УДК 629.7.08

Применение шаблонов проектирования в программном обеспечении системы планирования полетных заданий

Куликов А.М.

Компания "Раменское приборостроительное конструкторское бюро",

ул. Гурьева,2, Раменское, Московская область, 140103, Россия

e-mail: KulikovAM@mail.ru

Аннотация

В статье рассматривается модульная архитектура программного обеспечения системы планирования полётных заданий, в которой применена концепция MVC (модель-представление-контроллер) и другие шаблоны проектирования.

Ключевые слова: система планирования полётных заданий, архитектура программного обеспечения, шаблоны проектирования, паттерны, модель-представление-контроллер.

Введение

Системы планирования полётных заданий [1] различных летательных аппаратов имеют много общего, решают одинаковые прикладные задачи. Как правило, техническим заданием на любую систему планирования полётных заданий требуется отображение электронной карты местности (ЭКМ), отображение аэронавигационной информации (АНИ), наложение на ЭКМ условных знаков навигационных объектов, линии заданного пути (ЛЗП) и другой информации. Во всех системах планирования необходима возможность ввода данных об аэродромах, радиомаяках, маршруте, режиме и профиле полёта. Отличие систем планирования определяется тактико-техническими характеристиками объектов применения (видом

летательного аппарата), составом и характеристиками бортового оборудования, требованиями руководства по лётной эксплуатации, а также другими факторами и документами, регламентирующими действия пользователей системы. Полётные данные, подготовленные системой планирования, записываются на электронный носитель информации (см. рисунок 1) для последующей загрузки в бортовое радиоэлектронное оборудование (БРЭО) и используются в полёте при выполнении полётного задания (ПЗ).

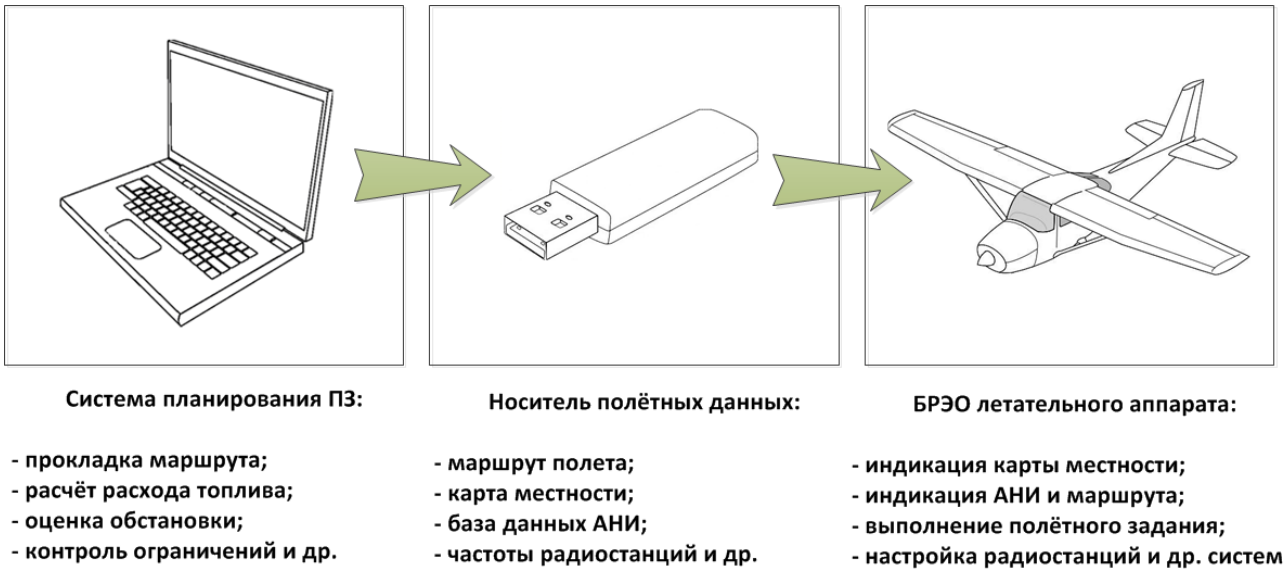

Рис. 1. Принципиальная схема подготовки полётных данных.

В разработанном ранее программном обеспечении (ПО) исходный код основных диалоговых окон системы компилировался в составе главного исполняемого модуля. Ряд программных подсистем был представлен монолитными программными библиотеками, в которых были реализованы как программные классы моделей предметной области, так и соответствующий пользовательский интерфейс. При этом наиболее востребованные данные были объединены в повсеместно используемую (глобальную) структуру. Внесение изменений в ПО приводило к необходимости перекомпиляции значительной его части. Отладка

внесённых изменений была затруднена из-за переплетения алгоритмов процедур чтения данных, расчётных функций и функций, реализующих диалог с пользователем. Адаптация такого ПО под другие виды летательных аппаратов и под требования другого заказчика занимала значительное время, было затруднено исключение из состава ПО невостребованной функциональности. Обилие глобальных переменных вкупе со статическим подключением программных библиотек требовало повышенного объема оперативной памяти, что затрудняло использование ПО на других программно-аппаратных платформах. Перспективное ПО должно было быть лишено указанных недостатков и стать легко адаптируемой основой для дальнейших разработок.

Предложенная архитектура

В программном обеспечении любой системы планирования полётных заданий можно выделить следующие подсистемы: картографическая подсистема, подсистема ввода навигационной обстановки, подсистема планирования полёта, подсистема подготовки полётных данных. Предложено все программные подсистемы строить по принципу (концепции) "модель-представление-контроллер" ("Model-View-Controller") [2], зарекомендовавшему себя как при разработке интернет-приложений, так и при разработке настольного ПО. Данная концепция предполагает архитектурное (модульное) разделение моделей предметной области (бизнес-логика), представлений моделей в диалоговых окнах (пользовательский интерфейс) и контроллеров, транслирующих пользовательский ввод с клавиатуры и от манипулятора типа "мышь" в события диалоговых элементов управления (кнопок, полей ввода, списков, полос прокрутки и т.п.).

Суть прикладных задач, расчётные функции, чтение, обработка и запись данных реализуются в программных модулях бизнес-логики. Вид диалоговых окон, логика отображения состояния моделей (данных) и смысловая обработка действий пользователя определяются в программных модулях пользовательского интерфейса. Функция контроллеров выполняется в современных операционных системах (ОС) на уровне самой ОС, поэтому реализация таких специальных программных модулей в прикладном настольном ПО не требуется.

Реализация бизнес-логики программно не зависит от представлений моделей предметной области в диалоговых окнах. Действия пользователя интерпретируются алгоритмами модулей пользовательского интерфейса в операции над моделями (создание моделей, вызов методов моделей и удаление моделей). Между моделями, как правило, есть смысловые связи. Операция над некоторой моделью приводит к ряду операций над другими моделями, которые также имеют свои представления в пользовательском интерфейсе. Оповещение представлений о событиях (изменениях), происходящих внутри моделей при функционировании системы, осуществляется по шаблону "Наблюдатель" ("Observer") [3]. Каждый "наблюдатель" структурно относится к соответствующей модели и позволяет динамически назначить "обработчик события" (функцию, которая должна быть вызвана при возникновении определённого события).

Связь подсистем предложено организовать по иерархическому принципу таким образом, чтобы подсистемы более низкого уровня не зависели от подсистем более высокого уровня. При этом в модуле бизнес-логики некоторой подсистемы допускается использование только модулей бизнес-логики подсистем более низкого

уровня, а обращений к модулям пользовательского интерфейса быть не должно. В модулях пользовательского интерфейса используются модули бизнес-логики и модули пользовательского интерфейса подсистем более низкого уровня.

На рисунке 2 для наглядности изображены только основные связи между модулями в предложенной архитектуре. Картографическая подсистема не зависит ни от одной из других подсистем (является подсистемой первого уровня). Модель базы данных (БД) цифровой картографической информации (ЦКИ) и модель ЭКМ реализуются в модуле бизнес-логики картографической подсистемы. Данные модели используются соответственно редактором БД ЦКИ и менеджером ЭКМ (пользовательский интерфейс). Модуль бизнес-логики картографической подсистемы используется в модулях подсистемы ввода навигационной обстановки (второй уровень) и во всех других подсистемах более высокого уровня.

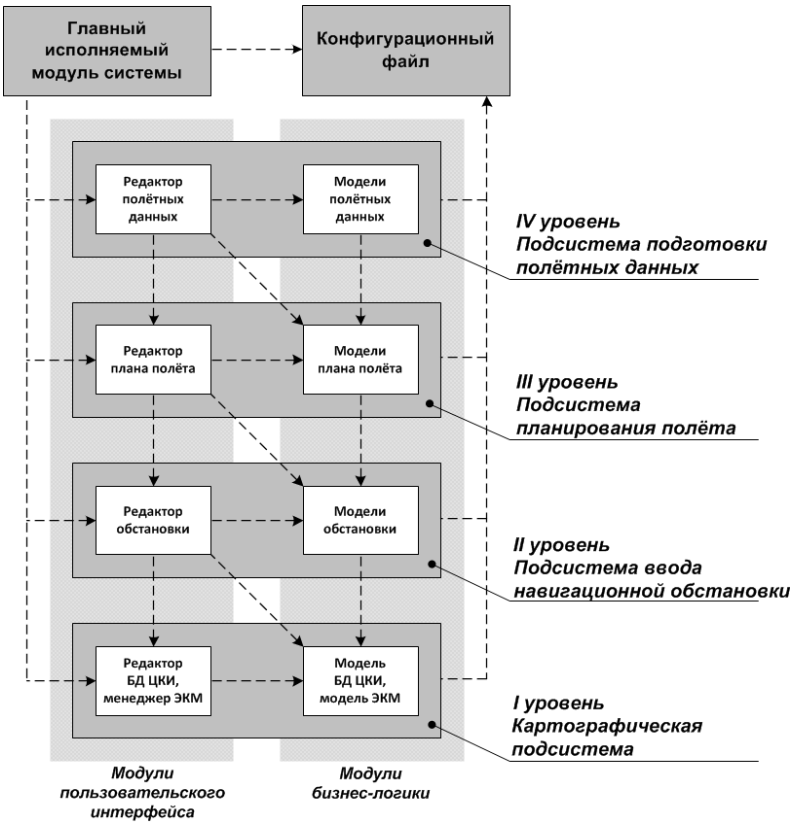

Рис. 2. Схема предложенной архитектуры.

Подсистема планирования полёта является подсистемой третьего уровня. В моделях плана полёта используются модели объектов навигационной обстановки и модуль бизнес-логики картографической подсистемы, а в редакторе плана полёта – ещё и соответствующие модули пользовательского интерфейса данных подсистем. Например, при открытии плана выполняется автоматический переход в соответствующий район, а при назначении точки (пункта) маршрута вызывается диалог выбора навигационного объекта. Подсистема подготовки полётных данных (четвертый уровень) использует модули всех остальных подсистем.

В зависимости от требований конкретного технического задания на некотором уровне может быть более одной программной подсистемы, а также могут быть подсистемы более высоких уровней. Например, подсистема послеполётного анализа является подсистемой пятого уровня, так как для автоматизированной оценки действий экипажа ей необходимы модели использованных полётных данных, а также функциональные возможности всех остальных подсистем.

Главный исполняемый модуль системы формирует главное меню, панель инструментов, осуществляет интеграцию программных подсистем в единый пользовательский интерфейс. На данном уровне в зависимости от выбранного пользователем режима работы принимается решение о задействовании подсистем, выполняется информационное связывание подсистем. При этом используются функции только модулей пользовательского интерфейса.

Разделение реализации бизнес-логики и пользовательского интерфейса позволяет выполнять тестирование модулей бизнес-логики независимо от модулей пользовательского интерфейса. Кроме того, при необходимости видоизменения

диалоговых окон по требованию нового заказчика могут разрабатываться новые модули пользовательского интерфейса, использующие имеющиеся модули бизнес-логики. И наоборот, при изменении алгоритмов прикладных задач или существенном изменении организации обработки данных могут быть заменены модули бизнес-логики с сохранением (или минимальной доработкой) имеющихся модулей пользовательского интерфейса. Но в целях постоянного совершенствования имеющихся наработок предпочтение должно отдаваться введению конфигурационных параметров в разработанные модули. Например, если в очередной заказанной системе не требуется реализованная ранее возможность трёхмерного представления наземной обстановки, то соответствующие элементы редактора обстановки должны скрываться установкой логического признака в конфигурационном файле, а не удаляться из файлов исходных кодов.

Программная реализация

Предложенная архитектура была реализована в одной из систем планирования полётных заданий производства ОАО "РПКБ". Программирование осуществлялось на языке C++ с использованием стандартной библиотеки STL (как правило, в модулях бизнес-логики) и библиотеки визуальных компонентов VCL (в модулях пользовательского интерфейса). Выбор среды "Borland C++ Builder 6.0" был обусловлен большим опытом её использования командой разработчиков. Усилия команды были сосредоточены на реализации выбранной архитектуры, но не на освоении нового инструментария, что дало определенный выигрыш в скорости выполнения работ. Кроме того, использовался ранее разработанный программный код путём вычленения из него расчётных функций и другой бизнес-логики.

Бизнес-логика картографической подсистемы представлена в разработанном ПО следующими основными программными классами: TTerrainRegion ("Район картообеспечения"), TTerrainSheet ("Лист карты местности"), TTerrainLayers ("Картографические слои"), TTerrainObject ("Картографический объект"). Экземпляры данных классов используются в классе TTerrainMapCore ("Ядро ЭКМ") при формировании изображения ЭКМ конкретного района местности. Но во всех подсистемах используется не класс TTerrainMapCore, а класс TTerrainMap, который повторяет public-часть объявления класса TTerrainMapCore, скрывая его private-часть (шаблон "Мост" ("Bridge") [3]). Исходный код перечисленных выше классов компилируется в динамически подключаемую библиотеку Terrain.dll.

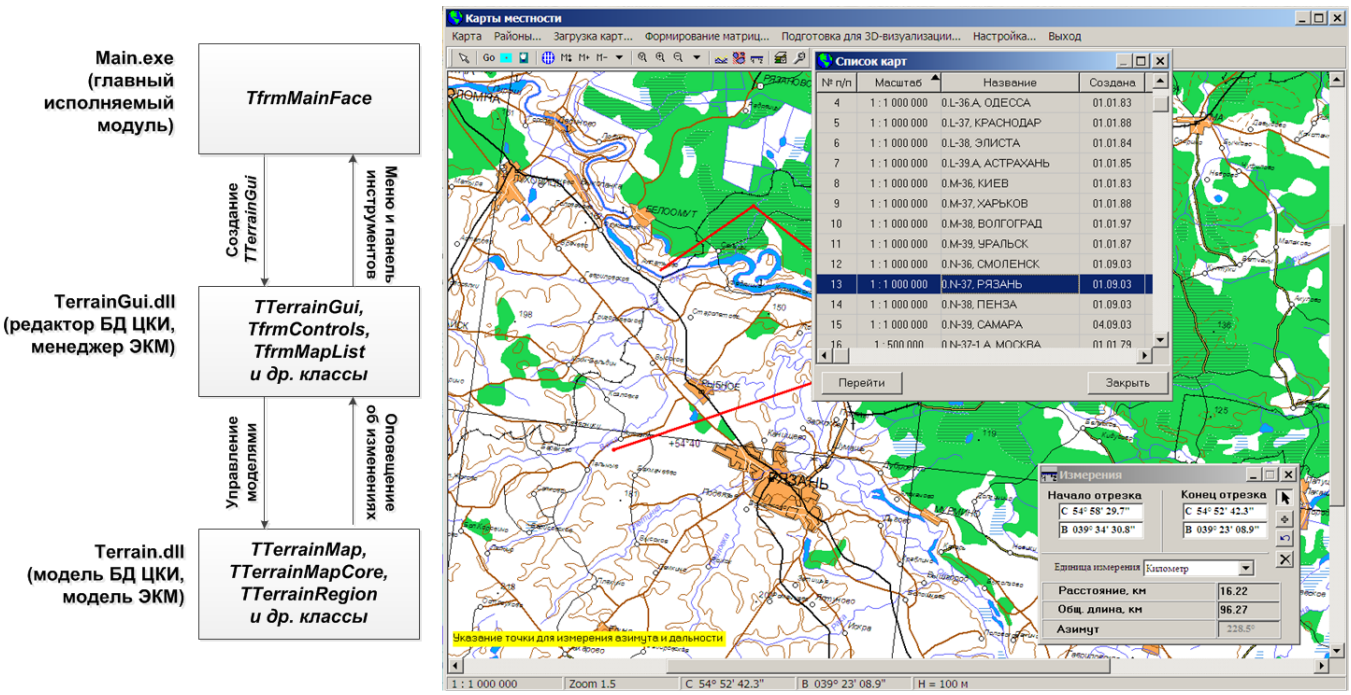

Рис. 3. Картографическая подсистема.

Диалоговые окна управления БД ЦКИ и менеджер ЭКМ реализованы в библиотеке TerrainGui.dll (Gui – Graphic user interface). Данная библиотека экспортирует (предоставляет для использования в других модулях) класс

TTerrainGui, который обеспечивает функционирование пользовательского интерфейса картографической подсистемы. Главный исполняемый модуль системы Main.exe создает экземпляр класса TTerrainGui и получает от него готовое меню и панель инструментов редактора БД ЦКИ и менеджера ЭКМ. Обработка выбора пунктов меню и нажатия кнопок панели инструментов осуществляется внутри TerrainGui.dll. При этом TerrainGui.dll выполняет необходимые операции с моделью БД ЦКИ и моделью ЭКМ, которые в свою очередь оповещают TerrainGui.dll о возникающих внутренних событиях. Внесение в картографическую подсистему новых пунктов меню и изменение её диалоговых окон не требуют перекомпиляции программного кода Main.exe.

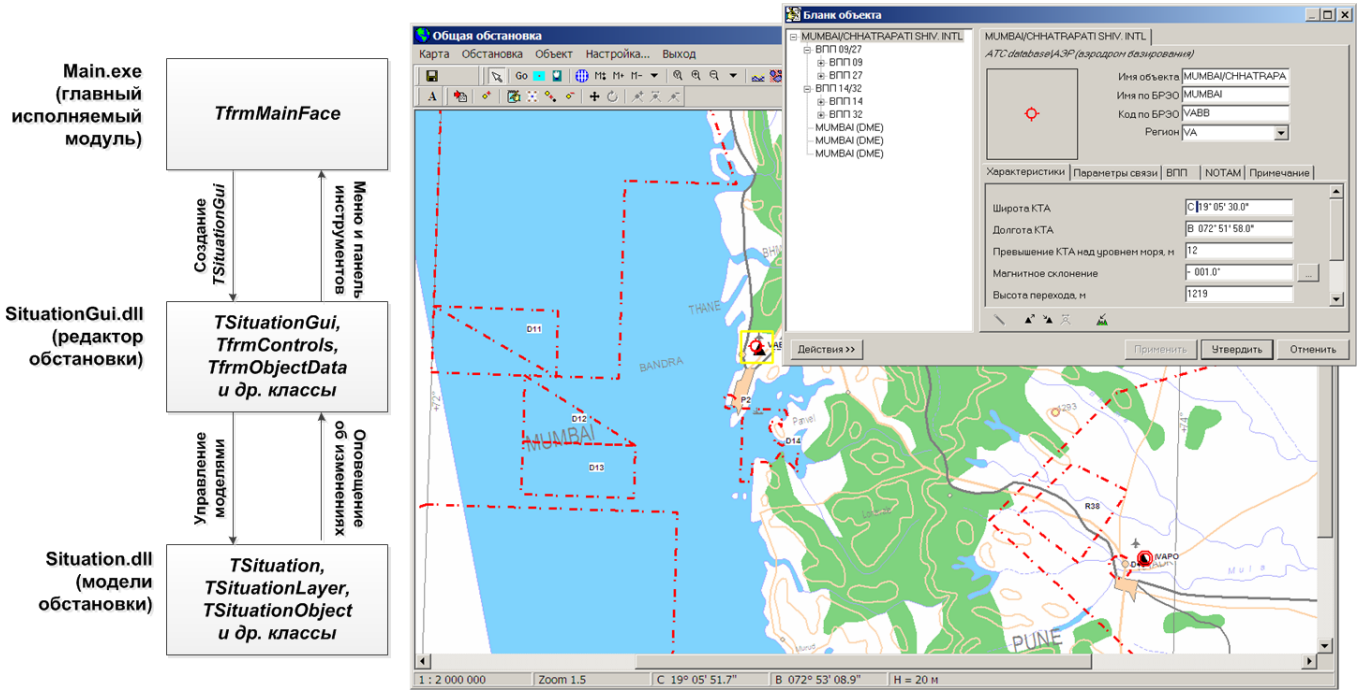

Рис. 4. Подсистема ввода навигационной обстановки.

Бизнес-логика подсистемы ввода навигационной обстановки (НО) реализована в библиотеке Situation.dll следующими основными программными классами: TSituationLayer ("Слой обстановки"), TSituationObject ("Объект обстановки"),

TSituationDrawParam ("Параметры отображения обстановки"), TSituationFilter ("Фильтр объектов обстановки"). Экземпляры данных классов инкапсулированы в экспортируемый класс TSituation ("Обстановка") и используются для работы с информацией НО и при формировании изображения НО на ЭКМ. При этом класс TSituation обеспечивает возможность одновременной работы с несколькими слоями обстановки (АНИ, пользовательская информация и т.п.).

Диалоговые окна ввода данных объектов обстановки, загрузки АНИ, просмотра и редактирования БД обстановки реализованы в библиотеке SituationGui.dll. Данная библиотека экспортирует класс TSituationGui, который обеспечивает обработку выбора соответствующих пунктов меню, нажатие кнопок панели инструментов и организацию интерактивной работы с НО на ЭКМ. Главный исполняемый модуль системы Main.exe создает экземпляр TSituationGui и передает ему указатель на экземпляр класса TTerrainMap. Загрузка SituationGui.dll в оперативную память осуществляется только при выборе пользователем такого режима работы системы, при котором требуется её использование. Возможность такой динамической загрузки библиотеки обеспечивается за счет экспортирования специальных функций создания и удаления экземпляра TSituationGui вместо экспортирования самого класса TSituationGui.

Бизнес-логика подсистемы планирования полёта реализована в библиотеке Plan.dll следующими основными программными классами: TAircraft ("Летательный аппарат"), TAircraftGroup ("Группа ЛА"), TRoute ("Маршрут"), TWaypoint ("Точка/пункт маршрута"), TRouteLeg ("Этап маршрута"), TFlightSegment ("Участок полёта"), TTrajectory ("Траектория"), TPlanDrawParam ("Параметры отображения

плана"). Экземпляры данных классов инкапсулированы в экспортируемый класс TPlan ("План") и используются для работы с информацией плана полёта, при отображении ЛЗП на ЭКМ, при формировании изображения профиля полёта.

Диалог ввода маршрута, диалог редактирования профиля полёта и другие диалоги планирования полёта реализованы в библиотеке PlanGui.dll. Взаимодействие главного исполняемого модуля системы Main.exe с библиотекой PlanGui.dll организовано аналогично его взаимодействию с SituationGui.dll. Главный модуль выполняет загрузку PlanGui.dll в оперативную память только при выборе пользователем такого режима работы системы, при котором требуется использование функций редактора плана полёта.

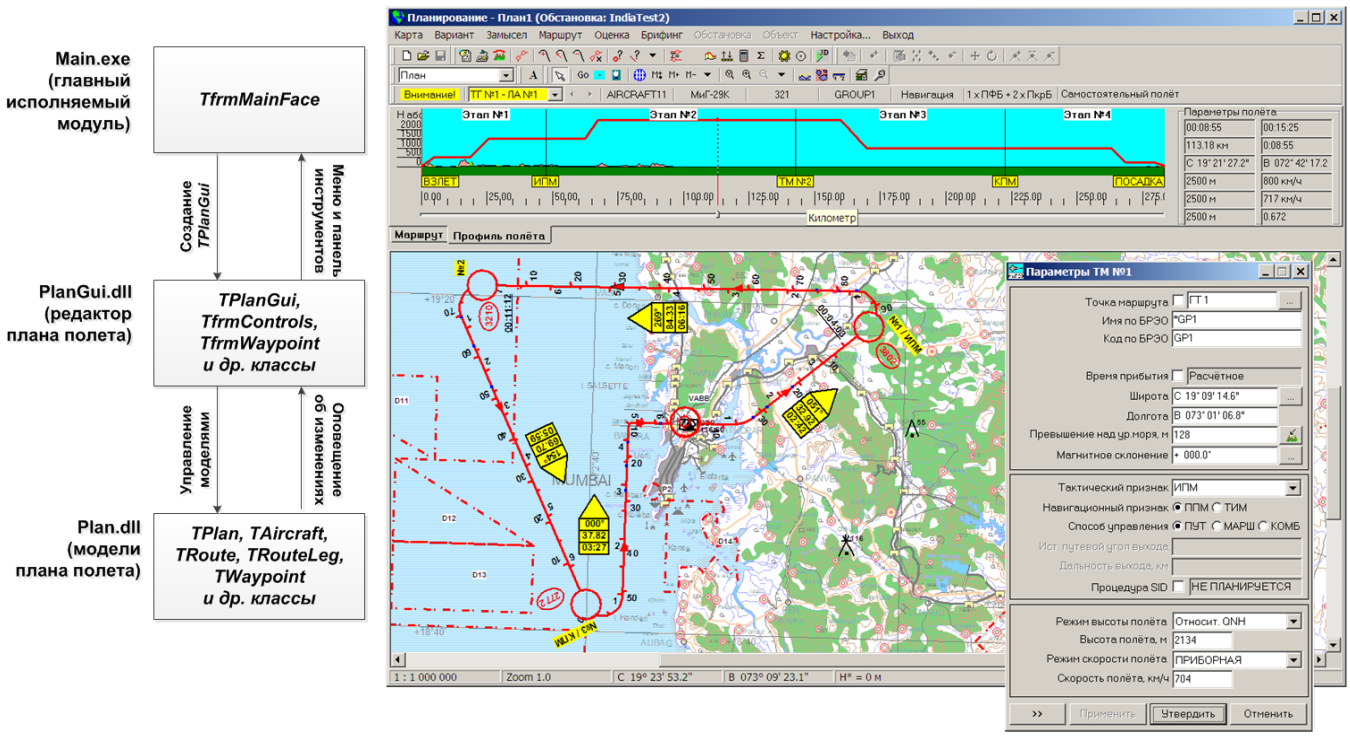

Рис. 5. Подсистема планирования полёта.

Создание экземпляра класса TPlanGui осуществляется вызовом специальной функции PlanGui.dll вместо прямого вызова конструктора TPlanGui. Инициализация, получение меню подсистемы и удаление экземпляра TPlanGui также осуществляется

с помощью специальных функций PlanGui.dll, а не с помощью методов TPlanGui. При инициализации экземпляру TPlanGui передаются указатели на экземпляры TTerrainMap и TSituation (классы бизнес-логики подсистем более низкого уровня). Такое динамическое подключение подсистемы позволяет минимизировать связанность главного модуля с подсистемами, упрощает тестирование системы, способствует оптимальному использованию оперативной памяти. По данному шаблону организовано взаимодействие Main.exe с подсистемой подготовки полётных данных (библиотека FlightDataGui.dll) и подсистемой послеполётного анализа (библиотека PostFlightGui.dll).

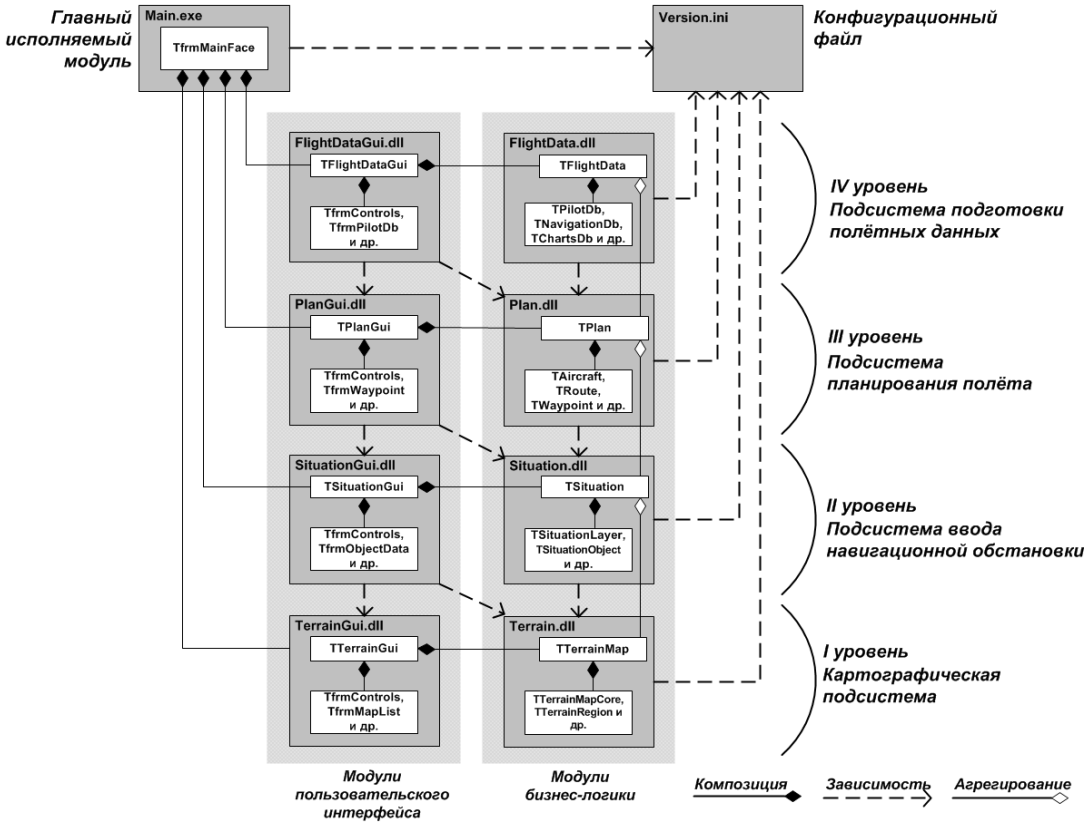

Рис. 6. Диаграмма основных программных классов.

Каждый архитектурный модуль реализован в ПО в виде программных классов одной или нескольких библиотек. На рисунке 6 для наглядности изображены программные классы только основных программных библиотек и связи между

ними. Экземпляры основных классов бизнес-логики (TTerainMap, TSituation, TPlan, TFlightData) инкапсулированы в соответствующие классы пользовательского интерфейса (TTerrainMap, TSituationGui, TPlanGui, TFlightDataGui), которые в свою очередь инкапсулированы в класс главного диалогового окна системы (TfrmMainFace). Данная инкапсуляция имеет характер "композиции" – связь типа "часть-целое", при которой экземпляры "частей" создаются вместе с экземпляром "целого" и уничтожаются вместе с ним. Информационное связывание бизнес-логики подсистем осуществляется путём "агрегирования" экземпляров основных классов бизнес-логики. Главный исполняемый модуль передаёт указатели экземпляров основных классов бизнес-логики (например, TTerrainMap) в подсистемы более высокого уровня. Тем не менее объектно-ориентированный подход позволяет создавать дополнительные экземпляры классов. Например, экземпляры TTerrainMap создаются для отображения ЭКМ в дополнительных диалоговых окнах системы, а не только в её главном окне.

Конфигурационные параметры вводились в программном коде, исходя из предыдущего опыта разработки систем планирования ПЗ. Прежде всего в части доступности тех или иных функциональных возможностей: расчёт расхода топлива, переключение системы измерений, трёхмерная визуализация, послеполётный анализ и т.п. Поэтому конфигурационный файл Version.ini используется как в модулях бизнес-логики, так и в моделях пользовательского интерфейса, а также в главном исполняемом модуле системы. Значения конфигурационных параметров определяются на этапе разработки и не подлежат редактированию в процессе эксплуатации ПО (выполняется проверка контрольной суммы файла Version.ini).

Заключение

В индустрии разработки программного обеспечения накоплен колоссальный опыт проектирования. Выпущено множество книг о теории и практике построения архитектуры программ. Знание шаблонов проектирования позволяет сэкономить время и улучшает взаимопонимание в команде разработчиков. Применяя шаблоны, мы не тратим время на создание оригинальных схем разбиения программного обеспечения на модули и классы, на организацию их взаимодействия. Но, с другой стороны, было бы ошибочным желание применить некоторый шаблон, не смотря на отсутствие такой практической целесообразности.

Предложенная архитектура ПО системы планирования ПЗ сама по себе является некоторым специализированным шаблоном. Теперь при разработке очередной системы имеется возможность не тратить время на "изобретение велосипеда", а сосредоточиться на реализации новых требований и прикладных задач, совершенствовать алгоритмы и улучшать пользовательский интерфейс. Программное разделение бизнес-логики и пользовательского интерфейса позволяет эффективнее использовать способности разработчиков: одни специализируются на разработке моделей предметной области, а другие – на реализации диалоговых окон. Кроме того, заложенные в архитектуре возможности позволяют перевести разработанное ПО на кроссплатформенные технологии. Для этого в программных модулях необходимо перейти на использование библиотеки Qt вместо VCL и прямого вызова функций WinAPI. А, благодаря повсеместному использованию класса TTerrainMap, смена технологии формирования изображения ЭКМ в основном потребует переработки только библиотеки Terrain.dll.

Разработанное ПО прошло межведомственные испытания, продемонстрировав стабильную работу и достаточное быстродействие. В настоящее время ПО передано в эксплуатацию и использовано при разработке нескольких других систем. Таким образом, получен программный каркас (платформа), при дальнейшем использовании и развитии которого могут быть значительно снижены сроки и себестоимость выпуска новых изделий, увеличена прибыль предприятия.

Библиографический список

1. Системы планирования и подготовки полетных заданий // Сайт ОАО "РПКБ" URL: http://www.rpkb.ru/lines-of-business/system-of-planning-and-preparation-of-flight-tasks/ (дата обращения: 23.09.2014).

2. Бурбек С. Программирование приложений в SmallTalk-80: Как использовать Model-View-Controller / Adobe Acrobat Reader, URL: http://www.math.rsu.ru/smalltalk/gui/mvc-rus.pdf (дата обращения: 23.09.2014).

3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования — СПб.: Питер, 2007. - 366 с.