**ГЛАВА 3 ПРОГРАММОЕ ОБЕСПЕЧЕНИЕ**

**3.1 Этапы разработки программного обеспечения**

От правильности выбора системного и прикладного программного обеспечения для разработки систем мониторинга зависит время, трудоемкость, эффективность и стоимость целевой системы.

Согласно современным требованиям [5,8] построение компьютерных систем включает, как минимум, четыре основных этапа:

* выбор операционной системы, согласно требованиям проекта;
* выбор программной платформы и языка, соответствующего целям задачи;
* разработка архитектуры ПО (компонент и связей между ними);
* реализация архитектуры (выбор программной платформы, программная реализация компонент, тестирование, описание этапов применения технологии);
* внедрение (использование ПО для решения прикладных задач).

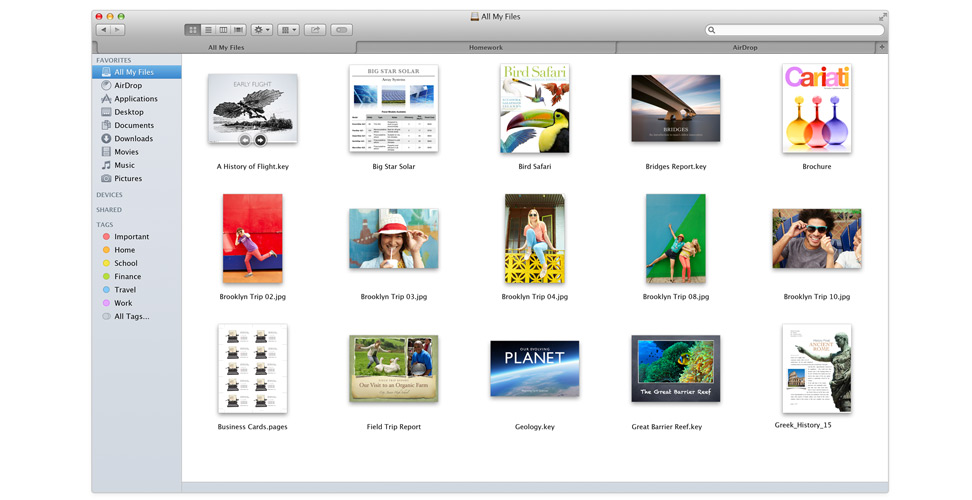
**3.2 Выбор программной платформы**

Правильный выбор программной платформы, адекватной свойствам и целям решаемой задачи, является важнейшей проблемой разработки компьютерных систем, т.к. от него зависит успех всего проекта. Неправильный выбор приводит к удорожанию проекта и усложнению поддержки жизненного цикла целевой системы. От выбора платформы так же зависит удобство и время отладки кодов программ, возможность использования ранее разработанных фрагментов программ, качество сервисной поддержки, трудоемкость документирования проекта и т.д. Поэтому вопрос выбора платформы для реализации разработанных в процессе дипломного проектирования моделей и алгоритмов требует отдельного рассмотрения.

В данной работе в качестве операционной системы была выбрана Mac OS X, которая значительно отличается от предыдущих версий Mac OS. Для реализации программной части использовался язык Objective-C в связке с фрэймворком Cocoa. Разработка велась в среде Xcode 5.1.1 [8]

**3.3 Обзор Mac OS X**

Основа системы — POSIX-совместимая операционная система Darwin, являющаяся свободным программным обеспечением. Её ядром является XNU (рекурсивный акроним от «X is Not Unix» — «X не Unix»), в котором используется ядро Mach и стандартные сервисы BSD. Все возможности Unix в OS X доступны через консоль.



**Рисунок 3.1 – Окно OS X**

Поверх этой основы в Apple разработано много проприетарных компонентов, таких как API Cocoa и Carbon, Quartz.

В OS X используется (как впрочем в любой UNIX-системе) вытесняющая многозадачность и защита памяти, позволяющие запускать несколько процессов, которые не могут прервать или повредить друг друга. На архитектуру OS X повлияла OpenStep, которая была задумана как портируемая операционная система. К примеру, NeXTSTEP была портирована с оригинальной платформы 68k компьютера NeXT, до того как NeXTSTEP была куплена Apple. Так и OpenStep была портирована на PowerPC в рамках проекта Rhapsody.

Основами OS X являются:

* подсистема с открытым кодом — Darwin (ядро Mach и набор утилит BSD).
* среда программирования Core Foundation (Carbon API, Cocoa API и Java API).
* графическая среда Aqua (QuickTime, Quartz Extreme и OpenGL).
* технологии CoreImage, CoreAudio и CoreData.

OS X также включает среду разработки программного обеспечения Xcode, которая позволяет разрабатывать программы на нескольких языках, включая Си, C++, Objective-C, Ruby, Java, Object Pascal. Она поддерживает компиляцию в так называемые «универсальные программы» (Universal Binary), которые могут запускаться на нескольких платформах (x86, PowerPC), так же, как «fat binaries» использовались для запуска одного приложения на 68k и PowerPC платформах.

**3.4 Обзор среды разработки Xcode**

Xcode — программа для разработки приложений под OS X и iOS, разработанная компанией Apple. Поставляется бесплатно на установочном диске OS X вместе с операционной системой, но устанавливается пользователем вручную. Последняя версия — 5.1.1, не поддерживается старыми версиями OS X. Последнюю версию Xcode можно бесплатно загрузить из Mac App Store.

Основным приложением пакета является встроенная среда разработки, которая называется Xcode. Помимо этого, пакет Xcode включает в себя большую часть документации разработчика от Apple и Interface Builder — приложение, использующееся для создания графических интерфейсов.

Пакет Xcode включает в себя изменённую версию свободного набора компиляторов GNU Compiler Collection (GCC, apple-darwin9-gcc-4.0.1, который с июля 2012 года больше не будет существовать в составе инструментария разработчиков для OS X (в Xcode 4.4), осенью GCC покинет и iOS (его не будет в Xcode 4.5) и поддерживает языки C, C++, Objective-C, Objective-C++, Java, AppleScript, Python и Ruby с различными моделями программирования, включая (но не ограничиваясь) Cocoa, Carbon и Java.

Interface Builder упрощает создание пользовательского интерфейса (UI) (рисунок 3.2). С его помощью можно легко, без написания кода, создать слои из окон, различные кнопки, ползунки и другие элементы управления. Затем надо превратить этот прототип UI в реальное приложение, добавив новые возможности. Xcode работает с Interface Builder в режиме реального времени, так что в графическом интерфейсе (Interface Builder) видно то, что программист пишет в Xcode.



**Рисунок 3.2 – Интерфейс Builder**

Можно легко создавать пользовательские интерфейсы, поскольку Cocoa был построен с использованием шаблона Model-View-Controller (MVC). На самом деле, UI фактически являются архивами объектов Cocoa, которые не требуют генерации кода. Изменения в интерфейсе пользователя (UI) не требуют перекомпиляции (перепроверки) кода, а изменения в коде, не требуют перекомпиляции UI.

**3.5 Обзор языка Objective-C**

Objective-C, известный также как Objective C, ObjC или Obj-C — компилируемый объектно-ориентированный язык программирования, используемый корпорацией Apple, построенный на основе языка Си и парадигм Smalltalk. В частности, объектная модель построена в стиле Smalltalk — то есть объектам посылаются сообщения.

Язык Objective-C является надмножеством языка Си, поэтому Си-код полностью понятен компилятору Objective-C.

Компилятор Objective-C входит в GCC и доступен на большинстве основных платформ. Язык используется в первую очередь для Mac OS X (Cocoa) и GNUstep — реализаций объектно-ориентированного интерфейса OpenStep. Также язык используется для iOS (Cocoa Touch).

ObjC был создан Брэдом Коксом в начале 1980-х в его компании Stepstone. Он пытался решить проблему повторного использования кода.

Целью Кокса было создание языка, поддерживающего концепцию software IC. Под этой концепцией понимается возможность собирать программы из готовых компонентов (объектов), подобно тому как сложные электронные устройства могут быть легко собраны из набора готовых интегральных микросхем.

При этом такой язык должен быть достаточно простым и основанным на языке С, чтобы облегчить переход разработчиков на него.

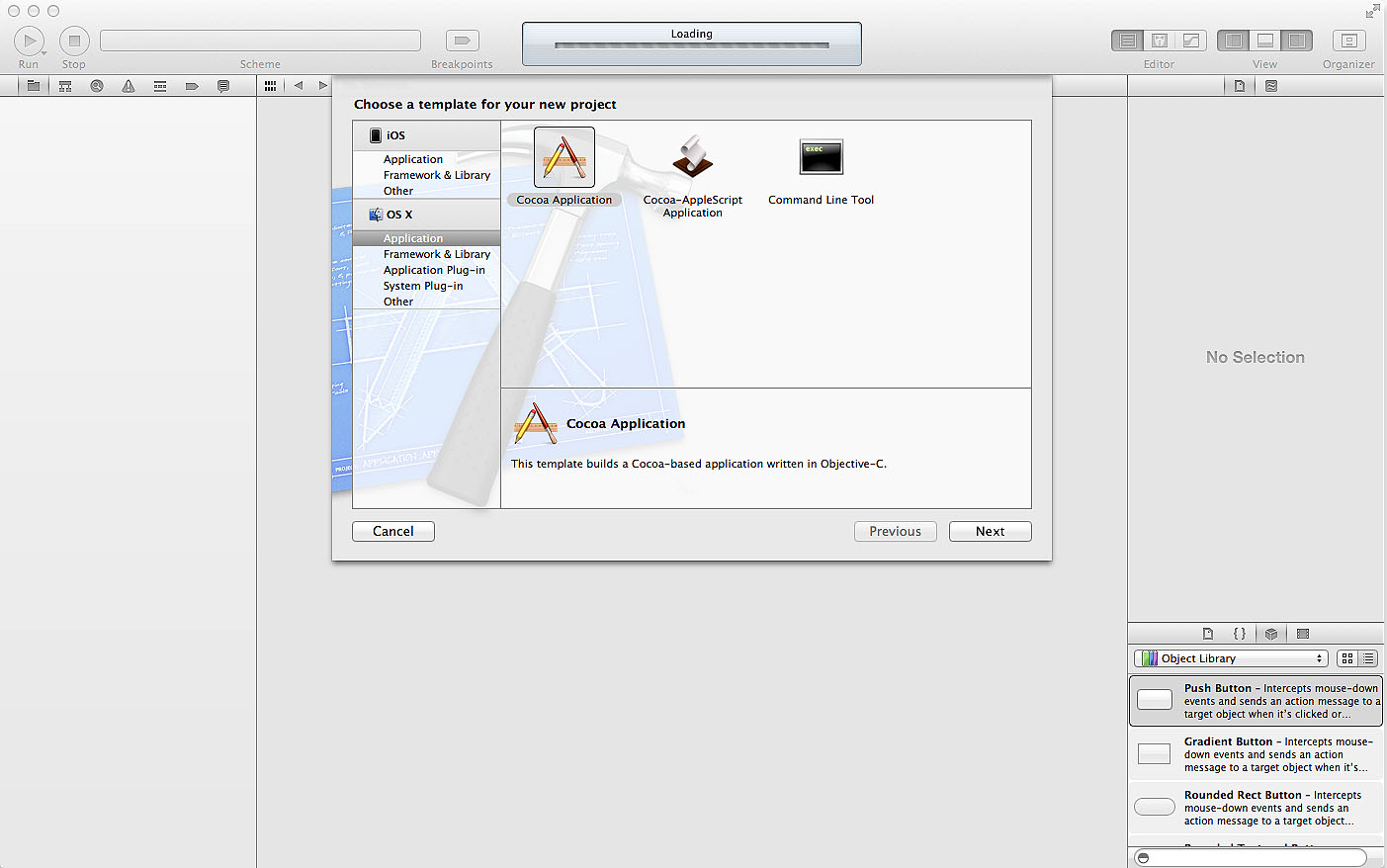
Одной из целей было также создание модели, в которой сами классы также являются полноценными объектами, поддерживалась бы интроспекция и динамическая обработка сообщений.

Получившийся в результате язык Objective-C оказался крайне прост — его освоение у С-программиста займет всего несколько дней. Он является именно расширением языка С — в язык С просто добавлены новые возможности для объектно-ориентированного программирования. При этом любая программа на С является программой и на Objective-C.

Одной из отличительных черт Objective-C является его динамичность — целый ряд решений, обычно принимаемых на этапе компиляции, здесь откладывается непосредственно до этапа выполнения.

**3.6 Обзор фрэймворка Сocoa**

Cocoa — родная объектно-ориентированная среда разработки приложений для операционной системы Mac OS X производства компании Apple (рисунок 3.3). Это один из пяти основных API, доступных в Mac OS X, — Cocoa, Carbon, Toolbox (для работы старых приложений Mac OS 9), POSIX и Java. Такие языки, как Perl, Python и Ruby не считаются основными, так как на них пока что пишется не так много серьёзных приложений для Mac OS X.



**Рисунок 3.3** – **Окно Cocoa**

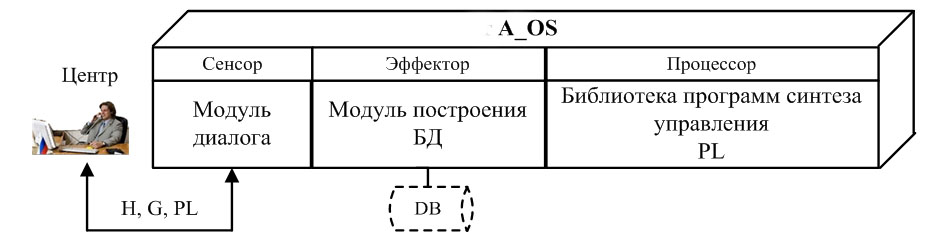
Приложения, использующие Cocoa, обычно разрабатываются с помощью среды разработки Apple Xcode (в прошлом называвшегося Project Builder) и Interface Builder с использованием языка Objective-C. Однако, среда Cocoa также доступна и при разработке на других языках, таких как Ruby, Python и Perl с помощью связующих библиотек (MacRuby, PyObjC и CamelBones соответственно). Также можно писать Cocoa-программы на Objective-C в обычном текстовом редакторе и вручную компилировать их с помощью GCC или make-сценариев для GNUstep.

С точки зрения конечного пользователя, Cocoa-приложения это приложения, написанные с использованием программной среды Cocoa. Такие приложения обычно имеют характерный внешний вид, поскольку эта среда во многом упрощает поддержку принципов «человечного интерфейса» Apple (Apple Human Interface Guidelines).

**3.7 Архитектура системы**

Архитектура системы включает модули, реализующие процессы решения задачи. Выше показано, что решение необходимо свести к пяти процессам, обеспечивающих в комплексе выполнение мониторинга, соответственно агентов должно быть пять.

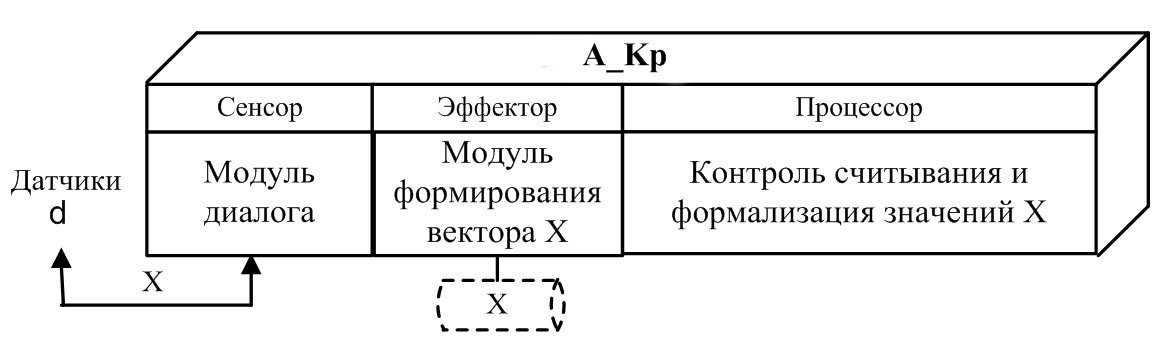
Синхронизация процессов осуществляется руководством организации, поэтому в первую очередь построим архитектуру модуля, формирующего модель иерархии и включающий атрибуты всех одушевленных и виртуальных (программы, устройства) участников организации(Рисунок 3.4).

****

**Рисунок 3.4** – **Архитектура модуля построения сцены**

Сенсорный элемент модуля A\_OS в диалоге в руководством организации получает всю необходимую информацию как о самом центре, так и об объектах наблюдения. Вся полученная информация аккумулируется в базе данных (DB). Программы анализа данных находятся в библиотеке PL, которая формируется на стадии компиляции агента.

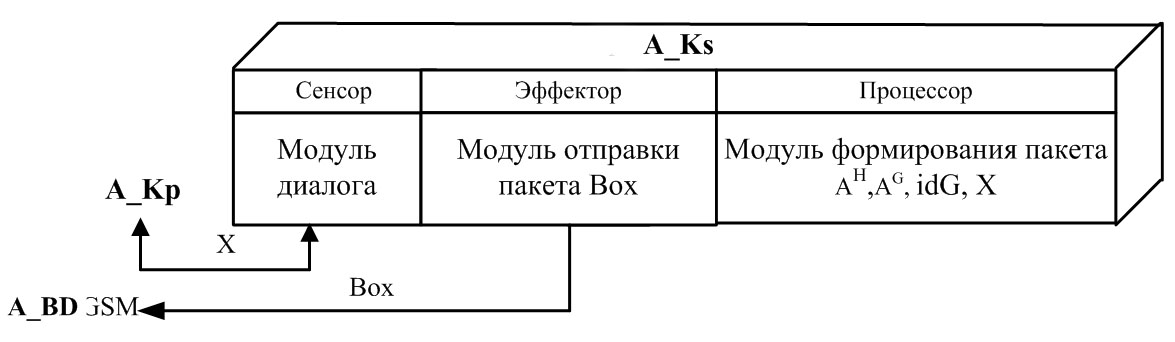
Первичным источником информации в мониторинге данного типа являются датчики. Соответственно, второй модуль должен получить значения признаков, формализовать их и передать агенту связи. Соответствующая архитектура представлена на рис.3.5.



**Рисунок 3.5** – **Архитектура модуля для формирования вектора значений диагностических показателей**

Работа модуля агента осуществляется, как правило, в дискретном режиме, т.е. значения показателей снимаются через определенный промежуток времени, установленный экспертом при разработке контроллера. Аппаратно модуль представлен комплексом датчиков и программируемым контроллером. Датчики измеряют значения <X> параметров X, контроллер формализует их и проводит первичную статистическую обработку.

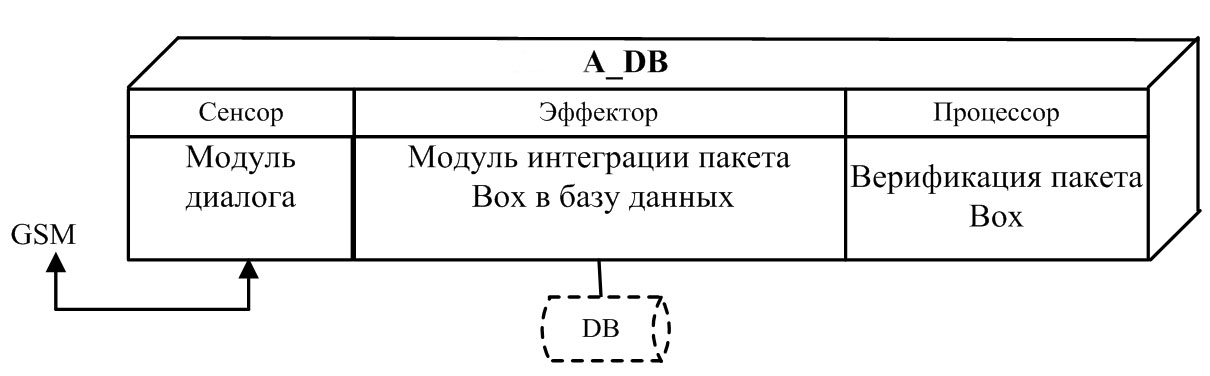
Далее активизируется модуль, который получает вектор X и формирует пакет-сообщение Box о текущем состоянии объекта для отправки в центр. Пакет содержит вектор показателей и идентификационные атрибуты объекта. Архитектура модуля A\_Ks, выполняющего эти операции, представлена на рис. 3.6.

** Рисунок 3.6** – **Архитектура модуля для формирования сообщения в центр**

Завершив формирование пакета, модуль A\_Ks выходит в сеть GSM по протоколу GPRS и отправляет сообщение Box на сервер в центр.

Аппаратно модуль A\_Ks представлен программируемым контроллером и средствами сопряжения с внешними коммуникациями.

Функции получения и интеграции пакета в базу данных DB выполняет агент A\_DB, архитектура которого представлена на рисунке 3.7.

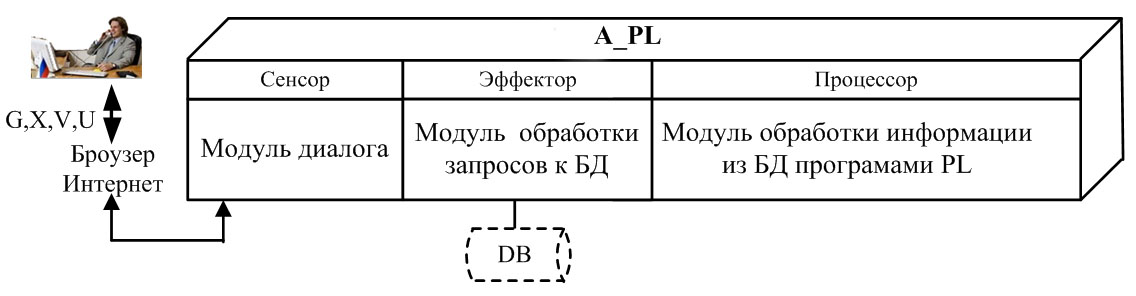
****

**Рисунок 3.7** – **Архитектура модуля для решения прикладных задач**

Агент анализирует входную информацию, идентифицирует задачу и объект, проверяет корректность значений вектора X, выполняет статистическую обработку, интегрирует результат в базу данных центра.

Информация, интегрированная в базу данных, может использоваться для решения задач синтеза состояния объектов, выработки управления, прогноза и т.д. Кроме того, она может использоваться в OLAP-анализе методами Data Mining и Knowledge Discovery. Доступ к базе осуществляется посредством абонентских диспетчерских программ, совместимых с интернет-браузерами MS Explorer, Opera, FireFox и др.

Обработку запросов пользователей к базе данных осуществляет агент A\_PL, архитектура которого представлена на рис. 3.8.

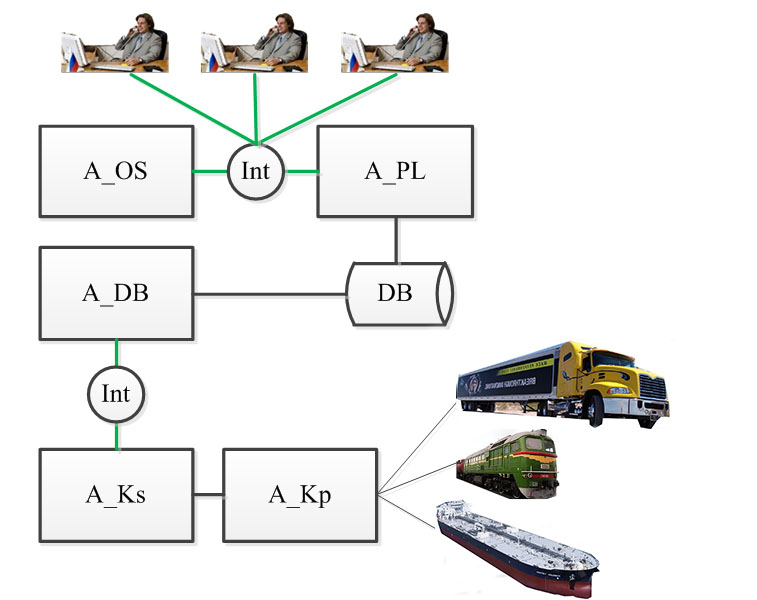
****

**Рисунок 3.8** – **Архитектура модуля обработки запросов и решения прикладных задач**

Доступ к ресурсам сервера (базе данных и пакету программ обработки) осуществляется после проверки входного кода пользователя. Использование общедоступных браузеров позволяет контролировать состояние удаленных объектов с помощью стационарных и портативных компьютеров, коммуникаторов, планшетов и т.д. Библиотека PL имеет модульную структуру, поэтому список решаемых задач постоянно расширяется.

Таким образом, архитектура системы мониторинга можно представить как совокупность пяти программных агентов, сервера, базы данных и стандартных коммуникационно-диалоговых средств Интернет. Такой подход позволяет модернизировать аппаратную и программную части каждого из участников мониторинга, не затрагивая остальных. При изменении требований коммуникаций достаточно перепрограммировать контроллер агента A\_Ks.

Пример построения архитектуры на основе разработанного подхода для удаленного мониторинга автопоездов, тепловозов, танкеров представлен на рис.3.9.



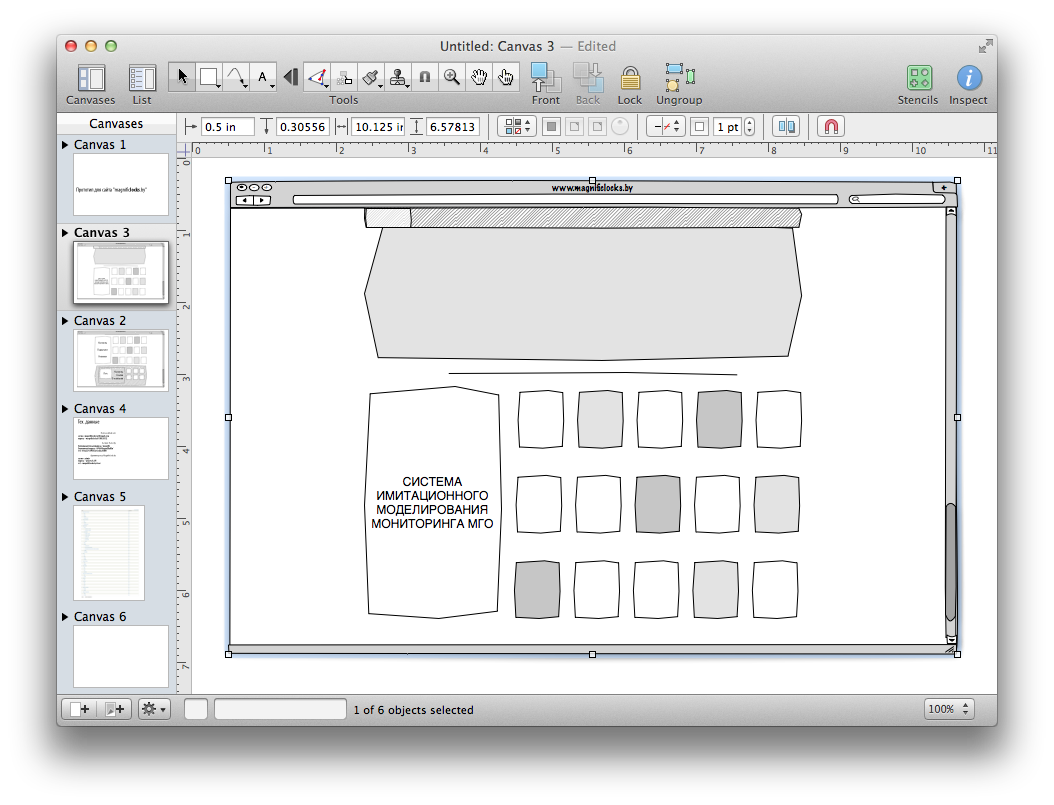
**Рисунок 3.9** – **Типовая архитектура системы мониторинга СМО**

Для практической реализации аналогичных архитектур разработан широкий спектр датчиков и программируемых контроллеров. В настоящее время датчики могут снимать в среднем около 20 различных параметров (температуру, вибрацию, давление, уровень топлива в баках, обороты двигателей, токи, местоположение абонента в GPS, ГЛОНАСС и т.д.), характерных для мобильных механизмов (тепловозов, вагонов, тягачей, грузовиков, танкеров и т.п.). Возможность программирования обеспечивает быструю настройку контроллеров на новую предметную область и выбор конфигурации датчиков.

Основной нерешенной проблемой, повышающей стоимость данного варианта архитектуры, является большой объем ручного труда при настройке программ обработки библиотеки PL на конкретную предметную область. При этом количество продукций при логическом выводе состояния V объектов G в зависимости от значений X, и синтез управления U в зависимости от V может доcтигать нескольких тысяч, что затрудняет не только отладку, но и понимание логики массива продукций неподготовленным пользователем.

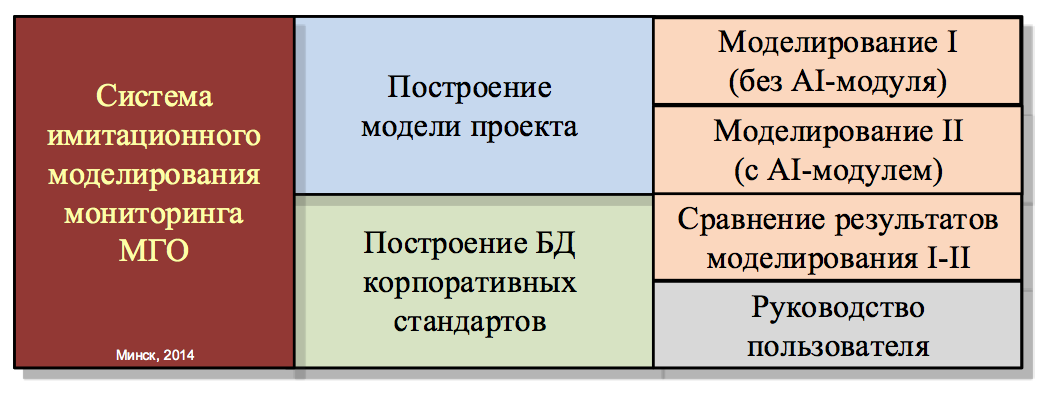
**3.8 Эскизы интерфейсов**

Разработка эскизов интерфейсов является трудоемкой операцией, которая обычно выполняется программистом или дизайнером вручную. Для максимального упрощения рисования часто достаточно сложных схем используется специализированная компьютерная система OmniGraffle, которая, несмотря на высокую стоимость широко используется. Рабочее окно OmniGraffle представлено на рисунке 3.10.



**Рисунок 3.10 – Рабочая область OmniGraffle**

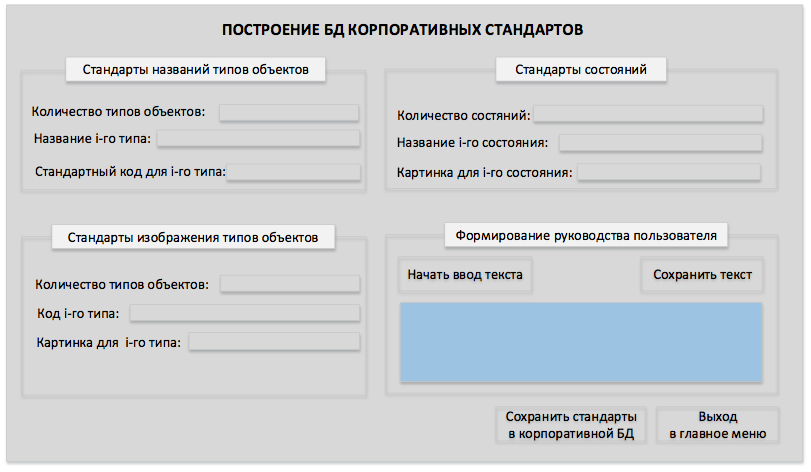
Разработанные эскизы реализации алгоритмов и визуализации результатов представлены на рисунках 3.11- 3.17.



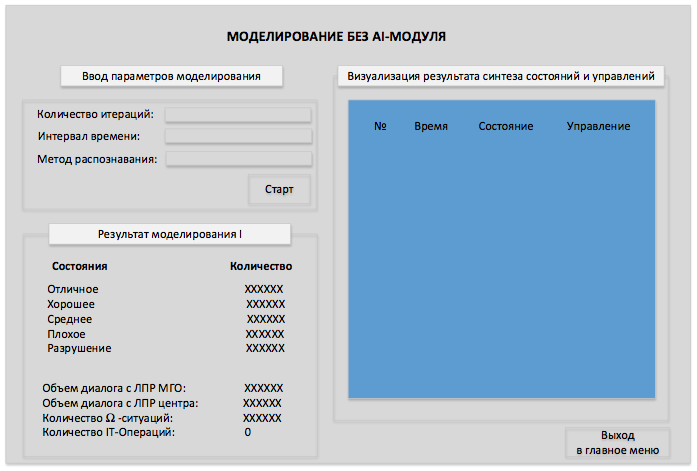
**Рисунок 3.11 – Эскиз главного меню**



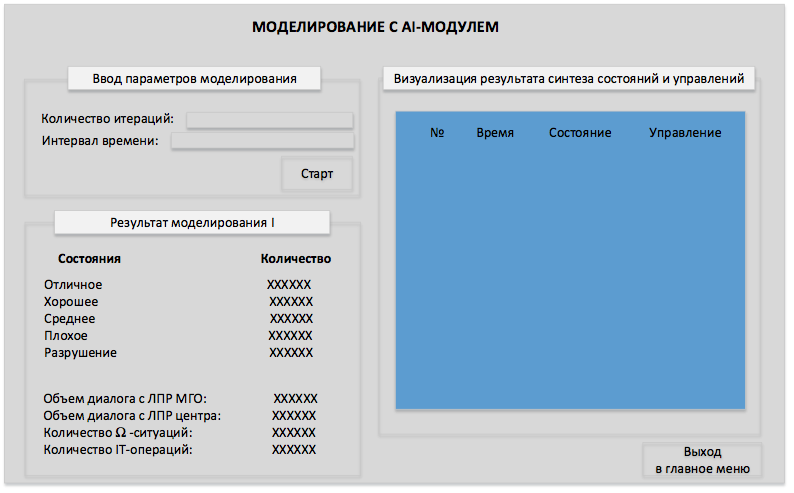
**Рисунок 3.12 – Эскиз режима построения модели**



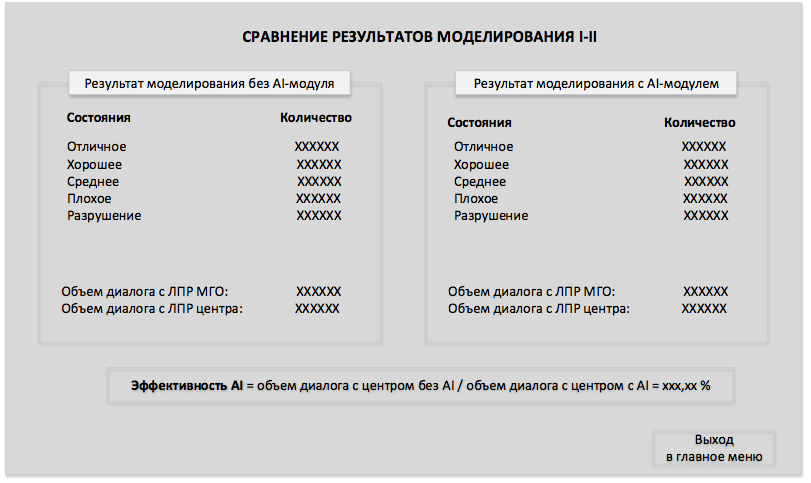
**Рисунок 3.13 – Эскиз режима создания стандартов**



**Рисунок 3.14 – Эскиз режима моделирования без AI-модуля**



**Рисунок 3.15 – Эскиз режима моделирования c AI-модулем**



**Рисунок 3.16 – Эскиз режима сравнения результатов**

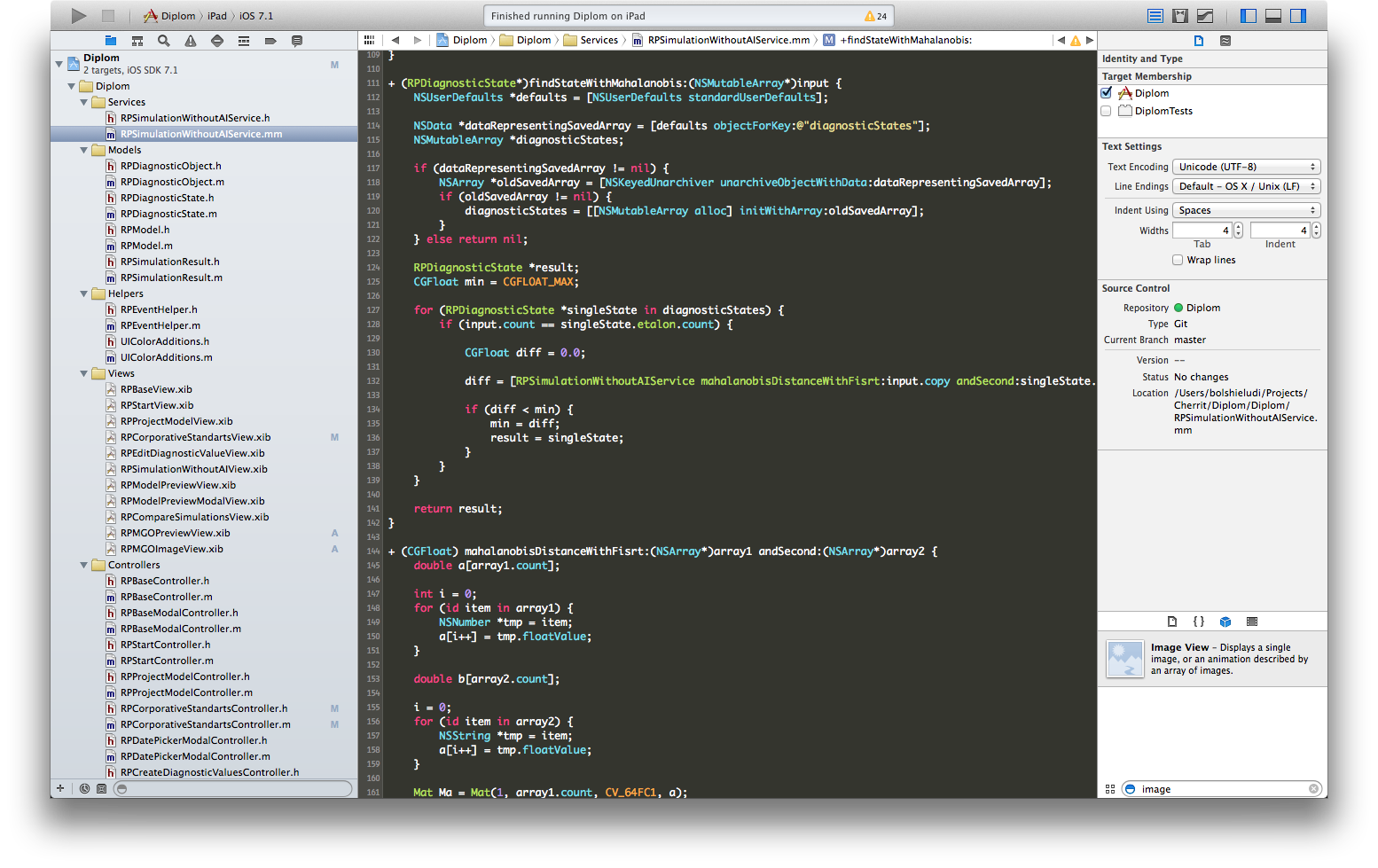


**Рисунок 3.17 – Эскиз руководства пользователя**

Приведенные выше эскизы могут быть реализованы в рамках различных подходов, например, в стиле Metro, развитом компанией Microsoft в Windows 8, или в любом другом стиле. В данной работе предлагается минималистический стиль мобильной операционной системы iOS 7, что обеспечивает наглядность выполнения всех этапов мониторинга в рамках 1-2 окон.

**3.9 Библиотека программ**

В процессе реализации архитектуры с использованием Mac OS X, Xcode, Objective-C, Cocoa (рисунок 3.18) была разработана библиотека Monitoring.



**Рисунок 3.18 – Окно разработки программ на языке Objective-C**

Библиотека содержит пять программных модулей:

- модуль построения сцены мониторинга;

- модуль анализа контрольных точек и формирования вектора значений диагностических показателей для объекта;

- модуль оценки объекта и синтеза соответствующего управляющего решения;

- модуль отправки сообщения в центр.

Библиотека реализована на языке Objective-C для Apple iPad. Она так же может быть реализована на любом современном объектно-ориентированном языке.

На основе библиотеки был разработан вариант системы для моделирования мониторинга СМО, главное меню которого представлено на рисунке 3.19.

****

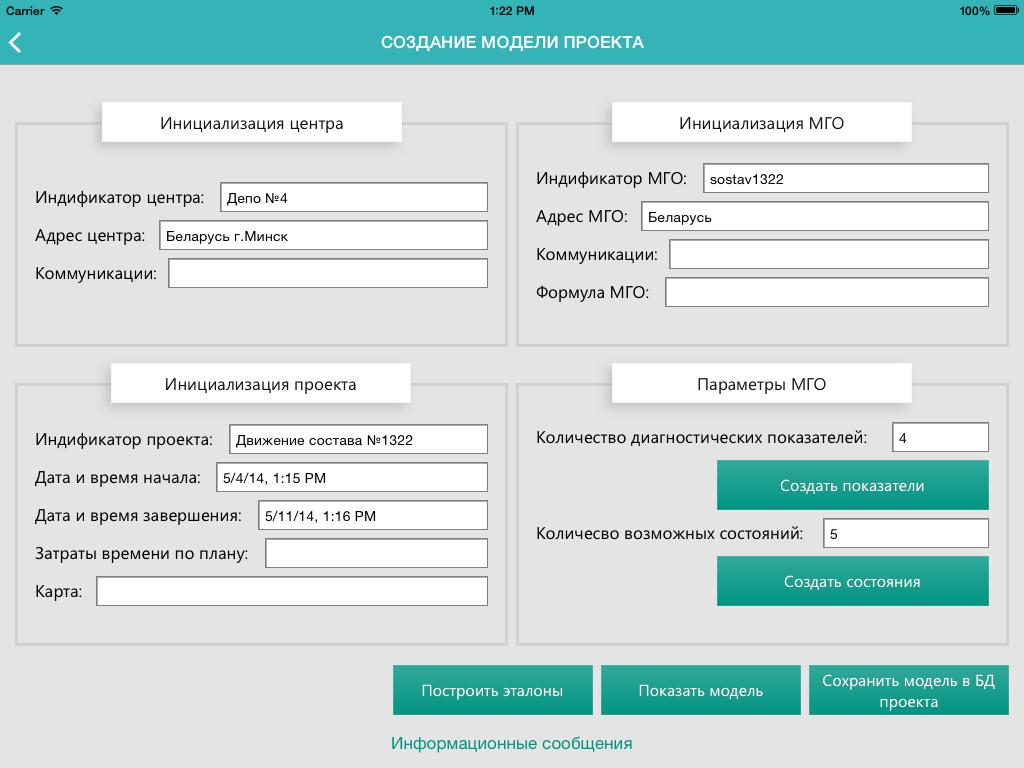
**Рисунок 3.19 – Главное меню системы**

Методика применения разработанной системы описана ниже.

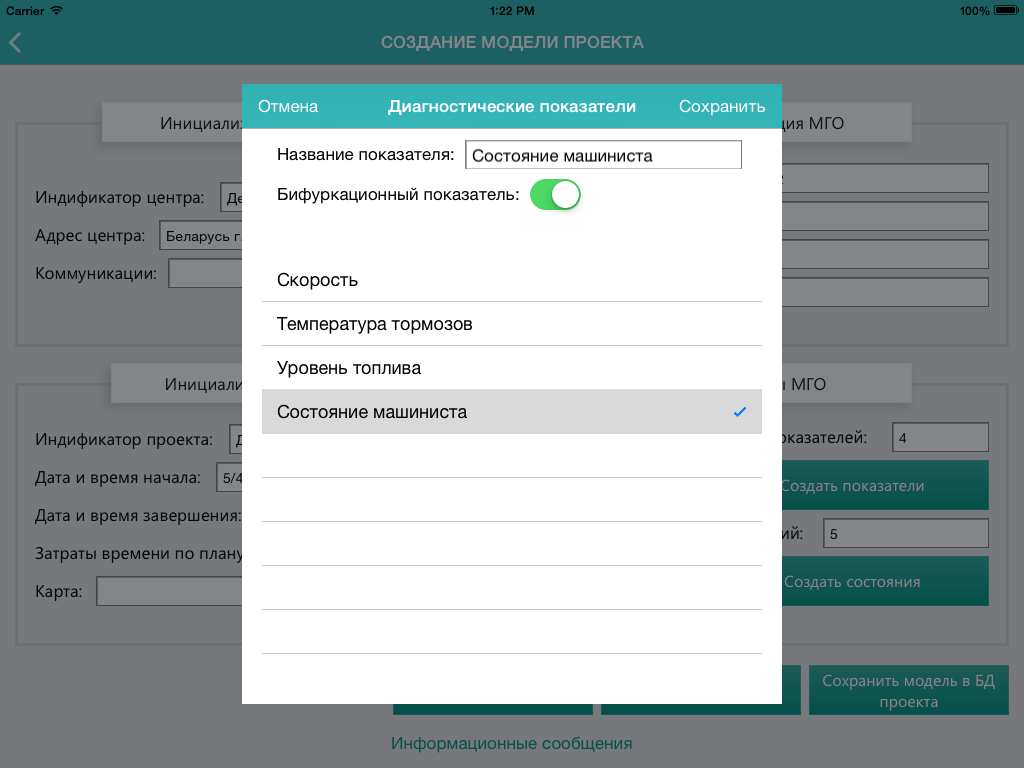
**3.10 Методика применения системы**

Методика применения системы включает пять шагов.

**Шаг 1.** Построение сцены (рисунок 3.20), задание диагностических показателей (рисунок 3.21) и диапазона их изменения. В сцене участвуют ЛПР Железнодорожного Депо 4 и железнодорожный состав из тепловоза и цистерны.

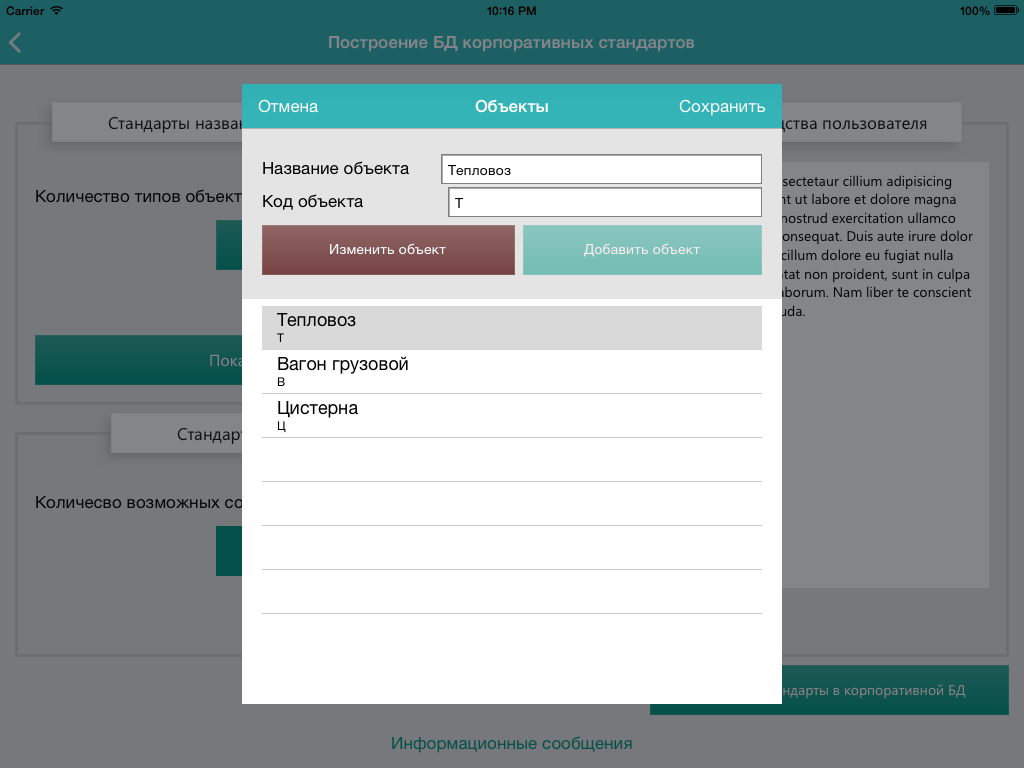


**Рисунок 3.20 – Построение сцены.**

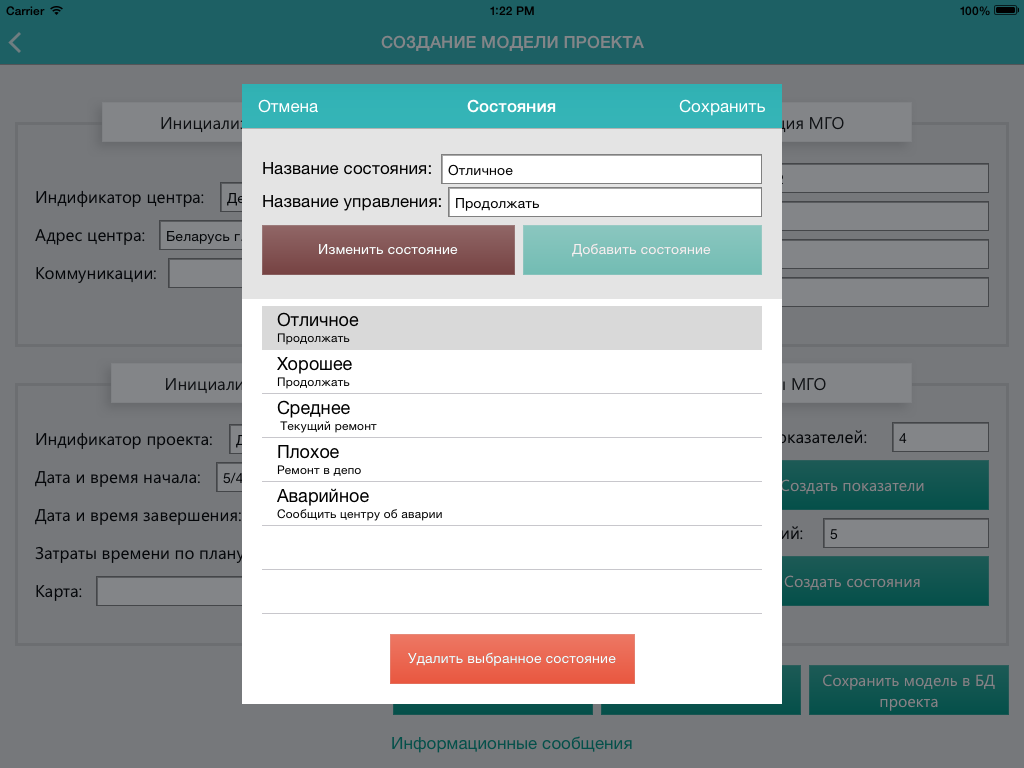


**Рисунок 3.21 – Задание диагностических показателей**

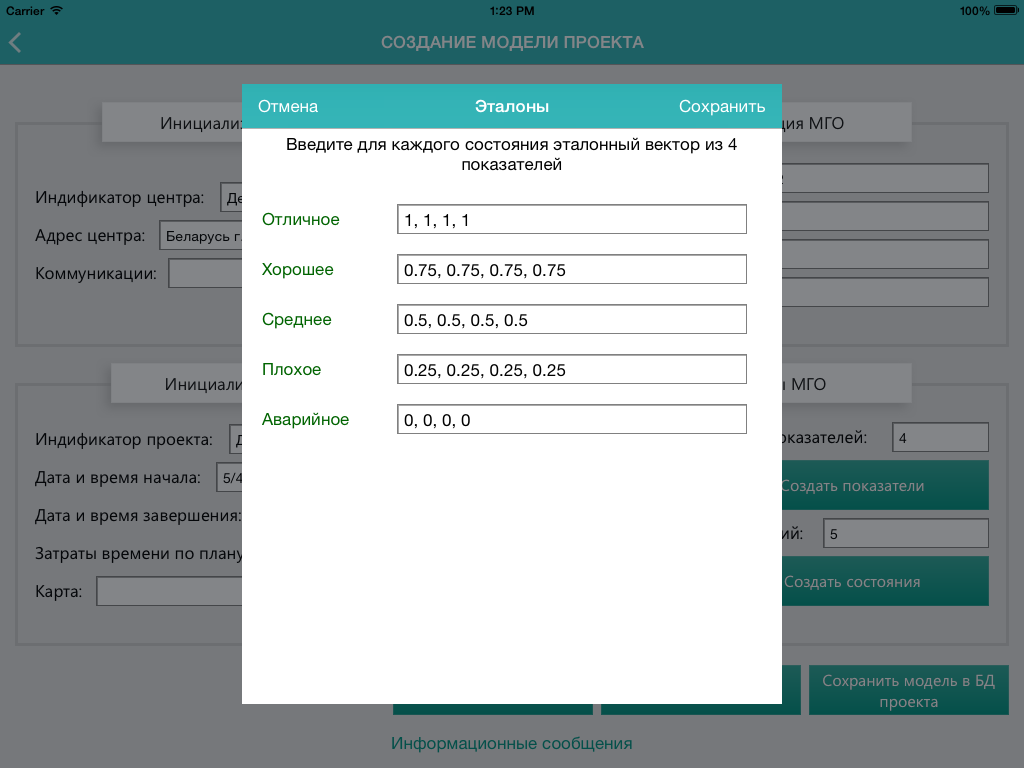
**Шаг 2**. Построение объектов (рисунок 3.22), состояний (рисунок 3.23), эталонов (рисунок 3.24), управлений.



**Рисунок 3.22 – Создание объектов**

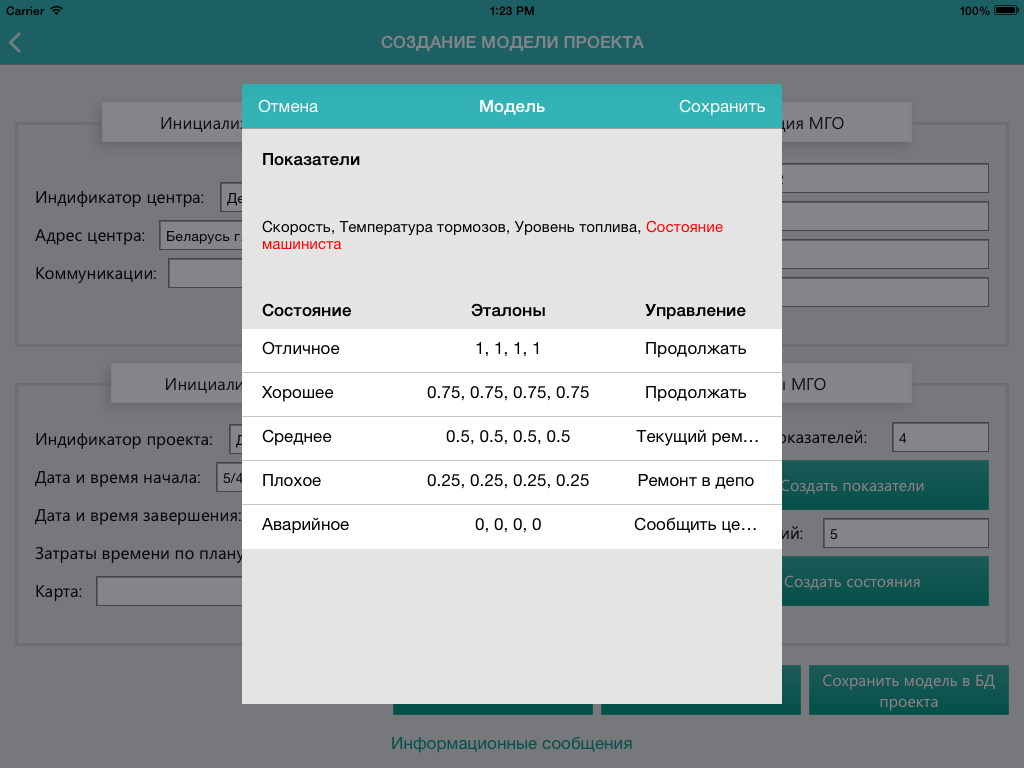


**Рисунок 3.23 – Ввод состояний**

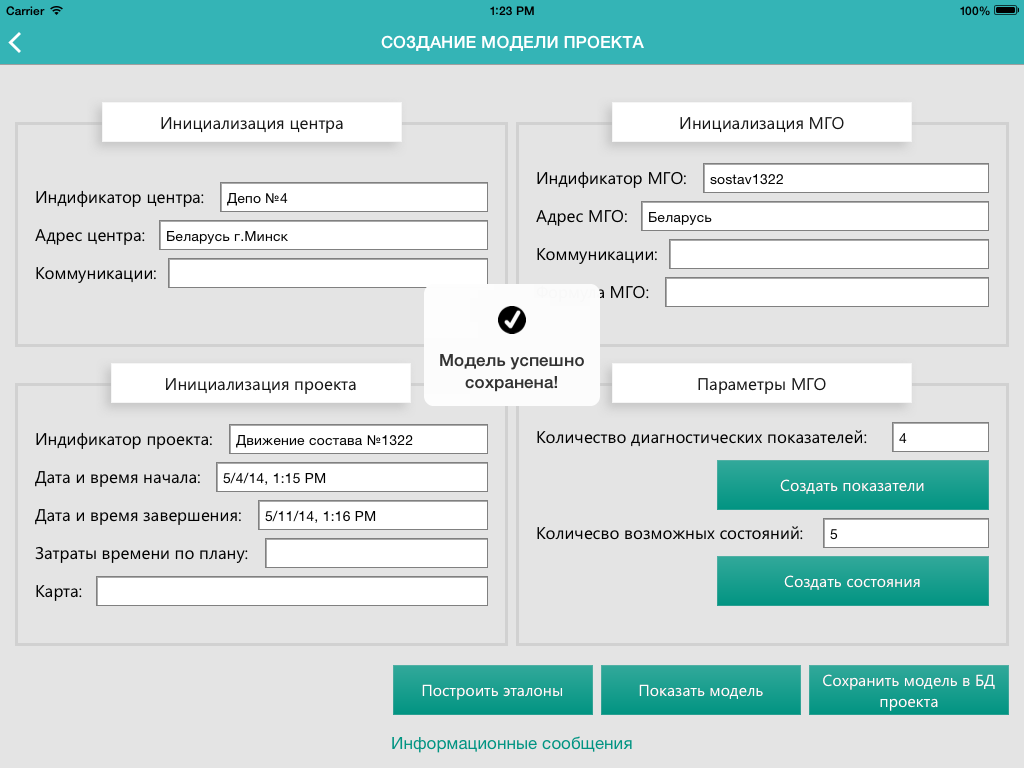


**Рисунок 3.24 – Ввод состояний**

**Шаг** **3**. Создание (рисунок 3.25) и сохранение модели в БД приложения (рисунок 3.26).

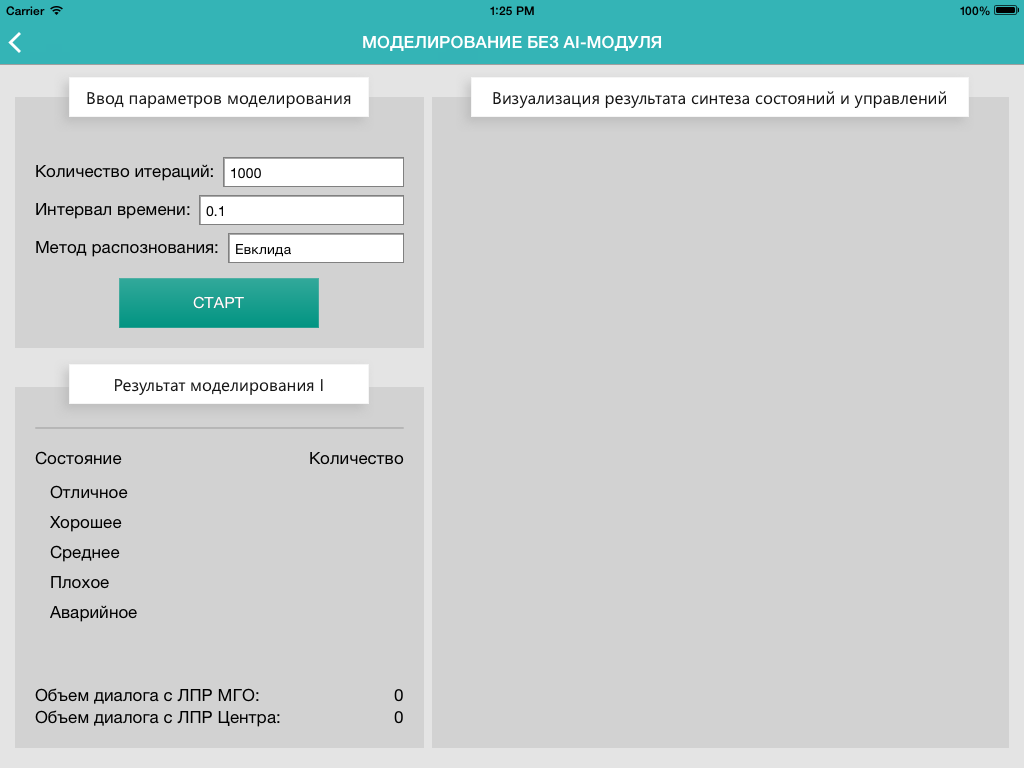


**Рисунок 3.25 – Созданные состояния, эталоны и управления**

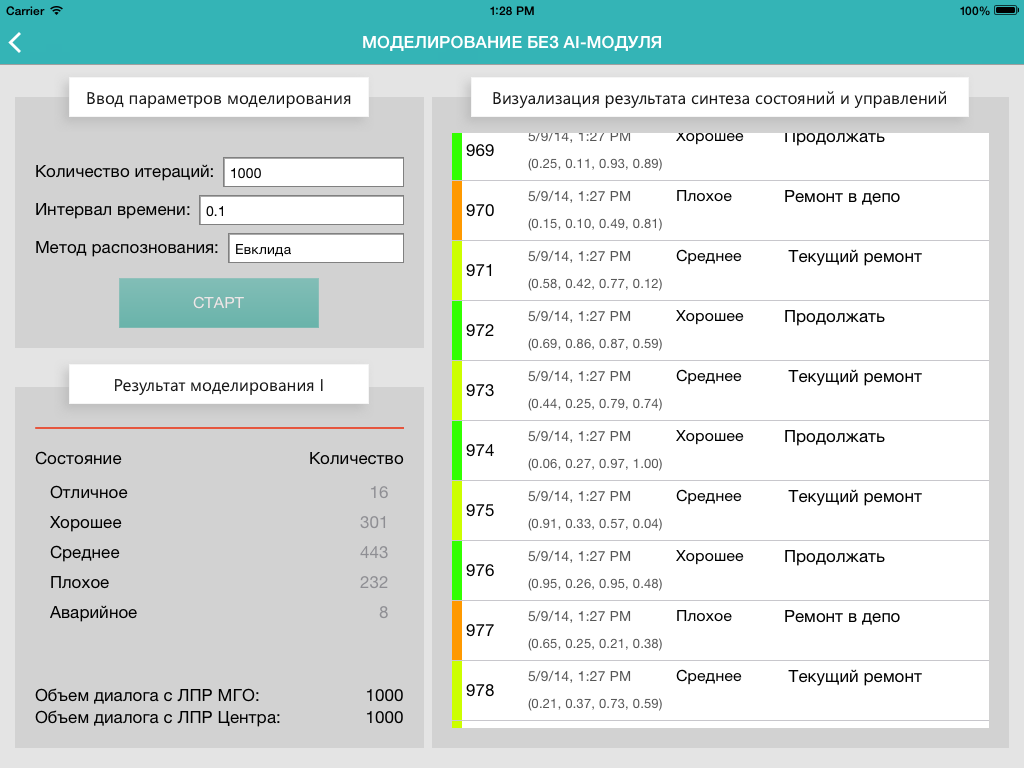


**Рисунок 3.26 – Сохранение модели в БД**

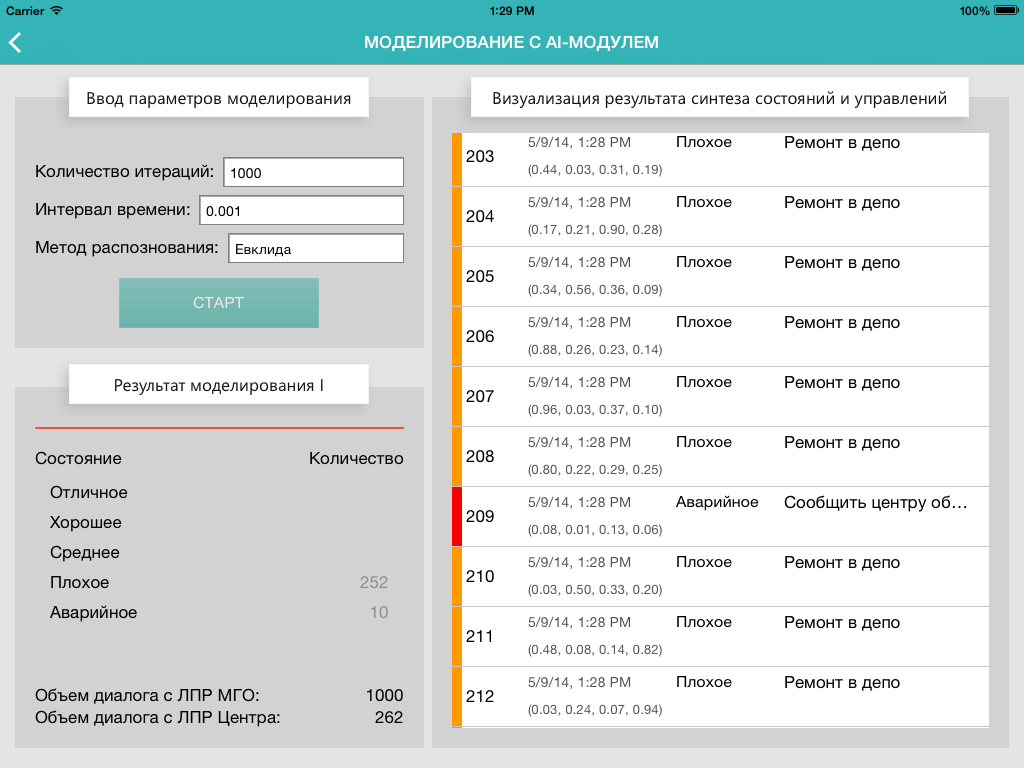
**Шаг 4.** Ввод количества итераций, интервала времени, а также метода распознавания (рисунок 3.27). Моделирование состояния объекта в диагностических точках без использования AI-модуля (рисунок 3.28) и с использованием AI-модуля (рисунок 3.29).



**Рисунок 3.27 – Ввод параметров моделирования**



**Рисунок 3.28 – Результат моделирования без AI-модуля**



**Рисунок 3.29 – Результат моделирования с AI-модулем**

В результате выполнения алгоритмов на экран выводится вся необходимая информация для ЛПР центра и ЛПР объекта наблюдения.

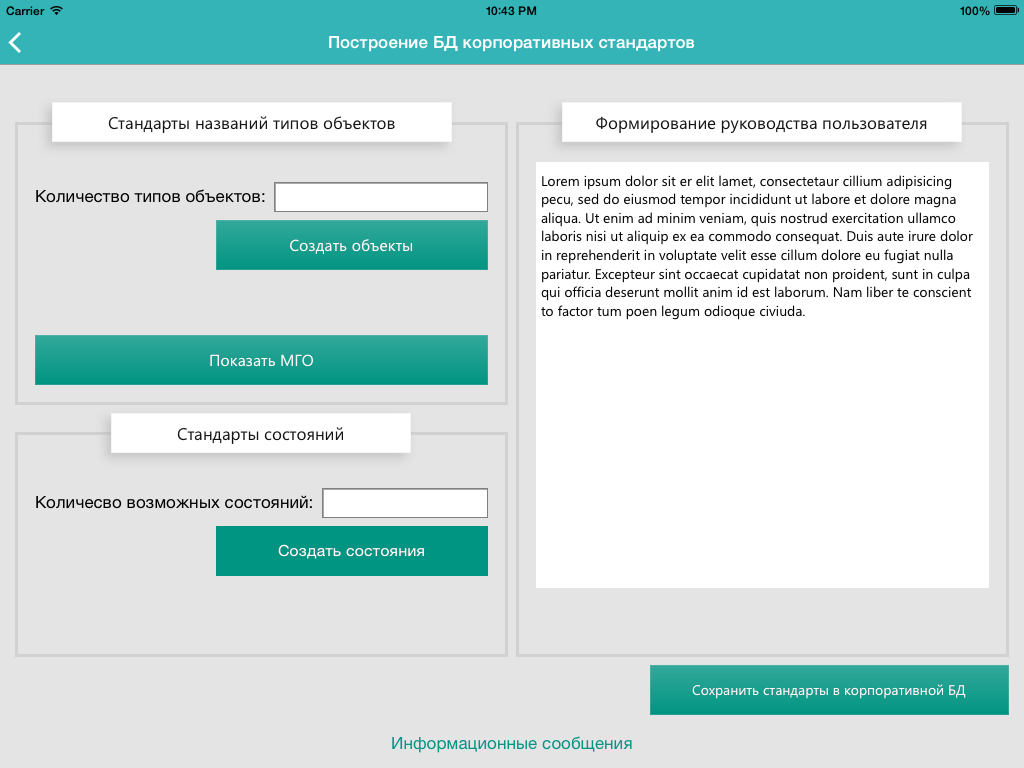
**Шаг 5.** Сравнение результатов моделирования без AI-модулем и с AI-модулем (рисунок 3.30).



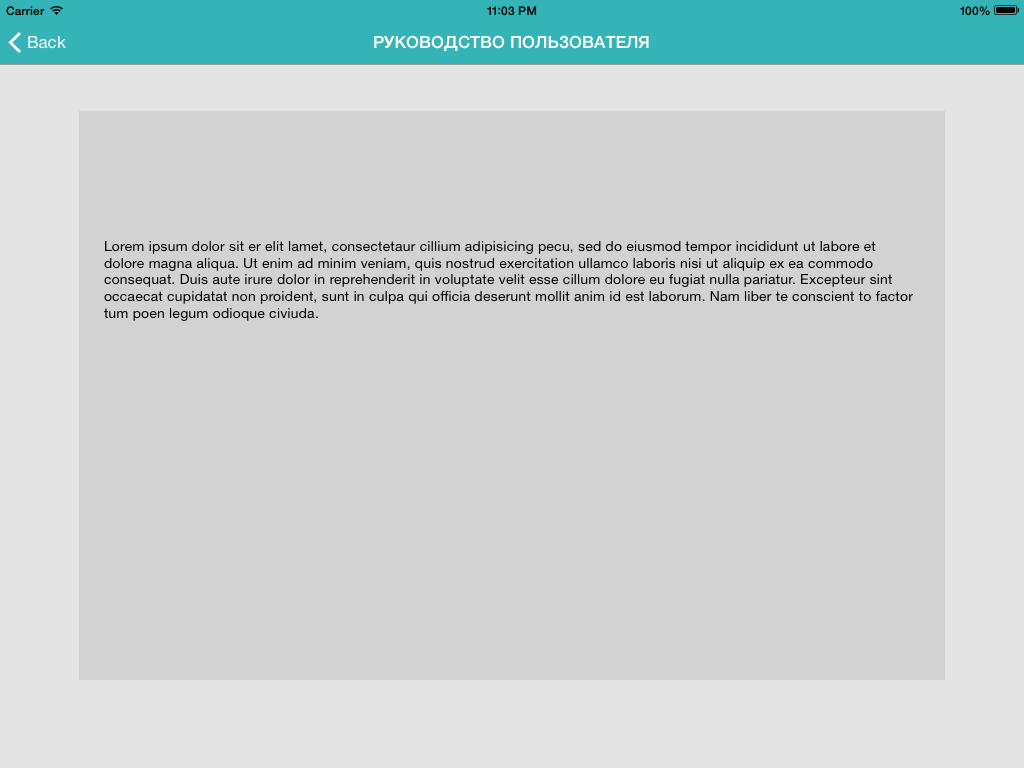
**Рисунок 3.30 – Сравнение результатов моделирования**

**Дополнительная функциональность.** Для удобства - можно ввести руководство пользователя (рисунок 3.31) и использовать его из главного меню (рисунок 3.32).

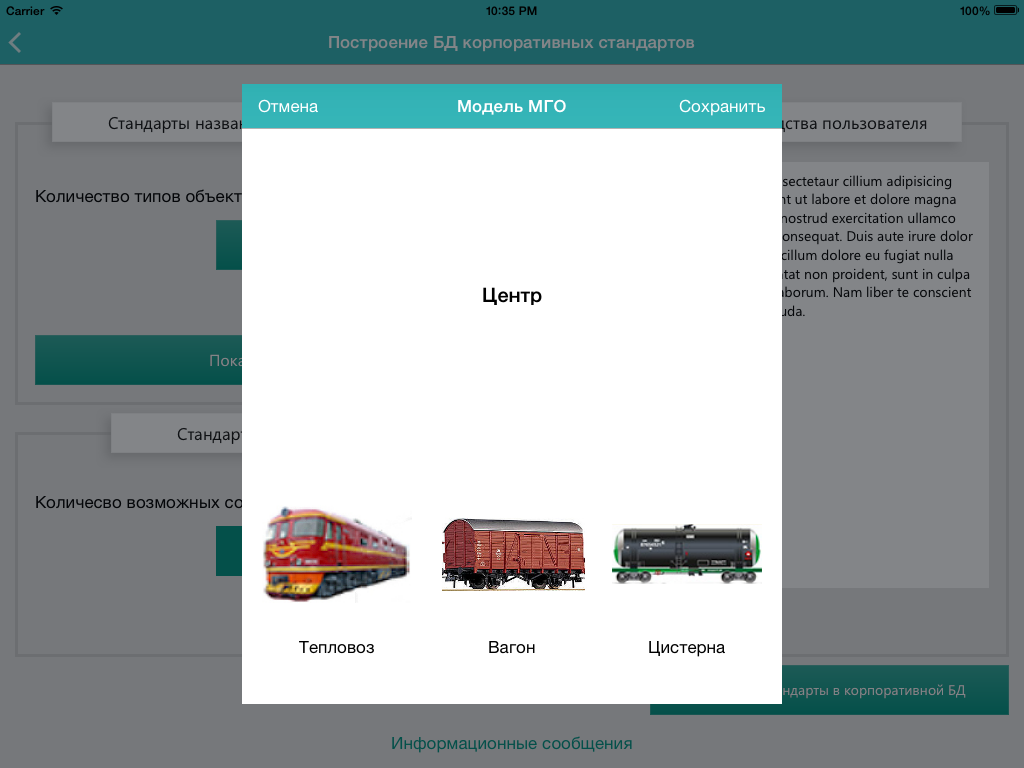
Существует возможность построить визуализацию сцены (рисунок 3.33).

****

**Рисунок 3.31 – Формирование руководства пользователя**

****

**Рисунок 3.32 – Руководства пользователя**



**Рисунок 3.33 – Визуализация сцены**

**3.11 Выводы**

В третьей главе были получены следующие результаты:

* сформулированы требования к архитектуре системы;
* обоснован выбор системного и прикладного ПО: Mac OS X, Xcode, Objective-C, Cocoa;
* средствами специализированной системы OmniGraffle разработан комплект эскизов интерфейсов, соответствующих архитектуре;
* средствами языка Objective-C разработана библиотека программ, реализующих архитектуру, на ее основе построена система Monitoring;
* проведена апробация системы на примере моделирования процесса организации мониторинга движения железнодорожного состава.