Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 607–622
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 607–622

DOI: 10.18255/1818-1015-2018-6-607-622

УДК 004.052, 519.179.2

Онтология процессов, ориентированная на верификацию

Гаранина Н. О.[1], **Ануреев И. С.**, **Боровикова О. И.**

Поступила в редакцию 20 июля 2018
После доработки 10 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. В статье представлена онтология процессов, близких взаимодействующим последовательным процессам Хоара. Она является частью интеллектуальной системы поддержки верификации свойств поведения таких процессов. Наше онтологическое представление процессов ориентировано как на применение формальных методов верификации, так и на извлечение информации из технической документации (с помощью нашей ранее разработанной системы извлечения информации из текстов на естественном языке). Мы описываем классы и домены онтологии, которые определяют взаимодействующие процессы. Эти процессы характеризуются множествами локальных и разделяемых переменных, списком действий над этими переменными, которые изменяют их значения, списком каналов взаимодействия процессов (которые, в свою очередь, характеризуются типом чтения сообщений, емкостью, способами записи и считывания, а также надежностью), списком коммуникационных действий для отправки сообщений. Помимо формального математического определения классов и доменов онтологии, приведены примеры описаний некоторых онтологических классов, а также типовых свойств и аксиом для них в редакторе Protégé на языке OWL с использованием правил вывода на языке SWRL. Для онтологического представления взаимодействующих процессов определяется их формальная операционная семантика, которая задается с использованием помеченной системы переходов. В интерливинговой модели она сводится к локальной операционной семантике отдельных экземпляров процессов. Представлена специализация онтологии для некоторых типов процессов из предметной области систем автоматического управления, моделирующих типовые функциональные элементы системы автоматического управления (датчики, сравнивающие устройства и регулирующие устройства), а также их комбинации. Понятия специализированной онтологии иллюстрируются на примере управляющей части системы розлива бутылок.

Ключевые слова: онтология, верификация, проверка моделей, процессы

Для цитирования: Гаранина Н. О., Ануреев И. С., Боровикова О. И., "Онтология процессов, ориентированная на верификацию", *Моделирование и анализ информационных систем*, **25**:6 (2018), 607–622.

Об авторах: Гаранина Наталья Олеговна, orcid.org/0000-0001-9734-3808, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: garanina@iis.nsk.su

Ануреев Игорь Сергеевич, orcid.org/0000-0001-9574-128X, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН, e-mail: anureev@iis.nsk.su

Боровикова Олеся Игнатьевна, orcid.org/0000-0001-5456-8513, мл. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН, e-mail: olesya@iis.nsk.su

Благодарности:
[1] Работа поддержана грантом РФФИ 17-07-01600.

Введение

Наша долгосрочная цель — комплексный подход к обеспечению качества систем взаимодействующих последовательных процессов (далее систем процессов) с помощью формальных методов, который включает извлечение формальных моделей и свойств систем из текстов технической документации, а также их верификацию. Наш подход ориентирован на использование проверки моделей (впоследствии и дедуктивных методов) в качестве средств формальной верификации. Недостатком существующих систем поддержки разработки и верификации требований является то, что они предлагают только ручную формулировку требований и описаний систем процессов [1, 15, 19, 20].

Разрабатываемое нами интеллектуальное средство поддержки формальной верификации систем процессов будет автоматизированным образом извлекать и порождать как формальное описание системы, так и требования к ней. В этой статье мы предлагаем онтологию систем процессов. Другим ключевым компонентом системы является онтология шаблонов требований, предложенная в [5]. Содержание этих онтологий, т. е. множества экземпляров их классов, является онтологическими описаниями, соответственно, некоторой системы процессов и требований к ней. Эти описания извлекаются из корпуса технической документации с помощью наших методов извлечения информации из текстов на естественном языке [2–4]. Набор описаний требований также может быть расширен описаниями типичных требований к корректности системы, автоматически порождаемых из ее онтологического описания.

Онтологические описания системы и требований являются основой для ее формальной верификации. Чтобы верифицировать систему процессов, необходимо сначала выбрать подходящий верификатор, основанный на проверке моделей, с учетом формальной семантики онтологического представления требований к онтологии. Если он существует, онтологическое описание системы транслируется в язык спецификации модели, а описание требований — в язык спецификации свойств (обычно это язык темпоральной логики), которые являются входными языками этого верификатора. Если подходящего верификатора не существует, разумно разработать специальный верификатор шаблонов требований на основе формальной семантики соответствующей онтологии, который использует специализированные упрощенные алгоритмы верификации. Если семантика онтологических представлений модели системы и требований к ней, а также входные языки средства верификации строго определены и обеспечена корректность трансляции этих представлений во входные данные верификатора, корректность проверки требований гарантируется.

Работа с требованиями предполагает не только формальную семантику шаблонов требований, но и представление этих шаблонов как на естественном языке, так и в графической форме. Эти представления важны, поскольку из-за неоднозначности естественного языка извлеченные и порожденные требования могут не соответствовать ожиданиям инженера требований и может потребоваться их ручная корректировка. Онтологии систем процессов и шаблонов требований с их формальной

семантикой являются основой для развития системы поддержки формальной верификации. Наш подход позволяет моделировать и верифицировать широкий спектр систем: от программных распределенных систем до бизнес-процессов. Основным ограничением подхода является сложность и, временами, невозможность *точного* извлечения информации из технической документации, и извлеченные модели систем и требований могут потребовать ручной коррекции.

Наша онтология систем процессов задает набор процессов. Известные подходы к онтологическому описанию процессов обычно решают задачи моделирования. Онтологии PSL [16], Cyc [18], SBPM [6, 7] и GFO [9] фокусируются на задачах в области планирования производства, планирования процессов, управления рабочим процессом, реинжиниринга бизнес-процессов, симуляции, реализации процессов, моделирования процессов и управления проектами. Важным преимуществом этих подходов является то, что они учитывают концепции предметных областей при описании производственных, инженерных и бизнес-процессов. Указание конкретного времени и места событий и действий, связанных с этими процессами, в некоторых из этих подходов позволяет описать онтологии планирования, представляющие абстрактные и исполняемые планы организации процессов, что является еще одним достоинством. Однако эти преимущества делают очень сложной верификацию логики поведения процессов, лежащих в основе этих онтологий. Тесная взаимосвязь предметно-ориентированных понятий с логикой поведения процессов приводит к усложнению извлечения последней для использования в системах верификации, основанных на проверке моделей. Более того, извлеченная логика поведения по-прежнему должна транслироваться на язык системы верификации, основанный, как правило, на системах переходов, которые работают с понятиями состояния и перехода. В частности, для онтологий планирования конкретное время и место событий и действий нужно уметь выражать в терминах состояний. Кроме того, эти онтологии не имеют формальной семантики для логики поведения процессов, что еще более усложняет задачу.

В этой статье в разделе 1. мы определяем онтологические классы и домены, которые характеризуют взаимодействующие процессы системы в целом. В разделе 2. мы даем формальную семантику этих классов, включающую их представление на языке OWL (Web Ontology Language [13]) и операционную семантику действий процессов системы. Синтаксис языков представления онтологий, в частности языка OWL, является декларативным, но строго определенная операционная семантика нашей онтологии позволяет легко транслировать представление модели процессов во входные языки средств верификации, в частности, SPIN [11]. В разделе 3. излагается методология использования нашей онтологии систем процессов общего вида для спецификации некоторых компонентов систем автоматического управления (САУ). В разделе 4. – заключение и планы развития и использования онтологии процессов.

1. Онтология

Мы считаем *онтологией* структуру, включающую следующие элементы: (1) конечное непустое множество *классов*, (2) конечное непустое множество *атрибутов дан-*

ных и атрибутов отношений и (3) конечное непустое множество *доменов атрибутов данных*. Каждый класс определяется множеством атрибутов. Атрибуты данных принимают значения из доменов, а значениями атрибутов отношения являются экземпляры классов. *Экземпляр класса* определяется набором значений атрибутов для этого класса. *Информационный контент* онтологии представляет собой множество экземпляров ее классов. Задача пополнения онтологии заключается в извлечении информационного контента этой онтологии из входных данных. В нашем случае входными данными для пополнения онтологии систем процессов является техническая документация. Чтобы пополнить эту онтологию, мы используем наши методы извлечения семантической информации из текстов на естественном языке [3]. Заметим, что некоторые значения атрибутов извлеченных экземпляров онтологии могут быть не определены. Система процессов описывается в нашей онтологии с помощью набора экземпляров.

Мы рассматриваем систему последовательных взаимодействующих процессов в духе CSP [10]. Процессы (описываемые классом *Process*) характеризуются множествами локальных и разделяемых переменных, списком действий над этими переменными, которые изменяют их значения, списком каналов взаимодействия процессов, а также списком коммуникационных действий для отправки сообщений. Переменные процессов (класс *Variable*) принимают значения основных типов (булевские, конечные подмножества целых чисел или строк для перечислимых типов). Исходные условия для значений переменных определяются их сравнением с константами. Действия процессов (класс *Action*) включают базовые операции над основными типами. Условие выполнимости каждого действия зависит от охранных условий (класс *Condition*) для значений переменных и содержимого отправленных сообщений. Процессы могут отправлять сообщения через каналы (класс *Channel*) при выполнении охранных условий (класс *Condition*). Каналы характеризуются типом чтения сообщений, емкостью, способами записи и считывания, а также надежностью. На рисунке 1 классы представлены белыми овалами. Отношения между классами показаны стрелками с именами в серых овалах. Эти стрелки обозначены сплошными линиями, если отношение является “один ко многим”, и пунктирными, если отношение является взаимнооднозначным. Атрибуты данных класса, помещенные в штрих-пунктирные прямоугольники, связаны с их классами штрих-пунктирными стрелками.

В следующих разделах мы детально опишем классы и домены нашей онтологии, определим их формальную семантику. Также мы приведем примеры описаний некоторых онтологических классов, их свойств и аксиом, заданных в редакторе Protégé [14] средствами языка OWL с использованием правил вывода на языке SWRL (Semantic Web Rule Language) [12]. Эти свойства и аксиомы задают как правила согласованности значений атрибутов разных классов, так и ограничения на значения, количество и тип атрибутов отдельного класса. Мы приведем несколько типовых свойств и аксиом для некоторых классов. С помощью набора правил SWRL мы задаем условия, которые обеспечивают в Protégé вывод новых знаний и проверку корректности и совместности онтологического описания набора процессов с помощью машины логического вывода Hermit [8].

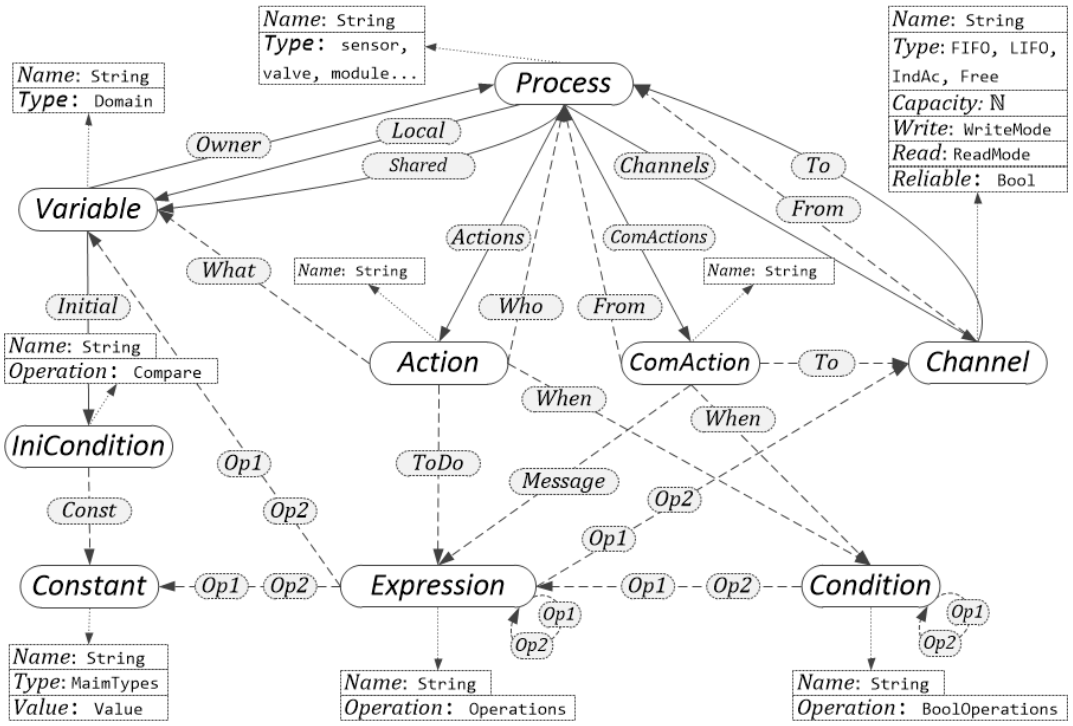

Рис. 1. Онтология систем процессов
Fig. 1. The Concurrent Process Ontology

2. Формальная семантика онтологии

Мы рассматриваем параллельную систему, которая является множеством последовательных процессов, взаимодействующих через разделяемые переменные и каналы ограниченной емкости. Значения разделяемых и локальных переменных, связанных с процессом, определяют его локальное состояние. В зависимости от предметной области задаются особенности взаимодействия процессов (синхронные, асинхронные) и характеристики каналов (FIFO, LIFO, надежные, ограниченные, и т.п.). Состояния канала задаются множеством сообщений, хранимым в нем.

Онтологическое описание O_S системы процессов S — это набор экземпляров онтологии, которые представляют процессы, каналы и другие элементы системы S. Мы определяем операционную семантику для O_S, используя стандартную модель помеченных систем переходов $M_S = (D_S, D_S^0, T_S, Act_S)$, где D_S — конечное непустое множество состояний, D_S^0 — непустое множество начальных состояний, T_S — отношение переходов между состояниями, Act_S — конечное множество меток переходов, описывающих действия системы. Семантика модели онтологического описания O_S — это частичная функция, которая отображает значения атрибутов экземпляров в элементы помеченной системы переходов $Sem : O_S \longrightarrow El(M_S)$.

Опишем множество $El(M_S)$. Пусть параллельная система S содержит конечное множество P_S процессов и конечное множество C_S ограниченных каналов. Пусть $\texttt{Domains} = \{\texttt{Integer}, \texttt{Bool}, \texttt{String}\}$ — конечная версия стандартных основных типов для значений переменных со стандартным приведением типов. Для процесса

$P \in P_S$ мы определим следующие множества: локальные переменные L_P, разделяемые переменные U_P, все переменные $V_P = L_P \cup U_P$, действия A_P, каналы C_P, коммуникационные действия S_P. Область определения переменной $v \in V_P$ есть $Val(T_P(v)) \subseteq \texttt{Values}$ для некоторого базового типа $T_P(v) \in \texttt{Domains}$. Пусть L_S — множество всех локальных переменных процессов из P_S, U_S — множество всех разделяемых переменных процессов из P_S, и функция означивания $V : L_S \cup U_S \longrightarrow 2^{\texttt{Values}}$ представляет значения переменных. Локальное состояние процесса P — это кортеж значений его переменных: $st_P = (V(v_1), ...V(v_{n_V}))$. Пусть $st_P(v) = V(v)$ для $v \in V_P$ — это значение переменной v в состоянии st_P. Начальные состояния процесса $st^0_P = (V^0(v_1), ...V^0(v_{n_V}))$ определяются начальными значениями его переменных $V^0 : L_S \cup U_S \longrightarrow 2^{\texttt{Values}}$. Локальное состояние канала c — это упорядоченный кортеж сообщений, хранимых в нем: $st_c = (m_1, ..., m_{n_c})$. Пусть $st_c(k) = m_k$ для $k \in [1..n_c]$ — содержимое k-го сообщения канала c в состоянии st_c. Начальное состояние всех каналов — пустой кортеж. Множество состояний D_S системы M_S — это декартово произведение значений локальных и разделяемых переменных процессов и состояний каналов. Множество начальных состояний D^0_S — это декартово произведение начальных состояний переменных процессов и состояний каналов. Мы считаем, что операционная семантика системы — это семантика чередования, т. е. ее состояние может изменяться только одним процессом системы. Следовательно, множество действий системы есть $Act_S = \cup_{P \in S} A_P \cup S_P$. В силу чередования отношение переходов T_S задается локальной операционной семантикой отдельных экземпляров процессов $P \in S_P$, которая определяется далее.

Опишем классы нашей онтологии. Далее для всех классов атрибут $\texttt{Name} \in \texttt{String}$ задает имя экземпляра класса. Имена атрибутов со множественными значениями заканчивается символом *.

Класс *Process* описывает процессы системы при помощи следующих атрибутов:
- $\texttt{Type}$: $\texttt{\{sensor, valve, ...\}} \in \texttt{String}$ содержит спецификатор процесса.
- $Local^*$: $Variable$ перечисляет переменные, видимые только в действиях процессов.
- $Shared^*$: $Variable$ перечисляет переменные, видимые процессам, их разделяющим.
- $Actions^*$: $Action$ перечисляет действия процесса над его переменными.
- $Channels^*$: $Channel$ перечисляет каналы, соединяющие процессы.
- $ComActions^*$: $ComAction$ перечисляет действия, отправляющие сообщения.

В этом классе не обязательно задание значений всех атрибутов. Однако, если $Local$ и $Shared$ не заданы, значение $Actions$ также не задано. Если значение $Channels$ не задано, значение $ComActions$ также не задано. Если задано значение $Channels$, то должны быть определены $Actions$ или $ComActions$. Если для процесса не заданы значения ни $Actions$, ни $ComActions$, ни $Channels$, то этот пассивный процесс считается *объектом*. Такие объекты удобно выделять для ясности описания системы и проверки его онтологической согласованности, но в процессе формальной верификации свойства поведения объектов не формулируются и не проверяются.

На рисунке 2 представлена иерархия классов онтологии процессов и фрагмент описания класса *Process*, описанный с помощью редактора *Protégé*. Слева, во вкладке “Class hierarchy” отображается иерархия заданных в онтологии классов. Вкладка “Usage: Process” для выбранного в иерархии класса описывает список его свойств и связей с другими классами и экземплярами онтологии, задаваемых атрибутами. Так, например, для класса *Process* указан фрагмент списка, содержащий имя про-

цесса `nameProcess`, связи с переменными и действиями (`localVar`, `actions`, `comActions`), а также отношения с классами *Channel*, *ComAction*, *Variable*, которые заданы в *Process* соответствующими атрибутами `fromChannel`, `fromComActions`, `owner`. Во вкладке "Description: Process" указываются дополнительные характеристики класса. Для экземпляров класса *Process* с помощью задания ограничений на атрибуты отношения определяются требования на количество понятий, связанных с этим классом: задание точно одного имени `nameProcess` и типа `typeProcess` процесса. Кроме того, с помощью следующих SWRL-правил можно задать требование определенности действий `a` или коммуникационных действий `ca` процесса `p` в случае определенности канала `ch` этого процесса:

```
Process(?p)^Channel(?ch)^channels(?p, ?ch)^differentFrom(?ch,NullChan)^
          Action(?a)^actions(?p,?a)^sameAs(?a,NullAct) ->
       ComAction(?ca)^comActions(?p,?ca)^differentFrom(?ca,NullComAct)

Process(?p)^Channel(?ch)^channels(?p,?ch)^differentFrom(?ch,NullChan)^
     ComAction(?ca)^comActions(?p,?ca)^sameAs(?ca,NullComAct) ->
                  Action(?a)^actions(?p,?a)^differentFrom(?a,NullAct)
```

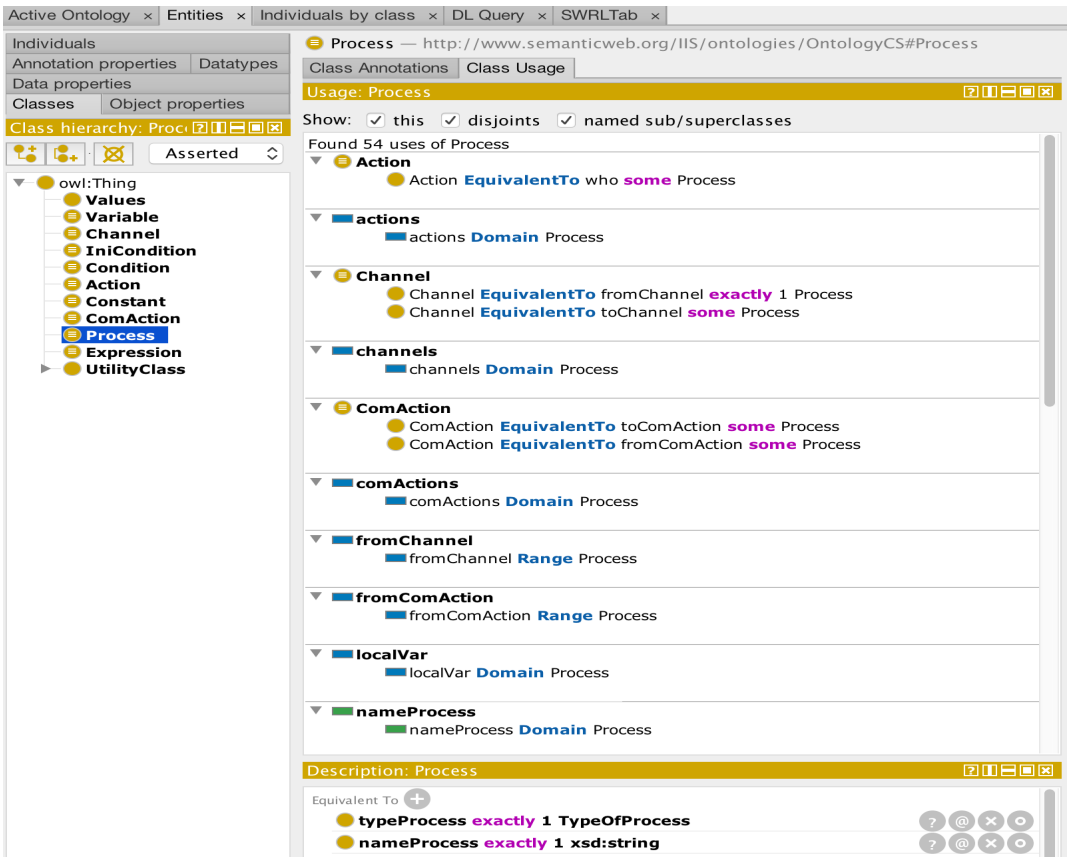

Рис. 2. Фрагмент описания онтологии процессов
Fig. 2. The Fragment of Description of the Concurrent Process Ontology

Для процесса P формальная операционная семантика его онтологического описания $Sem(P)$ определяется формальной семантикой онтологического описания его атрибутов: локальных переменных $Sem(P.Local)$, разделяемых переменных $Sem(P.Shared)$, действий $Sem(P.Actions)$, коммуникационных действий $Sem(P.ComActions)$, а также каналов $Sem(P.Channels)$. Процесс P может быть однозначно представлен в M_S посредством этих семантик. Для процесса P определим следующие множества: $L_P = \{l\}_{l \in Local}$, $U_P = \{u\}_{u \in Shared}$, $A_P = \{a\}_{a \in Actions}$, $C_P = \{c\}_{c \in Channels}$, $S_P = \{s\}_{s \in ComActions}$. Далее мы поочередно определяем семантику этих атрибутов.

Класс *Variable* описывает переменные процессов с помощью атрибутов:
- `Type`: `Domains` описывает тип переменной.
- $Initial^*$: $IniCondition$ описывает ограничения на начальное значение переменной.
- $Owner^*$: $Process$ ссылается на процессы, имеющие доступ к переменной.

Значения всех атрибутов этого класса, кроме $Initial^*$, должны быть заданы.

Ниже приведен фрагмент OWL-описания отношения `localVar` в формате *Turtle*. В данном примере для атрибута отношения `localVar` свойство обратной функциональности `InverseFunctionalProperty` гарантирует, что только для одного процесса *Process* переменная *Variable* является локальной:

```
:localVar rdf: type
          owl: ObjectProperty,
              owl: InverseFunctionalProperty;
          rdfs: domain: Process;
          rdfs: range: Variable.
```

Семантика переменной v процесса P задается таким образом: $Sem(v.Name) \in$ `String`, $Sem(v.Type) \in$ `Domains`, $Sem(v.Owner) \subseteq S_P$. Если $Sem(v.Owner) = \{P\}$, то $v \in L_P$. Семантика начального условия определяет множество начальных значений переменной $V^0(v)$:
– если $v.Initial = \emptyset$, то $V^0(v) = \{false\}$ для $v.Type =$ `Bool`, $V^0(v) = \{0\}$ для $v.Type \in$`Integer`, $V^0(v) = \{'' ''\}$ для $v.Type \in$ `String`;
– если $v.Initial \neq \emptyset$, то $V^0(v) = \cap_{ini \in v.Initial} Sem(ini)$.

Класс *IniCondition* задает начальные значения переменных с помощью следующих атрибутов:
- $IniConstant$: $\{Constant$, `Values`$\}$ определяет константу для сравнения.
- `Operation`: `Compare`$= \{<, >, =, \leq, \geq, \neq\}$ определяет сравнивающие действия.

Значения атрибутов $Const$ и `Operation` должны быть заданы.

Семантика начального условия ini переменной v — это подмножество множества ее значений: $Sem(ini.Name) \in$ `String`, и $Sem(ini) = \{val \in Val(v.Type) \mid val \circ Sem(Const)$ для $\circ \in \{<, >, =, \leq, \geq, \neq\}\}$, где $Sem(Const) = Sem(cs)$ для $cs \in Constant$, или $Sem(Const) = cs$ для $cs \in$ `Values`.

Класс *Constant* описывает константы и содержит следующие атрибуты:
- `Domain`: `Domains` описывает тип константы.
- `Value`: `Values` описывает значение константы.

Значения всех атрибутов должны быть определены.

Семантика константы cs — ее значение из множества возможных значений системы: $Sem(cs.Name) \in$ `String`, $Sem(cs.Domain) \in$ `Domains`, $Sem(cs.Value) \in$`Values`, и $Sem(cs) = cs.Value$.

Класс *Channel* описывает коммуникационные каналы с помощью атрибутов:
- $From$: $Process$ задает отправителя сообщений.
- To^*: $Process$ задает получателей сообщений из канала. Если получателей несколько, то канал — широковещательный.
- `Type`: `Order = {FIFO, LIFO, IndAc, Free}` определяет порядок чтения из канала, где `IndAc` означает доступ по индексу и `Free` означает случайный доступ.
- `Capacity`: `Integer` задает емкость (вместительность) канала.
- `Reliable`: `Bool` характеризует внешнюю надежность канала в том смысле, что сообщения не теряются по внешним причинам.
- `Write`∈`WriteMode = {NSent, Old, New, Some}` описывает режим записи при полном канале:
  - `NSent`: сообщение не записывается в канал, то есть теряется;
  - `New`: сообщение пишется вместо удаляемого сообщения с наибольшим индексом;
  - `Old`: сообщение записывается вместо удаляемого сообщения с индексом 1;
  - `Some`: сообщение записывается вместо некоторого удаляемого сообщения.
- `Read`∈`ReadMode = {Del, Keep, Tail, Head, Some}` задает режим чтения:
  - `Del`: сообщение удаляется из канала после прочтения;
  - `Keep`: сообщение не удаляется из канала после прочтения;
  - `Tail`: индекс сообщения после чтения становится наибольшим в канале;
  - `Head`: индекс сообщения после чтения становится равным 1;
  - `Some`: индекс сообщения после чтения становится произвольным.

Значения атрибутов $From$ и To должны быть определены. Мы задаем следующие значения атрибутов по умолчанию: `Type` = `FIFO`, `Capacity` = 1, `Write` = `NSent`, `Read` = `Del`, `Reliable` = $true$. Если у канала процесс P есть в атрибуте $From$ или To^*, то этот канал должен быть в атрибуте $Channels^*$ этого процесса.

Семантика канала c процесса P определяется следующим образом:
$Sem(c.Name) \in$ `String`, $Sem(c.From) = P$, и $Sem(s.To) \subseteq S_P$. Семантика локальных состояний канала и их изменений определена ниже в описаниях классов действий и коммуникационных действий.

Класс *Action* описывает действие, выполняемое процессом.
- Who: $Process$ указывает на процесс, который выполняет действие.
- $What$: $Variable$ ссылается на переменную, значение которой может измениться в результате действия.
- $ToDo$: $Expression$ описывает выражение, значение которого присваивается переменной из $What$.
- $When$: $Condition$ описывает булевскую формулу, которая задает условие выполнения действия в локальном состоянии.

Значения атрибутов Who, $What$ и $ToDo$ должны быть заданы. Если $When$ не задан, это означает, что действие может быть выполнено в любом локальном состоянии процесса. Значения атрибутов $ToDo$ и $When$ могут использовать только переменные процесса и сообщения из его каналов.

Следующее SWRL-правило задает требование того, что если атрибут `what` действия `a` процесса `p` включает переменную `v`, то ее атрибут `owner` включает `p`:

```
Action(?a) ^ Variable(?v) ^ Process(?p) ^ what(?a, ?v) ^ who(?a, ?p) ->
                                                        owner(?v, ?p)
```

Определим семантику действия a процесса P. Очевидно, $Sem(a.Name) \in$`String`, $Sem(a.Who) = P$, и $Sem(a.What) \in V_P$. Пусть $a.What = v$ — переменная, которая должна быть изменена действием a, и $C_a \subseteq C_P$ — каналы процесса, сообщения из которых читаются действием. Отношение перехода между локальными состояниями как процесса P, так и его каналов определяется следующим образом. Если локальные состояния процесса st_P и st_c каждого канала $c \in C_a$ таковы, что условие $a.When$ удовлетворено ($Sem(a.When) = true$), и выражение действия $a.ToDo$ имеет значение ($Sem(a.ToDo) \neq null$), то после выполнения действия a, новые локальные состояния st'_P процесса и его каналов таковы, что $st'_P(v) = Sem(a.ToDo) \wedge \forall x \in V_P \setminus \{v\} : st'_P(x) = st_P(x) \wedge \forall y \in C_P \setminus C_a : st'_y = st_y$, и новое состояние st'_c каждого канала c из C_a соответствует семантике операции чтения для каналов (см. следующий раздел). Заметим, что в процессе вычисления $Sem(a.When)$ и $Sem(a.ToDo)$, состояния каналов не меняются. Если $Sem(a.ToDo) = null$ или $Sem(a.When) \neq true$, все состояния остаются неизменными: $st'_P = st_P$, и $\forall c \in C_a : st'_c = st_c$.

Класс *Expression* описывает выражения, значения которых присваиваются переменным процесса в результате действий, и содержит следующие атрибуты.
- Атрибуты $Op1, Op2$: $\{Expression, Variable, Constant, Channel,$ `Values`$\}$ задают левый и правый операнды выражения.
- `Operation`: `Operations`$= \{+, -, ^*, \%, >, <, \leq, \geq, \neq, \neg, \wedge, \vee, \rightarrow\}$ определяет стандартные целочисленные и булевские операции над операндами.

Значения $Op1$ и $Op2$ должны использовать переменные и сообщения канала, доступные процессу. Значения атрибутов $Op1$ и `Operation` должны быть определены. Если $Op2$ не определен, то значение `Operation` должно быть унарной операцией.

Следующий набор SWRL-правил ограничивает значения операндов $Op1$ и $Op2$, входящих в выражение атрибута `toDo` действия `a` процесса `p`, переменными `v` и сообщениями канала `ch`, доступными для процесса:

```
who(?a, ?p) ^ Action(?a) ^ Process(?p) ^ toDo(?a, ?e1) ^
        Expression(?e1) ^ Expression(?e2) ^ subExp(?e1, ?e2)
                 op1Expression(?e2, ?v) ^ Variable(?v) -> owner(?v, ?p)

who(?a, ?p) ^ Action(?a) ^ Process(?p) ^ toDo(?a, ?e1) ^
        Expression(?e1) ^ Expression(?e2) ^ subExp(?e1, ?e2)
            op1Expression(?e2, ?ch) ^ Channel(?ch) -> channels(?ch, ?p)
```

Здесь `SubExp(?e1, ?e2)` является транзитивным отношением “быть подвыражением” на классе *Expression*.

Пусть для действия a процесса P в состоянии st_P выражение действия имеет вид $a.ToDo = e$. Семантикой этого выражения является значение базового типа или $null$: $Sem(e) \in$ `Values` $\cup\, null$.
– Для бинарных операций $\circ \in \{+, -, ^*, \%, >, <, \leq, \geq, \neq, \wedge, \vee, \rightarrow\}$:

$Sem(e) = Sem(e.Op1) \circ Sem(e.Op2)$;
– Для унарных операций $\circ \in \{-, \neg\}$: $Sem(e) = \circ Sem(e.Op1)$;
– $Sem(e) = null$ при $Sem(e.Op1) = null$, или $Sem(e.Op2) = null$.
Пусть $Op \in Op1, Op2$. $Sem(e.Op)$ определяется следующим образом:
– $e.Op = e' \in \{Expression, Constant, \texttt{Values}\}$: $Sem(e.Op) = Sem(e')$;
– $e.Op = w \in Variable$, $w \in V_P$: $Sem(e.Op) = st_P(w)$;
– $e.Op = c \in Channel$, $c \in C_a$: $Sem(e.Op) = Val$, где
  – $st_c = \emptyset$, то $Val = null$, и $st'_c = st_c$;
  – Если c.Type = `FIFO`, то $Val = st_c(1)$, и если c.`Read` равно
    – `Del`, то $st'_c = (m_2, ..., m_{n_c})$;
    – `Keep`, то $st'_c = st_c$;
    – `Tail`, то $st'_c = (m_2, ..., m_{n_c}, m_1)$;
    – `Head`, то $st'_c = st_c$;
    – `Free`, то $st'_c = (m_2, ..., m_1, ..., m_{n_c})$.
Аналогичным образом определяется семантика для других типов чтения.

Класс *Condition* описывает охранное условие для действий и коммуникационных действий процессов. Его описание и семантика почти такие же, как для класса *Expression*, за исключением того, что *Condition* использует только булевские операции и значения.

Класс *ComAction* описывает коммуникационные действия отправки сообщений и содержит следующие атрибуты:
- $From$: $Process$ задает процесс, который отправляет сообщение.
- To: $Channel$ задает канал, который доставляет сообщения получателю.
- $Message$: $Expression$ описывает отправленное сообщение.
- $When$: $Conditions$ описывает охранное условие для отправки сообщения.
Заметим, что наша модель взаимодействия процессов реализует получение сообщений по необходимости, т. е. чтение из канала выполняется, только когда процесс вычисляет охранное условие или выражение действия. Значения атрибутов $Message$, $From$ и To должны быть определены. Если $When$ не задан, это означает, что коммуникационное действие может выполняться в любом локальном состоянии процесса. Значения атрибутов $Message$ и $When$ могут использовать только переменные процесса и сообщения из его каналов.

Семантика действия s процесса P, отправляющего сообщение в состоянии st_P, определяется следующим образом: $Sem(s.Name) \in \texttt{String}$, $Sem(s.From) = P$, и $Sem(s.To) \in C_P$. Пусть $s.To = c$ — канал процесса для отправки сообщения, $st_c = (m_1, ..., m_{n_c})$ — его локальное состояние, и $C_s \subseteq C_P$ — каналы процесса, из которых сообщение читается. Пусть $Sem(s.Message) = Mes$, и $Sem(s.When) = Cnd$. Локальная операционная семантика отправки сообщений определяется следующим образом. Локальные состояния процесса P и неиспользуемые каналы не меняются: $st'_P = st_P$, и $\forall cl \in C_P \setminus (C_s \cup \{c\}) : st'_{cl} = st_{cl}$. Если локальные состояния P и каналов $cs \in C_s$ таковы, что $Mes = null$, или $Cnd \neq true$, то $\forall c \in C_P : st'_c = st_c$. Если условие $s.When$ выполнено ($Cnd = true$) и значение сообщения можно вычислить ($Mes \neq null$), то после выполнения коммуникационного действия s новое состояние каналов st'_{cs} для $cs \in C_s$ соответствует семантике операции чтения, определенной выше. Новое локальное состояние st'_c канала c зависит от емкости канала c.`Capacity`

и метода записи в канал c.`Write` следующим образом:
– Если $n_c < c$.`Capacity`, то $st'_c = (m_1, ..., m_{n_c}, Mes)$.
– Когда канал полон, если c.`Write` равно
  – `NSent`, то $\forall c \in C_P : st'_c = st_c$;
  – `Old`, то $st'_c = (Mes, m_2, ..., m_{n_c})$;
  – `New`, то $st'_c = (m_1, ..., m_{n_c-1}, Mes)$;
  – `Some`, то $st'_c = (m_1, ..., m_{i-1}, Mes, m_{i+1}, ..., m_{n_c})$, $i \in [1..n_c]$.
Если известна локальная семантика действий и коммуникационных действий каждого процесса, то может быть однозначно оперделено отношение перехода T_S.

3. Онтологическое моделирование функциональных элементов системы автоматического управления

В этом разделе мы моделируем в нашей онтологии некоторые типовые функциональные элементы САУ (датчики, сравнивающие устройства и регулирующие устройства) и представляем иллюстративный пример онтологического моделирования САУ. Состояние объекта управления, которым управляет САУ, характеризуется переменными состояния — количественными характеристиками, меняющимися во времени.

Датчик отслеживает изменение некоторой переменной состояния и посылает информацию об этом другим элементам САУ. Датчик моделируется в онтологии процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `sensor`. Атрибут $Shared^*$ содержит переменную mv, моделирующую переменную состояния, отслеживаемую датчиком (monitored variable). Атрибут $ComActions^*$ содержит действия по передаче значения переменной mv по каналам, содержащимся в атрибуте $Channels^*$.

Сравнивающее устройство сравнивает значение своего входного параметра с некоторым эталонным значением. Сравнивающее устройство представляется в онтологии процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `comparator`. Атрибут $Channels^*$ содержит каналы передачи результата сравнения значения входного параметра с эталонным значением, а также каналы, по которым процесс получает значения входного параметра. Атрибут $ComActions^*$ содержит действия по вычислению результата сравнения и его передачи по каналам.

Датчик с функцией сравнения, комбинирующий функции датчика и сравнивающего устройства, моделируется в онтологии процессом, свойства которого являются комбинацией свойств процессов, представляющих датчики и сравнивающие устройства. Атрибут `Type` этого процесса имеет значение `sensor-comparator`. Поскольку такой датчик сам отслеживает значение переменной состояния, нет необходимости в специальных каналах передачи этого значения.

Регулирующее устройство поддерживает некоторую заданную характеристику объекта управления. Оно непосредственно влияет на состояние объекта управления, изменяя его переменные в соответствии со значениями входных параметров регулирующего устройства. Моделирование регулирующего устройства в онтологии зависит от количества режимов его работы (например, “вкл.” и “выкл.”). Онтологическое представление регулирующего устройства с количеством режимов работы n

задается процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `regulator`. Атрибут $Local^*$ содержит переменную $mode$, которая моделирует режимы работы и принимает значения перечислимого типа, состоящего из n элементов. Атрибут $Shared^*$ содержит переменные, моделирующие переменные объекта управления, на которые влияет регулирующее устройство. Атрибут $Channels^*$ содержит каналы, по которым процесс получает значения входных параметров регулирующего устройства. Атрибут $Actions^*$ содержит действия, которые изменяют разделяемые переменные. Выражения, задаваемые атрибутом $ToDo$ этих действий и вычисляющие новые значения указанных переменных, зависят от значения локальной переменной $mode$ и сообщений, содержащих значения входных параметров регулирующего устройства.

Чтобы проиллюстрировать, как САУ моделируется в нашей онтологии, мы представим подсистему системы розлива бутылок из [17], описанную на естественном языке. Мы рассмотрим следующую упрощенную подсистему: датчики низкого и высокого уровня жидкости задаются процессами P_l и P_u, а впускной и выпускной клапаны — процессами P_i и P_b. Мы считаем, что все каналы имеют тип FIFO, емкость 1 и сообщения удаляются после прочтения.

Датчик низкого уровня моделируется процессом P_l=(`Type`: `sensor-comparator`, $Shared^*$: $\{mv\}$, $Channels^*$: $\{li, lb\}$, $ComActions^*$: $\{lo, lc\}$). Переменная mv задает информацию о текущем уровне жидкости. Она принимает целочисленные значения от $min.Value$ до $max.Value$, где константы min и max определяют уровни, когда бак почти пуст и когда он переполнен. Каналы li и lb соединяют датчик P_l с клапанами P_i и P_b, соответственно. Коммуникационное действие lo посылает сообщение “toOpen” впускному клапану P_i, когда бак почти пуст: $lo = (From$: P_l, To: li, $Message$: “toOpen”, $When$: $mv \leq min)$. Коммуникационное действие lc определяется симметричным образом.

Впускной клапан моделируется процессом P_i=(`Type`: `regulator`, $Local^*$: $\{mode\}$, $Shared^*$: $\{ev\}$, $Actions^*$: $\{inc, io, ic\}$, $Channels^*$: $\{li, ui\}$). Переменная $mode$ имеет перечислимый тип {“open”, “closed”}. Действие inc увеличивает уровень жидкости на 1: $inc = (Who$: P_i, $What$: mv, $ToDo$: $ev + 1$, $When$: $mode =$ “open”). Действие io открывает впускной клапан: $io = (Who$: P_i, $What$: $mode$, $ToDo$: “open”, $When$: $li =$ “toOpen”). Действие по закрытию клапана ic определяется симметричным образом. Процессы P_u и P_b определяются аналогичным образом.

4. Заключение

В данной статье мы представляем онтологию систем процессов, которая является одним из базовых компонентов интеллектуальной системы поддержки верификации. Наши методы извлечения информации из текстов технической документации позволяют пополнять эту онтологию экземплярами процессов, информация о которых обнаружена в этих текстах. Затем модуль трансляции должен транслировать онтологическое представление системы во входной язык некоторого верификатора. Чтобы обеспечить корректность трансляции и последующей верификации, в этой статье мы определяем формальную семантику нашей онтологии, связывая формальными правилами помеченную систему переходов со множеством экземпля-

ров процессов из онтологии. Формальная семантика онтологического представления процессов, ориентированная на формальные методы верификации, отличает наш подход от современных подходов к определениям онтологий процессов.

В дальнейшем мы планируем разработать системы онтологических шаблонов процессов для различных предметных областей: систем автоматического управления, бизнес-процессов и т. д. В рамках построения нашей интеллектуальной системы поддержки верификации мы также разработаем метод извлечения типичных требований к системе из онтологического представления процессов для пополнения онтологии требований.

Список литературы / References

[1] Autili M. et al., “Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar”, *IEEE Transactions on Software Engineering*, **41**:7 (2015), 620–638, https://doi.org/10.1109/TSE.2015.2398877.

[2] Garanina N., Sidorova E., “Context-dependent Lexical and Syntactic Disambiguation in Ontology Population”, *Concurrency, Specification and Programming (CS&P)*, Proc. of the 25th Int. Workshop (Humboldt-Universitat zu Berlin), 2016, 101–112.

[3] Garanina N., Sidorova E., Bodin E., “A Multi-agent Text Analysis Based on Ontology of Subject Domain”, *Perspectives of System Informatics. PSI 2014*, Proc. Int. Conference (St. Petersburg, Russia, June 24–27), Lecture Notes in Computer Science, **8974**, eds. Voronkov A., Virbitskaite I., Springer, Berlin, Heidelberg, 2015, 102–110, https://doi.org/10.1007/978-3-662-46823-4_9.

[4] Garanina N. et al., “Using Multiple Semantic Measures For Coreference Resolution In Ontology Population”, *International Journal of Computing*, **16**:3 (2017), 166–176.

[5] Garanina N., Zyubin V., Liakh T., “Ontological Approach to Organizing Specification Patterns in the Framework of Support System for Formal Verification of Distributed Program Systems”, *System Informatics*, **9** (2017), 111–132, http://dx.doi.org/10.31144/si.2307-6410.2017.n9.p111-132.

[6] Hepp M., Dumitru R., “An Ontology Framework for Semantic Business Process Management”, *Wirtschaftsinformatik Proc.*, (Karlsruhe, Germany, February 28 – March 2), 2007, 423–440, https://aisel.aisnet.org/wi2007/27/.

[7] Hepp M. et al., “Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management”, *e-Business Engineering (ICEBE 2005)*, Proc. Int. Conf. (Beijing, China, October 12–18), IEEE, 2005, 535–540, https://doi.org/10.1109/ICEBE.2005.110.

[8] *HermiT OWL Reasoner*, http://www.hermit-reasoner.com/.

[9] Herre H., “General Formal Ontology (GFO): A Foundational Ontology for Conceptual Modelling”, *Theory and Applications of Ontology: Computer Applications*, eds. Poli R., Healy M., Kameas A., Springer, Dordrecht, 2010, 297–345, https://doi.org/10.1007/978-90-481-8847-5_14.

[10] Hoare C.A.R., *Communicating sequential processes*, Prentice-Hall, 1985, 256 pp.

[11] Holzmann G.J., *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley Professional, 2003, 608 pp.

[12] Horrocks I. et al., *SWRL: A Semantic Web Rule Language combining OWL and RuleML*, http://www.w3.org/Submission/SWRL.

[13] *OWL Web Ontology Language Overview: W3C Recommendation 10 February 2004*, https://www.w3.org/TR/owl-features/, eds. D.L. McGuinness, F. Harmelen van..

[14] *Protégé. A free, open-source ontology editor and framework for building intelligent systems*, http://protege.stanford.edu/.

[15] Salamah S., Gates A.Q., Kreinovich V., "Validated patterns for specification of complex LTL formulas", *Journal of Systems and Software*, **85**:8 (2012), 1915–1929, https://doi.org/10.1016/j.jss.2012.02.041.

[16] Schlenoff C. et al., *The Process Specification Language (PSL): Overview and Version 1.0 Specification*, NIST Interagency/Internal Report (NISTIR), **6459**, 1999, https://www.nist.gov/publications/process-specification-language-psl-overview-and-version-10-specification.

[17] Shanmugham S.G., Roberts C.A., "Application of graphical specification methodologies to manufacturing control logic development: A classification and comparison", *International Journal of Computer Integrated Manufacturing*, **11**:2 (1998), 142–152, http://dx.doi.org/10.1080/095119298130886.

[18] Stuart A., Curtis J., "A Process Ontology", *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. EKAW 2002*, Proc. of the 13th Int. Conference (Sigüenza, Spain, October 01–04), Lecture Notes in Computer Science, **2473**, eds. Gómez-Pérez A., Benjamins V.R., Springer, Berlin, Heidelberg, 2002, 108–113, https://doi.org/10.1007/3-540-45810-7_13.

[19] Wong P.Y.H., Gibbons J., "Property Specifications for Workflow Modelling", *Integrated Formal Methods. IFM 2009*, Proc. Int. Conf. (Düsseldorf, Germany, February 16–19), Lecture Notes in Computer Science, **5423**, eds. Leuschel M., Wehrheim H., Springer, Berlin, Heidelberg, 2009, 166–180, https://doi.org/10.1007/978-3-642-00255-7_5.

[20] Yu J. et al., "Pattern based property specification and verification for service composition", *Web Information Systems. WISE 2006*, Proc. of 7th Int. Conf. (Wuhan, China, October 23–26), Lecture Notes in Computer Science, **4255**, eds. Aberer K. et al., Springer, Berlin, Heidelberg, 2006, 156–168, https://doi.org/10.1007/11912873_18.

Garanina N. O., Anureev I. S., Borovikova O. I., "Verification Oriented Process Ontology", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 607–622.

DOI: 10.18255/1818-1015-2018-6-607-622

Abstract. This paper presents the ontology of the concurrent processes close to Hoare communicating sequential processes. It is the part of the intellectual system for supporting verification of behavioural properties of these processes. Our ontological representation of the processes is oriented both to the application of formal verification methods and to the extraction of information from technical documentation (by our previously developed system of information extraction from a natural language text). We describe the ontology classes and domains that define communicating concurrent processes. These processes are characterized by sets of local and shared variables, a list of actions on these variables which change their values, a list of channels for the process communication (which, in turn, are characterized by the type of reading messages, capacity, ways of writing and reading, and reliability), and also a list of communication actions for sending messages. In addition to the formal mathematical definition of classes and domains of the ontology, examples of descriptions of some ontological classes as well as typical properties and axioms for them are specified in the editor Protégé in the OWL language with the use of the inference rules in the SWRL language. The formal operational semantics of communicating processes is determined on their ontological representation and is given as a labelled transition system. It is reduced to the local operational semantics of separate process instances in the interleaving model. We specialize several types of processes from the subject domain of automatic control systems that model the typical functional elements of the automatic control system (sensors, comparators and regulators) as well as their combinations. The concepts of the specialized ontology are illustrated by the example of a control part for a bottle-filling system.

Keywords: ontology, verification, model checking, concurrent processes

On the authors: Natalia O. Garanina, orcid.org/0000-0001-9734-3808, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: garanina@iis.nsk.su

Igor S. Anureev, orcid.org/0000-0001-9574-128X, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: anureev@iis.nsk.su

Olesya I. Borovikova, orcid.org/0000-0001-5456-8513, junior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: olesya@iis.nsk.su

Acknowledgments:
[1] The research has been supported by Russian Foundation for Basic Research (grant 17-07-01600).