвечает за активацию и индикацию активного состояния режимов самостоятельно. С этой целью он может использовать элементы панели инструментов, меню или стыкуемых панелей, которые он определяет. Интерфейсы режимов воплощают принципы шаблона «Модель-Вид-Контроллер».

Аналогично устроены интерфейсы расширений для панелей, предназначенных для редактирования структуры и свойств проекта. В реализациях этих панелей имеются точки расширения, которые подхватывают дополнительные стратегии из соответствующих сервисов плагинов, при необходимости добавляя в свой пользовательский интерфейс соответствующие элементы. Этим способом подключаются, например, функции создания слоев, редактирования их свойств.

Весь обмен текстовой информацией с расширениями ведется с использованием строк широких символов, которые соответствуют кодировке UTF-16 в операционной системе Windows и UTF-32 в Linux.

Расширение объектной модели проекта

С помощью реализуемых в плагинах сервисов можно дополнить модель проекта INTEGRO собственными разновидностями узлов древовидной структуры проекта, таких как двумерные и трехмерные слои и сцены, определить для них способы сохранения, визуализации, выполнения других процедур обработки, в том числе определяемых плагинами. Чтобы определить новую разновидность слоя или сцены, предназначенный для этого сервис плагина предоставляет основному приложению интерфейс фабрики объектов [8], реализующих стратегию представления данных в памяти для создаваемого узла. Фабрики, в свою очередь, создают экземпляры объектов-стратегий и передают их интерфейсы вызывающему коду. В вызывающем коде создается объект контекста, который занимает свое место в родительском узле проекта, и производится связывание (инжекция зависимости и передача владения) созданного объекта с его контекстом. Зависимость стратегии от контекста используется, в частности, для уведомления наблюдателей об изменении состояния узла. Также в контексте реализуется общая для узлов проекта функциональность: хранится идентификатор разновидности узла, идентификатор узла, признаки видимости, выбора узла, список дочерних элементов и другие данные – в зависимости от уровня узла в иерархической структуре проекта. Объекты стратегий освобождаются при удалении узлов проекта. Плагин может определять дополнительные интерфейсы для создаваемых им разновидностей узлов проекта.

Функции, ответственные за обработку узлов проекта, в том числе сохранение их состояния во внешнем формате, загрузку из внешнего формата и за отображение (рендеринг) их на двумерном или трехмерном виде, вынесены в отдельные интерфейсы и сервисы. Для каждого из видов обработки существует (или определяется плагином) отдельный сервис, предоставляющий стратегии для соответствующего вида обработки узлов. Так устраняется зависимость интерфейсов узлов от модели сохранения, рендеринга или иной обработки. Получив запрос на обработку узла, объект, реализующий стратегию, самостоятельно определяет его разновидность по идентификатору разновидности узла и (или) по наличию у него дополнительных интерфейсов и обрабатывает узлы поддерживаемых разновидностей.

Инструментарий разработчика

Плагинный каркас INTEGRO определяет интерфейсы между объектами, но не предоставляет их реализаций, кроме умолчательной (рис. 2). Для удобства разработки кода основного приложения и плагинов, а также для обеспечения повторного использования кода в дополнение к плагинному каркасу разработаны библиотеки классов и функций, составляющие инструментарий разработчика расширений INTEGRO. Сюда входят:

- библиотеки, обеспечивающие совместимость кода с различными платформами;
- объектно-ориентированные обертки и вспомогательные классы для функций программного интерфейса основного приложения;
- реализации, базовые классы и шаблоны реализации плагинных интерфейсов.

В части случаев на выбор разработчика предлагается более одного подхода к реализации интерфейсов. Например, для многих типов элементов пользовательского интерфейса имеются базовые классы, автоматически создающие компонент модели по умолчанию, и шаблоны, поддерживающие инжекцию зависимости от модели. Кроме того, разработчик может с целью повышения вычислительной эффективности кода самостоятельно реализовать интерфейсы с нуля или на основе сгенерированных заготовок реализаций.

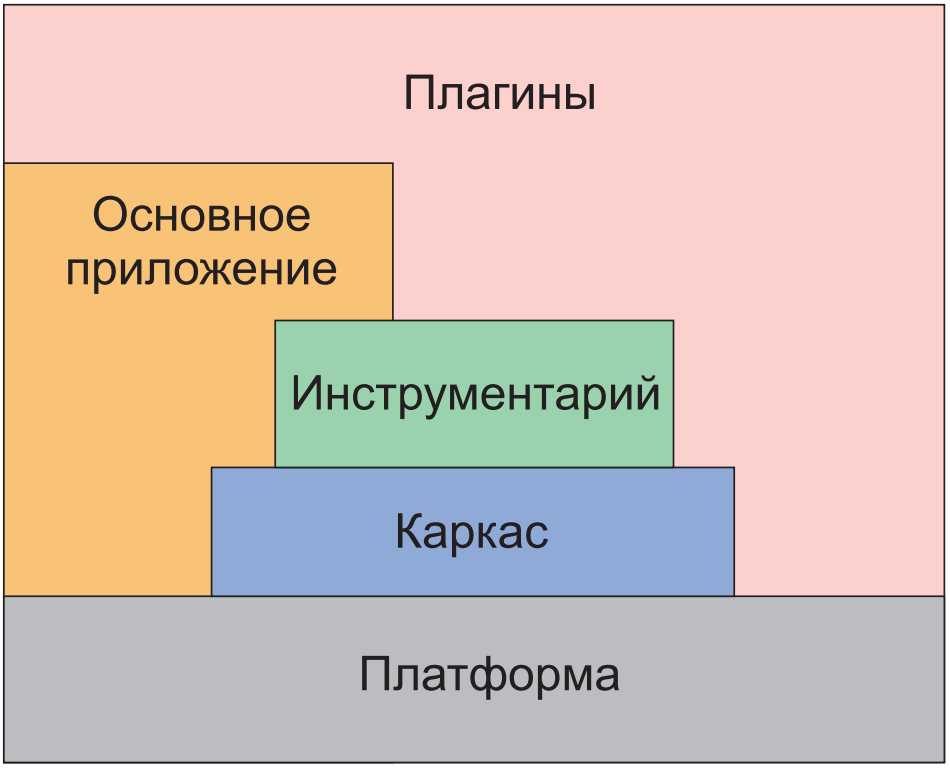


Рис. 2. Упрощенная схема плагинной архитектуры INTEGRO

Fig. 2. INTEGRO plugin architecture (simplified)

Заключение

Практика внедрения обновленной плагинной архитектуры в INTEGRO показывает, что поставленные перед ней цели были достигнуты. Точки расширения добавлялись в основное приложение поэтапно, одновременно с развитием системы плагинных интерфейсов. Эта работа не вызвала перерыва в поддержке системы и развитии других ее функций. Плагины устаревшей архитектуры поочередно переводились на новую, таким образом, прежняя плагинная технология была вытеснена из использования без ущерба для пользовательских качеств системы. Были разработаны плагины INTEGRO, предоставляющие развитые блоки разнообразной и взаимоувязанной функциональности. В частности в виде плагина реализован блок для работы с данными скважин, который создавала и который продолжает сопровождать отдельная группа разработчиков. Разработка многих плагинов ведется с использованием Microsoft Visual Studio. Создана реализация основного приложения на основе фреймворка Qt, заменившего VCL, при сборке которой стало возможным отказаться от применения инструментов Embarcadero. Ведутся небезуспешные эксперименты по переносу этой реализации на платформу Linux.

Таким образом, в INTEGRO внедрена плагинная архитектура, которая:

- переносима, то есть позволяет использовать один исходный код на разных платформах;
- не привязана к поставщику, то есть позволяет использовать инструменты разных разработчиков;
- «скромна», то есть не требует, чтобы приложение было основано на ней, а может встраиваться в имеющееся приложение;
- неинвазивна, то есть может быть применена к объектам без изменения их кода: свойства используемого интерфейсного адаптера таковы, что он может быть создан для любого объекта, имеющего адрес в памяти;
- объектно-ориентирована, то есть поддерживает принципы абстракции, инкапсуляции и полиморфизма, и использует соответствующие возможности языка программирования;
- эффективна, поскольку вводит минимум накладных расходов, связанных с вызовом методов, и не навязывает затратные модели управления временем жизни объектов;
- масштабируема в использовании, так как позволяет подключать новые модули, создавать

в них точки расширения и к ним также подключать новые модули;
- масштабируема в разработке, то есть позволяет дополнительно привлекать разработчиков как для создания плагинов, так и для разработки интерфейсов;
- сопровождаема: есть возможность независимо модифицировать основное приложение, плагины и двоичный интерфейс между ними.

Использование в интерфейсах строк широких символов позволяет реализовать поддержку кодировки UNICODE. Стало легче поддерживать единообразие во внешнем виде и поведении элементов управления, поскольку они определяются теперь главным образом основным приложением. Разработчик плагинов при этом избавлен от непосредственной работы со сторонними библиотеками визуальных компонентов.

Развитие плагинной архитектуры INTEGRO продолжается: разрабатываются новые интерфейсы, совершенствуется код основного приложения и плагинов, расширяется инструментарий разработчика, осваиваются альтернативные платформы.

ГИС INTEGRO готовится удовлетворить любые специфические потребности своих пользователей с помощью специально разработанных расширений.

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

ЛИТЕРАТУРА

1. Sutter H. Defining a Portable C++ ABI. – 2014. – URL: https://isocpp.org/files/papers/n4028.pdf (дата обращения: 31.07.2018).
2. Schmidt D.C. Model-Driven Engineering // IEEE Computer. – 2006. – V. 39, No. 2 (February). – P. 25-31. – URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.9720&rep=rep1&type=pdf (дата обращения: 31.07.2018).
3. Fowler M. GUI Architectures. – 2006. – URL: https://martinfowler.com/eaaDev/uiArchs.html (дата обращения: 31.07.2018).
4. Zelzer S. OSGi and C++. – 2012. – URL: http://blog.cppmicroservices.org/2012/03/29/osgi-and-c++/ (дата обращения: 31.07.2018).
5. The SWT FAQ // Eclipse devs. : The Platform for Open Innovation and Collaboration. – URL: http://www.eclipse.org/swt/faq.php (дата обращения: 31.07.2018).
6. QT developers. About Qt. – 2018. – URL: https://wiki.qt.io/About_Qt (дата обращения: 31.07.2018).
7. Eckstein R. Java SE Application Design with MVC. – 2007. – URL: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html (дата обращения: 31.07.2018).
8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software, 1994. – .417 p. – (Addison-Wesley Professional Computing Serias). – ISBN 0-201-63361-2.

REFERENCES

1. Sutter H. Defining a Portable C++ ABI. 2014. URL: https://isocpp.org/files/papers/n4028.pdf (date of access: 31.07.2018).
2. Schmidt D.C. Model Driven Engineering // IEEE Computer. 2006. V. 39, No. 2 (February). P. 25-31. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.9720&rep=rep1&type=pdf (date of access: 31.07.2018).
3. Fowler M. GUI Architectures. 2006. URL: https://martinfowler.com/eaaDev/uiArchs.html (date of access: 31.07.2018).
4. Zelzer S. OSGi and C++. 2012. URL: http://blog.cppmicroservices.org/2012/03/29/osgi-and-c++/ (date of access: 31.07.2018).
5. The SWT FAQ // Eclipse devs. : The Platform for Open Innovation and Collaboration. – URL: http://www.eclipse.org/swt/faq.php (date of access: 31.07.2018).
6. QT developers. About Qt. 2018. URL: https://wiki.qt.io/About_Qt (date of access: 31.07.2018).
7. Eckstein R. Java SE Application Design with MVC. – 2007. – URL: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html (date of access: 31.07.2018).
8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. 1994. 417 p. (Addison-Wesley Professional Computing Serias). ISBN 0-201-63361-2.