Титульный лист в отдельном файле.

РЕФЕРАТ

Отчёт 47 с., 10 рис., 7 табл., 6 источников, 3 прил.

Объект разработки — программный компонент, управляющий охранной сигнализацией.

Цель работы — разработка программного модуля для управления устройством охранной сигнализации в программном комплексе интеграции оборудования.

Программный модуль разрабатывается исходя из: протокола взаимодействия с устройством, протокола взаимодействия с системой сбора и обработки информации, интерфейсов программного ядра системы.

В результате работы разработан программный модуль и сценарий модульного тестирования.

Программный модуль будет интегрирован в систему охранной сигнализации.

Содержание

1 Определения

В настоящем отчете об УИР применяются следующие термины с соответствующими определениями:

Ant-Engine — утилита для автоматизации процесса подготовки к сборке *проекта* (например, разрешение зависимостей проекта от библиотек);

Mock-объект — объект, реализующий заданные аспекты моделируемого программного окружения. Это фиктивная реализация интерфейса, предназначенная исключительно для тестирования;

Артефакты — набор файлов, которые не компилируется в рамках текущего *проекта*, но необходимы для работы программы (например, библиотеки, предназначенные для конкретной платформы (динамически линкуемые, с расширением .dll, .so) или файлы данных *(например, конфигурационные)*);

Библиотечный файл — скомпилированные исходные коды (файлы с расширением .jar), не подлежащие компиляции при сборке *проекта*, реализующие функционал, используемый в нескольких *проектах;*

Дистрибутив — это файл, содержащий исполняемую программу, готовую к установке на компьютер, управляющий системой;

Инструментальный компьютер — предназначен для выполнения процедур компиляции программного обеспечения и анализа исходного кода;

Конфигурационные файлы — текстовые файлы, используемые приложениями для хранения настроек;

Модуль — набор исходных кодов *проекта*, который может компилироваться отдельно;

Платформа — множество аппаратно-программных конфигураций;

Проект — совокупность файлов с исходным кодом, набором *артефактов* и файлами для среды разработки, с помощью которых возможно получение всех необходимых выходных файлов (исполняемые, конфигурационные, справочные, установочные, ресурсные, библиотечные и т. д.) программного компонента или его части;

Рабочая копия — локальное дерево папок, содержащее проект, извлеченный из *SVN;*

Релиз — выходная структура *проекта*, содержащая все необходимые файлы (исполняемые, конфигурационные, библиотечные и т. д.), а также наборы дополнительных программ и служебных утилит, необходимых для сборки на их основе дистрибутива и/или установки этих файлов на компьютере пользователя и последующей эксплуатации.

2 Обозначения и сокращения

JDK — Java development kit (комплект разработки на Java);

JVM — Java virtual machine (виртуальная машина Java);

SVN — централизованная система управления версиями файлов;

АПИ — аппаратно-программный интерфейс;

АРМ — автоматизированное рабочее место;

БД — база данных;

ИК СФЗ — интегрированный комплекс средств и систем физической защиты;

КЗ — короткое замыкание;

КИТСФЗ — комплекс инженерно-технических средств и систем физической защиты;

КНЦ — концентратор датчиков;

КОП — код команды (код операции);

КОТ — код команды ответа (код ответа);

КСА — комплекс средств автоматизации;

ЛВС — локальная вычислительная сеть;

ПУ — процессор управления;

СКУД — система контроля и управления доступом;

СОС — система охранной сигнализации;

СПО — специальное программное обеспечение;

ССОИ — система сбора и обработки информации;

СУДОС — система управления доступом и охранной сигнализацией;

ТС — техническое средство.

3 Введение

Данная учебно-исследовательская работа заключается в проектировании, реализации и модульном тестировании программного модуля, управляющего устройством в охранной системе. Разработка ведётся исходя из:

- Протокола взаимодействия с устройством;
- Протокола взаимодействия с ССОИ;
- Требований к разработке модулей АПИ.

Протокол взаимодействия с устройством в данной системе унифицирует взаимодействие с различными типами охранных датчиков. Компонент разрабатывается с учётом интеграции в программный комплекс АПИ и взаимодействия с ССОИ.

4 Основная часть

4.1 Обзор работ связанных с УИР

4.1.1 Унифицированный протокол взаимодействия с ССОИ

Данный протокол необходим для унификации подключения контроллеров технических средств ИК СФЗ различных производителей к ССОИ КИТСФЗ.[1]

Аппаратно-программные интерфейсы

АПИ используются для подключения контроллеров технических средств СФЗ к ССОИ. АПИ имеет в своем составе аппаратный модуль, реализующий физический интерфейс подключения контроллера (RS-232, RS-485, RS-422, CAN и т.п.) и СПО АПИ. СПО АПИ должно[1]:

- обеспечивать обмен с ССОИ через ЛВС по унифицированному протоколу верхнего уровня;
- обеспечивать обмен информацией с контроллерами ТС через аппаратный интерфейс по специализированному протоколу ТС.

Платформой для установки АПИ являются ПУ, входящие в состав ССОИ. В ССОИ процессоры управления включены в состав КСА пунктов управления и участков контроля. ПУ представляет собой необслуживаемый компьютер с круглосуточным режимом работы. В ПУ устанавливаются (или подключаются) аппаратные модули АПИ, и на ПУ устанавливается СПО АПИ. На одном ПУ могут функционировать несколько АПИ. Обобщенная структурная схема АПИ приведена на рисунке 1.[1]

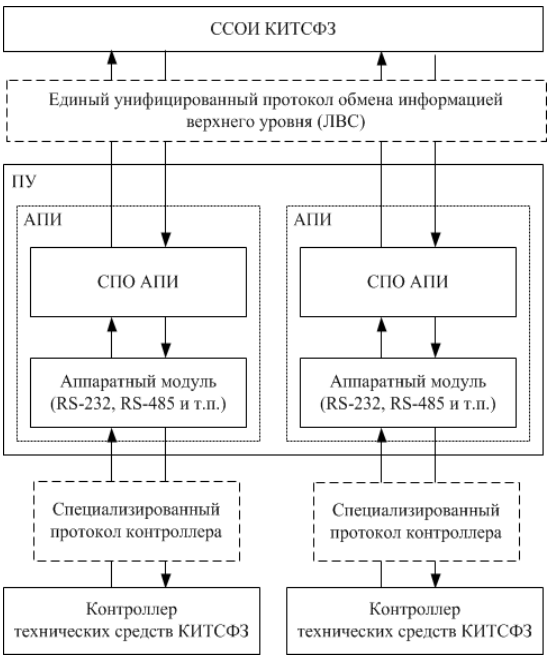

Рисунок 1 - Обобщённая структурная схема АПИ

Понятие модели типа устройства

Для унификации взаимодействия с различным оборудованием в едином унифицированном протоколе обмена информацией используется понятие модели типа устройства. Модель типа устройства — это совокупность информации (характеристик) одного типа устройств (прибора, контроллера, шлейфа сигнализации, видеокамеры и т.п.). Между моделями поддерживается отношения родитель-потомок. Схема модели представлена на рисунке 2.

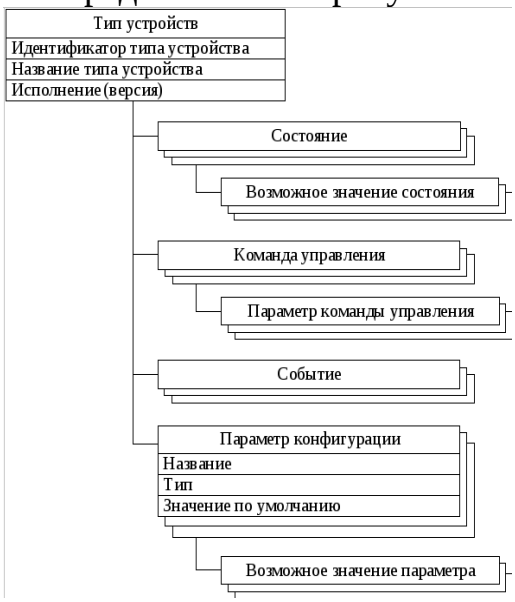

Рисунок 2 - Схема модели типа устройства

Модель типа устройства формируется производителем оборудования и передается в ССОИ по единому унифицированному протоколу верхнего уровня. Модель заносится в БД ССОИ, и в дальнейшем вся работа ССОИ с оборудованием строится на основе информации о модели устройства. АПИ является устройством, у которого есть дочерние типы устройств (например, порты RS-485), у которых в свою очередь есть свои дочерние типы устройств (например, контроллеры) и т. д.[1]

Варианты подключения технических средств СФЗ к ССОИ

ТС СФЗ должны подключаться к ССОИ по одному из следующих вариантов[1]:

- Если контроллер ТС имеет возможность подключения в локальную вычислительную сеть по транспортному протоколу TCP/IP, а программное обеспечение контроллер ТС реализует единый унифицированный протокол обмена информацией верхнего уровня, то контроллер ТС подключается напрямую в ЛВС ССОИ КИТСФЗ. В этом случае программное обеспечение ТС реализует функции АПИ. Модель оборудования при этом варианте формируется в контроллере ТС и передается в ССОИ по единому унифицированному протоколу обмена информацией верхнего уровня;
- Если ТС имеет возможность подключения по стандартным аппаратным интерфейсам (RS-232, RS-485, CAN и т.п.), то производителем контроллера ТС поставляется СПО АПИ, обеспечивающее обмен информацией с контроллером ТС через аппаратный интерфейс по специализированному протоколу с одной стороны и обмен с ССОИ через ЛВС по унифицированному протоколу верхнего уровня с другой стороны. СПО АПИ и стандартный аппаратный интерфейс устанавливается на ПУ, контроллер ТС подключается к стандартному аппаратному интерфейсу. Модель оборудования в этом случае формируется СПО АПИ и переда-

ется в ССОИ по единому унифицированному протоколу обмена информацией верхнего уровня;

- Если контроллер ТС подключается по уникальному аппаратному интерфейсу, то производителем контроллера ТС поставляется аппаратный интерфейсный модуль для подключения ТС к ПУ. Также, производителем контроллера ТС поставляются СПО АПИ, обеспечивающее обмен информацией с контроллером ТС через аппаратный интерфейс по специализированному протоколу с одной стороны и обмен с ССОИ через ЛВС по унифицированному протоколу верхнего уровня с другой стороны. СПО АПИ и аппаратный интерфейс устанавливается на ПУ, контроллер ТС подключается к аппаратному интерфейсу. Модель оборудования в этом случае формируется СПО АПИ и передается в ССОИ по единому унифицированному протоколу обмена информацией верхнего уровня.

4.1.2 Протокол взаимодействия с КНЦ

В данной системе непосредственного взаимодействия между датчиками и управляющим компьютером нет. Управляющий компьютер получает текущую информацию о состояниях датчиков от концентратора. Упрощённая схема физического подключения устройств представлена на рисунке 3.

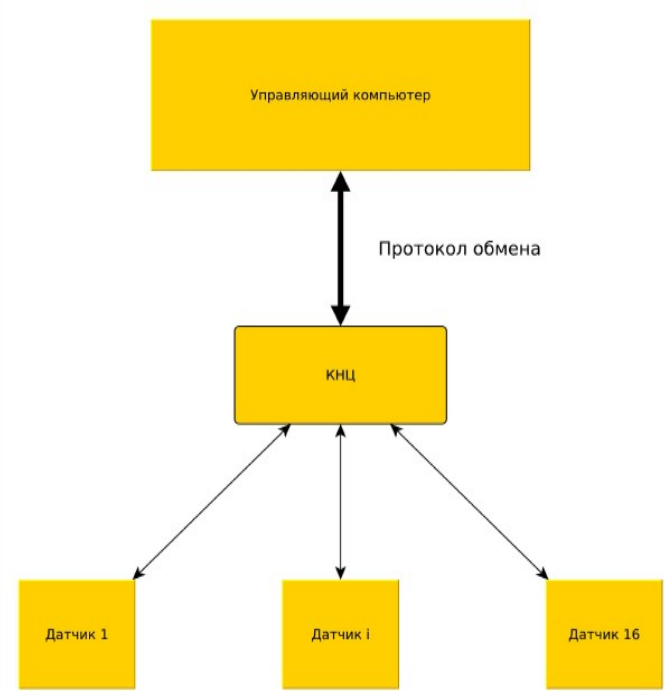

Рисунок 3 - Схема физического подключения датчиков

Далее будет описана часть протокола прикладного уровня взаимодействия между управляющим компьютером и КНЦ, которая касается сбора информации от датчиков.

Система команд блока КНЦ обеспечивает управление датчиками (максимальное число датчиков в аппаратной конфигурации — 16). Сокращённая система команд прикладного уровня приведена в таблице 1.[2]

В команде 77 биты первых двух байтов соответствуют текущим состояниям датчиков (B1[1] — датчики 1-8, начиная с младших битов; B2 — датчики 9-16, начиная с младших битов). Значение бита, равное 0, соответствует состоянию «Норма». Значение бита, равное 1, соответствует состоянию «Тревога». Состояние сохраняется в случае отсутствии связи с управляющим компьютером[2].

В командах 74, 76 **единичные** биты первых двух байтов соответствуют событиям изменения состояний датчиков (B1 — датчики 1-8, начиная с младших битов; B2 — датчики 9-16, начиная с младших битов), для команды 74 из «Нормы» в «Тревогу», для команды 76 из «Тревоги» в «Норму»[2].

В командах 74, 77 биты третьего и четвертого байтов дают уточнения для со-

1 Обозначение Bi значит i-ый байт в текущей команде.

стояния «Тревога» (команда 77), перехода в состояние «Тревога» (команда 74). Для значений битов, равных 1 в первом и втором байтах, соответствующие биты в третьем и четвертом байтах имеют значения, равные 0, для «КЗ» и 1 для «Обрыва»[2].

Команды 74, 76 приходят в результате возникновения события. Команда 77 по запросу от управляющего компьютера[2].

Аппаратная конфигурация датчиков предполагает, что они всегда находятся под охраной[2].

Таблица 1 - Команды прикладного уровня

КОП	**Посылки от концентратора**	**КОТ**	**Посылки на концентратор**
74	**Новая тревога** В1(младш.датч.), В2(старш.датч.) – (бит=1 – соотв. датчик перешел в состояние «Тревога») В3(младш.датч.), В4(старш.датч.) – (бит=1 – «Обрыв», бит=0 – «КЗ»)		
76	**Пропадание тревоги** В1(младш.датч.), В2(старш.датч.) – (бит=1 – соотв. датчик перешел в состояние «Норма»)		
77	**Текущее состояние датчиков** В1(младш.датч.), В2(старш.датч.) – (бит=1 – соотв. датчик находится в состоянии «Тревога») В3(младш.датч.), В4(старш.датч.) – (бит=1 – «Обрыв», бит=0 – «КЗ»)	77	**Запрос текущего состояния датчиков**
80	**Информация о параметрах концентратора** В1 — тип системы; В2 — значение, считываемое с перемычек (техническая информация, для пусконаладочных работ); В3 — конфигурация датчиков (младш.:1-8)(1-есть/0-нет); В4 — конфигурация датчиков (старш.: 9-16) (1-есть/0-нет).	80	**Запрос информации о параметрах концентратора**

4.2 Обзор инструментальных средств

4.2.1 Язык программирования Java

Для разработки программного модуля используется язык программирования Java. Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры[3]. Выделим существенные преимущества Java:

- Автоматическое управление сборкой мусора;
- Кроссплатформенность;
- Набор стандартных классов-коллекций (строки, массивы, карты);
- Встроенные в язык средства создания многопоточных программ.

Основные недостатки Java:

- Высокое потребление памяти, по-сравнению с аналогичной программой на C++;
- Низкая скорость работы приложений.

В текущей работе используется JDK версии 1.6.0u13. При разработке используется интегрированная среда разработки IntelliJ IDEA. IDEA предлагает удобные средства для:

- Рефакторинга исходного кода;
- Работы с системами контроля версий;
- Удобной логической организации Java приложений;
- Удобного запуска тестовых сценариев.

4.2.2 Библиотека для модульного тестирования JUnit

JUnit — библиотека для модульного тестирования Java программ, которая предоставляет возможности удобной и гибкой организации тестовых сценариев.

Модульное тестирование позволяет тестировать компоненты отдельно от других частей проекта. Среда разработки IDEA поддерживает простой запуск тестов с удобной настройкой параметров. В текущей работе используется JUnit версии 4.8.2.

Гибкость в первую очередь обеспечивается унифицированным созданием фикстур, которые помогают многократно использовать программный код[4]. Фактически, фикстура — метод класса, тестирующего некоторый функционал. Можно создать фикстуры следующих видов:

- Уровня класса (выполняются 1 раз: до или после выполнения всех тестовых методов);
- Уровня тестового метода (выполняются до или после каждого тестового метода).

4.2.3 Библиотека журналирования Log4j

Разрабатываемый программный компонент интегрируется в систему управления охранной системой. К этой системе предъявляется требование ведение журналов, содержащих важную информацию о работе программы, необходимой для диагностики и локализации неисправностей и ошибок. Для решения этой задачи используется библиотека Log4j версии 1.2.12. Её основным преимуществом является гибкость настроек, которые хранятся в специальном конфигурационном файле log4j.xml. При необходимости можно легко изменить логику ведения журналов не меняя исходный код проекта.

Журналирование заключается в вызове методов объекта класса Logger, отвечающих за нужный приоритет, и передаче в параметрах текста сообщения. В используемой библиотеке определены следующие приоритеты[5]:

- FATAL — произошла фатальная ошибка - у этого сообщения наивысший приоритет;
- ERROR — в программе произошла ошибка;
- WARN — предупреждение, что в программе что-то не так;

- INFO — просто полезная информация;
- DEBUG — детальная информация для отладки;
- TRACE — наиболее полная информация. трассировка выполнения программы. Наиболее низкий приоритет.

Логика ведения журналов описывается с помощью аппендеров. Аппендер — объект, который определяет: что нужно делать с журналируемыми сообщениями[5].

4.2.4 Система контроля версий и сборка проектов

Перед описанием системы контроля версий скажем несколько слов об инфраструктуре разработки программного обеспечения. Инфраструктура, обеспечивающая разработку программной системы и, в частности программного модуля, представлена на рисунке 4. Сервер приложений – компьютер, предоставляющий набор сервисов, необходимых при разработке программного обеспечения. Сервис **—** гостевая операционная система, запущенная в виртуальной машине Vmware Workstation, в которой запускается основное приложение. Примером основного приложения являются: система контроля версий SVN, система автоматизированной промышленной сборки Teamcity.

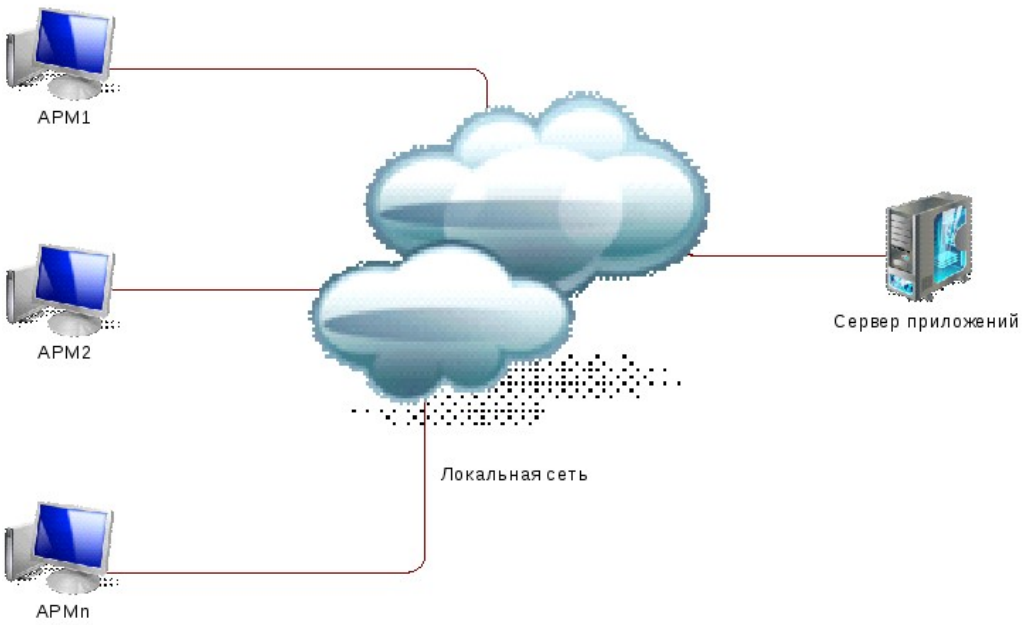

Рисунок 4 - Инфраструктура разработки ПО

Система контроля версий

Проект и артефакты хранятся на специальном сервере приложений с установленным сервисом SVN, который позволяет управлять файлами и каталогами, а также сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных и даёт возможность изучить историю всех изменений. При необходимости работы с проектом, его следует извлечь из SVN или обновить ранее извлеченную рабочую копию[6].

В таблице 2 приведена схема директорий хранилища SVN.

Таблица 2 - Схема хранилища SVN

Путь	Описание
/	Корневая папка svn-хранилища
/ DocRepo	Документация
/ LibRepo	Хранилище библиотек
/ ModuleRepo	Хранилище модулей
/ ProjectRepo	Хранилище проектов

Сборка и компиляция проектов

Используются следующие варианты сборки проектов:

1. **Сборка на локальном компьютере.** Данный метод используется только при разработке. Перед сборкой выполняются подготовительные действия, такие как загрузка библиотечных и конфигурационных файлов, от которых зависит собираемый проект. Эти действия выполняются с помощью программы Ant-Engine в среде разработки. Далее производится компиляция проекта.
2. **Сборка на удалённом инструментальном компьютере с помощью системы Teamcity.** Это так называемая промышленная сборка программы. Данный метод используется для автоматизированной сборки готовой к использованию системы.

4.3 Проектирование программного модуля

Проектирование программного модуля состоит из следующих этапов:

а) Создание модели типа устройства (см. раздел 4.3.1);

б) Определение тестовых воздействий и ожидаемых реакций на эти воздействия (см. разделы 4.4.3 и 4.4.4);

в) Определение интерфейсов взаимодействия с другими модулями;

г) Представление пункта в) в виде статической UML диаграммы классов и диаграммы последовательности (см. раздел 4.3.2);

д) Представление жизненного цикла объекта основного разрабатываемого класса в виде UML диаграммы состояний (см. раздел 4.3.2);

4.3.1 Описание модели датчика

В таблице 3 представлены команды от ССОИ к программному модулю датчика.

Таблица 3 - Команды от ССОИ

Порядковый номер	Команда	Параметры	Описание команды
-	Запрос состояния устройства	-	Запрос состояния устройства
-	Постановка устройства под охрану	-	Постановка устройства под охрану
-	Снятие устройства с охраны	-	Снятие устройства с охраны
0	Разблокировать	-	Разрешает поступление тревог от датчика
1	Блокировать	-	Блокирует поступление тревог от датчика

В таблице 4 представлены события, которые могут быть сгенерированы в ходе работы модуля датчика.

Таблица 4 - События датчика

Событие	Описание
Тревога	Тревога от устройства
Нет тревоги	Нет тревоги

В таблице 5 перечислены состояния датчика и их возможные значения.

Таблица 5 - Состояния датчика

Состояние	**Значения**
Режим загрузки	• Загрузка • Выгрузка • Загружено • Выгружено
Функционирование	• Включение • Выключение • Выключено • Исправно • Неисправно • Не определено
Охрана	• Да • Нет • Постановка • Не определено
Тревога	• Да • Нет • Блокировано • Не определено
По диагностике	• КЗ • Обрыв • Норма • Не определено

В таблице 6 перечислены параметры датчика. Их можно изменять из ССОИ.

Таблица 6 - Параметры датчика

Параметр	**Значение по умолчанию**	**Возможные значения**
Адрес	1	1..16
Блокировка тревоги после загрузки	Выключено	• Включено • Выключено

4.3.2 UML диаграммы

Основной исходный код, управляющий датчиком, представляет из себя класс Sensor (Датчик). На рисунке 5 представлена статическая диаграмма классов, связанных с классом Sensor.

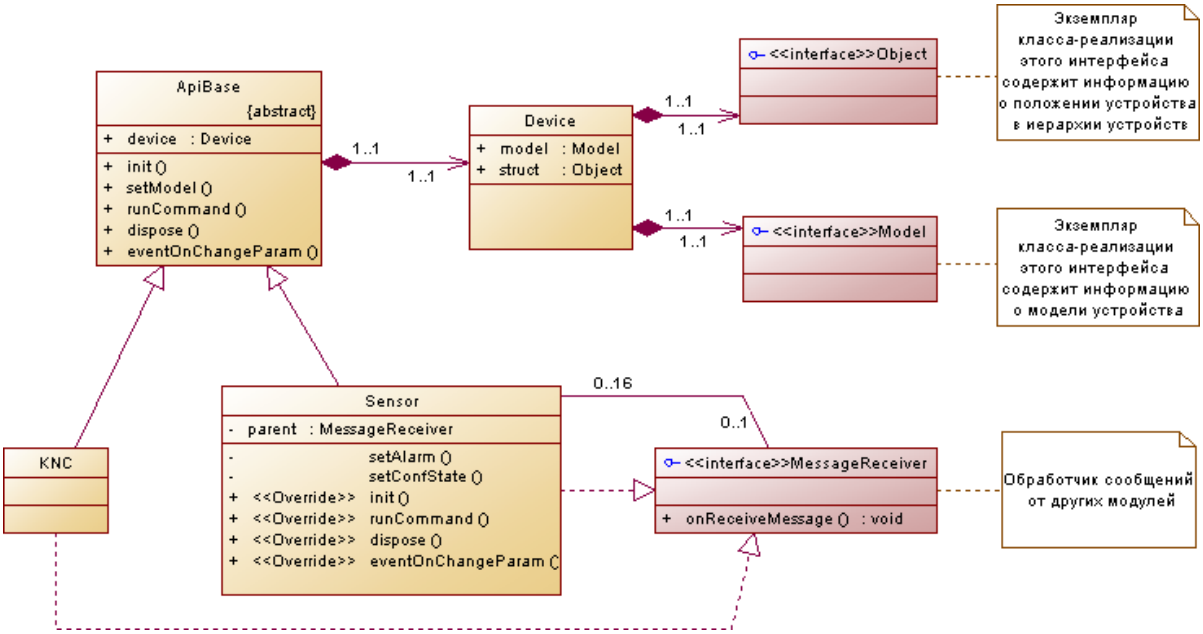

Рисунок 5 - Статическая схема классов

Для наглядности возможные состояния объекта класса Sensor изображены в виде UML диаграммы состояний на рисунке 6. Отметим следующее: состояния объекта класса Sensor – это не тоже самое, что состояния в модели типа устройства.

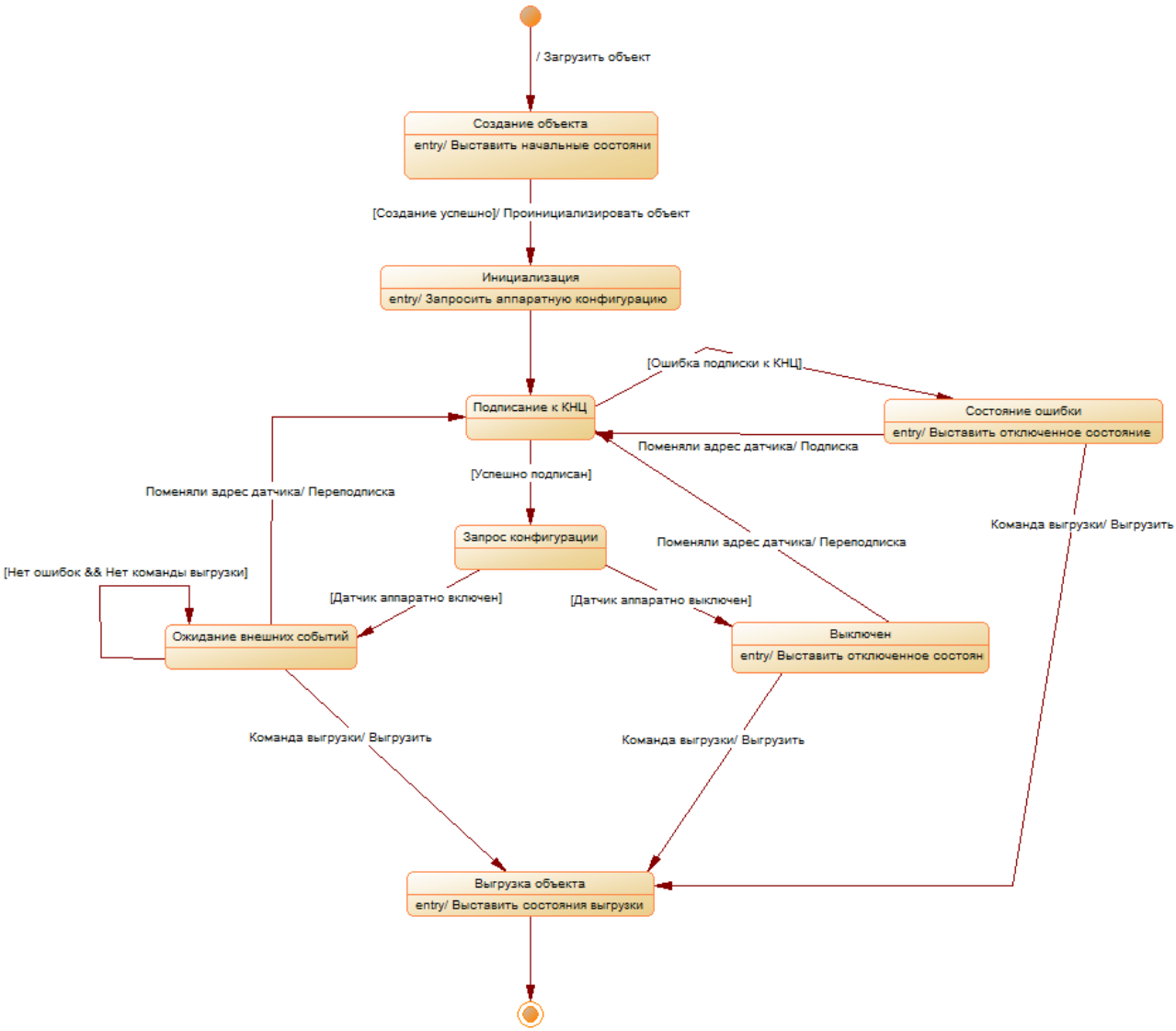

Рисунок 6 - Диаграмма состояний объекта Sensor

Объект Sensor получает сообщения (например сообщение о тревоге) от объекта класса КНЦ. Объект Sensor обрабатывает сообщения только от того объекта КНЦ, на который он подписан. Полный перечень сообщений обмена между объектами классов КНЦ и Датчика представлен в таблице 7. Отметим, что сообщения между объектами КНЦ и Датчика программные.

Таблица 7 - Сообщения между объектами КНЦ и Датчика

Запросы от объекта датчика		**Сообщения от КНЦ**	
Наименование	Описание	Наименование	Описание
QuerySubscribe	Запрос на подписку к КНЦ на получение сообщений	SensorAlarm	Сообщает о появлении тревоги/отсутствии тревоги. В случае тревоги в сообщении передается причина тревоги
QueryUnSubscribe	Запрос на отписку от получения сообщений КНЦ	SensorConfigState	Сообщает информацию о включении датчика в аппаратную конфигурацию
QueryConf	Принудительный запрос аппаратной конфигурации	SensorSubscribeResult	Сообщает результат попытки подписки датчика к КНЦ
QueryState	Принудительный запрос состояния по тревоге		

Передача сообщений между объектами синхронна (объект, передающий сообщение блокируется до поступления ответа). Необходимая последовательность вызовов для работы с объектом Датчика приведена в виде UML диаграммы на рисунке 7. Пример обмена сообщений успешно проинициализированного объекта Датчика и объекта КНЦ приведен на рисунке 8.

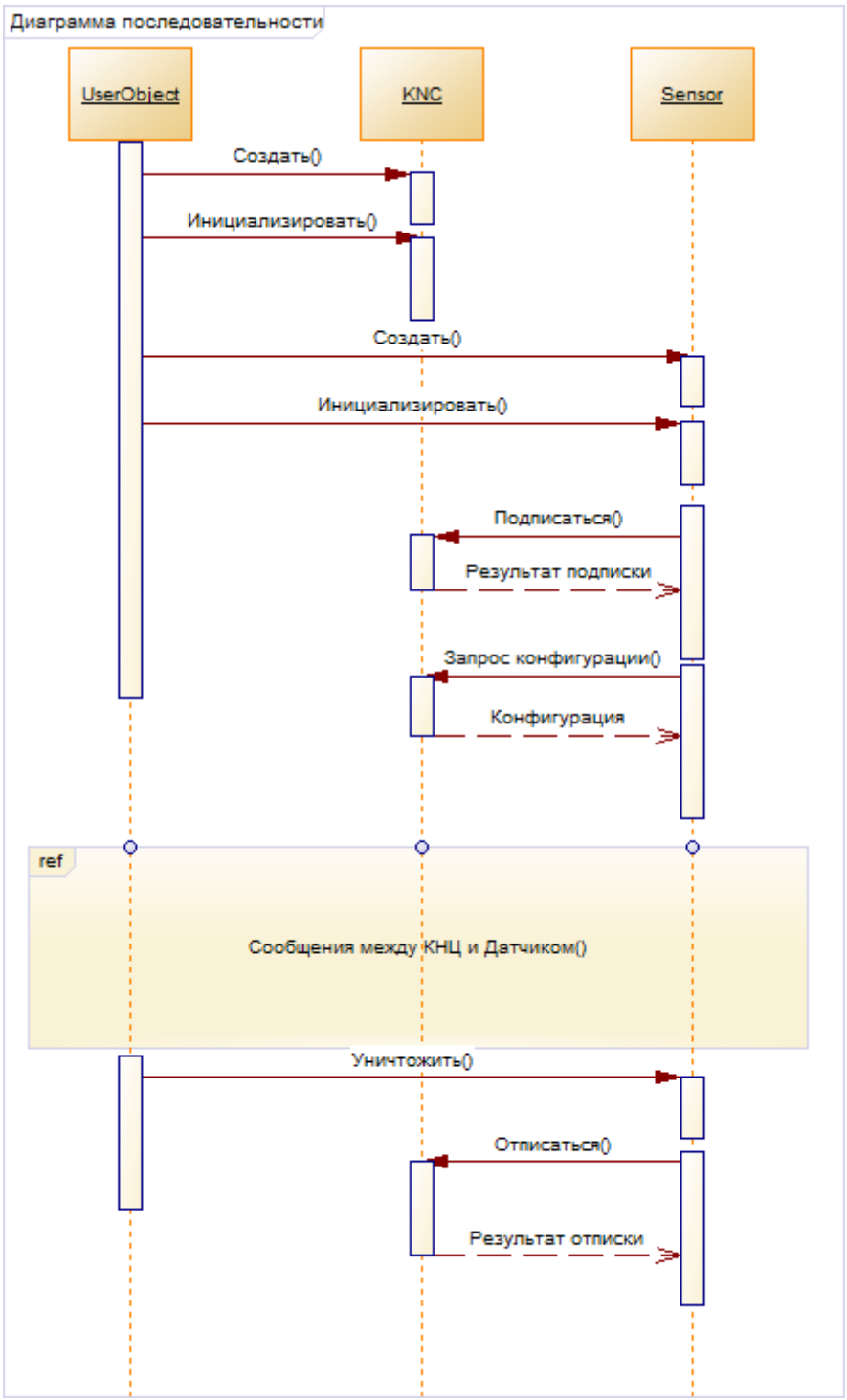

Рисунок 7 - Основная диаграмма последовательности взаимодействия

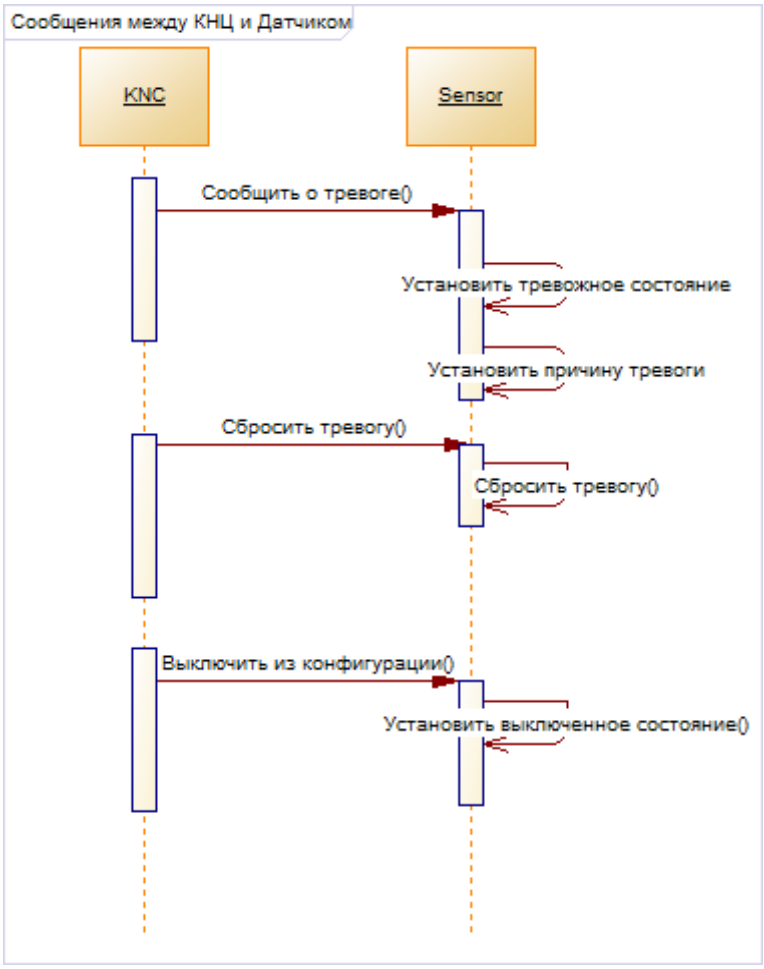

Рисунок 8 - Пример обмена сообщениями объектов КНЦ и Датчика

4.4 Проектирование сценария тестирования модуля управления датчиком

Проектирование тестового сценария состоит из следующих этапов:

а) Определение тестовых воздействий и ожидаемых реакций на эти воздействия (см. разделы 4.4.3 и 4.4.4);

б) Определение Mock-объектов и требований к ним (см. раздел 4.4.1);

в) Определение фикстур тестирования (см. раздел 4.4.2);

г) Определение схемы тестирования (см. раздел 4.4.5).

4.4.1 Mock-объекты

Для тестирования необходим имитатор работы класса КНЦ, который должен реализовать подписку объекта Датчика к объекту КНЦ.

4.4.2 Описание тестовых фикстур

а) Фикстуры уровня класса:

1) Метод, вызываемый перед началом тестирования должен создать объект имитатора КНЦ (и проинициализировать) и объект Датчика;
2) Метод, вызываемый после тестирования должен выгрузить объект имитатора КНЦ.

б) Фикстуры уровня методов:

1) До выполнения каждого тестового метода необходимо проинициализировать объект Датчика;
2) После выполнения каждого тестового метода необходимо выгрузить объект Датчика.

4.4.3 Тестирование загрузки и выгрузки объекта класса датчика

Тестирование загрузки

Необходимые действия для воспроизведения теста:

а) Создать объект Датчика;

б) Инициализировать объект датчика.

Ожидаемая реакция: объект Датчика должен перейти в состояние по загрузке: Загружен.

Тестирование выгрузки

Необходимые действия для воспроизведения теста:

а) Создать объект Датчика;

б) Инициализировать объект датчика;

в) Вызвать метод выгрузки объекта датчика.

Ожидаемая реакция: объект Датчика должен перейти в состояние по загрузке: Выгружен.

4.4.4 Тестирование реакции на поступление тревоги

а) Тест реакции датчика на поступление тревожного состояния в случае, если он готов обработать тревоги.

Необходимые действия для воспроизведения теста:

1) Создать объект Датчика и КНЦ;
2) Инициализировать объект Датчика;
3) Включить датчик в конфигурацию;
4) Подать тревогу (по причине КЗ).

Ожидаемая реакция: объект Датчика должен перейти в состояние по тревоге: Тревога, состояние по диагностике: КЗ.

б) Тест реакции датчика на поступление тревожного состояния в случае, если он блокирован.

Необходимые действия для воспроизведения теста:

1) Создать объект Датчика и КНЦ;
2) Инициализировать объект Датчика;
3) Включить датчик в конфигурацию;
4) Установить состояние по тревоге: Блокировка;
5) Подать тревогу (по причине КЗ).

Ожидаемая реакция: объект Датчика не должен поменять состояние по тревоге, состояние по диагностике: Норма.

в) Тест реакции датчика на поступление тревожного состояния в случае, если он выключен из конфигурации (такой сценарий возможен при сбоях в работе аппаратуры и/или программной системы)

Необходимые действия для воспроизведения теста:

1) Создать объект Датчика и КНЦ;
2) Инициализировать объект Датчика;
3) Выключить датчик из конфигурации;
4) Подать тревогу (по причине КЗ).

Ожидаемая реакция: объект Датчика не должен поменять состояния.

4.4.5 UML диаграммы

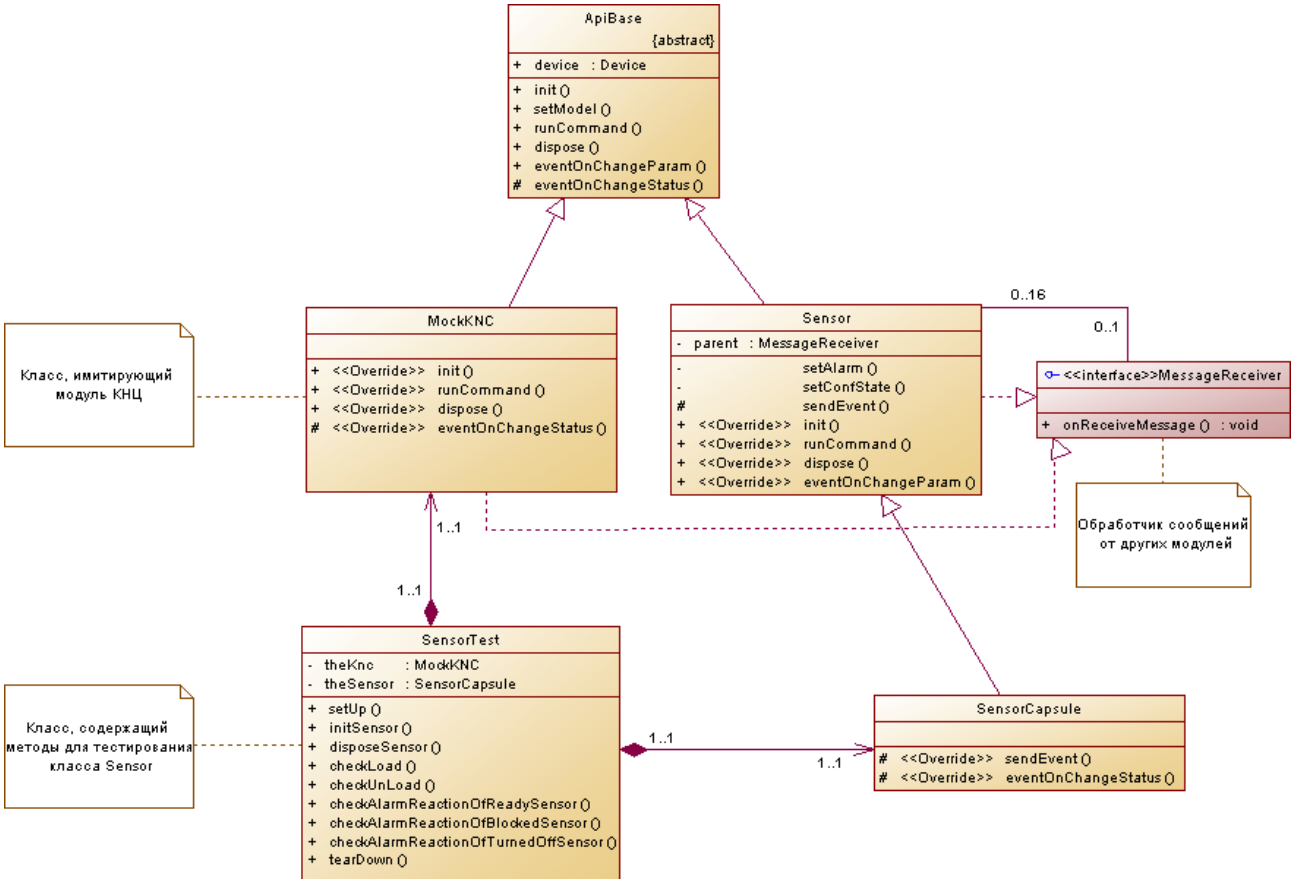

Рисунок 9 - Статическая диаграмма классов, участвующих в тестировании

4.5 Программная реализация

Программная реализация кода, управляющего датчиком, приведена в Приложении А. Программная реализация тестового сценария приведена в Приложении Б.

4.6 Результаты модульного тестирования

Сценарий тестирования запускается на исполнение в среде разработки. После запуска последовательно исполняются все тестовые методы. В случае возникновения результатов, отличающихся от ожидаемых, в среде разработки будет отображен проваленный тест и другая полезная информация. В случае успешного прохождения всех тестов среда разработки сообщает «All Tests Passed» и про-

цесс, выполнивший тесты, возвращает код 0 в операционную систему (см. рисунок 10).

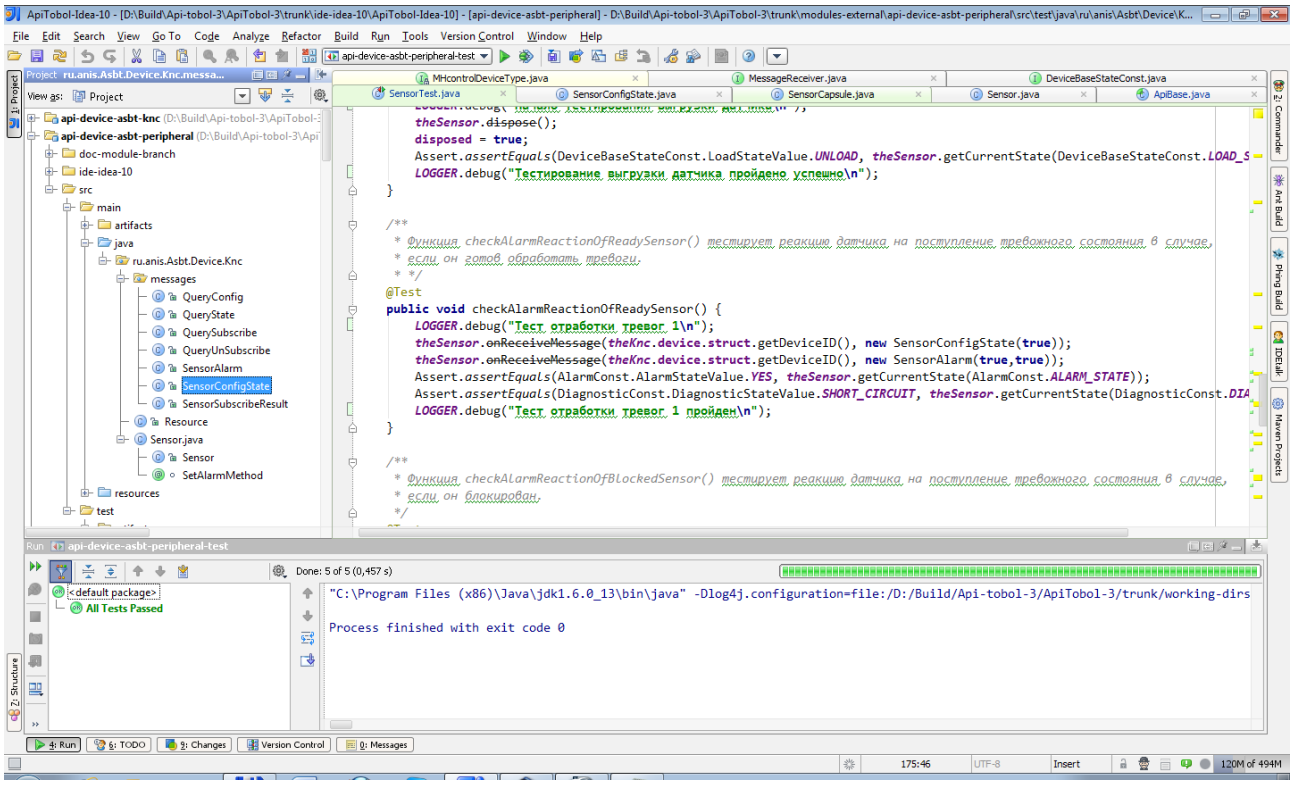

Рисунок 10 - Результат запуска тестов

Как уже было отмечено ранее, в системе ведется журналирование. В файле журнала можно посмотреть подробности процесса тестирования. Содержимое файла после запуска тестового сценария приведено в Приложении В.

5 Заключение

В рамках данной учебно-исследовательской работы изучены:

- Язык программирования Java и основные библиотеки;
- Среда разработки, система контроля версий SVN, система автоматизированной сборки программ;
- Единый протокол обмена информацией с ССОИ;
- Протокол обмена с КНЦ;
- Требования к разработке программных модулей АПИ;
- Исходный код ядра АПИ.

На основе изученных материалов и практических навыков спроектированы и реализованы:

- Программный модуль, управляющий охранной сигнализацией;
- Сценарий модульного тестирования базовых функций управления датчиками.

По результатам тестирования можно сказать, что базовый функционал работает верно.

Список использованных источников

1) СТО 1.1.1.04.007.0814-2009. Единый унифицированный протокол обмена верхнего уровня [Текст].- Введ. 2009
2) Протокол обмена КНЦ.- согл. 17.05.2013.- 12 стр.
3) Java [Электронный ресурс]. – Режим доступа : интернет : http://ru.wikipedia.org/wiki/Java. Дата обращения 21.05.2013
4) Гловер Э. Переходим на JUnit 4 [Электронный ресурс]. – Режим доступа : интернет : http://www.ibm.com/developerworks/ru/edu/j-junit4/section4.html. Дата обращения 21.05.2013
5) Принцип работы логеров и аппендеров [Электронный ресурс]. – Режим доступа : интернет : http://www.log4j.ru/articles/UmlExample.html. Дата обращения 21.05.2013
6) Кунг С. , Онкен Л., Ларге С.. TortoiseSVN: Клиент Subversion для Windows: Версия 1.6.3.- 08.06.2009.- 231 стр.

ПРИЛОЖЕНИЕ А

Исходный код класса Sensor

```
package ru.anis.Asbt.Device.Knc;
import org.apache.log4j.Logger;
import ru.anis.Algont.Commands.CPconfirmationType;
import ru.anis.Algont.Commands.CommandStatus;
import ru.anis.Algont.Commands.MHcontrolDeviceType;
import ru.anis.ApiTobol.ApiBase;
import ru.anis.ApiTobol.ApiConfigurator;
import ru.anis.ApiTobol.ApiDataSet.Model;
import ru.anis.ApiTobol.ApiDataSet.Param;
import ru.anis.ApiTobol.ApiServer;
import ru.anis.ApiTobol.CommonConst.DiagnosticConst;
import ru.anis.ApiTobol.CommonConst.SlaveConst;
import ru.anis.ApiTobol.Device.Device;
import ru.anis.ApiTobol.Messaging.MessageReceiver;
import ru.anis.ApiTobol.Test.RunTime;
import ru.anis.Asbt.Device.Knc.messages.SensorSubscribeResult;
import java.io.Serializable;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.math.BigInteger;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface SetAlarmMethod {
}
/**
 * Класс Sensor
 * @author Barbashov Pavel
 */
public class Sensor extends ApiBase implements DiagnosticConst, MessageReceiver {
    /*Consts*/
    private static final Logger LOGGER = Logger.getLogger(Sensor.class);
    private static final String PARAM_AFTER_LOAD_BLOCK = "Блокировка тревоги после загрузки";
    private static final String PARAM_SENSOR_ADDRESS = "Адрес";
    private static final String SINGLE_TEXT = "single";
    private static final String FALSE_TEXT = "False";

    private static final String COMMAND_DEV_STATE_QUERY = "Запрос состояния устройства";
    private static final String COMMAND_SET_GUARD = "Постановка устройства под охрану";
    private static final String COMMAND_UNSET_GUARD = "Снятие устройства с охраны";
    private static final String COMMAND_BLOCK_ALARM = "Блокировать";
```

```
private static final String COMMAND_UNBLOCK_ALARM = "Разблокировать";
private static final String CONFIGURATION_ERROR_EVENT = "Ошибка конфигурации";
private final ru.anis.Asbt.Device.Knc.messages.QueryConfig
      queryConfig = new ru.anis.Asbt.Device.Knc.messages.QueryConfig();
/*Fields*/
//private byte SensorAddr;
protected MessageReceiver parent = null;

/*Constructors*/
public Sensor(final Device devStruct) {
   super(devStruct);
   setCurrentState(FUNCTION_STATE, FunctionStateValue.PLAY_ON);
   setCurrentState(LOAD_STATE, LoadStateValue.LOADING);
   setCurrentState(GUARD_STATE, GuardStateValue.UNKNOWN);
   setCurrentState(ALARM_STATE, AlarmStateValue.UNKNOWN);
   setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.UNKNOWN);
   LOGGER.debug("Объект класса датчика создан с идентификатором " + device.struct.getDeviceID());
}

/*Private Methods*/
/**
 * Метод readyToHandleAlarm
 *
 * @return true, если модуль готов обработать тревоги от устройства
 *         false, если модуль не готов обработать тревоги от устройства
 */
private boolean readyToHandleAlarm() {
   return !getCurrentState(ALARM_STATE).equals(AlarmStateValue.BLOCKING) &&
         getCurrentState(FUNCTION_STATE).equals(FunctionStateValue.CORRECT) &&
         getCurrentState(GUARD_STATE).equals(GuardStateValue.YES) &&
         !getCurrentState(DIAGNOSTIC_STATE).equals(DiagnosticStateValue.UNKNOWN) &&
         getCurrentState(LOAD_STATE).equals(LoadStateValue.LOAD);
}
/**
 * Метод getParent
 *
 * @return ссылка на родительское устройство
 */
protected MessageReceiver getParent() {
   if (parent == null){
         ApiBase msrc = ApiConfigurator.getDevice(device.struct.getParentID());
         if(msrc instanceof MessageReceiver){
            parent = (MessageReceiver) msrc;
         } else {
            LOGGER.error(" ");
         }
```

```
    }
    return parent;
}

private void setOffState() {
    setCurrentState(FUNCTION_STATE, FunctionStateValue.OFF);
    setCurrentState(LOAD_STATE, LoadStateValue.LOAD);
    setCurrentState(GUARD_STATE, GuardStateValue.YES);
    setCurrentState(ALARM_STATE, AlarmStateValue.UNKNOWN);
    setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.UNKNOWN);
}
private String CurrentStatesToString(){
    return FUNCTION_STATE + ": " +getCurrentState(FUNCTION_STATE) + "\n" +
            LOAD_STATE  + ": " +getCurrentState(LOAD_STATE) + "\n" +
            GUARD_STATE  + ": " +getCurrentState(GUARD_STATE) + "\n" +
            ALARM_STATE   + ": " +getCurrentState(ALARM_STATE) + "\n" +
            DIAGNOSTIC_STATE  + ": " +getCurrentState(DIAGNOSTIC_STATE);
}
protected void sendEvent(String event){
    ApiServer.sendMessage(formatEvent(event));
}

/*Interface*/
/**
 * Метод init() инициализирует объект датчика
 * @param id - идентификатор устройства для которого вызвана инициализация
 * @throws Exception
 */
@Override
public void init(BigInteger id) throws Exception {
    LOGGER.debug("Начинается инициализация датчика с идентификатором " + device.struct.getDeviceID());
    LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " загружен");
    int newval;
    try{
        newval = Integer.valueOf(getCurrentParam(PARAM_SENSOR_ADDRESS).getValue());
        getParent().onReceiveMessage(device.struct.getDeviceID(),new ru.anis.Asbt.Device.Knc.messages.QuerySubscribe(newval));
    }
    catch (NumberFormatException e){
        LOGGER.warn("Неверный формат параметра!");
    }
}

/**
 * Метод  setModel() загружает модель в АПИ
 * @param model
 */
```

```
public static void setModel(final ru.anis.ApiTobol.ApiDataSet.Model model) {
    LOGGER.debug("Заполнение модели датчика");
    /*Заполнение параметров модели*/
    model.setParamArray(new Param[]{
        checkParamInModel(
            new String[]{
                PARAM_SENSOR_ADDRESS, SINGLE_TEXT, "1", FALSE_TEXT
            },
            new String[]{
                "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16"
            }
        ),
        checkParamInModel(
            new String[]{
                PARAM_AFTER_LOAD_BLOCK, SINGLE_TEXT, SlaveConst.OFF, FALSE_TEXT
            },
            null
        )
    });

    /*Заполнение событий модели*/
    model.setEventArray(
        checkEventInModel(
            model.getEventArray(),
            new String[][]{
                {
                    ALARM_EVENT, ALARM_EVENT, "0"
                },
                {
                    NO_ALARM_EVENT, NO_ALARM_EVENT, "0"
                },
                {
                    CONFIGURATION_ERROR_EVENT, CONFIGURATION_ERROR_EVENT, "0"
                }
            }
        )
    );

    /*Заполнение статусов состояний*/
    model.setStateArray(new Model.State[]
        {
            // Типовой статус загрузки
            getLoadState(),
            // Типовой статус функционирования
            getFunctionState(),
```

```
                checkStateInModel(GUARD_STATE,
                        "guard",
                        new String[]
                                {
                                        GuardStateValue.YES,
                                        GuardStateValue.NO,
                                        GuardStateValue.PLAY,
                                        GuardStateValue.UNKNOWN
                                }),

                checkStateInModel(ALARM_STATE,
                        "alarm",
                        new String[]
                                {
                                        AlarmStateValue.YES,
                                        AlarmStateValue.NO,
                                        AlarmStateValue.BLOCKING,
                                        AlarmStateValue.UNKNOWN
                                }),
                checkStateInModel(DIAGNOSTIC_STATE,
                        "diagnostic",
                        new String[]
                                {
                                        DiagnosticStateValue.NORM,
                                        DiagnosticStateValue.SHORT_CIRCUIT,
                                        DiagnosticStateValue.TEAR_OFF,
                                        DiagnosticStateValue.UNKNOWN
                                })
        });
    /*Заполнение команд*/
    model.setCommandArray(new Model.Command[]{
            checkComandInModel(new String[]{"StateQuery", COMMAND_DEV_STATE_QUERY, null}, null),
            checkComandInModel(new String[]{"SetGuard", COMMAND_SET_GUARD, null}, null),
            checkComandInModel(new String[]{"UnsetGuard", COMMAND_UNSET_GUARD, null}, null),
            checkComandInModel(new String[]{"BlockAlarm", COMMAND_BLOCK_ALARM, null}, null),
            checkComandInModel(new String[]{"UnblockAlarm", COMMAND_UNBLOCK_ALARM, null}, null)
    });
    LOGGER.debug("Завершение заполнения модели датчика");
}
/**
 * Функция runCommand() реализует реакцию на команды с ССОИ
 * @param mhdeviceControlType структура с командой и ее параметрами
 * @return
 */
@Override
public CPconfirmationType runCommand(MHcontrolDeviceType mhdeviceControlType) {
```

```
    String cmd = mhdeviceControlType.getCmdId();
    if (cmd.equals(COMMAND_DEV_STATE_QUERY)) {
        LOGGER.info("Датчик с идентификатором "+ device.struct.getDeviceID() +
                                    " получил команду " + COMMAND_DEV_STATE_QUERY);
        getParent().onReceiveMessage(device.struct.getDeviceID(), queryConfig);

    } else if (cmd.equals(COMMAND_SET_GUARD)) {
        LOGGER.warn("Датчик с идентификатором "+ device.struct.getDeviceID() +
                " получил не поддерживаемую команду " + COMMAND_SET_GUARD);
    } else if (cmd.equals(COMMAND_UNSET_GUARD)) {
        LOGGER.warn("Датчик с идентификатором "+ device.struct.getDeviceID() +
                " получил не поддерживаемую команду " + COMMAND_UNSET_GUARD);
        /*Не поддерживаемая команда*/
    } else if (cmd.equals(COMMAND_BLOCK_ALARM)) {
        LOGGER.info("Датчик с идентификатором "+ device.struct.getDeviceID() +
                                    " получил команду блокировки тревог");
        setCurrentState(ALARM_STATE, AlarmStateValue.BLOCKING);
    } else if (cmd.equals(COMMAND_UNBLOCK_ALARM)) {
        LOGGER.info("Датчик с идентификатором "+ device.struct.getDeviceID() +
                                    " получил команду разблокировки тревог");
        setCurrentState(ALARM_STATE, AlarmStateValue.NO);
    } else {
        LOGGER.warn("Датчик с идентификатором "+ device.struct.getDeviceID() +
                " получил не поддерживаемую команду");
    }

    if (getCurrentState(FUNCTION_STATE).equals(FunctionStateValue.CORRECT)){
        return super.runCommand(mhdeviceControlType, CommandStatus.OK);
    }
    return super.runCommand(mhdeviceControlType, CommandStatus.NO_CONNECTION);
}
/**
 * Метод dispose() выгружает датчик из АПИ
 */
@Override
public void dispose() {
   // try {
        getParent().onReceiveMessage(device.struct.getDeviceID(), new ru.anis.Asbt.Device.Knc.messages.QueryUnSubscribe());
        LOGGER.debug("Датчик с идентификатором " + device.struct.getDeviceID() +
                " отписался от устройства с идентификатором " + device.struct.getParentID());
   /* } catch (SubscriptionException e) {
        LOGGER.error("Датчик с идентификатором " + device.struct.getDeviceID() +
                " не смог отписаться от устройства с идентификатором " + device.struct.getParentID());

    } finally {
    */
```

```
        super.dispose();
        LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " выгружен");
    }
    /**
     * Метод setAlarm: устанавливает или сбрасывает состояние "Тревога", генерирует событие "Тревога" или "Нет тревоги",
     * В случае тревоги устанавливает состояние "по диагностике" в КЗ или Обрыв
     * В случае сброса тревоги устанавливает состояние "по диагностике" в Норма
     * <p/>
     * Если по каким-то причинам тревога не может быть обработана (см. фунцкию readyToHandleAlarm()),
     * то пишется соответствующее сообщение в лог
     *
     * @param isAlarm   == true - Тревога от датчика
     *                  == false - Нет тревоги
     * @param alarmType == true - КЗ
     *                  == false - Обрыв
     */
    @SetAlarmMethod
    private void setAlarm(boolean isAlarm, boolean alarmType) {
       boolean readyToHandleAlarm = readyToHandleAlarm();
       if (isAlarm && readyToHandleAlarm) {

          setCurrentState(ALARM_STATE, AlarmStateValue.YES);
          if (alarmType == true) {
             setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.SHORT_CIRCUIT);
             LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " в тревожном состоянии(КЗ)");
          } else {
             setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.TEAR_OFF);
             LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " в тревожном состоянии(Обрыв)");
          }
          sendEvent(ALARM_EVENT);
       } else if (!isAlarm && readyToHandleAlarm) {
          setCurrentState(ALARM_STATE, AlarmStateValue.NO);
          setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.NORM);
          sendEvent(NO_ALARM_EVENT);
          LOGGER.info("Пропадание тревоги на датчике с идентификатором " + device.struct.getDeviceID());
       } else if (isAlarm){
          LOGGER.info("Тревога с датчика "  + device.struct.getDeviceID() +  " не может быть обработана.");
          LOGGER.info("Текущие состояния:" + CurrentStatesToString());
       } else {
          LOGGER.error("Пропадание тревога на датчике "  + device.struct.getDeviceID() +  " не может быть обработано.");
          LOGGER.error("Текущие состояния:" + CurrentStatesToString());
       }
    }
    /**
     * Метод setConfState обрабатывает информацию о включении и выключении датчика. Устанавливает соответствующее состояние.
     *
```

```
     * @param isCorrect == true, если пришла команда, что датчик включен аппаратно (исправен)
     *                  == false, если пришла команда, что датчик выключен аппаратно
     */
    private void setConfState(boolean isCorrect) {
        if (isCorrect) {
            LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " включили в конфигурацию");
            setCurrentState(FUNCTION_STATE, FunctionStateValue.CORRECT);
            setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.NORM);
        } else {
            LOGGER.info("Датчик с идентификатором " + device.struct.getDeviceID() + " выключили из конфигурации");
            setCurrentState(FUNCTION_STATE, FunctionStateValue.OFF);
            setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.UNKNOWN);
        }
    }
    /**
     * Функция onReceiveMessage() реализует обработку сообщений от устройств, на которые подписан объект датчика
     * @param id - идентификатор источника сообщения
     * @param msg - сообщение
     */
    @Override
    public void onReceiveMessage(Serializable id, Object msg) {
        LOGGER.debug("Пришло сообщение от концентратора");
        if (device.struct.getParentID().equals(id)) {
            if (msg instanceof ru.anis.Asbt.Device.Knc.messages.SensorAlarm) {
                setAlarm(((ru.anis.Asbt.Device.Knc.messages.SensorAlarm) msg).isAlarm(), ((ru.anis.Asbt.Device.Knc.messages.SensorAlarm) msg
).getAlarmType());
            }
            else if (msg instanceof ru.anis.Asbt.Device.Knc.messages.SensorConfigState) {
                setConfState(((ru.anis.Asbt.Device.Knc.messages.SensorConfigState) msg).isOn());
            }
            else if (msg instanceof SensorSubscribeResult) {
                if (((SensorSubscribeResult) msg).returnCode() == 0){  /*Успешная подписка*/
                    LOGGER.debug("Успешная подписка датчика с идентификатором " + device.struct.getDeviceID() +
                            " к устройству с идентификатором " + device.struct.getParentID());
                    /*Успешно подписались поэтому устанваливаем следующие статусы: */
                    setCurrentState(FUNCTION_STATE, FunctionStateValue.PLAY_ON);
                    setCurrentState(LOAD_STATE, LoadStateValue.LOAD);
                    setCurrentState(GUARD_STATE, GuardStateValue.YES);
                    setCurrentState(ALARM_STATE, AlarmStateValue.UNKNOWN);
                    setCurrentState(DIAGNOSTIC_STATE, DiagnosticStateValue.UNKNOWN);
                    LOGGER.debug("Датчик с идентификатором "+ device.struct.getDeviceID() + " запросил свою конфигурацию");
                    getParent().onReceiveMessage(device.struct.getDeviceID(), new ru.anis.Asbt.Device.Knc.messages.QueryConfig());
                }
                else if (((SensorSubscribeResult) msg).returnCode() == 1){ /*Не успешная подписка к КНЦ*/
                    setOffState();
                    LOGGER.error("Ошибка подписки датчика с идентификатором " + device.struct.getDeviceID() +
```

```
                    " к устройству с идентификатором " + device.struct.getParentID()+" адрес занят.");
            }
            else {
                setOffState();
                LOGGER.error("Неизвестная ошибка подписки датчика с идентификатором " + device.struct.getDeviceID() +
                        " к устройству с идентификатором " + device.struct.getParentID());
            }
        }
        else {
            LOGGER.error("Не поддерживаемое сообщение от концентратора");
        }
    } else {
        LOGGER.warn("Получено сообщение не от родительского устройства");
    }
}
/**
 * Метод eventOnChangeParam() обрабатывает изменение параметров. Изменить можно только параметр "Адрес датчика"
 * @param paramName  название
 * @param oldValue старое значение
 * @param newValue новое значение
 */
@Override
public void eventOnChangeParam(String paramName, String oldValue, String newValue) {
    if (paramName.equals(PARAM_SENSOR_ADDRESS)) {
        int newval;
        try{
            newval = Integer.valueOf(newValue);
            if (setCurrentParam(paramName, newValue)) {
                LOGGER.info("Адрес датчика с идентификатором " + device.struct.getDeviceID() + " сменился с " +
                        oldValue + " на " + newValue);
                /*Отписка*/
                getParent().onReceiveMessage(device.struct.getDeviceID(), new ru.anis.Asbt.Device.Knc.messages.QueryUnSubscribe());
                setOffState();
                /*Подписка*/
                getParent().onReceiveMessage(device.struct.getDeviceID(), new ru.anis.Asbt.Device.Knc.messages.QuerySubscribe(newval));
            } else {
                LOGGER.warn("Ошибка смены адреса датчика с идентификатором "+ device.struct.getDeviceID());
            }
        }
        catch(NumberFormatException e){
             LOGGER.warn(e);
        }

    } else {
        LOGGER.info("Попытка изменения параметра "+ paramName +" у датчика с идентификатором " + device.struct.getDeviceID());
    }}}
```

ПРИЛОЖЕНИЕ Б

Исходный код сценария тестирования

```
package ru.anis.Asbt.Device.Knc;
import org.apache.log4j.Logger;
import org.junit.*;
import ru.anis.Algont.Commands.MHcontrolDeviceType;
import ru.anis.ApiTobol.ApiDataSet.ApiDataSetDocument;
import ru.anis.ApiTobol.ApiDataSet.Model;
import ru.anis.ApiTobol.CommonConst.AlarmConst;
import ru.anis.ApiTobol.CommonConst.DeviceBaseStateConst;
import ru.anis.ApiTobol.CommonConst.DiagnosticConst;
import ru.anis.Asbt.Device.Knc.messages.SensorAlarm;
import ru.anis.Asbt.Device.Knc.messages.SensorConfigState;
import java.math.BigInteger;
/**
 * Created by IntelliJ IDEA.
 * User: Barb
 * Date: 14.05.13
 * Time: 14:10
 * Класс SensorTest содержит тесты датчика КНЦ
 * @author Barbashov Pavel
 */
public class SensorTest {
    private static final Logger LOGGER = Logger.getLogger(SensorTest.class);
    private static Sensor theSensor;
    private static ru.anis.ApiTobol.ApiDataSet.ApiDataSetDocument config = null;
    private static MockKnc16 theKnc;
    private static final String xmlStr= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
            "<ApiDataSet xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xsi:noNamespaceSchemaLocation=\"ApiDataSet_2008.xsd\">\n" +
            "    <LoadConfig producer=\"ФГУП СНПО Элерон\" objectId=\"44073070852589891594474271620633941051\">\n" +
            "<Model _type_=\"ru.anis.Asbt.Device.Knc.Knc16\"\n" +
            "type=\"КНЦ16\"\n" +
            "model=\"1.0\">\n" +
            "<ChildModel _type_=\"ru.anis.Asbt.Device.Knc.Sensor\"\n" +
            "type=\"Датчик КНЦ\"\n" +
            "model=\"1.0\">\n" +
            "</ChildModel>\n" +
            "</Model>\n" +
            "</LoadConfig>\n" +
            "</ApiDataSet>";
    private boolean disposed;
    private static final String knc16Id = "1";
    private static final String sensorId = "2";
```

```
/**
 * Функция setUp() инициализирует все, что нужно для проведения тестирования
 * @throws Exception
 */

@BeforeClass
public static void setUp() throws Exception {
    LOGGER.debug("Начало тестирования модуля датчика\n");
    LOGGER.debug("Начало инициализации тестовой фикстуры\n");
    config = ApiDataSetDocument.Factory.parse(xmlStr);

    Model sensorModel = config.getApiDataSet().getLoadConfig().getModel().getChildModelArray(0);
    Model kncModel = config.getApiDataSet().getLoadConfig().getModel();

    Sensor.setModel(sensorModel);
    MockKnc16.setModel(kncModel);

    ru.anis.ApiTobol.ApiDataSet.Object sensorObject = ru.anis.ApiTobol.ApiDataSet.Object.Factory.newInstance();
    sensorObject.setDeviceID(new BigInteger(sensorId));
    sensorObject.setName("Датчик КНЦ");
    sensorObject.setParentID(new BigInteger(knc16Id));
    sensorObject.addNewCurentParam().setParam("Адрес");

    ru.anis.ApiTobol.Device.Device devInfoSensor = new ru.anis.ApiTobol.Device.Device();
    devInfoSensor.model = sensorModel;
    devInfoSensor.struct = sensorObject;

    ru.anis.ApiTobol.ApiDataSet.Object knc16Object = ru.anis.ApiTobol.ApiDataSet.Object.Factory.newInstance();
    knc16Object.setDeviceID(new BigInteger(knc16Id));
    knc16Object.setName("КНЦ16");
    knc16Object.setParentID(new BigInteger("0"));
    ru.anis.ApiTobol.Device.Device devInfoKnc = new ru.anis.ApiTobol.Device.Device();
    devInfoKnc.model = kncModel;
    devInfoKnc.struct = knc16Object;

    theKnc = new MockKnc16(devInfoKnc);
    theSensor = new SensorCapsule(devInfoSensor,theKnc);

    ((SensorCapsule)theSensor).setCurrentParam("Адрес", "1");

  //  System.out.print(theSensor.getCurrentParam("Адрес").getValue());

    LOGGER.debug("Окончание инициализации тестовой фикстуры\n");
}

/**
```

```
 * Функция initSensor() инициализирует объект датчика перед каждого теста
 */
@Before
public void initSensor() {
    try {
        theSensor.init(new BigInteger(sensorId));
        disposed = false;
    } catch (Exception e) {
        disposed = true;
        e.printStackTrace();
        Assert.fail("Ошибка загрузки датчика");
    }
}

/**
 * Функция disposeSensor() выгружает объект датчика после каждого теста
 */
@After
public void disposeSensor() {
    if (!disposed){
        theSensor.dispose();
        disposed = true;
    }
}

/**
 * Функция checkLoad() тестирует загрузку объекта датчика
 */
@Test
public void checkLoad() {
    LOGGER.debug("Начало тестирования загрузки датчика\n");
    Assert.assertEquals(DeviceBaseStateConst.LoadStateValue.LOAD, theSensor.getCurrentState(DeviceBaseStateConst.LOAD_STATE));
    LOGGER.debug("Тестирование загрузки датчика пройдено успешно\n");
}

/**
 * Функция checkUnLoad() тестирует выгрузку объекта датчика
 */
@Test
public void checkUnLoad() {
    LOGGER.debug("Начало тестирования выгрузки датчика\n");
    theSensor.dispose();
    disposed = true;
    Assert.assertEquals(DeviceBaseStateConst.LoadStateValue.UNLOAD, theSensor.getCurrentState(DeviceBaseStateConst.LOAD_STATE));
    LOGGER.debug("Тестирование выгрузки датчика пройдено успешно\n");
}
```

```
    /**
     * Функция checkAlarmReactionOfReadySensor() тестирует реакцию датчика на поступление тревожного состояния в случае,
     * если он готов обработать тревоги.
     * */
    @Test
    public void checkAlarmReactionOfReadySensor() {
        LOGGER.debug("Тест отработки тревог 1\n");
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorConfigState(true));
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorAlarm(true,true));
        Assert.assertEquals(AlarmConst.AlarmStateValue.YES, theSensor.getCurrentState(AlarmConst.ALARM_STATE));
        Assert.assertEquals(DiagnosticConst.DiagnosticStateValue.SHORT_CIRCUIT,
theSensor.getCurrentState(DiagnosticConst.DIAGNOSTIC_STATE));
        LOGGER.debug("Тест отработки тревог 1 пройден\n");
    }

    /**
     * Функция checkAlarmReactionOfBlockedSensor() тестирует реакцию датчика на поступление тревожного состояния в случае,
     * если он блокирован.
     */
    @Test
    public void checkAlarmReactionOfBlockedSensor() {
        LOGGER.debug("Тест отработки тревог 2\n");
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorConfigState(true));
        MHcontrolDeviceType mHcontrolDeviceType = MHcontrolDeviceType.Factory.newInstance();
        mHcontrolDeviceType.setCmdId("Блокировать");
        theSensor.runCommand(mHcontrolDeviceType);
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorAlarm(true,true));
        Assert.assertEquals(AlarmConst.AlarmStateValue.BLOCKING, theSensor.getCurrentState(AlarmConst.ALARM_STATE));
        Assert.assertEquals(DiagnosticConst.DiagnosticStateValue.NORM, theSensor.getCurrentState(DiagnosticConst.DIAGNOSTIC_STATE));
        LOGGER.debug("Тест отработки тревог 2 пройден\n");
    }

    /**
     * Функция checkAlarmReactionOfTurnedOffSensor() тестирует реакцию датчика на поступление тревожного состояния в случае,
     * если он выключен из конфигурации
     */
    @Test
    public void checkAlarmReactionOfTurnedOffSensor() {
        LOGGER.debug("Тест отработки тревог 3\n");
        String cur_alarm_state = theSensor.getCurrentState(AlarmConst.ALARM_STATE);
        String cur_diagnotic_state = theSensor.getCurrentState(DiagnosticConst.DIAGNOSTIC_STATE);
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorConfigState(false));
        theSensor.onReceiveMessage(theKnc.device.struct.getDeviceID(), new SensorAlarm(true,true));
        Assert.assertEquals(cur_alarm_state, theSensor.getCurrentState(AlarmConst.ALARM_STATE));
        Assert.assertEquals(cur_diagnotic_state, theSensor.getCurrentState(DiagnosticConst.DIAGNOSTIC_STATE));
```

```
        LOGGER.debug("Тест отработки тревог 3 пройден\n");
    }

    /**
     * Функция tearDown() выполняет завершающие действия после завершения тестов
     * @throws Exception
     */
    @AfterClass
    public static void tearDown() throws Exception {
        theKnc.dispose();
        LOGGER.debug("Тестирование модуля датчика КНЦ закончено успешно!\n");
    }

    public static Sensor getTheSensor() {
        return theSensor;
    }
}
```

ПРИЛОЖЕНИЕ В

Журнал после запуска сценария тестирования

2013-05-23 16:32:15,890 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Начало тестирования модуля датчика

2013-05-23 16:32:15,890 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Начало инициализации тестовой фикстуры

2013-05-23 16:32:16,121 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Заполнение модели датчика

2013-05-23 16:32:16,158 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Завершение заполнения модели датчика

2013-05-23 16:32:16,203 DEBUG [ru.anis.Asbt.Device.Knc.MockKnc16] Загружен Mock-объект КНЦ16 с идентификатором 1

2013-05-23 16:32:16,207 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Объект класса датчика создан с идентификатором 2

2013-05-23 16:32:16,208 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Окончание инициализации тестовой фикстуры

2013-05-23 16:32:16,212 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Начинается инициализация датчика с идентификатором 2

2013-05-23 16:32:16,212 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 загружен

2013-05-23 16:32:16,212 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Успешная подписка датчика с идентификатором 2 к устройству с идентификатором 1

2013-05-23 16:32:16,213 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 запросил свою конфигурацию

2013-05-23 16:32:16,213 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Начало тестирования загрузки датчика

2013-05-23 16:32:16,213 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тестирование загрузки датчика пройдено успешно

2013-05-23 16:32:16,214 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 отписался от устройства с идентификатором 1

2013-05-23 16:32:16,214 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выгружен

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Начинается инициализация датчика с идентификатором 2

2013-05-23 16:32:16,216 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 загружен

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Успешная подписка датчика с идентификатором 2 к устройству с идентификатором 1

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 запросил свою конфигурацию

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Начало тестирования выгрузки датчика

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 отписался от устройства с идентификатором 1

2013-05-23 16:32:16,216 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выгружен

2013-05-23 16:32:16,216 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тестирование выгрузки датчика пройдено успешно

2013-05-23 16:32:16,217 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Начинается инициализация датчика с идентификатором 2

2013-05-23 16:32:16,217 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 загружен

2013-05-23 16:32:16,217 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Успешная подписка датчика с идентификатором 2 к устройству с идентификатором 1

2013-05-23 16:32:16,217 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 запросил свою конфигурацию

2013-05-23 16:32:16,217 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 1

2013-05-23 16:32:16,217 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,218 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 включили в конфигурацию

2013-05-23 16:32:16,218 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,218 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 в тревожном состоянии(КЗ)

2013-05-23 16:32:16,218 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 1 пройден

2013-05-23 16:32:16,218 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 отписался от устройства с идентификатором 1

2013-05-23 16:32:16,218 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выгружен

2013-05-23 16:32:16,219 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Начинается инициализация датчика с идентификатором 2

2013-05-23 16:32:16,219 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 загружен

2013-05-23 16:32:16,219 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Успешная подписка датчика с идентификатором 2 к устройству с идентификатором 1

2013-05-23 16:32:16,219 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 запросил свою конфигурацию

2013-05-23 16:32:16,219 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 2

2013-05-23 16:32:16,219 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,219 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 включили в конфигурацию

2013-05-23 16:32:16,229 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 получил команду блокировки тревог

2013-05-23 16:32:16,233 INFO [ru.anis.ApiTobol.ApiBase] Команда "Блокировать" выполнена

2013-05-23 16:32:16,244 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,244 INFO [ru.anis.Asbt.Device.Knc.Sensor] Тревога с датчика 2 не может быть обработана.

2013-05-23 16:32:16,245 INFO [ru.anis.Asbt.Device.Knc.Sensor] Текущие состояния:Функционирование: Исправно
Режим загрузки: Загружено
Охрана: Да
Тревога: Блокировано
По диагностике: Норма

2013-05-23 16:32:16,245 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 2 пройден

2013-05-23 16:32:16,245 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 отписался от устройства с идентификатором 1

2013-05-23 16:32:16,245 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выгружен

2013-05-23 16:32:16,245 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Начинается инициализация датчика с идентификатором 2

2013-05-23 16:32:16,245 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 загружен

2013-05-23 16:32:16,245 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Успешная подписка датчика с идентификатором 2 к устройству с идентификатором 1

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 запросил свою конфигурацию

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 3

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,246 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выключили из конфигурации

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Пришло сообщение от концентратора

2013-05-23 16:32:16,246 INFO [ru.anis.Asbt.Device.Knc.Sensor] Тревога с датчика 2 не может быть обработана.

2013-05-23 16:32:16,246 INFO [ru.anis.Asbt.Device.Knc.Sensor] Текущие состояния:Функционирование: Выключено
Режим загрузки: Загружено
Охрана: Да
Тревога: Не определено
По диагностике: Не определено

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тест отработки тревог 3 пройден

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 отписался от устройства с идентификатором 1

2013-05-23 16:32:16,246 INFO [ru.anis.Asbt.Device.Knc.Sensor] Датчик с идентификатором 2 выгружен

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.MockKnc16] Выгружен Mock-объект КНЦ16 с идентификатором 1

2013-05-23 16:32:16,246 DEBUG [ru.anis.Asbt.Device.Knc.SensorTest] Тестирование модуля датчика КНЦ закончено