Модель конфигуратора интеллектуального решателя, основанного на концептуальной модели задачи

М.И. Кошкарев, В.В. Нечаев

Аннотация **– В работе рассматривается подсистема конфигурирования (конфигуратор) интеллектуального решателя, основанного на концептуальной модели задачи для интеллектуальной системы информационной поддержки бионических технологий. Такая подсистема дает возможность оптимизировать использование ресурсов в процессе функционирования ИРЗ, оптимизировать время выполнения поставленных задач, восстанавливать поврежденные модули при работе в сети решателей.**

Ключевые слова **– Интеллектуальный решатель задач, конфигурационное моделирование, концептуальная модель задачи, модульная архитектура.**

I. Введение

Рассматривается один из классов систем искусственного интеллекта – интеллектуальные решатели задач (ИРЗ), которые, как правило, реализуются в виде программной системы. ИРЗ имеет два уровня интерпретаций в области систем искусственного интеллекта (СИИ). Первый уровень – это класс СИИ, а второй уровень – один из подсистем различных классов интеллектуальных систем. В обоих случаях ИРЗ имеет идентичную цель – реализация решений задач, поставленных пользователем, при поддержке СИИ. Принципиальным отличием ИРЗ, рассматриваемых в настоящей работе, является разделение пользовательского (внешнего) и программного (внутреннего) представления задачи. В качестве внешнего и внутреннего представления предлагается использовать концептуальную модель задачи (КМЗ) [1]. Такой подход дает возможность отделить пользовательское представление задачи от внутреннего ее представления. При этом возникает возможность унифицировать ИРЗ, определить парадигму φ программных модулей (БФМ) и механизмы настройки ИРЗ на разные типы пользовательских задач.

Статья получена 01.12.2016 г.

Исследование выполнено федеральным государственным бюджетным образовательным учреждением высшего образования «Московский технологический университет» (МИРЭА) за счет гранта Российского научного фонда (проект №14-11-00854)

Нечаев В. В., к.т.н, д.ф.-м.н., профессор, зав. лаб. МТУ МИРЭА
(e-mail: nechaev@mirea.ru)

Кошкарёв М. И., аспирант МТУ МИРЭА
(e-mail: kmi_89@mail.ru)

Актуальность создания конфигуратора для (ИРЗ) обусловлена тем, что многие из модулей могут занимать большое количество ресурсов и тем самым затормаживать работу системы в целом. В совокупности с концептуальной моделью задачи система дает возможность ИРЗ формировать такие свойства, как модульность, масштабируемость и интероперабельность.

Целью работы является создание модели подсистемы управления программными модулями, которая сможет проводить структурную адаптацию программных средств ИРЗ в целях его реконфигурирования.

Можно выделить следующие случаи структурной адаптации:

1) изменение функционала модуля без изменения общей структуры;

2) исключение одного модуля и включение на его место другого;

3) изъятие из системы и изменение режима работы оставшихся модулей;

4) добавление нового модуля;

4а) добавление нового модуля с изменением, уже существующего.

В случае создания распределенной сети ИРЗ, такая система должна осуществлять удаленную загрузку модулей от соседних фрагментов сети, повысив тем самым их восстанавливаемость и живучесть. Для достижения поставленной цели необходимо выполнить следующие задачи:

– проанализировать модульный состав ИРЗ;

– разработать алгоритмы работы целевой системы на основе методов конфигурирования [2];

– разработать модель модуля ИРЗ, используя КМЗ [1,3];

– привести примеры реализации такой системы на одном из языков высокого уровня.

Целевой результат данной работы – модель БФМ ИРЗ, определенного как конфигуратор, с описанными выше функциями, а также примеры их реализации на языке Java.

II. Базовые функциональные модули ИРЗ

В первую очередь выберем признаки оснований для классификации БФМ ИРЗ. Основания для классификации БФМ ИРЗ:

– обязательность: обязательные модули и дополнительные модули;

– сложность: простые (жесткие) модули и адаптивные (комплексные, конфигурируемые) модули;

– визуализация: модуль с наличием графического пользовательского представления и без.

Признак обязательности. По этому признаку модули разделяются на обязательные для функционирования системы по принципу необходимости и достаточности (такие как модуль планирования (МП), модуль управления базой знаний (СУБЗ) и др.), а также модули, приносящие определенные преимущества функционирования системы, но удовлетворяющие, только принципу достаточности.

Признак сложности. По этому признаку модули разделяются на конфигурируемые, с возможностью реконфигурации по целям, функциям или задачам и не конфигурируемые (без такой возможности). Также этот признак можно определить как функциональность: моно- и полифункциональные модули. *Монофункциональный модуль* имеет одну цель, задачу и функцию и, следовательно, является простым. *Полифункциональный модуль* имеет, как минимум более одной задачи, а следовательно, является конфигурируемым.

Визуализация. По этому признаку разделение осуществляется на модули с визуальным представлением результата работы пользователю и без визуализации.

Представим результаты классификации по выделенным выше основаниям в табличной форме, отметим что для формирования таблицы, используется список модулей из работы [4].

Следует иметь в виду: указанные свойства должны быть учтены при разработке архитектуры на программном уровне, при этом должны быть разработаны также и специальные классы, дифференцирующие базовые функциональные модули в системе.

III. Структурная адаптация системы на основе модульного состава

Предлагается рассмотреть режимы функционирования интеллектуального решателя, в которых происходит взаимодействие решателя и КМЗ на логическом уровне. Это необходимо для того чтобы разработать граф состояний решателя в ходе процесса решения задачи. Для удобства введем обозначения этих режимов функционирования (R):

R 0 – ожидание выполнения подзадачи;

R 1 – анализ задачи;

R 2 – планирование решения задачи;

R 3 – анализ плана решения;

R 4 – Решение (выполнение операций планирования);

R 5 – анализ результата решения задачи.

Далее для каждого режима функционирования составим таблицу взаимодействия: «Режим – Задача», а также связь с другими режимами.

A. Анализ задачи, загруженной в среду решателя

Для того чтобы произвести конфигурирование по модулям, необходимо проанализировать модульный состав ИРЗ в каждом из режимов функционирования (РФ) [4]. Результаты такого анализа представлены в таблице 2. Определим список режимов функционирования ИРЗ

Режимы функционирования:

1. Постановка задачи пользователем
2. Загрузка пользовательской задачи
3. Создание и базовое конфигурирование интеллектуального агента задачи
4. Функционирование интеллектуального агента и его конфигурирование:
   - 4.1. Анализ задачи
   - 4.2. Планирование задачи
   - 4.3. Анализ плана решения
   - 4.4. Выполнение операций планирования (решение)
   - 4.5. Оценка решения задачи
   - 4.6. Формирование объяснения действий (пользователю)
5. Представление результатов решения пользователю
6. Изменение базы знаний после выполненного решения
7. Обучение с экспертом

В таблице 2 использованы следующие обозначения:

«+» – обязательный модуль

«-» – в модуле нет необходимости

«V» – дополнительный модуль

Анализ данных, представленных в таблице 2, даёт возможность сделать следующий вывод: в зависимости от вида конфигурации данный метод даст возможность сэкономить от 30 % до 85 % ресурсов системы (при условии, что каждый модуль занимает одинаковое количество ресурсов).

На рисунке 1 показаны сущности, разработанные для классификации модулей в системе по признакам, рассмотренным в таблице 1. Для разработки за основу взят шаблон проектирования Interface (интерфейс) [5]. В качестве примера рассмотрены два модуля, реализующих разный набор интерфейсов. Для каждого базового функционального модуля разработан абстрактный класс, обязательно реализующий два из четырёх классов-интерфейсов. Все четыре класса интерфейса, в свою очередь, расширяют класс-интерфейс модуль, который и является базовым для любого модуля в системе. «Класс-интерфейс» модуля в свою очередь определяет обязательную для реализации функцию bool isRequred(), которая возвращает булево значение, определяющее является ли модуль обязательным для работы системы.

В библиотеке компонент системы описаны все классы, отвечающие за каждый из БФМ. Все классы являются абстрактными. Однако они реализуют часть функций, таких как isRequred() и getSystemType(), которые необходимы, для конфигурирования и работы системы. Они должны быть одинаково определены вне зависимости от дальнейшей реализации модуля в системе. При запуске системы конфигуратор выполняет ряд проверок. При проверке библиотек модулей системы БФМ конфигуратор выполняет проверку факта, что все модули, функция isRequred() которых

возвращает *true*, имеют реализацию, по меньшей мере, одним модулем. В случае отсутствия одного или нескольких модулей, система: либо переходит в режим удаленной загрузки модулей, если реализована распределенная сеть решателей, либо режим ограниченного функционирования (ошибки). Результатом данной проверки будут списки модулей, необходимых для загрузки в каждом из режимов функционирования. В дальнейшем при переходе из одного режима функционирования в другой классы будут загружаться в систему по мере необходимости.

IV. Конфигурирование комплексных модулей

Рассмотренные выше модули определялись как неделимый – целостный элемент системы. В этом разделе БФМ рассматривается разложенным по трём стратам: целевой (Z), функциональной (F) и задачной (T). Следует отметить, что выделенные страты не являются независимыми. Также необходимо иметь ввиду, что в данном разделе рассматривается только класс комплексных КБФМ, выделенный в первой части данной работы. Тогда КБФМ представляется следующей схемой (рисунок 2).

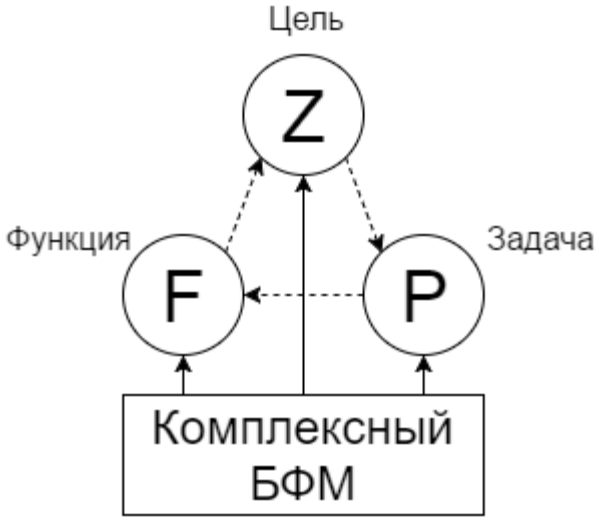

Рисунок 2 – Комплексное представление базового функционального модуля

Для данного представления необходимо определить содержательное наполнение каждого символа: цель Z – идеальный образ (модель конечного результата), функция F – что должен делать КБФМ для достижения цели, а задача T определяет, как реализована функция в соответствии с целью. Поскольку каждый компонент из представленных на рис. 2 компонентов Z, F, T, - это многосортные множества, постольку они могут быть определены соответствующими кортежами. Таким образом модуль КБФМ определяется следующими кортежами:

$$\begin{cases} Z = < Z_1, \dots, Z_v, \dots, Z_N >, \text{где: } v = \overline{1, \dots, N}; \quad N \geq 1. \\ F = < F_1, \dots, F_q, \dots, F_Q >, \text{где: } q = 1, 2, \dots, Q; Q \geq 2. \\ T = < T_1, \dots, T_l, \dots, T_L >, \text{где: } l = 1, 2, \dots, L; L \geq 2. \end{cases}$$

На основе этих кортежей системный модуль может быть представлен следующей концептуальной моделью:

$M = \langle Z, F, T, r \rangle$, где r - отношение.

С учетом приведенных выше записей сформируем стратифицированную структуру организации КБФМ, представленную на рисунке 3.

Замечание. В рассматриваемой схеме задача обозначена символом T (от англ. Task – задание), так как она ориентирована на реализацию в среде ИРЗ.

На основе предложенной модели возникает возможность представления комплексного модуля его внутренней структурой. Для реализации этой возможности следует разработать специальные алгоритмы и их реализацию на программном уровне (листинге 1).

Листинг 1. Класс DefaultAnalyzer

```
public class DefaultAnalyzer extends
AnalyzerAbstract{
    @Override
    public ProblemModel treat(ProblemModel
problemModel) {
        return null;
    }
    @Override
    public boolean isRequired() {
        return super.isRequired();
    }
    @Target(name = "component", functions =
{"func1forTarget1"})
    public boolean
analyzeComponent(ProblemModel problemModel){
        …
    }
    @Function(name = "func1forTarget1",tasks
= {})
    public boolean
firstTaskAnalyzeComponent(){
        …
    }
    @Target(name = "adequate", functions =
{})
    public boolean
analyzeAdequate(ProblemModel problemModel){
        …
    }
    @Target(name = "solution", functions =
{})
    public boolean
analyzeSolution(ProblemModel problemModel){
        …
    }
```

Как следует из Листинга 1, над некоторыми из функций объявлены аннотации [6] Target, Function и Task с параметрами. По этим аннотациям конфигурируется каждый из сложных модулей. Перед загрузкой модуля в момент переключения режима функционирования, сначала загружается bytecode (набор команд для Java Virtual Machine) класса реализующего данный модуль [6]. Затем проводиться анализ (выбирается программная функция или класс с текущей целью, создается список функций с аннотациями необходимых задач). На следующем шаге создается необходимый класс, содержащий только те функции, которые необходимы для достижения требуемой цели. Подобные аннотации могут быть расставлены как внутри одного класса, так и сразу в нескольких классах модуля. Как следствие, конфигурирование модуля будет происходить на уровне загрузки классов, т.е. будут загружаться только те классы, которые необходимы для

выполнения текущей задачи.

V. Конфигурирование «оптимального» модуля системы

На основе алгоритма конфигурирования системного модуля (КСМ) предложен алгоритм формирования оптимального модуля системы. Модель рассматриваемых ИРЗ представляет собой набор модулей с несколькими реализациями каждого из них (см. рисунок 4).

Каждый из этих модулей может быть разработан для оптимизации той или иной цели, задачи или функции. Под «оптимальным модулем» подразумевается такой, который ориентирован на достижение необходимой цели, а также имеет минимальное время выполнения задач и функций, требуемых для реализации поставленной цели. Стоит уточнить, что данный модуль является фиктивным, т.е. создаваться он будет только на время работы программы в оперативной памяти системы из нескольких реализаций одного и того же абстрактного модуля. Алгоритм функции конфигурирования оптимального модуля представлен на рисунке 5.

Рисунок 5 – Алгоритм функции конфигурирования оптимального модуля

Рассмотрим более подробно алгоритм оптимизации модуля системы. Предположим, что в системе существует три реализации информационно-поискового процессора: реализация поиска в глубину, поиска в ширину и поиска с итеративным углублением.

В информационно поисковом процессоре, должны быть определены следующие функции (цели работы модуля):

1) поиск объекта по имени:
2) поиск объекта по структуре;
3) поиск всех объектов заданного класса.

В ходе работы системы конфигуратор анализирует полученные им системные данные. Исходные данные для анализа можно представить в форме таблицы 3

Таблица 3 – Исходные данные для оптимизации модулей системы

цель	**модуль 1**	**модуль 2**	**…**	**модуль m**
цель 1	t_{11}	$\mathbf{t_{21}}$	…	t_{m1}
цель 2	t_{12}	t_{22}	…	$\mathbf{t_{m2}}$
…	…	…	…	…
цель n	$\mathbf{t_{1n}}$	t_{2n}	…	t_{mn}

Время t является средним временем выполнения каждой цели. Аналогично такие же исходные данные могут быть получены для задач и функций. Жирным шрифтом выделено минимальное время каждой из целей. Тогда для создания оптимального модуля выбираются необходимые функции каждой из реализаций, и создается фиктивный модуль, который будет вызываться на время работы программы. Анализ исходных данных и формирование модуля будет происходить каждый раз при запуске (инициализации) системы, а также в ходе работы системы при каждом переходе из одного состояния в другое. Исходные данные можно получить с помощью шаблона проектирования Proxy (Заместитель) [5], пример такого класса представлен в листинге 2.).

Листинг 2. Класс AnalyzerProxy

```
package componentLibrary.analizer;
import
componentLibrary.basic.problemModel.ProblemMo
del;
import java.util.HashMap;
import java.util.Map;
public class AnalyzerProxy extends
AnalyzerAbstract {
    final AnalyzerAbstract analyzer;
    final Map<String, Long> map = new
HashMap<String, Long>();
    public AnalyzerProxy(AnalyzerAbstract
analyzer) {
        this.analyzer = analyzer;
    }
    @Override
    public ProblemModel treat(ProblemModel
model) {
        long t = System.nanoTime();
        try {
            return analyzer.treat(model);
        } finally {
            map.put("treat",System.nanoTime()
- t);
        }
    }
    @Override
    public Class getSystemType() {
        long t = System.nanoTime();
        try {
            return analyzer.getSystemType();
        } finally {

map.put("systemType",System.nanoTime() - t);
```

```
        }}
    @Override
    public void configure(String targetName)
{
        long t = System.nanoTime();
        try {
            analyzer.configure(targetName);
        } finally {

map.put("configure",System.nanoTime() - t);
        }
    }
    @Override
    public boolean isRequired() {
        long t = System.nanoTime();
        try {
            return analyzer.isRequired();
        } finally {

map.put("isRequired",System.nanoTime() - t);
        }
    }
}
```

В качестве входного параметра объект такого класса получает реализацию настоящего класса и подменяет все её функции на свои так, чтобы вычислить время их работы, а затем помещает их в контейнер для хранения. По окончанию работы модуля эти данные передаются системе для последующего анализа. Стоит отметить, что сторонний разработчик реализации модуля при рассматриваемой архитектуре может не вносить в свой код никаких изменений, подмена классов будет от него скрыта.

VI. Восстановление системы с помощью модуля конфигурации в сети распределенных решателей

Восстановление системы в сети интеллектуальных решателей может быть достигнуто путем удаленной загрузки модулей. Так как система является модульной и реконфигурируемой, то повреждение модуля не должно влиять на работу системы в целом, а затрагивать только те режимы функционирования, в которых он участвует. Под повреждением модуля подразумевается не способность системы загрузить такой модуль в систему независимо от того какие причины этому предшествовали. Для своего самовосстановления системе понадобится обратиться к одному из интеллектуальных решателей в сети и послать запрос на получение модуля, поврежденного в исходной системе. Затем данный модуль загружается в систему. Стоит отметить, что система должна проводить анализ своих действий над полученным модулем. На текущий момент предлагается обязательные модули системы сохранять на жесткий диск, а дополнительные загружать и выгружать по мере необходимости получая у других решателей сети.

VII. Модель конфигуратора ИРЗ

Модель конфигуратора должна выполнять все необходимые функции, рассмотренные выше: реконфигурация системы, реконфигурация модулей, сбор аналитической информации о модулях, восстановление системы, ответ на запросы модулей другими системами.

Предлагается следующая модель модуля реализующего эти функции (рисунок 6).

Логический уровень получает запросы на переключение системы из одного режима в другой и запрашивает необходимые модули на физическом уровне, при этом не имея данных(адреса), откуда этот модуль получен

VIII. Заключение

В работе рассмотрена концептуальная модель конфигуратора ИРЗ. Приведена необходимая информация для анализа БФМ и создания библиотеки компонентов. На основе проведенного анализа, описаны алгоритмы работы конфигуратора для реконфигурирования системы, отдельных модулей, а также создания оптимального модуля системы и удаленной загрузки модулей. Приведены примеры реализации отдельных функций и структур на языке Java. Предложена концептуальная модель конфигуратора, разделенная на логический и физический уровни.

БИБЛИОГРАФИЯ

[1] Нечаев В. В. «Концептуальное модельное представление задачи как системы» // Информационные Нечаев В.В. Конфигурационное моделирование: часть I. Теоретические аспекты: Учебное пособие / Государственное образовательное учреждение высшего профессионального образования «Московский государственный институт радиотехники, электроники и автоматики (технический университет)». – М.:2007 – 92 с.

[2] Нечаев В.В. «Раскрытие неопределенности системной задачи, представленной концептуальной моделью» // Информационные технологии. – 2012, № 12 (157). - с. 46-52.

[3] Нечаев В.В., Кошкарёв М.И. «Интеллектуальные решатели задач: сравнительный анализ и архитектурная модель» // Информационные и телекоммуникационные технологии с. 51-61 № 21, 2014 г.

[4] Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. «Приемы объектно-ориентированного проектирования. Паттерны проектирования.» - СПб: Питер, 2001. – 368 с.: ил. (Серия «Библиотека программиста»)

[5] Bruce Eckel «Thinking in Java», 4th Edition, Prentice-Hall PTR, 2006.

[6] Горбатов В.А. «Теория частично упорядоченных систем» М.: Советское радио, 1976. – 336 с.

Таблица 1 – Модули интеллектуального решателя задач

№	Название модуля	Обязательность	Сложность	Визуализация
1	Пользовательский интерфейс	+	+	+
2	Интеллектуальный интерфейс	+	+	-
3	Система управления базой знаний	+	-	-
4	Модуль планирования	+	-	-
5	Вычислительный процессор	+	+	-
6	Модуль объяснения решений	+	+	-
7	Информационно-поисковый процессор	+	+	-
8	Подсистема конфигурации ИРЗ	+	-	-
9	Система управления задачами	+	-	-
10	Интеллектуальный агент задачи	+	-	-
11	Анализатор	+	-	-
12	Модуль информационного поиска в интернете	-	-	-
13	Модуль интеллектуального анализа данных	-	-	-
14	Модуль понимания ЕЯ	-	-	+
15	Интеллектуальная диалоговая система	-	-	+
16	Модуль распознавания изображения	-	-	+
17	Модуль синтеза речи	-	-	-
18	Модуль распознавания речи	-	-	-
19	Модуль когнитивной графики	-	-	+
20	Логический процессор	+	+	-

Таблица 2 – исходные данные для анализа модульного состава РФ системы

№	Название модуля	1	2	3	4						5	6	7
					1	2	3	4	5	6			
1	Пользовательский интерфейс	+	+	-	**V**	-	-	-	-	-	+	-	+
2	Интеллектуальный интерфейс	-	+	+	-	-	-	-	-	-	**V**	-	-
3	Система управления базой знаний	+	+	-	+	+	+	+	-	+	-	+	+
4	Модуль планирования	-	-	-	-	+	+	-	-	+	-	-	-
5	Вычислительный процессор	-	-	-	-	-	+	+	-	-	-	-	-
6	Модуль объяснения решений	-	-	-	-	-	-	-	-	+	-	-	-
7	Информационно-поисковый процессор	+	+	-	+	+	+	+	-	+	-	**V**	-
8	Подсистема конфигурации ИРЗ	+	+	+	+	+	+	+	+	+	+	+	+
9	Система управления задачами	-	+	+	-	-	-	-	-	-	+	-	-
10	Интеллектуальный агент задачи	-	-	+	+	+	+	+	+	+	-	+	-
11	Анализатор	-	-	-	+	-	+	-	+	-	-	-	-
12	Модуль информационного поиска в интернете	-	**V**	-	**V**	**V**	-	-	-	-	-	-	-
13	Модуль интеллектуального анализа данных	**V**	**V**	-	**V**	**V**	-	-	-	-	-	-	-
14	Модуль понимания ЕЯ	**V**	**V**	-	**V**	**V**	-	-	-	-	-	-	-
15	Интеллектуальная диалоговая система	**V**	**V**	-	**V**	-	-	-	-	-	-	-	-
16	Модуль распознавания изображения	**V**	**V**	-	**V**	-	-	-	-	-	-	-	-
17	Модуль синтеза речи	**V**	**V**	-	**V**	-	-	-	-	-	**V**	-	-
18	Модуль распознавания речи	**V**	**V**	-	**V**	-	-	-	-	-	-	-	-
19	Модуль когнитивной графики	**V**	**V**	-	**V**	-	-	-	-	-	-	-	-
20	Логический процессор	-	-	-	**V**	-	+	+	-	-	-	-	-

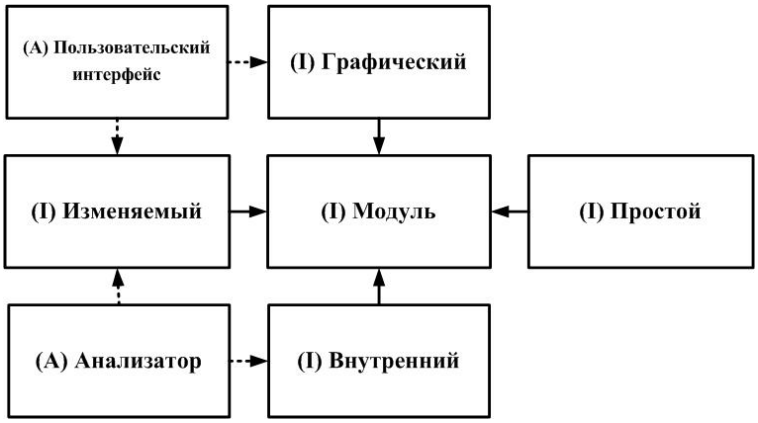

Рисунок 1 – Диаграмма классов базовых функциональных модулей

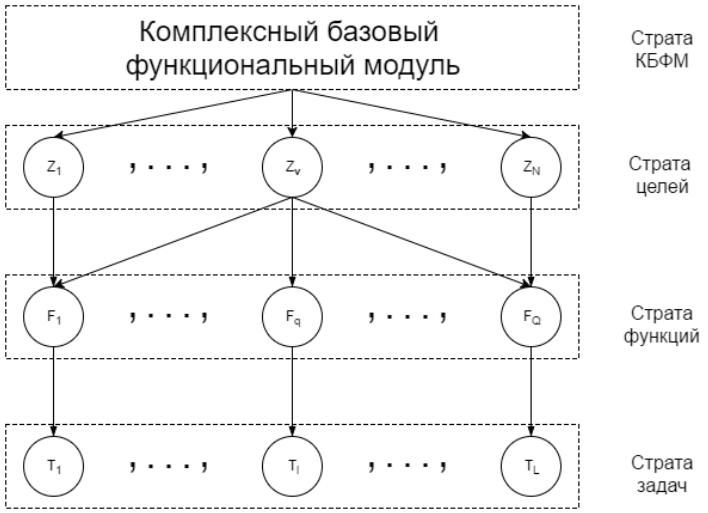

Рисунок 3 – Структура модуля (Цели-Функции-Задачи)

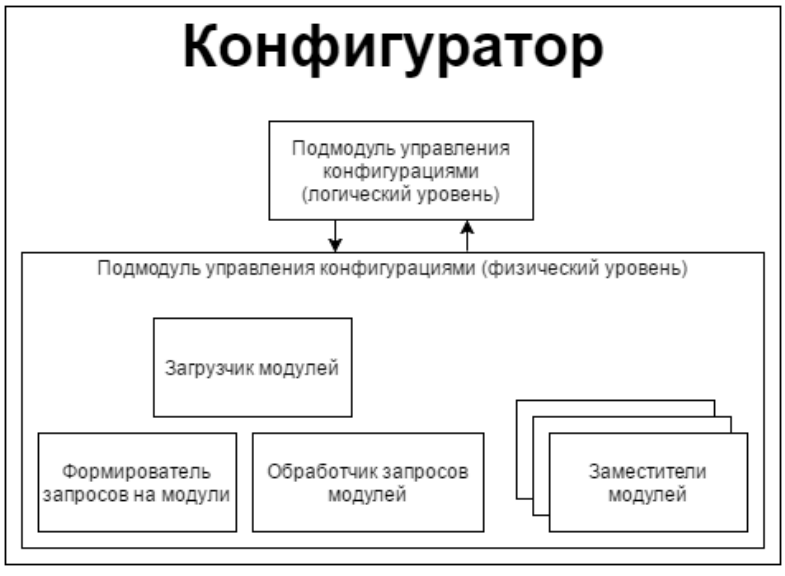

Рисунок 6 – Концептуальная модель конфигуратора

Modelling the configuration subsystem for the intellectual problem solver which utilises conceptual task representation

M.I. Koshkarev, V.V. Nechaev

Abstract – The paper deals with the sub-system of configuration for a predictive intellectual solver based on a conceptual model of the problem to the intelligent system of information support of bionic technology. The described sub-system makes it possible to optimize the use of resources in the process of functioning of the intellectual solver, optimize the execution of tasks, and repair damaged units while working in the network solvers.

Keywords – intellectual problem solver, configuration modelling, conceptual model of a problem, modular architecture.