Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
Петрозаводский государственный университет
Математический факультет
Кафедра информатики и математического обеспечения

Отчет по дисциплине «Верификация ПО»

Программная система автоматизированной разработки приложений для интеллектуальных пространств на платформе Smart-M3

Выполнил:
студент 6 курса группы 22608 С. А. Марченков

_____
подпись

Лектор:
к.т.н., доцент К. А. Кулаков
Итоговая оценка: _____

_____
подпись

Петрозаводск

2015

Содержание

1 Объект тестирования

SmartSlog [1, 2] – программная система автоматизированной разработки приложений для интеллектуальных пространств на платформе Smart-M3 [3, 4, 5]. По онтологическому описанию (OWL формат) SmartSlog предоставляет библиотеку, которая позволяет писать код процессоров знаний (knowledge processor - KP) в OWL терминах классов, свойств и индивидов. Она имитирует онтологическое описание с помощью локальных структур

данных выбранного языка программирования (ANSI C, C#), а также реализует прикладной интерфейс (API) для взаимодействия KP с интеллектуальным пространством (ИП).

SmartSlog библиотека выступает в качестве высокоуровневого KP интерфейса (KPI), который в отличии от низкоуровневого KPI, оперирующего RDF-триплетами в качестве основных единиц операций взаимодействия KP с ИП, предоставляет более высокоуровневые функции, принимающие в качестве параметров и возвращающие структуры библиотеки SmartSlog.

Рис. 1: Схема использования SmartSlog.

Схема использования SmartSlog для версии C# представлена на рис. 1. SmartSlog предоставляет интерфейс KPI, содержащий более программно-специфический доступ к функциям, реализованных непосредственно в онтологической библиотеки SmartSlog. Также интерфейс содержит описание структур данных, которые используется в библиотеки. Благодаря данному интерфейсу программист может написать свою KPI обертку (KPI Wrapper) для доступа к методам библиотеки, используя любой KPI, содержащий базовые функции протокола SSAP.

Библиотека SmartSlog при тестировании функций, работающих с ИП, будет использовать C# KPI, функции которого также будут протестированы. C#KPI включает: (а) модуль реализации SSAP-клиента, (б) модуль формирования и обработки XML сообщений с RDF-данными. Подробнее план и стратегия тестирования будут рассмотрены ниже.

2 План тестирования

2.1 Описание классов

В API библиотеки SmartSlog будут тестироваться следующие классы:

- Individual: класс для локальной работы с индивидом и его свойствами.

  Участие в блочном тестировании: все представленные функции.

  - Получение всех свойств данных по заданному онтологическому свойству.

    Collection<string> GetDataProperties(Property property);

  - Получение первого найденного свойства данных по заданному онтологическому свойству.

    string GetDataProperty(Property property);

  - Получение всех объектных свойств данных по заданному онтологическому свойству.

    Collection<Individual> GetObjectProperties(Property property);

  - Получение первого найденного объектного свойства данных по заданному онтологическому свойству.

    Individual GetObjectProperty(Property property);

  - Получение всех значений по заданному онтологическому свойству.

    Collection<object> GetProperties(Property property);

  - Получение всех значений свойств.

    Collection<object> GetProperties();

  - Получение первого найденного значения по заданному онтологическому свойству.

    object GetProperty(Property property);

  - Удаление всех значений свойств по заданному онтологическому свойству.

    void RemoveProperties(Property property);

  - Удаление первого найденного значения свойства данных с заданным значением строки по заданному онтологическому свойству.

    void RemoveProperty(Property property, string value);

  - Удаление первого найденного значения объектного свойства с заданным значением индивида по заданному онтологическому свойству.

    void RemoveProperty(Property property, Individual value);

  - Удаление первого найденного значения свойства по заданному онтологическому свойству.

    void RemoveProperty(Property property);

- Установка заданного значения свойства типа строки с проверкой кардинальности по заданному онтологическому свойству.
  void SetProperty(Property property, string value);
- Установка заданного значения свойства типа индивида с проверкой кардинальности по заданному онтологическому свойству.
  void SetProperty(Property property, Individual value);
- Обновление значения свойства данных по заданному онтологическому свойству, необходимы ненулевые значения свойства.
  void UpdateProperty(Property property, string oldValue, string newValue);
- Обновление значения объектного свойства по заданному онтологическому свойству, необходимы ненулевые значения свойства.
  void UpdateProperty(Property property,
  Individual oldValue, Individual newValue);

• Pattern: класс, представляющий шаблон поиска в ИП.
  Участие в блочном тестировании: все представленные функции.

- Добавление проверяемого свойства со значением данных.
  void AddProperty(Property property, string value);
- Добавление проверяемого свойство с объектным значением.
  void AddProperty(Property property, Individual value);
- Добавление проверяемого свойство со шаблоном поиска.
  void AddProperty(Property property, Pattern value);
- Добавление непроверяемого свойство со значением данных.
  void AddUnProperty(Property property, string value);
- Добавление непроверяемого свойство с объектным значением.
  void AddUnProperty(Property property, Individual value);
- Получение всех свойств без значений.
  List<Property> GetOntProperties();
- Получение всех непроверяемых свойств без значений.
  List<Property> GetOntUnProperties();
- Получение всех проверяемых свойств.
  List<object> GetProperties()

- Получение вех значений проверяемых свойств.
  List<object> GetPropertyValues();
- Получение всех непроверяемых свойств.
  List<object> GetUnProperties();
- Получение всех значений непроверяемых свойств.
  List<object> GetUnPropertyValues();
- Удаление проверяемого свойства с заданным значением данных.
  void RemoveProperty(Property property, string value);
- Удаление проверяемого свойства с заданным объектным значением.
  void RemoveProperty(Property property, Individual value);
- Удаление непроверяемого свойства с заданным значением данных.
  void RemoveUnProperty(Property property, string value);
- Удаление непроверяемого свойства с заданным объектным значением.
  void RemoveUnProperty(Property property, Individual value);
- Установка идентификатора шаблону поиска.
  void SetUUID(string uuid);

- OntlogyClass: используется для представления класса сущности онтологии.
  Участие в блочном тестировании: не участвует, так как представляет из себя структуру, имитирующую онтологический класс, с методами добавления и удаления элементов.
  Содержит следующие поля (элементы):
  - Список индивидов.
    List<Individual> individuals;
  - Список супер-классов.
    List<OntologyClass> superClasses;
  - Список свойств.
    private List<Property> properties;

- Property: класс, представляющий свойство из онтологии.
  Участие в блочном тестировании: не участвует, так как представляет из себя структуру, имитирующую свойство онтологии, с методами доступа к элементам и их со-

здания.

Содержит следующие поля (элементы):

  - Тип свойства.
    bool isObject;
  - Максимальная кардинальность.
    int maxCardinality;
  - Минимальная кардинальность.
    int minCardinality;
  - Список свойств родителей.
    List<Property> parentProperties;

- PropertyValue: класс для хранения пар индивид-свойство.
  Участие в блочном тестировании: не участвует, так как представляет из себя структуру, имитирующую свойство со значением, с методами добавления и удаления элементов.
  Содержит следующие поля (элементы):

  - Свойство.
    Property property;
  - Состояние свойства (добавленное, удаленное и т.д.).
    State state;
  - Значение свойства.
    object value;

- Repository: хранит информацию о сущностях. Участие в блочном тестировании: не участвует, так как представляет собой специализированное хранилище всех используемых в библиотеки объектов (сессии, свойства, индивиды и т.д.).
  О его работоспособности можно говорить по результатам других тестов.

- OwlMapping: преобразования между низкоуровневыми RDF тройками и индивидами, свойствами, классами.
  Участие в блочном тестировании: не участвует, так как представляет из себя проверенный ранее класс с методами для преобразования низкоуровневых структур данных (rdf тройки) в высокоуровневые (индвиды, свойства, классы) и обратно.

- Node: связующее звено между ИП и программистом.

  Участие в блочном тестировании: участвует только методы CreateIndividual и CreatePattern, остальные функции будут проверяться в интеграционных тестах при взаимодействии с C# KPI.

  - Создание индивида по заданному онтологическому классу.
    Individual CreateIndividual(OntologyClass ontologyClass);
  - Создание шаблона поиска по заданному онтологическому классу.
    Pattern CreatePattern(OntologyClass ontologyClass);
  - Проверка наличия индивида со свойствами в ИП.
    bool Exist(Individual individual);
  - Получение всех индивидов из ИП по заданному онтологическому классу.
    Collection<Individual> GetIndividuals(OntologyClass ontologyClass);
  - Получение всех индивидов из ИП по заданному шаблону поиска.
    Collection<Individual> GetIndividuals(Pattern pattern);
  - Получения значения первого найденного свойства заданного индивида из ИП.
    object GetProperty(Individual individual, Property property);
  - Вставка индивида со свойствами и их значениями.
    void Insert(Individual individual);
  - Вставка индивида со свойствами, которые указаны в шаблоне поиска.
    void Insert(Individual individual, Pattern pattern);
  - Подключение к ИП.
    void Join();
  - Отключение от ИП.
    void Leave();
  - Заполнить индивид из ИП.
    void Populate(Individual individual);
  - Удалить индивид из ИП.
    void Remove(Individual individual);
  - Удаление свойства индивида из ИП по заданному объектному значению свойства.
    void RemoveProperty(Individual individual, Property property, Individual value);

– Удаление свойства индивида из ИП по заданному значению свойства данных.
  void RemoveProperty(Individual individual, Property property, string value);
– Установка свойства индивида в ИП по заданному объектному значению свойства.
  void SetProperty(Individual individual, Property property, Individual value);
– Установка свойства индивида в ИП по заданному значению свойства данных.
  void SetProperty(Individual individual, Property property, string value);
– Обновление индивида в ИП.
  void Update(Individual individual);
– Обновление индивида в ИП, обновляется только заданные свойства в шаблоне поиска.
  void Update(Individual individual, Pattern pattern);
– Обновление свойства индивида в ИП по заданному объектному значению свойства, необходимо новое и старое значение.
  void UpdateProperty(Individual individual, Property property,
  Individual oldValue, Individual newValue);
– Обновление свойства индивида в ИП по заданному значению свойства данных, необходимо новое и старое значение.
  void UpdateProperty(Individual individual, Property property,
  string oldValue, string newValue);

На рис. 2 представлены связи между классами в библиотеки SmartSlog. Пользователь не может создавать экземпляр класса Individual, такие экземпляры можно получать через класс Node, используя функцию CreateIndividual. В этом классе определены все локальные действия над свойствами индивидов. Класс Node, в свою очередь, является связующим звеном (можно сказать фасадом) между ИП и клиентским программистом. В нем определены функции по регистрации сущностей (классов, свойств), получения нового индивида, работа с удаленными свойствами. Класс Repository хранит информацию о сущностях, является «одиночкой», т.е. существует один экземпляр этого класса и он доступен всем классам системы.

Для имитирования структур данных онтологии используется классы OntlogyClass, Property и PropertyValue. Класс Pattern предоставляет набор функция для создания и использования шаблонов поиска в ИП. Данный класс определяет локальные действия

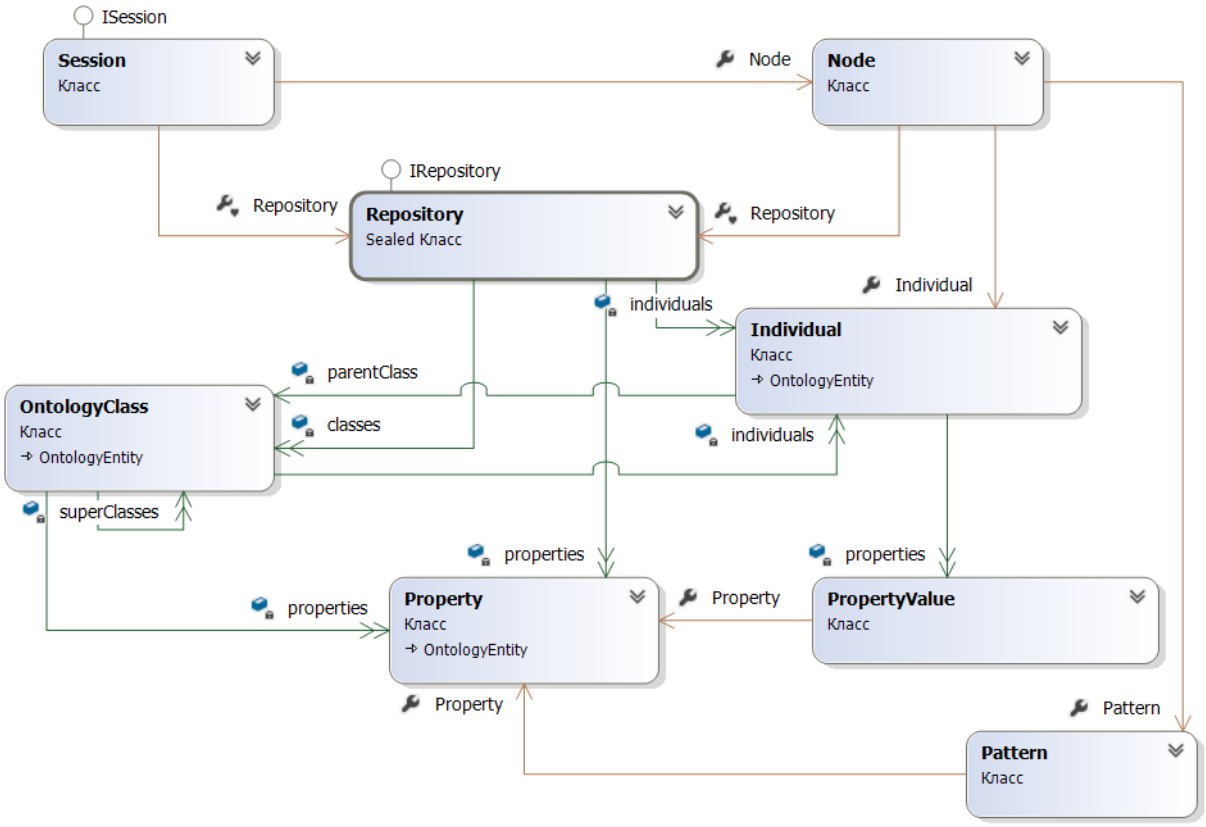

Рис. 2: Диаграмма классов SmartSlog.

над шаблоном поиска, однако через класс Node можно совершать удаленные операции с использованием шаблонов поиска.

Для удаленных операций с ИП SmartSlog использует низкоуровневый C# KPI, который содержит функции для работы с ИП. Функции C# KPI вызываются только в классе Node. C# KPI будет использоваться в интеграционном тестировании, он содержит следующие классы:

- KPICore: определяет основные функции для взаимодействия с ИП.

  Участие в блочном тестировании: не участвует, так как данный класс является сторонним, гарантируется его правильность.

- SSAP_XMLTools: формирование и обработка XML сообщений с RDF-данными.

  Участие в блочном тестировании: не участвует, так как данный класс является сторонним, гарантируется его правильность.

На рис. 3 представлены связи между классами в C# KPI. Класс KPICore является основным, в нем определены все операции протокола взаимодействия С ИП: join, leave, remove, insert, update, query. Данный класс взаимодействует с классом SSAP_XMLTools

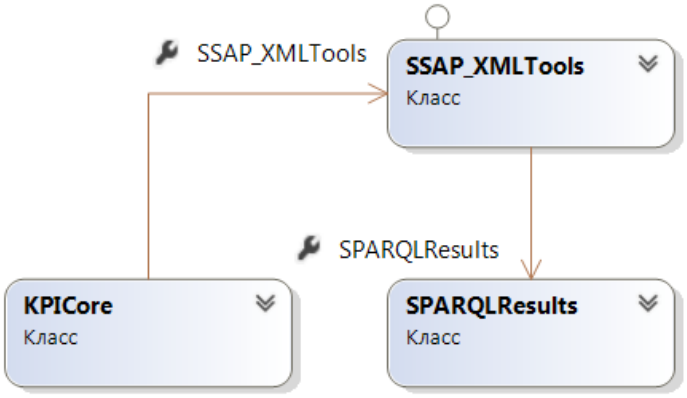

Рис. 3: Диаграмма классов C# KPI.

для формирования и обработки XML сообщений. С помощью методов send и receive класс KPICore выполняет сетевое взаимодействие с ИП, отправляя и получая XML сообщения.

2.2 Стратегия тестирования

Для тестирования языков платформы .NET в основном используются следующие инструменты: NUnit, XUnit, MbUnit. Проведя оценку данных инструментов, выбор сделан был в пользу NUnit для библиотеки C#. На данный момент существует расширение для Visual Studio 2013, что позволяет совмещать разработку программного кода и тестирования в одной среде. Для использования NUnit в проекте необходимо добавить к его решению расширения NUnit и NUnit Test Adapter. После этого, все тесты будут отображаться в самой среде в специальном окне для тестов.

Тестовый класс, используемый данный фреймворк состоит из следующих обязательных атрибутов:

- TestFixture – представляет класс, который содержит набо тест кейсов.
- Test – представляет тест кейс, внутри класса TestFixture.

Также часто используются атрибуты для выполнения некоторых действий до тест кейсов и после:

- TestFixtureSetUp – представляет набор действий, которые выполняется до выполнения всех тест кейсов (один раз).
- SetUp – представляет набор действий, которые выполняются до выполнения каждого тест кейса.

- TestFixtureTearDown – представляет набо действий, которые выполняется после выполнения всех тест кейсов (один раз).
- TearDown – представляет набор действий, которые выполняются после каждого тест кейса.

Таким образом для организации тестирования с использованием NUnit необходимо создать тестовый класс, помеченный атрибутом TestFixture, и набор методов тестового класса, помеченных атрибутом Test, также, по необходимости, задать методы для выполнения действий до и после тест кейсов. Каждый тестовый метод (тест кейс) класса должен иметь идентификатор public и возвращать тип void.

Все тест кейсы в NUnit строятся на так называемых утверждениях, которые обеспечивают проверку поставленных требований. При невыполнении какого-либо утверждения тест кейс считается проваленным. Утверждения представляют собой методы статического класса Assert. С помощью таких методов можно устроить всевозможные проверки тестируемых функции.

Также с помощью атрибута ExpectedException можно задавать ожидаемое исключение для выбранного тест кейса. Это полезно для функций, которые в случаи ошибки не изменяют какие-либо данные или возвращают значение, а генерируют исключения. Тесты с атрибутом ExpectedException будут использоваться.

Для того чтобы использовать методы и атрибуты фреймворка NUnit необходимо подключить его в файл C# класса с помощью команды:

```
using NUnit.Framework;
```

Для проведения блочного тестирования необходимо определить обозначить все возможные входные данные, соответствующие им выходные данные, а также генерируемые исключения. Также в блочном тестировании будет использоваться тестирование с использованием больших данных. Такие тесты будут разработаны для локальных функций класса Individual для работы со свойствами (установка и получение).

На рис. 4 представлена схема взаимодействия классов при выбранной схеме интеграционного тестирования. Для каждого модуля необходимо определить зависимые модули и проверить правильность их взаимодействия. В интеграционном тестирование планируется проверить взаимодействие основных классов при выполнении функций, выполняющих операции с ИП с помощью низкоуровневого C# KPI. Все методы разных модулей будут вызываться последовательно, в порядке выполнения тестируемой функции. Как видно, функции Node также взаимодействуют с классами Individual, Pattern, Repository,

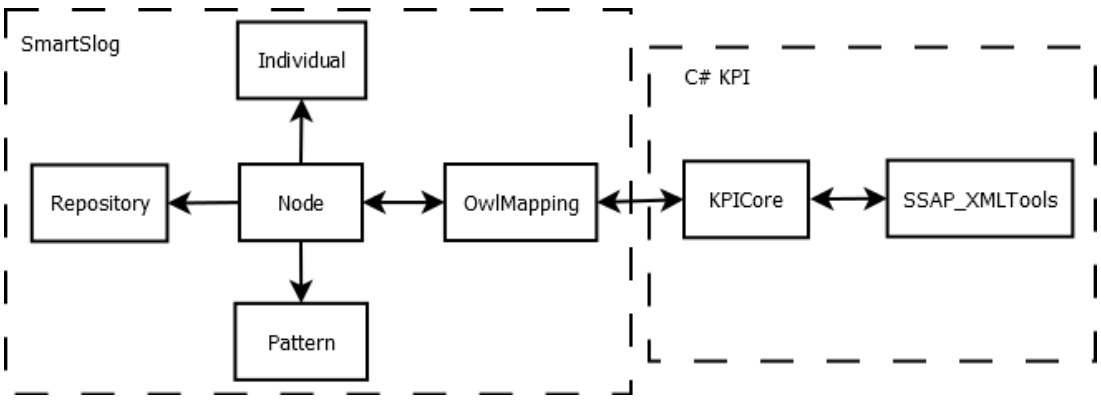

Рис. 4: Схема взаимодействия классов в интеграционных тестах.

так как в некоторых случаях приходиться локально сохранять данные. Все взаимодействие с KPICore происходит через класс OwlMapping, так как C# KPI оперирует с низкоуровневыми RDF тройками, а SmartSlog с высокоуровневыми индивидами, классами и свойствами.

При проведении интеграционного тестирования тесты будут разделены на группы. Каждая из групп организует свой порядок взаимодействия классов. Описание групп и их порядок взаимодействия классов описан ниже.

- *Not joined.*

  Содержит тесты с проверкой взаимодействия классов при вызове всех основных функций класса Node (Remove, Insert, GetProperty, Exist, SetProperty, GetIndividuals, RemoveProperty, Populate, Update) на генерирование исключений об ошибках при выполнении данных функций без установки сетевого подключения с ИП (без вызова функции Join).

  Порядок взаимодействия:

  $Node \rightarrow OwlMapping \rightarrow KPICore$

- *Individual/Property.*

  Содержит тесты с проверкой взаимодействия классов при вызове функций класса Node при работе с индивидами и свойствами.

  Порядок взаимодействия:

  $[Node \rightarrow Repository + Individual] \rightarrow OwlMapping \rightarrow KPICore \rightarrow$

  $\rightarrow SSAP\_XMLTools \rightarrow KPICore \rightarrow OwlMapping \rightarrow Node$

- *Pattern.*

  Содержит тесты с проверкой взаимодействия классов при вызове функций класса

Node при работе с шаблонами знаний.

Порядок взаимодействия:

$[Node \rightarrow Pattern] \rightarrow OwlMapping \rightarrow KPICore \rightarrow SSAP\_XMLTools \rightarrow$

$\rightarrow KPICore \rightarrow OwlMapping \rightarrow Node$

- *Other.*

  Содержит тесты с проверкой взаимодействия классов при вызове функций Exist и Populate, а также специализированные тесты для функций подключения к ИП.

  Порядок взаимодействия:

  $Node \rightarrow OwlMapping \rightarrow KPICore \rightarrow SSAP\_XMLTools \rightarrow KPICore$

Для проведения аттестационного тестирования библиотека SmartSlog должна выполнять следующий набор функций.

1. Локальные функции (без участия C# KPI):
   (a) работа с сущностями (регистрация, создание индивидов);
   (b) работа со свойствами (получение, установка, удаление, обновление).
2. Удаленные функции (с участием C# KPI):
   (a) управление доступов к ИП (подключение, отсоединение);
   (b) работа со свойствами (получение, установка, удаление, обновление);
   (c) расширенная работа с индивидами (вставка, удаление, обновление, заполнение);
   (d) поддержка запросов (получение индивидов по шаблонам поиска, получение индивидов определенного класса).

На каждую группу функций должно быть реализовано демонстрационное тестирующее приложение с использованием библиотеки SmartSlog версии C# и низкоуровневого C# KPI с использованием Mircosoft Visual Studio 2013.

2.3 Подход к тестированию

Тестирование будет проходить в автоматическом режиме, с помощью VS2013 и фреймворка NUnit. Тестировщику необходимо запустить тесты в проекте для тестирования и ожидать результата. После выполнения всех тестов тестировщику будет предоставлен

результат тестирования, в котором будет видно какие тесты прошли успешно, а какие нет. В случаи ошибки, тестировщик составляет отчет об ошибки.

2.4 Критерий прохождения тестов

Применим для блочного и интеграционного тестирования.
Тест считается успешно пройденным, если ожидаемый и фактический результаты совпадают. Если тест завершается неудачей, то перед принятием решения целесообразно проверить правильность самого теста. Если тест завершился неудачей и тест реализован правильно, то производится заключение о найденной ошибке. Ожидаемый результат должен присутствовать в описании каждого теста (описание будет приводится в разделе *Тесты*).

2.5 Критерий приостановления тестирования

Применим для блочного и интеграционного тестирования.
Тестирование должно быть приостановлено, если количество непройдённых тестов превысит 10% от их общего количества.

2.6 Критерий возобновления работы

Применим для блочного и интеграционного тестирования.
После получения заявки на повторное тестирование, необходимо начать тестирование с самого начала. Тестирование с места, когда была приостановлена работа, считается недопустимым.

2.7 Участники тестирования

Для тестирования достаточно одного тестировщика, обладающего навыками работы со следующими инструментами:

1. Microsoft Visual Studio 2013
2. NUnit 2.6.3

3 Тесты

Типы тестов:

- П – простой
- О – общий
- С – специальный
- Н – негативный
- К – краевой

3.1 Блочные тесты

Класс Individual

Метод: Individual(OntologyClass oc, Session s [, string uuid])

Результат: экземпляр класса Individual, исключение, если аргумент не верен

Тест 1: IncorrectIndividualTest

Тип: Н

Входные данные: существующий онтологический класс, нулевое значение сессии

Алгоритм:

1. Создать экземпляр класса Individual с указанными входными данными

Ожидаемый результат: исключение System.ArgumentNullException

Тест 2: IncorrectIndividuaUuidlTest

Тип: Н

Входные данные: существующий онтологический класс, существующая сессия, нулевое значение идентификатора

Алгоритм:

1. Создать экземпляр класса Individual с указанными входными данными

Ожидаемый результат: исключение System.ArgumentNullException

Метод: RemoveProperty(Property property, Individual value)

Результат: удаление свойства с указанным значением индивида

Тест 3: RemovePropertyIndTest

Тип: О

Входные данные: существующие свойство, существующий индивид

Алгоритм:

1. Создать индивид

2. Установить 2 значения данного свойства с индивидом

3. Удалить свойство с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: данное значения должно быть удалено, количество значений данного свойства равно 1

Тест 4: RemoveIncorrectPropertyIndTest

Тип: С

Входные данные: существующие свойство, нулевой индивид

Алгоритм:

1. Создать индивид

2. Установить свойство

3. Удалить свойство с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: удаление не происходит

Тест 5: RemoveIncorrectPropertyIndCardinalityViolationTest

Тип: О

Входные данные: существующие свойство с минимальной кардинальностью 1 с существующим значением индивида

Алгоритм:

1. Создать индивид

2. Установить 1 значение данного свойства с индивидом

3. Удалить одно значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: удаление не произошло, исключение System.ArgumentOutOfRangeException

Метод RemoveProperties(Property property)

Результат: удаление всех значений у указанного свойства

Тест 6: RemoveAllPropertiesTest

Тип: О

Входные данные: существующие свойство

Алгоритм:

1. Создать индивид
2. Установить 3 значения данного свойства
3. Удалить значения данного свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значения свойства удалены, количество значений данного свойства 0

Метод: RemoveProperty(Property property, string value)

Результат: удаление свойства с указанным значением строки

Тест 7: RemovePropertyStringTest

Тип: O

Входные данные: существующие свойство с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 3 значения данного свойства со строкой
3. Удалить одно значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: данное значения должно быть удалено, количество значений данного свойства равно 2

Тест 8: RemoveIncorrectPropertyStringCardinalityViolationTest

Тип: O

Входные данные: существующие свойство с минимальной кардинальностью 1 с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 1 значение данного свойства со строкой
3. Удалить одно значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: удаление не произошло, исключение
System.ArgumentOutOfRangeException

Метод: SetProperty(Property property, string value)

Результат: установка свойства с указанным значением строки

Тест 9: SetOneGetPropertiesTest

Тип: С

Входные данные: существующие свойство с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 100 значений данного свойства с одинаковой строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 1

Тест 10: SetDifGetPropertiesTest

Тип: П

Входные данные: существующие свойство с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 2 различных значений данного свойства со строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 2

Тест 11: SetManyGetPropertiesTest

Тип: С

Входные данные: существующие свойство с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 1000 различных значений данного свойства со строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 1000

Тест 12: SetGetIncorrectPropertyCardinalityViolationTest

Тип: С

Входные данные: существующие свойство с максимальной кардинальностью 1 с существующим значением строки

Алгоритм:

1. Создать индивид
2. Установить 3 различных значений данного свойства со строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла только первого значения свойства, исключение System.ArgumentOutOfRangeException

Метод: SetProperty(Property property, Individual value)

Результат: установка свойства с указанным значением строки

Тест 13: SetOneObjectGetPropertiesTest

Тип: С

Входные данные: существующие свойство с существующим значением индивида

Алгоритм:

1. Создать индивид
2. Установить 100 значений данного свойства с одинаковым индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 1

Тест 14: SetDifGetObjectPropertiesTest

Тип: П

Входные данные: существующие свойство с существующим значением индивида

Алгоритм:

1. Создать индивид
2. Установить 2 различных значений данного свойства с индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 2

Тест 15: SetManyGetObjectPropertiesTest

Тип: С

Входные данные: существующие свойство с существующим значением индивида

Алгоритм:

1. Создать индивид
2. Установить 1000 различных значений данного свойства с индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла, количество значений данного свойство равно 1000

Тест 16: SetGetIncorrectObjectPropertyCardinalityViolationTest

Тип: С

Входные данные: существующие свойство с максимальной кардинальностью 1 с существующим значением индивида

Алгоритм:

1. Создать индивид
2. Установить 3 различных значений данного свойства с индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: установка произошла только первого значения свойства, исключение System.ArgumentOutOfRangeException

Метод: UpdateProperty(Property property, string oldvalue, string newValue)

Результат: обновление старого значения свойства на новое

Тест 17: UpdateNullParametersTest

Тип: Н

Входные данные: существующие свойство с нулевыми значениями свойств

Алгоритм:

1. Создать индивид
2. Установить значение данного свойства со строкой
3. Обновить значение данного свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: обновление не произошло, исключение System.ArgumentNullException

Тест 18: UpdateIncorrectParametersDataTest

Тип: Н

Входные данные: нулевое значение свойства с пустыми значениями строк

Алгоритм:

1. Создать индивид
2. Обновить значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.ArgumentException

Тест 19: UpdateToSameValueDataTest

Тип: Н

Входные данные: существующие значение свойства с непустыми значениями строк

Алгоритм:

1. Создать индивид
2. Установить одному свойству 2 значения со строками A и B
3. Обновить значение свойства A на значение свойства B с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: произойдет обновление значения свойства A на значение свойства B, количество значений у данного свойство равно 1

Метод: UpdateProperty(Property property, Individual oValue, Individual nValue)

Результат: обновление старого значения свойства на новое

Тест 20: UpdateNullObjectParametersTest

Тип: Н

Входные данные: существующее свойство с нулевыми значениями свойств

Алгоритм:

1. Создать индивид
2. Установить значение данного свойства с индивидом
3. Обновить значение данного свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: обновление не произошло, исключение System.ArgumentNullException

Тест 21: UpdateIncorrectParametersObjectTest

Тип: О

Входные данные: нулевое значение свойства с одинаковыми значениями индивидов

Алгоритм:

1. Создать индивид
2. Обновить значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.ArgumentException

Тест 22: UpdateToSameValueObjectTest

Тип: О

Входные данные: существующее значение свойства с существующими значениями индивидов

Алгоритм:

1. Создать индивид
2. Установить одному свойству 2 значения с индивидами A и B
3. Обновить значение свойства A на значение свойства B с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: произойдет обновление значения свойства A на значение свойства B, количество значений у данного свойство равно 1

Класс Node

Метод: Individual CreateIndividual(OntologyClass ontologyClass, string uri)

Результат: создание индивида заданного онтологического класса и заданным uri

Тест 23: CreateIndividualUuidTest

Тип: П

Входные данные: существующий онтологический класс с непустым uri

Алгоритм:

1. Создать индивид с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: создание ненулевого индивида с заданным uri

Метод: Pattern CreatePattern(OntologyClass ontologyClass)

Результат: создание шаблона поиска заданного онтологического класса

Тест 24: CreatePatternTest

Тип: П

Входные данные: существующий онтологический класс

Алгоритм:

1. Создать шаблон поиска с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: создание ненулевого шаблона поиска

Тест 25: CreatePatternNotRegisteredOntologyClassTest

Тип: П

Входные данные: незарегистрированный онтологический класс

Алгоритм:

1. Создать шаблон поиска с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: нулевой шаблона поиска

Класс Patten

Метод: AddUnProperty(Property property, string value)

Результат: добавление незарегистрированного свойства со значением строкой

Тест 26: AddUnPropertyDataTest

Тип: О

Входные данные: существующее свойство со значением строкой

Алгоритм:

1. Создать шаблон поиска

2. Добавить значение свойства со строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: свойство установлено

Метод: AddUnProperty(Property property, Individual value)

Результат: добавление незарегистрированного свойства со значением индивидом

Тест 27: AddUnPropertyObjectTest

Тип: O

Входные данные: существующее свойство со значением индивидом

Алгоритм:

1. Создать шаблон поиска
2. Добавить значение свойства с индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: свойство установлено

Метод: List<object> GetUnPropertyValues()

Результат: получения списка значений всех добавленных незарегистрированных свойств

Тест 28: GetUnPropertyValuesTest

Тип: O

Входные данные: –

Алгоритм:

1. Создать шаблон поиска
2. Добавить 2 незарегистрированных свойства с различными значениями (по 1 на свойство)
3. Получить список значений незарегистрированных свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список значений свойств получен, общее количество значений равно 2

Метод: List<Property> GetOntUnProperties()

Результат: получение списка всех добавленных незарегистрированных свойств

Тест 29: GetOntUnPropertyValuesTest

Тип: O

Входные данные: –

Алгоритм:

1. Создать шаблон поиска
2. Добавить 2 незарегистрированных свойства с различными значениями (по 1 на свойство)

3. Получить список незарегистрированных свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список свойств получен, общее количество различных свойств равно 2

Метод: AddProperty(Property property, string value)

Результат: добавление зарегистрированного свойства со значением строкой

Тест 30: AddPropertyDataTest

Тип: О

Входные данные: существующее свойство со значением строкой

Алгоритм:

1. Создать шаблон поиска
2. Добавить значение свойства со строкой с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: свойство установлено

Метод: AddProperty(Property property, Individual value)

Результат: добавление зарегистрированного свойства со значением индивидом

Тест 31: AddPropertyObjectTest

Тип: О

Входные данные: существующее свойство со значением индивидом

Алгоритм:

1. Создать шаблон поиска
2. Добавить значение свойства с индивидом с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: свойство установлено

Метод: List<object> GetPropertyValues()

Результат: получение списка значений всех добавленных зарегистрированных свойств

Тест 32: GetPropertyValuesTest

Тип: О

Входные данные: –

Алгоритм:

1. Создать шаблон поиска

2. Добавить 2 зарегистрированных свойства с различными значениями (по 1 на свойство)

3. Получить список значений зарегистрированных свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список значений свойств получен, общее количество значений равно 2

Метод: List<Property> GetOntProperties()

Результат: получения списка всех добавленных зарегистрированных свойств

Тест 33: GetOntPropertyValuesTest

Тип: О

Входные данные: –

Алгоритм:

1. Создать шаблон поиска

2. Добавить 2 зарегистрированных свойства с различными значениями (по 1 на свойство)

3. Получить список зарегистрированных свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список свойств получен, общее количество различных свойств равно 2

Метод: RemoveProperty(Property property, string value)

Результат: удаление зарегистрированного свойства со значением строкой

Тест 34: RemovePropertyDataPatternTest

Тип: О

Входные данные: существующее свойство с существующим значением строки

Алгоритм:

1. Создать шаблон поиска

2. Добавить зарегистрированное свойства со значением строки

3. Удалить заданное значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалено, общее количество значений свойства равно 0

Тест 35: RemovePropertyIncorrectDataPatternTest

Тип: С

Входные данные: существующее свойство с несуществующим значением строки

Алгоритм:

1. Создать шаблон поиска
2. Добавить зарегистрированное свойства со значением строки
3. Удалить заданное значение строку с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство не удалено, общее количество значений свойства равно 1

Метод: RemoveProperty(Property property, Individual value)

Результат: удаление зарегистрированного свойства со значением индивида

Тест 36: RemovePropertyObjectPatternTest

Тип: О

Входные данные: существующее свойство с существующим значением индивида

Алгоритм:

1. Создать шаблон поиска
2. Добавить зарегистрированное свойства со значением индивида
3. Удалить заданное значение индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалено, общее количество значений свойства равно 0

Тест 37: RemovePropertyIncorrectObjectPatternTest

Тип: С

Входные данные: существующее свойство с несуществующим значением индивида

Алгоритм:

1. Создать шаблон поиска
2. Добавить зарегистрированное свойства со значением индивида

3. Удалить заданное значение индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство не удалено, общее количество значений свойства равно 1

Метод: RemoveUnProperty(Property property, string value)

Результат: удаление незарегистрированного свойства со значением строкой

Тест 38: RemoveUnPropertyDataPatternTest

Тип: O

Входные данные: существующее свойство с существующим значением строки

Алгоритм:

1. Создать шаблон поиска
2. Добавить незарегистрированное свойства со значением строки
3. Удалить заданное значение свойства с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалено, общее количество значений свойства равно 0

Тест 39: RemoveUnPropertyIncorrectDataPatternTest

Тип: C

Входные данные: существующее свойство с несуществующим значением строки

Алгоритм:

1. Создать шаблон поиска
2. Добавить незарегистрированное свойства со значением строки
3. Удалить заданное значение строку с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство не удалено, общее количество значений свойства равно 1

Метод: RemoveUnProperty(Property property, Individual value)

Результат: удаление незарегистрированного свойства со значением индивида

Тест 40: RemoveUnPropertyObjectPatternTest

Тип: O

Входные данные: существующее свойство с существующим значением индивида

Алгоритм:

1. Создать шаблон поиска
2. Добавить незарегистрированное свойства со значением индивида
3. Удалить заданное значение индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалено, общее количество значений свойства равно 0

Тест 41: RemoveUnPropertyIncorrectObjectPatternTest

Тип: С

Входные данные: существующее свойство с несуществующим значением индивида

Алгоритм:

1. Создать шаблон поиска
2. Добавить незарегистрированное свойства со значением индивида
3. Удалить заданное значение индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство не удалено, общее количество значений свойства равно 1

Метод: SetUUID(string uuid)

Результат: установка идентификатора

Тест 42: SetGetUUIDPatternTest

Тип: П

Входные данные: существующее значение строки

Алгоритм:

1. Создать шаблон поиска
2. Установить идентификатор с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: идентификатор установлен

3.2 Интеграционные тесты

Класс Node

Метод: bool Exist(Individual individual)

Результат: проверка на существование индивида в ИП

Тест 43: ExistNotJoinTest

Тип: H

Группа взаимодействия: Not joined

Входные данные: существующее значение индивида

Алгоритм:

1. Создать индивида
2. Проверить на существование заданного индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 44: ExistIndividualTest

Тип: O

Группа взаимодействия: Other

Входные данные: существующее значение индивида

Алгоритм:

1. Создать A и B индивиды разных онтологических классов
2. Присоединиться к ИП
3. Поместить индивид A в ИП
4. Проверить на существование индивида A с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: A - true, B - false

Метод: Insert(Individual individual)

Результат: публикация индивида в ИП

Тест 45: InsertNodeNotJoinTest

Тип: H

Группа взаимодействия: Not joined

Входные данные: существующее значение индивида

Алгоритм:

1. Создать индивида

2. Вставить индивид с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: object GetProperty(Individual individual, Property property)

Результат: получение свойства индивида из ИП

Тест 46: GetPropertyNodeNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид с существующим свойством

Алгоритм:

1. Создать индивида

2. Получить значение свойства индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 47: GetPropertyNodeTest

Тип: О

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с существующим свойством

Алгоритм:

1. Создать 2 индивида разных онтологических классов

2. Установить локально одному из них значение строку, другому индивид

3. Получить значение свойств индивидов с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: 2 установленных значения свойств (строка и индивид)

Тест 48: GetPropertyNotRegisteredIndTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: незарегистрированный индивид с существующим свойством

Алгоритм:

1. Создать незарегистрированный индивид
2. Установить локально значение строку или индивид
3. Получить значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.ArgumentException

Метод: Remove(Individual individual)

Результат: удаление индивида из ИП

Тест 49: RemoveNodeNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид

Алгоритм:

1. Создать индивида
2. Удалить индивид с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: Collection<Individual> GetIndividuals(OntologyClass ontologyClass)

Результат: получение списка индивидов заданного онтологического класса из ИП

Тест 50: GetIndividualsNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий онтологический класс

Алгоритм:

1. Создать несколько индивидов
2. Получить список индивидов с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 51: GetIndividualsTest

Тип: О

Группа взаимодействия: Individual/Property

Входные данные: существующий онтологический класс

Алгоритм:

1. Создать 3 различных индивида одного онтологического класса
2. Присоединиться к ИП
3. Вставить 3 созданных индивида в ИП
4. Получить список индивидов с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список индивидов из ИП заданного онтологического класса, количество индивидов равно 3

Тест 52: GetIndividualsIncorrectTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: нулевой онтологический класс

Алгоритм:

1. Присоединиться к ИП
2. Получить список индивидов с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.ArgumentException

Метод: Collection<Individual> GetIndividuals(Pattern pattern)

Результат: получение списка индивидов по шаблону поиска из ИП

Тест 53: GetIndividualsNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий шаблон поиска

Алгоритм:

1. Создать несколько индивидов
2. Создать шаблон поиска
3. Получить список индивидов с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 54: GetIndividualsPatternDataTypeTest

Тип: О

Группа взаимодействия: Pattern

Входные данные: существующий шаблон поиска

Алгоритм:

1. Создать 3 индивида одного онтологического класса
2. Задать 3 различных свойства с различными значениями строк (по значению на каждого индивида)
3. Создать шаблон поиска такого же онтологического класса, что и индивиды
4. Установить шаблону поиска одного из заданных свойств со значением строки
5. Присоединиться к ИП
6. Получить список индивидов по шаблону поиска с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список индивидов, содержащий 1 индивид, соответствующий шаблону поиска

Тест 55: GetIndividualsPatternObjectTypeTest

Тип: О

Группа взаимодействия: Pattern

Входные данные: существующий шаблон поиска

Алгоритм:

1. Создать 3 индивида одного онтологического класса
2. Задать 3 различных свойства с различными значениями индивидов (по значению на каждого индивида)
3. Создать шаблон поиска такого же онтологического класса, что и индивиды
4. Установить шаблону поиска одного из заданных свойств со значением индивида
5. Присоединиться к ИП
6. Получить список индивидов по шаблону поиска с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: список индивидов, содержащий 1 индивид, соответствующий шаблону поиска

Метод: RemoveProperty(Individual individual, Property property, Individual value)

Результат: удаление значения объектного свойства у индивида из ИП

Тест 56: RemovePropertyNodeNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид с существующим объектным значением свойства

Алгоритм:

1. Создать индивида
2. Установить объектное значение свойства
3. Удалить объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 57: RemovePropertyNodeNullParamTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с нулевым свойством и значением индивидом

Алгоритм:

1. Создать индивида
2. Присоединиться к ИП
3. Удалить объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.ArgumentException

Тест 58: RemoveObjectPropertyNodeTest

Тип: О

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с существующим свойством и значением индивидом

Алгоритм:

1. Создать индивида
2. Установить локально индивиду свойство со значением индивида
3. Присоединиться к ИП
4. Поместить индивид со свойством в ИП
5. Удалить объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалилось из ИП

Метод: RemoveProperty(Individual individual, Property property, string value)

Результат: удаление значения свойства строки у индивида из ИП

Тест 59: RemovePropertyNodeNullParamStringTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с нулевым свойством и значением строки

Алгоритм:

1. Создать индивида
2. Присоединиться к ИП
3. Удалить строковое значение свойства индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.ArgumentException

Тест 60: RemovePropertyNodeIncorrectTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с существующим свойством и несуществующим значением строки

Алгоритм:

1. Создать индивида
2. Установить строковое значение свойства
3. Присоединиться к ИП

4. Удалить строковое значение свойства индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: значение свойства не удалится

Тест 61: RemoveDataPropertyNodeTest

Тип: О

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с существующим свойством и строковым значением

Алгоритм:

1. Создать индивида
2. Установить локально индивиду свойство со строковым значением
3. Присоединиться к ИП
4. Поместить индивид со свойством в ИП
5. Удалить строковое значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: значение свойство удалилось из ИП

Метод: SetProperty(Individual individual, Property property, Individual value)

Результат: установка значения объектного свойства у индивида в ИП

Тест 62: SetPropertyNodeNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид с заданным объектным значением свойства

Алгоритм:

1. Создать индивида
2. Установить объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Тест 63: SetObjectPropertyNodeIncorrectCardinalityTest

Тип: Н

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с заданным объектным значением свойства с максимальной кардинальностью 1

Алгоритм:

1. Создать индивида
2. Установить локально одно объектное значение свойства
3. Подключиться к ИП
4. Установить второе объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: исключение System.InvalidOperationException

Тест 64: SetObjectPropertyNodeTest

Тип: О

Группа взаимодействия: Individual/Property

Входные данные: существующий индивид с заданным объектным значением свойства

Алгоритм:

1. Создать индивида
2. Подключиться к ИП
3. Установить объектное значение свойства индивида с помощью тестируемого метода с указанными входными данными

Ожидаемый результат: свойство установлено в ИП

Метод: Populate(Individual individual)

Результат: заполнение индивида значениями свойств из ИП

Тест 65: PopulateNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид

Алгоритм:

1. Создать индивида

2. Заполнить индивид с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: Update(Individual individual)

Результат: обновление индивида в ИП

Тест 66: UpdateNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид

Алгоритм:

1. Создать индивида

2. Обновить индивид с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: Update(Individual individual, Pattern pattern)

Результат: обновление индивида в ИП по шаблону поиска

Тест 67: UpdatePatternNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид с существующим шаблоном поиска

Алгоритм:

1. Создать индивида

2. Создать шаблон поиска

3. Обновить индивид с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: UpdateProperty(Individual individual, Property property, string oldValue, string newValue)

Результат: обновление значения со строкой свойства индивида в ИП

Тест 68: UpdatePropertyNotJoinTest

Тип: Н

Группа взаимодействия: Not joined

Входные данные: существующий индивид с существующим старым значением строки, новое значение строка

Алгоритм:

1. Создать индивида
2. Установить старое значение свойства со строкой индивида
3. Обновить свойство данных индивида с помощью тестируемого метода с указанными входными данными без присоединения к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: Join()

Результат: подключение к ИП

Тест 69: JoinRepeatTest

Тип: Н

Группа взаимодействия: Other

Входные данные: -

Алгоритм:

1. Подключиться к ИП
2. Заснуть на 10 мс
3. Подключиться к ИП

Ожидаемый результат: исключение System.InvalidOperationException

Метод: Leave()

Результат: отключение от ИП

Тест 70: LeaveRepeatTest

Тип: Н

Группа взаимодействия: Other

Входные данные: -

Алгоритм:

1. Подключиться к ИП
2. Заснуть на 10 мс

3. Отключиться от ИП

4. Заснуть на 10 мс

5. Отключиться от ИП

Ожидаемый результат: исключение System.InvalidOperationException

3.3 Аттестационные тесты

1. Локальные функции: работа с сущностями

Тип теста: О

Алгоритм:

1. Создать индивид

2. Установить uuid

Ожидаемый результат: индивид с заданным uuid

2. Локальные функции: работа со свойства

Тип теста: О

Алгоритм:

1. Создать индивид

2. Установить uuid

3. Установить свойство со значением A

4. Обновить свойство со значения A на значение B

5. Удалить свойство со значением B

6. Установить свойство со значением C

7. Получить значение свойства

Ожидаемый результат: значение C

3. Удаленные функции: управление доступом к ИП

Тип теста: С

Алгоритм:

1. Создать узел (класс Node)
2. Подключить узел к ИП
3. Отключить узел от ИП
4. Проверить сетевое соединение узла с ИП

Ожидаемый результат: соединение отсутствует

4. Удаленные функции: работа со свойствами

Тип теста: О

Алгоритм:

1. Создать индивид
2. Установить uuid
3. Подключиться к ИП
4. Установить свойство со значением A
5. Обновить свойство со значения A на значение B
6. Удалить свойство со значением B
7. Установить свойство со значением C
8. Получить значение свойства

Ожидаемый результат: значение C

5. Удаленные функции: расширенная работа с индивидами

Тип теста: О

Алгоритм:

1. Создать индивид
2. Установить uuid
3. Локально установить свойство со значением A
4. Подключиться к ИП
5. Заполнить индивид из ИП

6. Поместить индивид в ИП

7. Локально установить свойство со значением B

8. Обновить индивид в ИП

9. Удалить индивид из ИП

Ожидаемый результат: индивид отсутствует в ИП

6. Удаленные функции: поддержка запросов

Тип теста: O

Алгоритм:

1. Создать индивиды A1 и B1

2. Локально установить свойство со значением A индивиду A1

3. Локально установить свойство со значением B индивиду B1

4. Подключиться к ИП

5. Поместить индивиды A1 и B1 в ИП

6. Создать шаблон поиска со свойством с значением B

7. Получить список индивидов по созданному шаблону поиска

Ожидаемый результат: индивид B1 со значением B

4 Покрытие кода

Сложность современного программного обеспечения и инфраструктуры сделало невыполнимой задачу проведения тестирования со 100% тестовым покрытием. Таким образом покрытие кода тестами в сложных программных проектах должна составлять, по крайней мере, 80..100%.

Расчет тестового покрытия относительно исполняемого кода программного обеспечения проводится по формуле:

$$\mathrm{Tcov} = \frac{\mathrm{Ltc}}{\mathrm{Lcode}} * 100\% \tag{1}$$

где:

- Tcov - тестовое покрытие

- Ltc - количество строк кода, покрытых тестами
- Lcode - общее количество строк кода

Одним из инструментов, определяющих степень покрытия кода и являющимся открытым программным обеспечением, является PartCover.

Текущей вариант отчета о покрытие кода библиотеки SmartSlog для языка C# представлен на рис. 5. На данный момент покрытие кода не превосходит 80%, но уже протестирована значительная его часть, что позволяет использовать данные тесты для регулярного выполнения и проверки после рефакторинга.

Summary

Parser:	PartCover23Parser
Assemblies:	1
Files:	11
Coverage:	75.4%
Covered lines:	1356
Coverable lines:	1799
Total lines:	4448

Assemblies

PetrSU.SmartSlog	**75.4%**	
PetrSU.SmartSlog.Individual	95.2%	
PetrSU.SmartSlog.Node	78.9%	
PetrSU.SmartSlog.OntologyClass	42.9%	
PetrSU.SmartSlog.OwlMapping	46.8%	
PetrSU.SmartSlog.Pattern	90.4%	
PetrSU.SmartSlog.Property	100%	
PetrSU.SmartSlog.PropertyValue	100%	
PetrSU.SmartSlog.Repository	69.7%	

Рис. 5: Покрытие кода библиотеки SmartSlog

5 Примеры реализации тестов

Ниже будет приведен тест кейс в классе тестирования локальных функций библиотеки SmartSlog версии языка C# для обеспечения тестирования функций установки и получения свойств (типа string) индивида с использованием фреймворка NUnit:

```
[TestFixture]
public class LocallyTests
{
        Node locallyNode = null;
        OntologyStructure os = null;
```

```
        [TestFixtureSetUp]
        protected void SetUp()
        {
                locallyNode = new Node("Test", spaceId, address, port);
                os = new OntologyStructure(locallyNode);
        }

        [Test]
        public void SetGetPropertyTest()
        {
                Individual money = locallyNode.CreateIndividual(os.Money);
                money.SetProperty(os.Currency, "US");

                Assert.AreEqual("US", money.GetDataProperty(os.Currency));
                Assert.AreEqual("US", (string)money.GetProperty(os.Currency));
                Assert.IsNull(money.GetObjectProperty(os.Currency);
        }
}
```

Данный тест кейс SetGetPropertyTest() состоит из двух частей: PreConditions и Test Case Description. Третья часть PostConditions отсутствует. В первой части PreConditions происходят действия для проведения основной проверки, т.е. происходит создание индивида с помощью функции CreateIndividual, затем происходит установка свойства индивиду с помощью функции SetProperty. Во второй части происходят действия переводящие систему из одного состояния в другое и проверки поставленных требований. В данном примере эта часть выполняется в наборе утверждений Assert фреймворка NUnit.

Иногда необходимо ожидаемый результат теста может быть исключением специализированного типа, с помощью фреймворка это можно проверить с помощью атрибута ExpectedException:

```
[Test]
[ExpectedException("PetrSU.SmartSlog.SemanticException")]
public void SetPropertyNodeIncorrectCardinalityTest()
{
        Individual money = node.CreateIndividual(os.Money);
    Assert.IsNotNull(money, "CreateIndividual␣failed.");

    money.SetProperty(os.Falsity, "YES");
    node.Join();
```

```
    node.SetProperty(money, os.Falsity, "NO");
}
```

Данный тест проверяет кардинальность свойства os.Falsity, в данном случаи оно равно 1, поэтому попытка установить второе значение свойству тому же самому индивиду генерирует исключение, которое обрабатывается как ожидаемый результат данного теста.

Все остальные тест кейсы, разработанные для библиотеки, строятся аналогично тем, которые были приведены выше. Результаты выполнения некоторых тестов могут быть проанализированы вручную, путем сравнения опубликованных данных В ИП и данных отладочного вывода.

6 Отчет о выполнении тестирования

Таблица 1: Результаты блочного и интеграционного тестирования

Номера тестов	**Класс**	**Метод**	**Кол-во тестов**	**Кол-во ошибок**	**Дата проведения**
1,2	Individual	Individual	2	0	20.01.2015
3,4,5, 7,8	Individual	RemoveProperty	5	0	20.01.2015
6	Individual	RemoveProperties	1	0	20.01.2015
9,10,11, 12,14, 15,16	Individual	SetProperty	7	0	20.01.2015
17,18,19, 20,21,22	Individual	UpdateProperty	6	0	20.01.2015
17,18,19, 20,21,22	Individual	UpdateProperty	6	0	20.01.2015
23	Node	CreateIndividual	1	0	20.01.2015
24,25	Node	CreatePattern	2	0	20.01.2015
26,27	Pattern	AddUnProperty	2	0	20.01.2015
28	Pattern	GetUnPropertyValues	1	0	20.01.2015

Продолжение таблицы 1

Номера тестов	Класс	Метод	Кол-во тестов	Кол-во ошибок	Дата проведения
29	Pattern	GetOntUnProperties	1	0	20.01.2015
30,31	Pattern	AddProperty	2	0	20.01.2015
32	Pattern	GetPropertyValues	1	0	20.01.2015
33	Pattern	GetOntProperties	1	0	20.01.2015
34,35, 36,37	Pattern	RemoveProperty	4	0	20.01.2015
38,39, 40,41	Pattern	RemoveUnProperty	4	0	20.01.2015
42	Pattern	SetUUID	1	0	20.01.2015
43,44	Node	Exist	2	0	20.01.2015
45	Node	Insert	1	0	20.01.2015
46,47,48	Node	GetProperty	3	0	20.01.2015
49	Node	Remove	1	0	20.01.2015
50,51,52, 53,54,55	Node	GetIndividuals	6	2 (Отчет об ошибке №1)	20.01.2015
56,57,58, 59,60,61	Node	RemoveProperty	6	0	20.01.2015
62,63,64	Node	SetProperty	3	0	20.01.2015
65	Node	Populate	1	0	20.01.2015
66,67	Node	Update	2	0	20.01.2015
68	Node	UpdateProperty	1	0	20.01.2015
69	Node	Join	1	0	20.01.2015
70	Node	Leave	1	0	20.01.2015

Таблица 2: Результаты аттестационного тестирования

Номер теста	**Результат**	**Дата проведения**
1	пройден	20.01.2015

Продолжение таблицы 2

Номер теста	Результат	Дата проведения
2	пройден	20.01.2015
3	пройден	20.01.2015
4	пройден	20.01.2015
5	пройден	20.01.2015
6	провален (Отчет об ошибке №1)	20.01.2015

7 Отчет об ошибке №1

Не работает функция GetIndividuals класса Node с использованием шаблона поиска в качестве аргумента.

Тесты:

54 – GetIndividualsPatternDataTypeTest, 55 – GetIndividualsPatternObjectTypeTest.

Условия:

Версия библиотеки SmartSlogKPlib_NET_v0.21beta, ОС Windows 7 с использованием Visual Studio 2013, NUnit 2.6.3.

Приоритет:

Критический

Алгоритм воспроизведения ошибки:

1. Создать несколько индивидов одного онтологического класса с различными данными типа string (data property) или Individual (object property).

2. Создать шаблон поиска (Pattern) этого онтологического класса и установить ему в значение одно из свойств типа string или Individual.

3. Получить список индивидов по шаблону поиска с помощью функции GetIndividuals класса Node.

Ожидаемый результат:

Получить только те индивиды, которые соответствуют шаблону поиска (соответствуют установленному свойству).

Фактический результат:

Все существующие индивиды онтологического класса шаблона поиска.

Воспроизводимость:

Всегда.

Дата:

20.01.2015.

8 Результаты

В ходе разработки и выполнения тестов были получены следующие результаты. Результаты блочного тестирования представлены в таблице 3. Результаты интеграционного тестирования представлены в таблице 4.

Перечень обнаруженных ошибок:

1. GetIndividualsPatternDataTypeTest – из созданных нескольких индивидов с разными свойствами данных типа string пытаемся с помощью созданного шаблона поиска с конкретным значением свойства выбрать один индивид с этим свойством. В результате при вызове функции GetIndividuals передавая ей в качестве аргумента созданный шаблон поиска, получаем весь список созданных индивидов данного онтологического класса.

2. GetIndividualsPatternObjectTypeTest – из созданных нескольких индивидов с разными свойствами данных типа Individual пытаемся с помощью созданного шаблона поиска с конкретным значением свойства выбрать один индивид с этим свойством. В результате при вызове функции GetIndividuals передавая ей в качестве аргумента созданный шаблон поиска, получаем весь список созданных индивидов данного онтологического класса.

Таблица 3: Результаты блочного тестирования

Класс	**Количество тестов**	**Количество ошибок**
Individual	27	0
Pattern	17	0
Node	3	0

Таблица 4: Результаты интеграционного тестирования

Группа тестов	**Количество тестов**	**Количество ошибок**
Not joined	12	0
Individual/Property	9	0
Pattern	3	2
Other	4	0

В локальных функциях (без участия C# KPI) (a)-(c) аттестационного тестирования не выявлено ошибок. В удаленных функциях (с участием C# KPI) (a)-(c) также не выявлено ошибок. Однако удаленных функции (d), связанные с поддержкой запросов на получение индивидов по шаблонам поиска, не прошли аттестационного тестирования. Найденные ошибки являются критическими и требуют скорейшего исправления.

Таким образом, после проведения тестирования, можно сказать, что все основные функции библиотеки SmartSlog, как локальные, так и с использованием низкоуровневого C# KPI проходят все тесты. Однако существует проблемы с работой шаблонов поиска. Их основная функциональность, делать выборку из ИП по определенным свойствам и индивидам, не прошла проверки.

Список литературы

1. D. Korzun, A. Lomov, P. Vanag, J. Honkola, and S. Balandin, "Generating modest high-level ontology libraries for Smart-M3," in *Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, Oct. 2010, pp. 103–109.

2. "SmartSlog: free development software downloads at SourceForge.net," Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smartslog/

3. D. G. Korzun, S. I. Balandin, V. Luukkala, P. Liuha, and A. V. Gurtov, "Overview of Smart-M3 principles for application development," in *Proc. Congress on Information Systems and Technologies (IS&IT'11), Conf. Artificial Intelligence and Systems (AIS'11)*, vol. 4. Moscow: Physmathlit, Sep. 2011, pp. 64–71.

4. J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform," in *Proc. IEEE Symp. Computers and Communications (ISCC'10).* IEEE Computer Society, Jun. 2010, pp. 1041–1046.

5. "Smart-M3: Free development software downloads at SourceForge.net," Release 0.9.5beta, Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smart-m3/