УДК 004.052, 519.179.2

# Онтологический подход к организации шаблонов требований в рамках системы поддержки формальной верификации распределенных программных систем

*Н.О. Гаранина (Институт систем информатики им. А.П. Ершова),*

*В.Е. Зюбин (Институт автоматики и электрометрии),*

*Т.В. Лях (Институт автоматики и электрометрии)*

В статье описывается структура онтологии шаблонов требований, извлекаемых из текстов технической документации. Эта онтология комбинирует шаблоны известных классификаций требований с новыми шаблонами. Язык онтологии допускает запись булевых комбинаций шаблонов следующих типов: качественных, реального и ветвящегося времени, с комбинированными событиями, количественными характеристиками событий и простыми утверждениями о данных. Приведены примеры требований к реальной системе управления вакуумированием Большого солнечного вакуумного телескопа. Изложена схема интеллектуальной системы поддержки формальной верификации программных распределенных систем.

*Ключевые слова*: шаблоны требований, инженерия требований, темпоральные логики, онтология

## 1. Введение

Данная работа выполняется в рамках проекта “Методы извлечения формальных спецификаций программных систем из текстов технических заданий и их верификация”. Проект посвящён проблеме обеспечения качества программных систем с помощью формальных методов. В рамках проекта планируется разработать комплексный подход к извлечению формальных моделей и свойств распределенных программных систем из текстов технической документации с последующей их верификацией. Под распределенной программной системой (РПС) здесь мы понимаем систему, состоящую из множества параллельно исполняемых взаимодействующих компонент.

Комплексный подход подразумевает создание системы поддержки формальной верификации распределенных программных систем. Известны системы поддержки разработки требований на основе шаблонов [1, 13, 16, 18–20], но они позволяют только ручное

формулирование требований с последующей визуализацией и определением точной формальной семантики. Разрабатываемая нами система предполагает автоматическое порождение требований к системе с последующей ручной коррекцией. Порождаемые требования также имеют формальную семантику, выраженную формулами некоторой темпоральной или модальной логики, определение на естественном языке и визуализацию. В настоящей работе разработана онтология требований и определена их формальная семантика. Онтология требований сочетает известные шаблоны требований, адаптированные и расширенные новыми шаблонами для более выразительного и единообразного представления.

Систематизированное задание требований и их формальной семантики как формул темпоральных логик является хорошо исследованной задачей. Известны различные классификации формул темпоральных логик. Наиболее ранняя классификация [12] является синтаксической и касается самых общих свойств программных систем (живость, безопасность, справедливость, блокировка). Ставшая классической система шаблонов из [4] отражает наиболее характерные качественные требования к системам различного назначения. При этом каждый шаблон описан на естественном языке и дана его формализация на языках CTL, LTL [3], квантифицированных регулярных выражений и графического представления GIL. В работах [8, 11] эти шаблоны были расширены на случай систем реального времени и вероятностных систем соответственно. В работах [13, 16] предложены варианты шаблонов составных событий. В [2] авторы дополняют качественные и временные шаблоны шаблонами, отражающими количественные характеристики появления событий, а также шаблоном данных, впервые упомянутым в [9]. Все упомянутые работы оперируют только шаблонами, выразимыми в логике линейного времени LTL и её реальновременных и вероятностных расширениях, однако в статье [14] отмечается необходимость в некоторых случаях использовать ветвящееся время и логику CTL с соответствующими расширениями. Недавняя работа [1] комбинирует описания классических шаблонов с вероятностными и шаблонами реального времени и даёт их описание на ограниченном английском языке. Такие описания можно использовать для генерации правил извлечения требований из текстов технической документации. В работе [20] классификация шаблонов также представлена в виде онтологии, однако набор шаблонов свойств систем весьма ограничен, а формализация семантики самих паттернов вовсе отсутствует.

В рамках разработки онтологии требований полезно организовать уже известные системы шаблонов в единую онтологическую классификацию, добавив некоторые шаблоны

и области действия, и выработать соответствующую онтологию собственно требований (классы, отношения, домены), наполнение которой уникально для каждого отдельного комплекта технической документации. Такой комплексный систематизированный подход к представлению шаблонов спецификаций позволяет с помощью небольшого набора атрибутов описывать широкий спектр свойств (требований) взаимодействующих параллельных систем. Эта широта важна потому, что для одной и той же системы бывает необходимо задавать как простые, легко верифицируемые свойства типа достижимости выделенного состояния системы, так и сложные свойства, зависящие от реального времени работы компонент системы. Возможность формулировать такие разнообразные свойства в рамках одного формализма повышает качество поддержки процесса разработки таких систем, поскольку позволяет целиком охватывать всю картину требований к системе. Онтология требований будет представлена на языке OWL.

Шаблоны требований нашей онтологии имеют строгую семантику, выраженную формулами темпоральных логик CTL, LTL и их расширений реального времени, а также мю-исчисления (MuC), что позволяет, с одной стороны, чётко выражать требования, а с другой — однозначно определять подходящий метод верификации. Онтология требований допускает расширение свойствами систем, которые не описываются темпоральными логиками, однако опускают эффективную верификацию. Классификация практически-значимых поведенческих свойств распределенных программных систем, используемых в технической документации, которые могут быть эффективно верифицированы в моделях, позволит упростить процедуру проверки соответствия модели РПС и требований к ней. Онтология требований пополняется из текстов технической документации. На настоящий момент она содержит 13 классов требований и 15 отношений между ними с параметрами, конкретные значения которых извлекаются из технической документации.

В данной работе представлена онтология требований, которые могут задаваться булевой комбинацией шаблонов следующих типов: качественных; реального времени; ветвящегося времени; допускающих комбинированные события; позволяющих учитывать количественные характеристики событий, а также простые утверждения о данных. Добавлены шаблоны, задающие свойства оптимальности и устойчивости разрабатываемой системы к нежелательному поведению среды. Следующий раздел 2 описывает классы и отношения онтологии требований с примерами их формальной семантики. В разделе 3 приведены примеры требований к реальной системе управления вакуумированием Большого солнеч-

ного вакуумного телескопа. В следующем разделе 4 изложена схема интеллектуальной системы поддержки разработки программных распределенных систем. В заключении 5 мы обсуждаем дальнейшие планы.

**Благодарности**. Исследование поддержано Российским Фондом Фундаментальных Исследований (грант № 17-07-01600).

## 2. Онтология требований

Мы считаем *онтологией* структуру, включающую следующие элементы: (1) конечное непустое множество *классов*, (2) конечное непустое множество *атрибутов-данных и атрибутов-отношений*, и (3) конечное непустое множество *доменов атрибутов-данных*. Каждый класс определяется набором атрибутов. Атрибуты-данные принимают значения из домена, а значениями атрибутов-отношений являются экземпляры классов. *Экземпляр класса* определяется набором значений атрибутов этого класса. Класс $c_2$ является *подклассом* $c_1$ если и только если все экземпляры класса $c_2$ являются также экземплярами класса $c_1$. Подкласс наследует все атрибуты родительского класса. *Информационный контент* онтологии — это набор экземпляров её классов. Задача пополнения заданной онтологии состоит в извлечении информационного контента этой онтологии из входных данных. В нашем случае такими данными для пополнения онтологии программной системы и онтологии требований является техническая документация. Для пополнения обеих онтологий мы планируем использовать нашу систему семантического извлечения информации из текстов на естественном языке [5–7]. Кроме того, контент онтологии требований также может пополняться из онтологии программной системы с помощью специального транслятора, разработка которого запланирована на ближайшее время.

В следующих подразделах дано подробное описание классов и доменов нашей онтологии требований. В таблицах, перечисляющих атрибуты классов, имена классов выделены курсивом, имена доменов и их значения – телетайпом. Для пояснения семантики требований определим следующие понятия. Мы рассматриваем *модель системы* в духе CSP [10] как параллельное исполнение последовательных процессов, каждый из которых задаётся сменой своих состояний. *Состояния объекта-процесса* определяются набором значений его переменных. Взаимодействие объектов-процессов происходит путём обмена сообщениями и через изменение разделяемых переменных. *Состоянием системы* является набор состояний объектов-процессов, т.е. мгновенное описание значений их перемен-

Table 1

**Спецификации**

| | Operation | S1 | S2 |
|---|---|---|---|
| *SimSpec* | $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ | *Proposition* *SimSpec* | *Proposition* *SimSpec* |
| *Spec* | | *Pattern* *Spec* | *Pattern* *Spec* |

ных. Далее мы будем использовать понятие *события-утверждения* (или просто *события*), которое может иметь место в определённых состояниях системы. В данной статье мы рассматриваем примеры формальной семантики шаблонов, а задача полного описания семантики имеет технический характер.

## 2.1. Спецификации

Спецификации представленные в таб.2.1, задают требования к системе. Простые спецификации требований, задаваемые классом *SimSpec*, являются булевыми комбинациями событий-утверждений, представленных классом *Proposition*, который описан ниже. Эти комбинации строятся с помощью атрибута-данного "Operation", используемого для булевых операторов, и пары атрибутов-отношений "S1" и "S2" для двух операндов. Например, в классе *SimSpec* экземпляр $ss$ = (Operation: $\rightarrow$, S1: $pr_{Start}$, S2: $pr_{AllReady}$), где $pr$ — экземпляры класса *Proposition*, обозначает спецификацию требования $Start \rightarrow AllReady$. Такие простые спецификации используются при задании сложных спецификаций, включающих шаблоны, и ограничений на исполнение простых событий класса *Proposition*. Класс *Spec* является подклассом *SimSpec*. Он представляет спецификации, являющиеся булевой комбинацией простых спецификаций *SimSpec* и шаблонов, представленных классом *Pattern*. Например, в классе *Spec* экземпляр $s$ = (Operation: $\rightarrow$, S1: $pr_{Start}$, S2: $ptU_{AllRight}$), где $pr$ — экземпляр класса *Proposition*, а $pt_U$ — экземпляр класса *Pattern*, обозначает спецификацию требования, которая выражается в логике LTL как $Start \rightarrow \mathbf{G}AllRight$.

Table 2

**Шаблоны**

| *Pattern* | Kind | S1 | S2 | Space | T.Type | Fr.Sco | Fr.Time | Fr.Qnt | Cstr |
|---|---|---|---|---|---|---|---|---|---|
| *Occurence* | `Univ` `Exist` `Abs` | *Prop* | | | `linear` `branch` | *Scope* | *Bound* | *Bound* | *Prop* |
| *Order* | `Prec` `Resp` | | *Prop* | `Novlap` `Rovlap` `Lovlap` | | | | | |

## 2.2. Шаблоны

Шаблоны, представленные в таб.2.2 основаны на классических шаблонах появления *Occurence* и порядка *Order* из [4]. Они снабжены рядом атрибутов-спецификаторов, позволяющих задавать различные дополнительные ограничения, такие как время появления относительно определённых событий-утверждений ("FrameScope"), время выполнения как самого шаблона, так и его событий, по логическим часам ("FrameTime") и количество повторений выполнения самого шаблона, либо его событий ("FrameQuantity"). Шаблоны также могут быть снабжены атрибутом "TimeType" для определения линейного, либо ветвящегося времени шаблона. Утверждения, которые не должны выполняться одновременно с данным шаблоном, задаются значением атрибута "Constraint".

*Шаблоны появления* описывают появление некоторого события в процессе работы системы с помощью класса *Occurence*, который является наследником класса *Pattern* и может использовать все его атрибуты. Событие задаётся атрибутом-отношением "S1" со значением в классе *Proposition*, а тип появления события — атрибутом "Kind"∈{`Absence, Existence, Universality`}. Семантика этих шаблонов для линейного времени и при отсутствии ограничений описывается следующим образом:

- `Universality`: $S_1$ имеет место всегда.
  - LTL: $\mathbf{G}S_1$.
- `Existence`: $S_1$ когда-нибудь случится.
  - LTL: $\mathbf{F}S_1$.
- `Absence`: $S_1$ никогда не случается.
  - LTL: $\mathbf{G}\neg S_1$.

С помощью *шаблонов порядка* можно выразить относительное появление двух событий в процессе работы системы используя класс $Order$, который является наследником класса $Pattern$. Первое событие шаблона задаётся атрибутом-отношением "S1", а второе — атрибутом-отношением "S2" со значениями в классе $Proposition$. Тип порядка задаётся с помощью атрибута "Kind" $\in$ {`Precedence, Response`}. Семантика этих шаблонов для линейного времени и при отсутствии ограничений описывается следующим образом:

- `Precedence`: если случилось $S_1$, то до этого когда-то случилось $S_2$.
  - LTL: $\mathbf{F}S_1 \rightarrow \neg S_1 \mathbf{U}(S_2 \wedge \neg S_1)$.
- `Response`: если выполнено $S_1$, то потом когда-нибудь выполнится $S_2$.
  - LTL: $\mathbf{G}(S_1 \rightarrow \mathbf{F}S_2)$.

Кроме того для шаблонов из $Order$ на события "S1" и "S2" могут накладываться ограничения одновременности выполнения "Space": {`NonOverlap, RightOverlap, LeftOverlap`}:

- `NonOverlap`: $S_1$ и $S_2$ никогда не случаются одновременно.
  - LTL: $\mathbf{G}\neg(S_1 \wedge S_2)$.
- `RightOverlap`: $S_1$ всегда случается раньше $S_2$ и некоторое конечное время они имеют место одновременно.
  - LTL: $\mathbf{G}(S_1 \wedge \neg S_2 \mathbf{U}(S_1 \wedge S_2 \mathbf{U} \neg S_1 \wedge S_2))$.
- `LeftOverlap`: $S_2$ всегда случается раньше $S_1$ и некоторое конечное время они имеют место одновременно.
  - LTL: $\mathbf{G}(S_2 \wedge \neg S_1 \mathbf{U}(S_2 \wedge S_1 \mathbf{U} \neg S_2 \wedge S_1)))$.

Семантика всех предыдущих шаблонов описывалась формулами логики линейного времени LTL, с использованием значения `linear` спецификатора "TimeType". Однако бывает полезно использовать шаблоны семантика которых описывается формулами логики ветвящегося времени CTL, но не формулами LTL [14]. Шаблоны *ветвящегося времени* задаются с помощью значения `branch` спецификатора "TimeType". Их семантика при отсутствии ограничений описывается следующим образом:

- `Universality+branch`: всегда, рано или поздно, есть хотя бы один вариант работы системы, в котором $S_1$ имеет место всегда.
  - CTL: $\mathbf{AFEG}S_1$.
- `Existence+branch`: всегда есть хотя бы один вариант работы системы, в котором $S_1$ когда-нибудь имеет место.
  - CTL: $\mathbf{AGEF}S_1$.

Table 3

**Утверждения-события**

| | | | | |
|---|---|---|---|---|
| *Variable* | Name | Domain | | |
| | `string` | `Dom` | | |
| *Data* | Var1 | Var2 | Opr | Quantifier |
| | *Variable* | *Variable* | $\{=, <, >\}$ | $\forall, \exists$ |
| *Case* | Kind | Name | Condition | |
| | `state` `event` | `string` | *SimpleSpec* | |
| *Complex* | Kind | Type | Cases | |
| | `one` `parallel` `serial` `eventual` | `strict` `free` `hold` | *Data* *Case* | |

- `Absence+branch`: всегда, рано или поздно, есть хотя бы один вариант работы системы, в котором $S_1$ никогда не имеет места.
  – CTL: $\mathbf{AFEG}\neg S_1$.
- `Precedence+branch`: если случилось $S_2$, тогда есть хотя бы один вариант работы системы, при котором $S_1$ случилось раньше $S_2$.
  – CTL: $\mathbf{AF}S_2 \rightarrow (\neg S_2\mathbf{EU}(S_1 \wedge \neg S_2))$.
- `Response+branch`: если случилось $S_1$, тогда есть хотя бы один вариант работы системы, при котором случится $S_2$.
  – CTL: $\mathbf{AG}(S_1 \rightarrow \mathbf{EF}S_2)$.

## 2.3. Утверждения

Экземпляры класса *Proposition* (таб. 2.3) описывают выделенные состояния системы. Семантикой этих экземпляров является то, что утверждения о данных (класс *Data*), события (класс *Case*), либо составные события (класс *Complex*) имеют место в некоторых состояниях системы. Все эти классы являются подклассами класса *Proposition*. Одинаковых наследуемых атрибутов данных эти классы не имеют, но наследуют все отношения родительского класса.

С помощью служебного класса $Var$ задаются переменные системы, значения которых могут определять состояния или события. Константами являются переменные с одноэлементной областью определения. $Data$ — это утверждения о переменных системы, представленными атрибутами отношениями "Var1" и "Var2". Они задают сравнение их значений с помощью атрибута "Opr", возможно, с учётом квантора из значений атрибута "Quantifier". Утверждения об именованных состояниях и событиях системы с помощью класса $Case$, где атрибуты "Kind" и "Name" определяют тип и строковое имя, а атрибут-отношение "Condition" служит для задания условия пуска, представленного экземпляром класса $SimSpec$. Составные события определяются с помощью класса $Complex$, который использует значения атрибута "Kind", задающего вид комбинации событий, и атрибута "Type", задающего её тип, а также атрибута-отношения "Cases", задающего собственно множество комбинируемых событий. Семантику комбинаций событий можно описать следующим образом, основываясь на работах [13, 16]. Пусть $G = \{e_1, \dots e_n\}$ – это множество утверждений о переменных и событиях, представленные экземплярами классов $Data$ и $Case$.

- `one`($G$):
  - `strict`: хотя бы одно из событий множества $G$ имеет место.
    - LTL: $orE = e_1 \vee \dots \vee e_n$.
  - `free`: хотя бы одно из событий множества $G$ будет иметь место.
    - LTL: $notE \wedge notE\mathbf{U}orE$, где $notE = \neg e_1 \wedge \dots \wedge \neg e_n$.
- `parallel`($G$):
  - `strict`: все события множества $G$ имеют место.
    - LTL: $andE = e_1 \wedge \dots \wedge e_n$.
  - `free`: все события множества $G$ будут иметь место.
    - LTL: $notE \wedge notE\mathbf{U}andE$.
- `serial`($G$):
  - `strict`: события множества $G$ имеют место в заданном порядке, по одному в последовательных состояниях.
    - LTL: $xE = e_1 \wedge (\mathbf{X}e_2 \wedge (\mathbf{X}e_3 \dots \wedge \mathbf{X}e_n))$.
  - `hold`: события множества $G$ имеют место в заданном порядке, по одному в последовательных состояниях, и все следующие события не имеют места в этот момент.
    - LTL: $xhE = e_1 \wedge notE_2^n \wedge (\mathbf{X}e_2 \wedge notE_3^n \wedge (\mathbf{X}e_3 \wedge notE_4^n \dots \wedge \mathbf{X}e_n))$, где $notE_i^n =$

$\bigwedge_{j=i}^{n} \neg e_j$.

– `free`: события множества $G$ будут иметь место по одному в заданном порядке.

– LTL: $notE \wedge notE\mathbf{U}xhE$.

- `eventual`($G$):
  - `strict`: события множества $G$ имеют место в заданном порядке, в различных и возможно не последовательных состояниях.
    - LTL: $fE = e_1 \wedge \mathbf{X}(\neg e_2 \mathbf{U} e_2 \wedge \mathbf{X}(\neg e_3 \mathbf{U} e_3 \ldots \wedge \mathbf{X}(\neg e_n \mathbf{U} e_n)))$.
  - `hold`: события множества $G$ будут иметь место в заданном порядке, в различных и возможно не последовательных состояниях, и все следующие события не имеют места в этот момент.
    - LTL: $fhE = e_1 \wedge notE_2^n \wedge (notE_2^n \mathbf{U} e_2 \wedge notE_3^n \wedge (notE_3^n \mathbf{U} e_3 \wedge notE_4^n \ldots \wedge \neg e_n \mathbf{U} e_n))$.
  - `free`: события множества $G$ будут иметь место в заданном порядке, в различных и возможно не последовательных состояниях.
    - LTL: $notE \wedge notE\mathbf{U}fhE$).

Представленное множество шаблонов составных событий мы планируем расширить шаблонами, которые учитывают выполнимость событий после их однократного выполнения.

## 2.4. Границы

Экземпляры классов, приведённых в таблице 2.4, задают ограничения на выполнение шаблонов относительно событий (класс *Scope*), временных рамок (класс $Time$) и количества выполнений ($Quantity$).

Класс *Scope* определяет область действия шаблонов, используя атрибуты-отношения "S1" и "S2" для спецификации событийных границ, а тип ограничений задаётся атрибутом "Kind"$\in$ `Scopes ={globally, before, after, between, after-until, start, regular, final}`. Семантика событийных границ выполнения шаблонов относительно событий основана на работах [4, 17]. Приведём семантику событийных границ для шаблона универсальности, когда событие $S$ выполняется всегда:

- `globally`: шаблон выполняется в течении всего времени работы системы.
  - LTL: $\mathbf{G}S$.
- `before`: шаблон выполняется в течении всего времени работы системы до первого появления события $S_1$.
  - LTL: $\mathbf{F}S_1 \rightarrow S\mathbf{U}S_1$.

Table 4

**Границы**

| *Scope* | Kind | Space | S1 | S1 |
|---|---|---|---|---|
| | `Scopes` | `Overlapping` | *Props* | *Props* |
| *Bound* | Kind | Bound1 | Bound2 | Bound3 |
| | `Situation` | *Quantity* *Time* | *Quantity* *Time* | *Quantity* *Time* |
| *Quantity* | Kind | Num1 | Num2 | |
| | `point` `minimum` `maximum` `interval` | `Integer` | `Integer` | |
| *Time* | Duration | Periodic | | |
| | *Quantity* | *Quantity* | | |

- `after`: шаблон выполняется выполняется в течении всего времени работы системы после первого появления события $S_1$.
  – LTL: $\neg S\mathbf{U}(S_1 \wedge \mathbf{G}S)$.
- `between`: шаблон выполняется между первым появлением события $S_1$ и следующим после этого появлением события $S_2$.
  – LTL: $\neg S\mathbf{U}(S_1 \wedge \neg S_2 \wedge (S\mathbf{U}S_2))$.
- `after-until`: шаблон выполняется выполняется между первым появлением события $S_1$ и следующим после этого появлением события $S_2$ либо в течении всего последующего времени работы системы.
  – LTL: $\neg S\mathbf{U}(S_1 \wedge \neg S_2 \wedge (S\mathbf{W}S_2))$.
- `start`: шаблон выполняется, пока событие $S_1$ не обозначит конец начальной фазы работы системы
  – LTL: $\neg S_1 \rightarrow S\mathbf{W}S_1$.
- `regular` шаблон всегда выполняется выполняется между появлением события $S_1$ и следующим после этого появлением события $S_2$ либо в течении всего последующего времени работы системы.
  – LTL: $\mathbf{G}(S_1 \wedge \neg S_2 \rightarrow (S\mathbf{W}S_2))$.

- `final` шаблон выполняется после того, как событие $S_1$ обозначит начало финальной фазы работы системы.
  - LTL: $(\mathbf{FG}S_1 \wedge \mathbf{F}S) \rightarrow \neg S\mathbf{U}S_1)$.

Для дополнительных ограничений на одновременное исполнение событийный границ "S1", "S2" и событий шаблона используется атрибут "Space"$\in$ `Overlapping` $= \{$`NonOverlap`$(i,j)$, `RightOverlap`$(i,j)$, `LeftOverlap`$(i,j) \mid i,j \in \{S_{1^s}, S_{2^s}, S_{1^p}, S_{2^p}\}\}$, где $S_{1^s}$ и $S_{2^s}$ — событийные границы, а $S_{1^p}$ и $S_{2^p}$ — события шаблона, и семантика конструкций из `Overlapping` аналогична семантике ограничений одновременности выполнения шаблонов "Space" из предыдущего раздела.

В отличие от событийных границ, которые накладываются на исполнение шаблона в целом, временные и количественные границы могут определяться как для шаблона в целом, так и для его событий. Для спецификации этого используется класс *Bound*, где атрибут "Kind"$\in$ `Situation` $= \{$`whole`, `S1`, `S2`, `whole+S1`, `whole+S2`, `S1+S2`, `whole+S1+S2`$\}$ определяет сочетание накладываемых ограничений, а сами ограничения задаются атрибутами-отношениями "Bound1", "Bound2" и "Bound3", и могут быть как количественными (класс *Quantity*), так и временными (класс $Time$).

Число появлений шаблона или события задаётся экземпляром класса *Quantity* с помощью атрибута-спецификатора "Kind" и натуральных числовых атрибутов "Num1" и "Num2". Приведём семантику ограничений на число появлений при Num1=2 и Num2=3 (легко обобщается на произвольные натуральные числа) для шаблона существования, когда событие $S$ выполнится когда-нибудь:

- `point` (ровно Num1 раз):
  - $S$ выполнится когда-нибудь ровно 2 раза.
  - LTL: $Pt_2 = Now_2 \vee (\neg S\mathbf{U}Now_2)$, где $Now_2 = S \wedge \mathbf{X}Pt_1$ и $Pt_1 = Now_1 \vee \neg S\mathbf{U}Now_1$, а $Now_1 = S \wedge \mathbf{XG}\neg S$;
- `minimum` (хотя бы Num1 раз):
  - $S$ выполнится когда-нибудь хотя бы 2 раза.
  - LTL: $Min_2 = \mathbf{F}(S \wedge \mathbf{XF}S)$;
- `maximum` (не более Num1 раз):
  - $S$ выполнится когда-нибудь не более 2 раз.
  - LTL: $Max_2 = G\neg S \vee Pt_1 \vee Pt_2$;
- `interval` (от Num1 до Num2 раз):

– $S$ выполнится когда-нибудь не более 3 и не менее 2 раз.

– LTL: $Int_{2,3} = Min_2 \wedge Max_3$.

Класс *Time* позволяет задавать временные границы (атрибут-отношение “Duration” с классом *Quantity*) и временную периодичность (атрибут-отношение “Periodic” с классом *Period*) появления шаблона или события. Приведём семантику временных ограничений, выраженную формулами логики MTL [15], для шаблона существования, когда событие $S$ выполнится когда-нибудь:

- Duration:
  - `point` (ровно через Num1):
    - $S$ выполнится через $Num_1$ единиц времени.
    - MTL: $\neg S \mathbf{U}_{Num_1} S$;
  - `minimum` (минимум через Num1):
    - $S$ выполнится не раньше $Num_1$ единиц времени.
    - MTL: $Min^d_{Num_1}(S) = \neg S \mathbf{U}_{Num_1}(\neg S \rightarrow \mathbf{F} S)$,
  - `maximum` (максимум через Num1):
    - $S$ выполнится не позже $Num_1$ единиц времени.
    - MTL: $Max^d_{Num_1}(S) = \mathbf{F}_{Num_1} S$.
  - `interval` (между Num1 и Num2):
    - $S$ выполнится не раньше $Num_1$ и позже $Num_2$ единиц времени.
    - MTL: $Min^d_{Num_1}(S) \wedge Max^d_{Num_2}(S)$.
- Periodic:
  - `point` (ровно каждые Num1):
    - $S$ выполняется каждые $Num_1$ единиц времени.
    - MTL: $\mathbf{G}(\neg S \mathbf{U}_{Num_1}(S \wedge \mathbf{X} \neg S))$.
  - `minimum` (минимум каждые Num1):
    - $S$ выполняется не реже каждых $Num_1$ единиц времени.
    - MTL: $Min^p_{Num_1}(S) = \mathbf{GF}_{Num_1} S$,
  - `maximum` (максимум каждые Num1):
    - $S$ выполняется не чаще каждых $Num_1$ единиц времени.
    - MTL: $Max^p_{Num_1}(S) = \mathbf{G}(\neg S \mathbf{U}_{Num_1}(\neg S \rightarrow \mathbf{F}(S \wedge \mathbf{X} \neg S)))$.
  - `interval` (между Num1 и Num2):
    - $S$ выполняется не реже $Num_2$ и не чаще $Num_1$ единиц времени.

Table 5

**Среда**

| *Environment* | Kind | Env | Sys |
|---|---|---|---|
| | `{BadBeh,Opt}` | *Spec* | *Spec* |

– MTL: $G(Min^{p}_{Num_2}(S) \wedge Max^{p}_{Num_2}(S))$.

Отметим, что могут быть одновременно заданы все ограничения, как событийные, так и количественные с временными.

## 2.5. Окружение

Класс *Environment* позволяет моделировать отношения поведений системы и её окружения. Атрибут “Kind” задаёт вид этих отношений, а атрибуты “Env” и “Sys” — поведение окружения и системы, соответственно, со следующей семантикой:

- `BadBeh` (плохое поведение): если окружение всегда может следовать плохой спецификации $Env$, то система всё равно следует хорошей спецификации $Sys$.
  – CTL: $\mathbf{EF}Env \wedge \mathbf{AG}Sys$.
- `Opt` (оптимальность): как бы ни вело себя окружение, система может реагировать так, что оптимальное спецификация $Sys$ для системы и спецификация $Env$ для окружения будут достигнуты
  – Свойство невыразимо в логиках CTL или LTL, но выразимо в $\mu-$исчислении. Доказательство этого факты выходит за рамки данной работы.

Неформально говоря, первый тип отношений окружения и системы полезен, чтобы убедиться, что определённый (нежелательный) сценарий поведения окружения действительно смоделирован, но ошибок системы при этом не возникает. Второй тип, выражающий свойство оптимального поведения системы заключается в том, что при любом поведении окружения система может достигнуть нужного состояния.

Заметим, что не всякая комбинация значений атрибутов классов построенной онтологии имеет смысл. Например, шаблон *ограниченной универсальности* $BU$ события $P$ мог бы быть специфицирован следующим образом: $BU = Occurrence(Kind : universality, S1 : P, TimeType : linear, FrameQuantity : (Kind : whole, Bound3 : (Kind : point, Num1 = 5)))$, но это понятие не имеет осмысленной семантики.

Онтология требований может быть представлена в графическом виде. На рис.1 классы

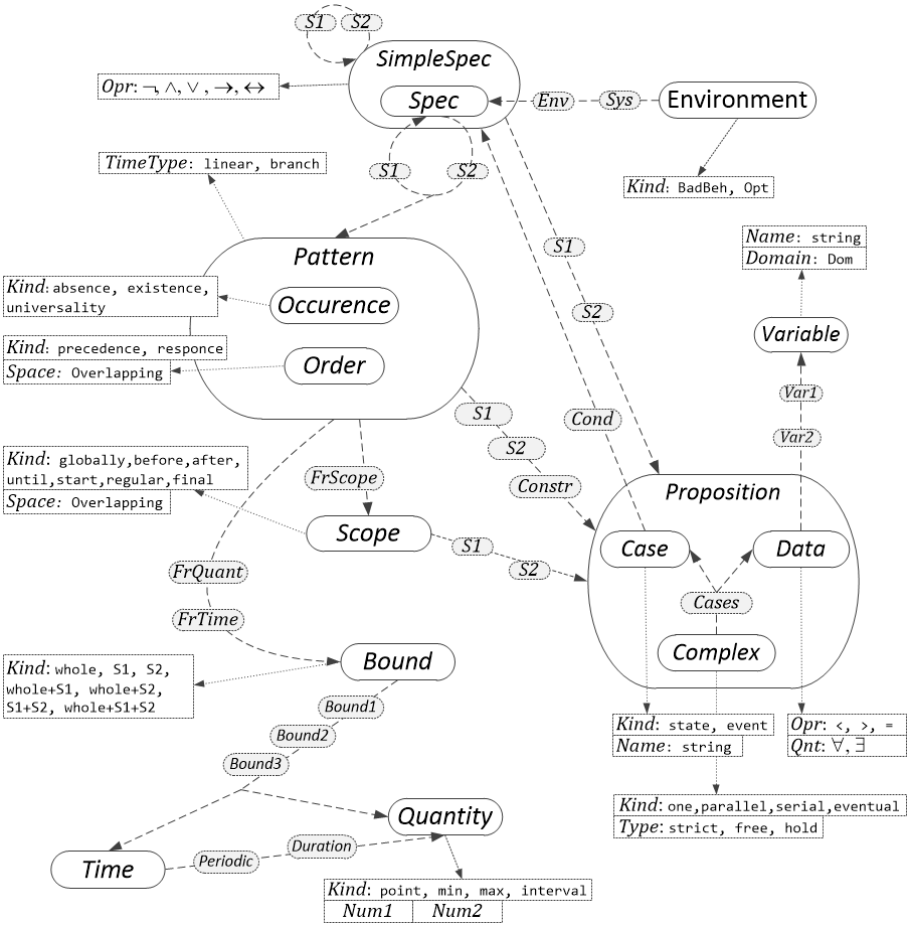


*Рис. 1.* Онтология требований

представлены в виде белых овалов и их подклассы находятся внутри них. Отношения между классами помечены штриховыми стрелками, а названия отношений помещены внутри серых овалов. Атрибуты классов располагаются в прямоугольниках и связаны с соответствующими классами штрихпунктирными стрелками.

## 3. Примеры требований

Приведём примеры требований к автоматической системе управления вакуумированием Большого солнечного вакуумного телескопа (БСВТ)[21].

Система вакуумирования БСВТ (рис. 2) содержит следующие компоненты: 1) труба телескопа, 2) пневмоустройство, 3) датчик давления в трубе телескопа, 4) клапан подключения вакуумного насоса к трубе телескопа, 5) датчик давления в патрубке вакуумного насоса, 6) отсечной клапан соединения вакуумного насоса с атмосферой (сапун), 7) датчик температуры воды в рубашке охлаждения вакуумного насоса, 8) вентилятор, 9) вакуумный насос, 10) датчик температуры воды в системе климат-контроля, 11) насос охлажде-

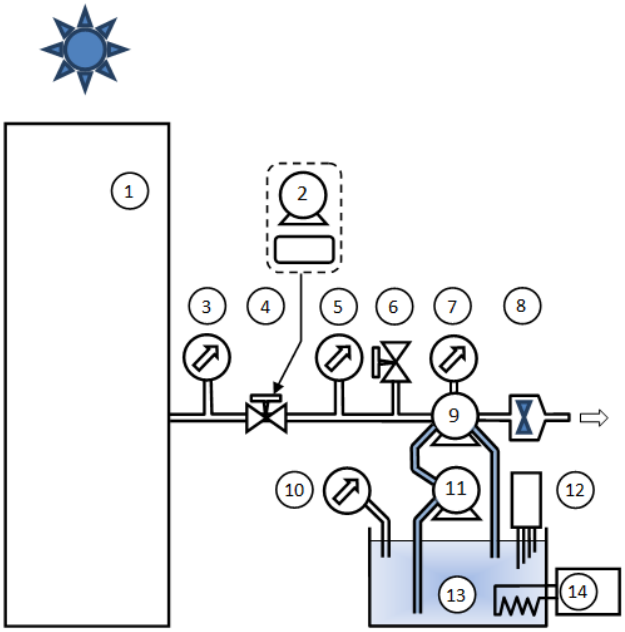


*Рис. 2.* Подсистема вакуумирования БСВТ

ния, 12) датчики уровня воды в системе климат-контроля, 13) система климат-контроля, 14) нагреватель воды в системе климат-контроля.

Описание требований в нашей работе включает в себя формулирование их на естественном языке (RUS), с помощью темпоральных логик (LTL, MTL) и в виде экземпляра онтологии шаблонов требований (ONT). Если область действия шаблона явно не задана, то мы считаем, что шаблон выполняется всегда.

**Пример 1**. Требование отсутствия составного события.

RUS: *S1 = Никогда одновременно не включается насос P9 и не открывается затворный клапан V4.*

Пусть состояния системы, в которых включён насос, обозначаются с помощью события $Pump$, выполняющегося в них, а открывание затворного клапана описывается исполнением одного из событий: вызова открытия $V4.in$ и собственно открытия $V4.out$.

LTL: $S1_F = \mathbf{G}\neg(Pump \wedge V4.in \wedge V4.out)$.

ONT: $S1_O = Occurrence(\text{Kind: } \texttt{Absence}, \text{S1: } E_1, \text{TimeType: } \texttt{linear})$, где

- составное событие $E_1 = Complex$ (Kind: `strict`, Type: `parallel`, Cases: $\{Pump, V4.in, V4.out\}$), при этом
  - каждое из простых событий $e \in \{Pump, V4.in, V4.out\}$ – это экземпляр $e = Case(\text{Kind: } \texttt{event}, \text{Name: } e)$.

В этих записях экземпляров и далее опущены имена атрибутов, значения которых не

заданы. Далее будем обозначать экземпляры простых событий именами этих событий.

**Пример 2**. Требование появления события между двумя другими событиями.

RUS: *Между сигналами датчика уровня воды S12 "Воды недостаточно" и "Воды достаточно" всегда посылается сигнал отключения нагревателя H14.*

Пусть состояния системы, в которых датчик S12 обнаруживает, что воды недостаточно, обозначаются с помощью события $S12.low$, если же воды достаточно, то $S12.norm$, а сигнал отключения нагревателя H14 описывается исполнением события $H14.tooff$.

LTL: $S2_F = \mathbf{G}(\neg H14.tooff \rightarrow$

$(\neg H14.tooff \mathbf{U}(S12.low \wedge \neg S12.norm \rightarrow (\neg S12.norm \mathbf{U} H14.tooff \wedge \neg S12.norm))))$.

ONT: $S2_O = Occurrence$

(Kind: `Existence`, S1: $H14.tooff$, TimeType: `linear`, FrameScope: $Sc_2$), где

- область действия $Sc_2 = Scope$(Kind: `after-until`, S1: $S12.low$, S2: $S12.norm$).

**Пример 3**. Требование реакции на событие в течение заданного времени.

RUS: *Если пришёл сигнал на клапан V5, то он откроется до истечения таймаута TV5.*

Пусть состояния системы, в которых приходит сигнал на клапан V5, обозначаются с помощью события $V5.toOpen$, а в которых этот клапан открыт – $V5.Open$.

MTL: $S3_F = \mathbf{G}(V5.toOpen \rightarrow \mathbf{F}_{\leq TV5} V5.Open)$.

ONT: $S3_O = Order$

(Kind: `Response`, S1: $V5.toOpen$, S2: $V5.Open$, TimeType: `linear`, FrameTime: $B_3$), где

- ограничение по времени, в течении которого должно выполниться второе событие шаблона $B_3 = Bound$(Kind: `S2`, Bound2: $T_3$), где
  - время ограничения $T_3 = Time$(Duration: $Q_3$) с
    - количественными характеристиками времени

$Q_3 = Quantity$(Kind: `maximum`, Num1: $TV5$).

**Пример 4**. Требование однократной реакции на составное событие в течение заданного времени.

RUS: *Всегда при нормальной работе вакуумного насоса P9, если падение давления в трубе сидеростата (S4) меньше уровня D04 за время ToutD4, и при этом давление в трубе сидеростата (S4) все еще больше критического уровня давления P04, не более, чем за время ToutM4 генерируется сообщение об ошибке. Это сообщение генерируется только один раз.*

Обозначим состояния системы, в которых насос работает нормально, как $PumpNorm$,

изменение давления и значение давления в трубе сидеростата S4 задаётся условно-целочисленными переменными $D4$ и $P4$, локальный счётчик времени — целочисленной переменной $T4$, а событие генерирования сообщения об ошибке обозначается как $AlarmPS4$.

MTL: $S4_F = \mathbf{G}(PumpNorm \wedge T4 > ToutD4 \wedge D4 < D04 \wedge P > P04 \rightarrow \mathbf{F}_{\leq ToutM4} AlarmPS4 \wedge \neg AlarmPS4 \mathbf{U}(PumpNorm \wedge T4 > ToutD4 \wedge D4 < D04 \wedge P > P04))$.

ONT: $S4_O = Order$(Kind: `Response`, S1: $E_4$, S2: $AlarmPS4$, TimeType: `linear`, FrameTime: $Bt_4$, FrameQuantity: $Bq_4$), где

- составное событие $E_4 = Complex$ (Kind: `strict`, Type: `parallel`, Cases: $\{PumpNorm, D_1, D_2, D_3\}$), где
  - утверждения о данных:

    $D_1 = Data$(Var1: $T4$, Var2: $ToutD4$, Opr: `>`),

    $D_2 = Data$(Var1: $D4$, Var2: $D04$, Opr: `<`), $D_3 = Data$(Var1: $P$, Var2: $P04$, Opr: `>`),

    где переменные и константы:

    $T4 = Variable$(Name: $T4$, Domain: $int$),

    $ToutD4 = Variable$(Name: $ToutD4$, Domain: $\{ToutD4\}$),

    $D4 = Variable$(Name: $D4$, Domain: $int$),

    $D04 = Variable$(Name: $D04$, Domain: $\{D04\}$),

    $P = Variable$(Name: $P$, Domain: $int$),

    $P04 = Variable$(Name: $P04$, Domain: $\{P04\}$);
- ограничение по времени, в течении которого должно выполниться второе событие шаблона, $Bt_4 = Bound$(Kind: `S2`, Bound2: $T_4$), где
  - время ограничения $T_4 = Time$(Duration: $Qt_4$) с
    - количественными характеристиками времени

      $Qt_4 = Quantity$(Kind: `maximum`, Num1: $ToutM4$),
- количественные характеристики исполнения второго события шаблона

  $Bq_4 = Bound$(Kind: `S2`, Bound2: $Qq_4$) с $Qq_4 = Quantity$(Kind: `point`, Num1: 1).

## 4. Концепция интеллектуальной системы поддержки разработки распределенных программных систем.

Интеллектуальная система поддержки формальной верификации распределенных программных систем включает следующие компоненты.

1. Системы семантического извлечения информации (СИИ).

2. Онтология программной системы (ОПС).
3. Онтология требований (ОТ).
4. Редактор онтологий ОПС и ОТ.
5. Транслятор содержания ОПС в язык подходящего инструмента верификации.
6. Модуль извлечения требований из ОПС.
7. Транслятор содержания ОТ в высказывания на естественном языке (ТЯ).
8. Транслятор содержания ОТ в графическое представление (ТГ).
9. Транслятор содержания ОТ в формулы логики спецификаций (ТЛ).
10. Онтология логик спецификаций (ОЛС).
11. Инструмент верификации шаблонов требований (ВШ).

Наша система семантического извлечения информации (1) позволяет извлекать данные из текстов на естественном языке в виде экземпляров онтологии предметной области [5–7]. Онтология программной системы (2) содержит описание программной системы как параллельно взаимодействующих последовательных процессов, временных и причинно-следственных отношений между ними. Онтология требований (3) содержит описание требований, относящихся к корректности программной системы. Содержание онтологии ОПС извлекается из технической документации с использованием СИИ, а содержание онтологии ОТ — с помощью модуля извлечения требований из ОПС и из технической документации с использованием СИИ. Транслятор содержания ОПС (5) в язык инструмента верификации служит для задания программной системы на входном языке выбранного инструмента верификации. Модуль извлечения требований (6) из ОПС, основываясь на причинно-следственных и временных отношениях процессов, описанных в ОПС, и онтологии ОТ, извлекает типичные требования корректности. Трансляторы требований ТЯ и ТГ (7,8), представляющие экземпляры ОТ как высказывания на ограниченном подмножестве естественного языка и в графическом виде, служат для облегчения задачи понимания и спецификации требований. Планируется также разработка редакторов результата трансляции и обратных трансляторов в ОТ. Транслятор требований ТЛ (9) в формулы логики спецификаций определяет формальную семантику требований. Формальная семантика позволяет выбрать способ и инструмент верификации требований к программной системе, информация о которых содержится в онтологии логик спецификаций ОЛС (10). В отличие от других онтологий нашей системы, онтология ОЛС пополняется вручную и содержит сведения о темпоральных и модальных логиках спецификаций,

отношениях между ними и известных инструментах верификации. Инструмент верификации шаблонов требований ВШ (11) использует алгоритмы верификации шаблонов, обладающие меньшей трудоёмкостью, чем стандартные алгоритмы верификации общего вида. В силу неоднозначности естественного языка и трудоёмкости точного задания требований корректности программных систем все инструменты системы поддержки формальной верификации, кроме верификатора шаблонов, не являются полностью автоматическими, т.е. результат их работы может требовать дополнительного ручного анализа.

## 5. Заключение

В работе предложен начальный вариант онтологии требований, которая допускает описание требований следующего типа: качественные; реально-временные; количественные; учитывающие составные события, а также утверждения о данных. Эта онтология может быть расширена несколькими способами. Независимым от предметной области способом является описание не только шаблонов комбинаций событий, но и обобщение этих комбинаций до поведения в духе CSP, предложенное в [19]. Кроме того, специализированные предметные области, такие как безопасность, агентные модели, могут потребовать свои шаблоны спецификаций.

Очевидным следующим шагом является задание полной формальной семантики шаблонов. Особого внимания требует формальная семантика шаблонов, содержащих в том или ином виде оператор Until и его варианты, т.к. можно получить некорректную спецификацию за счёт несоответствия правых границ времён появления событий. Эта семантика будет использоваться в трансляторе содержания ОТ в формулы логики спецификаций. Трансляторы в высказывания на естественном языке и в графическое представление зависят от формальной семантики и должны разрабатываться после её полного определения. Отметим, что за счёт неоднозначности естественного языка возможно неоднозначное соответствие шаблонов и результатов их трансляции в естественный язык. Представление шаблонов требований с помощью графических формальных языков, например GIL, может помочь выявить и исправить некорректные требования. Разработка модуля извлечения требований из онтологии программной системы и инструмента верификации шаблонов требований также зависит от формальной семантики шаблонов. Независимой задачей является построение онтологии логик спецификаций.

## Список литературы

1. **Autili M., Grunske L., Lumpe M., Pelliccione P., Tang A.** *Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar* // IEEE Transactions on Software Engineering, Volume: 41, Issue: 7, July 1 2015, P. 620–638.
2. **Bianculli D., Ghezzi C., Pautasso C., Senti P.** *Specification Patterns from Research to Industry: A Case Study in Service-Based Applications* // Proc. of 34th International Conference on Software Engineering (ICSE), 2012, P. 968–976.
3. **Clarke E.M., Grumberg O., Peled D.** *Model Checking.* // MIT Press, 1999. 324 p.
4. **Dwyer M. B., Avrunin G. S., Corbett J. C.** *Patterns in property speciÏcations for finite-state verification* // Proc. of the 21st Int. Conf. on Software Engineering, IEEE Computer Society Press, 1999, P. 411–420.
5. **Garanina N., Sidorova E., Bodin E.** *A Multi-agent Text Analysis Based on Ontology of Subject Domain* // In: Perspectives of System Informatics. LNCS Vol .8974, 2015, P. 102-110.
6. **Garanina N., Sidorova E.** *Context-dependent Lexical and Syntactic Disambiguation in Ontology Population* // Proc. of the 25th International Workshop on CS&P. Rostock, Germany, Sep. 28-30, 2016. – Humboldt-Universitat zu Berlin, 2016, P. 101–112.
7. **Garanina N., Sidorova E., and Kononenko I.** *A Distributed Approach to Coreference Resolution in Multiagent Text Analysis for Ontology Population* // Springer International Publishing AG 2018 A. K. Petrenko and A. Voronkov (Eds.): PSI 2017, LNCS 10742, P. 1–16, 2018.
8. **Grunske L.** *Specification patterns for probabilistic quality properties* // Proc. of the 30th international conference on Software engineering. - ICSE '08. - New York, NY, USA: ACM, 2008. - P. 31-40.
9. **Halle S., Villemaire R., Cherkaoui O.** *Specifying and validating data-aware temporal web service properties* // IEEE Trans. Softw. Eng., 2009, vol. 35, no. 5, P. 669-–683.
10. **Hoare, C. A. R.** *Communicating sequential processes* // Communications of the ACM. 21 (8): P. 666–677
11. **Konrad S., Cheng B. H. C.** *Real-time specification patterns* // Proc. of the 27th International Conference on Software Engineering. ACM, 2005, P. 372—381.
12. **Manna Z., Pnueli A.** *The Temporal Logic of Reactive and Concurrent Systems* // Springer-Verlag, 1991. 427 p.
13. **Mondragon O., Gates A. Q., Roach S.** *Prospec: Support for Elicitation and Formal Specification of Software Properties* // Proc. of Runtime Verification Workshop, ENTCS, Vol. 89, Elsevier, 2004. P. 67–88.
14. **Post A., Menzel I., Podelski A.** *Applying restricted english grammar on automotive requirements: does it work? A case study* // Proc. of 17th international working conference on Requirements engineering: foundation for software quality. Vol. 6606 of LNCS. Berlin, Heidelberg: Springer-Verlag, 2011. P. 166–180.
15. **Koymans R.** *Specifying Real-Time Properties with Metric Temporal Logic* // Real-Time Systems, November 1990, Volume 2, Issue 4, P. 255–299
16. **Salamah S., Gates A. Q., Kreinovich V.** *Validated templates for specification of complex LTL*

*formulas* // J. of Syst. and Soft., 2012, V. 85, n. 8. P. 1915–1929.

17. **Shoshmina I. V** *Developing formal temporal requirements to distributed program systems* // Proc. of Seven Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV 2016) June 14-15, 2014 in St. Petersburg, Russia. - System Informatics. - 2016. - N.8. - P. 21-31.
18. **Smith M.H., Holzmann G.J., Etessami K.** *Events and Constraints: A Graphical Editor for Capturing Logic Requirements of Programs* // Proc. of Proceedings Fifth IEEE International Symposium on Requirements Engineering 27-31 Aug. 2001, P. 14-22
19. **Wong P.Y.H., Gibbons J.** *Property Specifications for Workflow Modelling* // Proc. of Integrated Formal Methods. IFM 2009. Lecture Notes in Computer Science, vol 5423. Springer, Berlin, Heidelberg
20. **Yu J., Manh T. P., Han J. et al.** *Pattern based property specification and verification for service composition* // Proc. of 7th International Conference on Web Information Systems Engineering (WISE). Vol. 4255 of LNCS. Springer-Verlag, 2006. P. 156–168.
21. **Лях Т.В., Зюбин В.Е., Сизов. М.М.** *Опыт применения языка Reflex при автоматизации Большого солнечного вакуумного телескопа* // Журнал «Промышленные АСУ и контроллеры». 2016. №7. С. 37-43.