An Ontology-Based Approach to Support Formal Verification of Concurrent Systems

Natalia Garanina[1,2,3](✉), Igor Anureev[1,2], Elena Sidorova[1,3], Dmitry Koznov[4], Vladimir Zyubin[2,3], and Sergei Gorlatch[5]

[1] A.P. Ershov Institute of Informatics Systems, Novosibirsk, Russia
{garanina,anureev,lena}@iis.nsk.su
[2] Institute of Automation and Electrometry, Novosibirsk, Russia
[3] Novosibirsk State University, Novosibirsk, Russia
zyubin@iae.nsk.su
[4] St. Petersburg University, Saint Petersburg, Russia
d.koznov@spbu.ru
[5] University of Muenster, Münster, Germany
gorlatch@uni-muenster.de

Abstract. Formal verification ensures the absence of design errors in a system with respect to system's requirements. This is especially important for the control software of critical systems, ranging from automatic components of avionics and spacecrafts to modules of distributed banking transactions. In this paper, we present a verification support framework that enables automatic extraction of a concurrent system's requirements from the technical documentation and formal verification of the system design using an external or built-in verification tool that checks whether the system meets the extracted requirements. Our support approach also provides visualization and editing options for both the system model and requirements. The key data components of our framework are ontological descriptions of the verified system and its requirements. We describe the methods used in our support framework and we illustrate their work for the use case of an automatic control system.

Keywords: Ontology · Information extraction · Formal verification · Requirement engineering · Formal semantics

1 Introduction

Our long-term goal is a comprehensive approach to support practical formal verification of safety-critical concurrent systems. Such approach should include understandable representations of both concurrent systems and their requirements (graphical and in a limited natural language), as well as tools for editing these representations and navigating over them. Also a support for information extraction is necessary because of large volumes of technical documentation,

https://doi.org/10.1007/978-3-030-54994-7_9

especially for legacy software systems. The use of practical formal verification methods should greatly improve quality assurance for safety-critical systems.

Various tools for specification and verification support have been suggested for different kinds of safety-critical systems. A commercial requirement engineering tool for embedded systems Argosim [25] allows software engineers to test the systems and perform inconsistency checking for requirements, but formal verification and information extraction are not supported. Software Cost Reduction toolset [32], Model Based Systems Engineering [27], RoboTool [14] are used for development, simulation and formal verification in avionics, space-crafts, robots and other control systems, but they do not use information extraction. SEVA [13] extracts information from natural language queries and can reason about system requirements, but no formal verification is offered. An approach to both information extraction and formal verification is suggested in [20], but it is restricted to a very special application field.

Our approach to supporting formal verification is based on patterns, because many requirements on real-world systems have recurring formulations with similar properties. Systems for supporting the development and verification of requirements based on patterns are an active topic of research [1,15,17,19,21,22]. Patterns are parameterized expressions in natural language that describe typical requirements for the behaviour of a system. Usually, parameters of patterns are system events or their combinations. For example, in pattern "The event *Restart* will occur", *Restart* is a parameter. The key property of patterns is that they have precisely-defined formal semantics. Patterns make it easier for developers to specify and verify typical system requirements. The drawback of the current support systems is that they offer only manual formulation of requirements and description of its formal semantics, sometimes with visualization. The first approach to employ an ontology as a knowledge organization method for patterns [22] does not yet use all benefits of the ontological knowledge representation.

Our envisaged advantage over the state-of-the-art approaches is that our framework supports a user-friendly, integrated strategy for the quality assurance of concurrent systems using a flexible tool that supports model extraction, correction, and verification, together with textual explanation and visualization of requirements. In particular, our support framework offers the following functionality: 1) constructing the model of a concurrent system and the system requirements by extracting information about them from technical documentation and/or using development tools and/or using information from experts via questionnaires, 2) generating typical requirements from the internal description of this extracted/constructed model, 3) representing the extracted/constructed requirements in a mathematical, linguistic and graphical manner, and editing these representations, 5) checking the integrity and consistency of the model's and requirements' representations, 6) translating the model representation into the input language of a suitable verifier. A special feature in our framework is automatic generation of requirements both from technical documentation and from the internal description of the concurrent system model, which considerably simplifies the work of requirement engineers. However, due to the linguistic ambiguities in technical documentation, a correction of extracted requirements and models is usually required; it is accomplished using the editors of our system.

In comparison to the state-of-the-art support systems, we suggest flexible customization of system components in four aspects. This flexibility is based on the intensive use of ontologies for representing knowledge about concurrent systems and their requirements, and especially on the formal semantics of these ontologies. First, due to the formal semantics, we can apply various methods of model/requirements extraction and construction taking into account various methods of formal verification. Second, it is possible to address various kinds of concurrent systems: from telecommunication protocols to cyber-physical systems, by defining the corresponding formal semantics for system models: finite/abstract state machines, hybrid systems, probabilistic automata [2,11], etc. Third, to specialize system descriptions for a particular subject domain we can use ontology axioms and rules. Forth, the requirements semantics also can be customised by choosing appropriate logics: LTL, CTL, PLTL [2], etc. The ontological representations allows us to check the integrity and consistency of a model and requirements' descriptions; it also naturally supports the term consistency between them.

The current configuration of our support framework uses an ontology-driven, rule-based approach to information extraction [5], labelled transition systems as formal semantics for concurrent models [9], and logics LTL, CTL and their real-time extensions as formal semantics for requirements [7].

The following Sect. 2 outlines our framework for supporting practical formal verification as a whole. Section 3 describes our ontology-driven methods used for information extraction, and illustrates them with a use case of a bottle-filling system. Section 4 defines the Requirement and Process Ontologies used for the internal representation of systems and their requirements. Sections 5–7 describe the methods used for processing requirements. Section 8 discusses our current framework's limitations that we plan to address in future work.

2 The Framework for Supporting Formal Verification

Figure 1 shows an overview of our framework for supporting formal verification of concurrent systems that automatically extracts and generates system requirements. The key components of the framework are the Process Ontology [9] and the Requirement Ontology based on patterns [7]. We use ontologies for an internal representation of concurrent systems and their requirements, because ontologies are convenient for systematizing knowledge, and they facilitate formulating and checking non-trivial consistency properties. Moreover, there are several well-developed tools for creating, editing and checking ontologies [26,29]. In our case, the contents of ontologies are descriptions of a particular concurrent system model and the requirements it must meet. These data must be acquired by Data Acquisition modules and then verified by Data Verification modules. After expert analysis of verification result, we can make corrections to particular development artefacts, such as high-level requirements and specifications, design specifications, software code, etc.

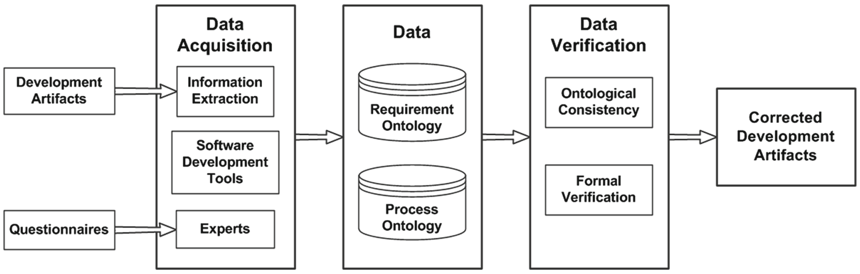


Fig. 1. The framework for supporting formal verification

Data Acquisition. System and requirement descriptions can be extracted from technical documentation containing development artifacts or constructed using concurrent software development tools (e.g. IBM Rhapsody [30]). These descriptions can be refined by domain experts via answering questions from ontology-based questionnaires. For populating the ontologies, we use our Information Extraction System [4–6] described in Sect. 3. The extracted requirements can be extended with typical requirements, automatically generated from the ontological description of the system (see Sect. 5). The descriptions of the system model and the requirements are the basis for formal verification.

Data Verification. The extracted description of the system model and its requirements may be incomplete or incorrectly constructed due to insufficiency of information presented in technical documentation or incorrect software development process. The Ontological Consistency procedure verifies the integrity and consistency of the constructed instances of the Process and Requirement Ontologies. In addition, as a rule, a large number of requirements are formulated for concurrent systems. Therefore, for the Requirement Ontology, it is also reasonable to check the semantic consistency of a requirement set using standard ontological methods. The output of these checking procedures are sets of incorrectly constructed entities of the considered concurrent system, as well as incorrectly formulated or inconsistent requirements. This procedure, described in Sect. 6, executes a simple pre-checking, before time-consuming formal verification. To formally verify a system, we choose a suitable verifier taking into account the formal semantics of the ontology-based requirement representation. If such a verifier is available, we translate the ontological description of the system into the model specification input language of the verifier, and the requirements' description is translated into the input language of the verifier (usually, this language is some temporal logic). If no suitable verifier is available, then our framework exploits the special verification algorithms for specific patterns.

Requirement Processing. Let us sketch the ontology processing activities necessary for the framework operation described in Sect. 7. Dealing with requirements involves representing them in three ways. The mathematical representation as formulas of some logic enables formal verification. The current formal representation is LTL and CTL with real-time extensions. The language and graphical representations both help a requirement engineer to understand formal representations. Due to the ambiguity of the natural language, it is possible that the extracted and generated requirements may not meet the engineer's expectations, and manual corrections are required. These corrections can use the editors for ontology representations, as well as the editors for formal, language and graphical representations.

In this paper, we do not consider methods for the formal verification module and ontology-based questionnaires. In the following sections, we describe in more detail the main ontologies and other parts of our framework.

3 The Information Extraction System

Figure 2 shows the general scheme of our information extraction system that takes technical documentation as input and searches for concepts and relations to populate the Process and Requirement Ontologies as described below. We use a rule-based, multi-agent approach to implement this system [5].

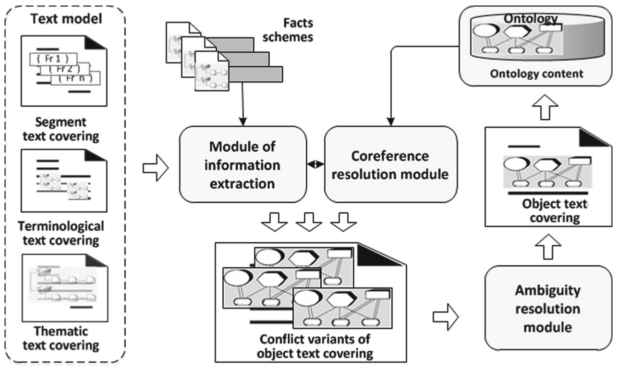

Text model
{ Fr 1 }
{ Fr 2 }
{ Fr n }
Segment text covering
Terminological text covering
Thematic text covering
Facts schemes
Ontology
Ontology content
Module of information extraction
Coreference resolution module
Object text covering
Conflict variants of object text covering
Ambiguity resolution module

Fig. 2. The information extraction and ontology population system

The process of information extraction includes the preliminary (lexical) step and the main (ontological) step. At the lexical step, the system constructs a text model that includes the terminological, thematic, and segment coverings of the input text. The terminological covering is the result of lexical text analysis that extracts the terms of a subject domain from the text and forms lexical objects using semantic vocabularies. The segment text covering is a division of the input text into formal fragments (clauses, sentences, paragraphs, headlines, etc.) and genre fragments (document title, annotation, glossary, etc.). The thematic covering selects text fragments of a particular topic. The construction of a thematic covering is based on the thematic classification methods. At the lexical step, the system constructs objects that represent instances of concepts and relations of the domain ontology from the lexical objects. Our system uses the ontology population rules which are automatically generated from lexico-syntactic patterns formulated by experts taking into account the ontology and language of a subject domain. Each lexico-syntactic pattern describes a typical situation for the subject area in terms of specific subject types of objects in the situation. These lexico-syntactic patterns constrain morphological, syntactic, genre, lexical, and semantic characteristics of the objects. The outputs of modules of disambiguation [4] and co-reference resolution [6] are used to choose the best version of text analysis for populating the ontology with a consistent set of instances of subject domain concepts and relations found in the input text.

To implement the described extraction technology for analyzing concurrent systems and their requirements, we create a knowledge base that includes: 1) a genre model of the input text for constructing segments, 2) a semantic dictionary, and 3) lexico-syntactic patterns for the subject domain. We illustrate this approach below, using the subject domain of automatic control systems (ACS).

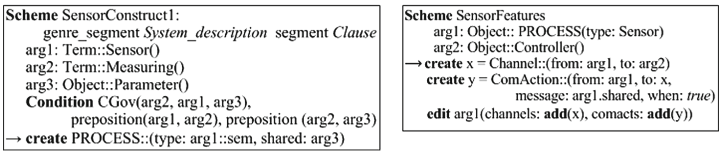
We consider Technical Documentation (TD) as a set of documents used for the design, creation and use of any technical objects. TDs have strong genre features: they do not contain figurative expressions, evaluative adjectives, almost no adverbs, the natural language ambiguity is compensated by the use of previously defined terms, etc. For this genre, we mark out sub-genre *Purpose* (the description of the system and its elements with respect to goals and functions) and sub-genre *Scenario* (the description of sequences of actions of automatic processes and the corresponding input and output states of the system). The detection of these genre fragments is based on a set of lexical markers which indicate that these sub-genres are located in the headings of the text.

The main component of the knowledge base of our information extraction system is a semantic dictionary. The system of lexico-semantic characteristics in the dictionary provides the connection of subject vocabulary with the ontology elements. For the ACS subject domain, we define the following lexical-semantic classes of lexical units in the dictionary:

- the vocabulary for the names of entities (objects, substances, technical devices and their parts, software products and their components);
- the vocabulary for naming situations:
  - state predicates (absence, be, contain),
  - event predicates for representing automatic processes and actions (move, rotate, feed, warm up, turn on, stop),
  - functional predicates (used for, provide),
  - mental predicates (control, measure, monitor, determine);
- the parametric vocabulary:
  - the names of qualitative/quantitative parameters (e.g., level, position),
  - the numbers and units of measurement, lexical names of reference scores (e.g., low/high, given position),
  - the predicates of quantitative change (e.g., fall, grow, normalize, etc.);
- the reference designation:
  - as proper names (e.g., Large Solar Vacuum Telescope – LSVT),
  - as unique numeric identifiers for designating referents of objects (e.g., temperature is measured by sensor (12)).

For constructing the lexico-syntactic patterns for descriptions of technological processes, we define the following types of situations: 1) actions leading the system or its components into enabled/disabled state; 2) activity characteristics for the functionality of the system or its components; 3) states of the values of quantitative or qualitative parameters; 4) processes for changing the parameters of system elements; and 5) information transfer processes. Most of situations and lexical names described in the TD texts are universal. Therefore, the generated lexico-syntactic patterns can be used for a large class of technical objects. Our developed methods are focused on the analysis of a linear text, however, after small changes, they can be applied to tables or TD schemas with language labels.

Illustrative Example. Let us illustrate our ontology-based approach to support formal verification with a system documentation text taken from [18]. This technical documentation describes the work of a bottle-filling system and includes several requirements on the system. We use two lexico-syntactic patterns shown in Fig. 3 to extract an ontology object corresponding to a sensor from the following text: "`Two sensors`^{arg1} `are also attached to the tank to read`^{arg2} `the fluid level`^{arg3} `information.`" With terms and ontology objects **arg1**, **arg2**, and **arg3**, satisfying the syntactical **Condition**, the SensorConstruct1-pattern creates an object of class *Process* with predefined attribute values following the ACS-ontology structure, as explained in the next section. The SensorFeatures pattern evaluates the attribute values of this sensor object using only the ACS-ontology structure without the input text.



Scheme SensorConstruct1:
    genre_segment *System_description* segment *Clause*
  arg1: Term::Sensor()
  arg2: Term::Measuring()
  arg3: Object::Parameter()
  Condition CGov(arg2, arg1, arg3),
      preposition(arg1, arg2), preposition (arg2, arg3)
→ **create** PROCESS::(type: arg1::sem, shared: arg3)

Scheme SensorFeatures
  arg1: Object:: PROCESS(type: Sensor)
  arg2: Object::Controller()
→ **create** x = Channel::(from: arg1, to: arg2)
  create y = ComAction::(from: arg1, to: x,
      message: arg1.shared, when: *true*)
  edit arg1(channels: **add**(x), comacts: **add**(y))

Fig. 3. Example: the lexico-syntactic patterns for extracting objects for "sensors"

4 The Ontologies

We consider an ontology as a structure that includes the following elements: (1) a finite, non-empty set of classes, (2) a finite, non-empty set of data attributes and relation attributes, and (3) a finite, non-empty set of domains of data attributes. Each class is defined by a set of attributes. Data attributes take values from domains, and relation attributes' values are instances of classes. An information content of an ontology is a set of instances of its classes formed by taking particular values of their attributes. In our case, the input data for populating the Process and Requirement Ontologies is technical documentation.

We represent the classes of our ontologies, their properties and axioms using the system Protégé [29] with the OWL language [28] and the SWRL language

[31]. These properties and axioms define the rules for checking the correctness of attribute values. Using the SWRL rules, we define the conditions used in Protégé for checking the correctness and consistency of ontological descriptions by the Hermit inference engine [26].

The Process Ontology. The Process Ontology [9] is used for an ontological description of a concurrent system by a set of its instances. We consider a concurrent system as a set of communicating processes that are described by the class *Process* and are characterized by: 1) their type for a subject-domain description (e.g. sensor, controller); 2) sets of local and shared variables; 3) a list of actions on these variables which change their values; 4) a list of channels for the process communication; and 5) a list of communication actions for sending messages. The process variables (class *Variable*) and constants (class *Constant*) take values in domains from a set consisting of basic types (Booleans, finite subsets of integers or strings for enumeration types) and finite derived types. Initial conditions of the variable values can be defined by comparison with constants. The actions of the processes (class *Action*) include operations over variables' values. The enabling condition for each action is a guard condition (class *Condition*) for the variable values and the contents of the sent messages. The processes can send messages via channels (class *Channel*) under the guard conditions. The communication channels are characterized by the type of reading messages, capacity, and modes of writing and reading. Currently, we define the Process Ontology formal semantics as a labelled transition system [9].

The classes of the Process Ontology are universal: they do not take into account the features of a subject domain. In order to describe specific-domain process ontology, we use ontology axioms and SWRL-rules. In the next subsection, we give an example of an SWRL-rule which restricts the Process Ontology for typical elements of automatic control systems (ACS), such as simple and complex sensors, controllers, actuators and the controlled object.

The ACS Process Ontology. The SWRL-rules impose the following restrictions on sensors. Sensors must read the observed values from the variables shared with the controlled object and they cannot change it. They have outgoing channels connecting them with controllers and communication actions for sending messages to the controllers. There is at least one controller and a shared variable associated with each sensor. Simple sensors have no local variables and actions: they can observe exactly one variable shared with the controlled object and send the observed value unchanged to controllers. Complex sensors can process observable and local variables to produce output for controllers.

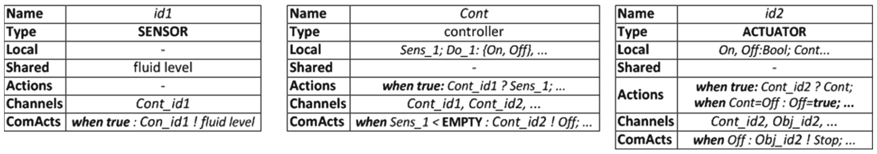
Controllers, actuators and controlled objects are also restricted by the corresponding SWRL-rules. Controllers and actuators must not have shared variables. Controllers must have output channels connecting them with other controllers and actuators, and input channels connecting them with sensors and actuators. Actuators must have output channels connecting them with controllers and the controlled object, and input channels connecting them with controllers. There must be at least one sensor and at least one actuator connected with a controller via input and output channels, respectively. There must be at least one controller and controlled object connected with an actuator through input and output channels, respectively. A controlled object must be connected with actuators by input channels. There must be at least one shared variable, one sensor and one actuator associated with a controlled object.

The following SWRL rule establishes the existence of a connection between a sensor and a single controller in an automatic control system:

```
Process(?p)^Process(?q)^type(?p,Sensor)^type(?q,Controller) ->
Channel(?c)^channels(?p, ?c)^channels(?q, ?c)
```

In [8], we describe ontology axioms and SWRL-rules for ACS in detail.

The lexico-syntactic patterns that extract information have to follow the restrictions mentioned above. In particular, for the bottle-filling system from [18] the patterns in Fig. 3 generate the sensor-process with name *id1* shown in Fig. 4 that has to send the observable value of the fluid level to the controller-process named *Cont* via channel *Cont_id1*. The attribute values in bold capital letters correspond to particular words in the input text, and other attribute values are generated automatically using the subject domain restrictions without direct text correspondence. For our text, a single controller-process is created at the beginning of model extraction automatically. Other lexico-syntactic patterns produce the actuator-process *id2* for the bottom valve that must be closed when the fluid level is low. This closing action is controlled by the controller-process that sends to *id2* the *Off*-message when the sensor *id1* reports the low fluid level. Sending communication actions are in ComActs-attributes and receiving communication actions are in Actions-attributes.



Name	*id1*
Type	**SENSOR**
Local	-
Shared	fluid level
Actions	-
Channels	*Cont_id1*
ComActs	***when true*** *: Con_id1 ! fluid level*

Name	*Cont*
Type	controller
Local	*Sens_1; Do_1: {On, Off}, ...*
Shared	-
Actions	***when true:*** *Cont_id1 ? Sens_1; ...*
Channels	*Cont_id1, Cont_id2, ...*
ComActs	***when*** *Sens_1* < **EMPTY** *: Cont_id2 ! Off; ...*

Name	*id2*
Type	**ACTUATOR**
Local	*On, Off:Bool; Cont...*
Shared	-
Actions	***when true:*** *Cont_id2 ? Cont;* ***when*** *Cont=Off : Off=***true; ...**
Channels	*Cont_id2, Obj_id2, ...*
ComActs	***when*** *Off : Obj_id2 ! Stop; ...*

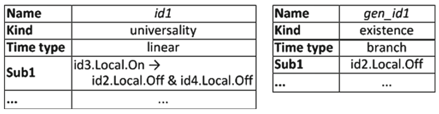
Fig. 4. Example: the processes of the bottle-filling system

The Requirement Ontology. Let us define how requirements are described by specification patterns. Requirements are expressed using standard Boolean connections of five basic patterns defining the appearance of certain events which can be considered as a particular combination of parameter values of the model. These patterns are: *Universality* (the event always takes place), *Existence* (the event will occur sometime), *Absence* (the event will never occur), *Precedence* (one event surely precedes another), and *Response* (one event always causes another). The patterns and their events can be constrained by eventual, time, and quantitative restrictions. Requirements expressed by these patterns have formal semantics as formulas of temporal logics LTL, CTL and their real-time variants [2] with temporal operators over events specified as Boolean combinations of propositions. These semantics unambiguously express requirements, and they precisely define the corresponding verification method.

Using ontologies for organizing a set of system requirements makes it possible to accurately systematize knowledge about them due to a hierarchical structure of concepts and relations. With our approach to the system requirements' presentation, the user can rely on a small set of class attributes to describe a wide range of properties of concurrent systems. This variability in expressing requirements is important, because for the same system it is necessary to specify both simple, easily verifiable properties (e.g. reachability), and complex properties that depend on the execution time of the system components. The possibility to formulate such different properties within a single formalism increases the quality of support for the development of complex systems as it covers the entire picture of the system requirements. Moreover, the ontological representation of a set of requirements enables its consistency checking. The output of our system of information extraction is a content of a certain ontology of a subject domain.

Our Requirement Ontology [7] organizes the existing systems of specification patterns [3,12] into unified structure and contains 12 classes and 17 relations between them. This ontology is designed to specify the requirements of system models described by the Process Ontology. Events of such systems occur in discrete time and the processes are completely dependent on the observed system states (including the current time), but may be non-deterministic.

In Fig. 5, the instance *id1* of the Requirement Ontology is produced by our IE-system from the following text fragment: `‘‘Both the bottle-filling and the heating operations are prohibited when the fluid is pumped to the filler tank’’` [18]. The formal semantic of this requirement are given by the LTL formula $\mathbf{G}(id3.Local.On \rightarrow id2.LocalOff \wedge id4.Local.Off)$, where *id3* and *id4* are identifiers for the inlet valve and the heating steam valve, respectively. The other requirement *gen_id1* is automatically generated by the Requirement Extraction Module described in the next section.



Name	*id1*
Kind	universality
Time type	linear
Sub1	id3.Local.On → id2.Local.Off & id4.Local.Off
...	...

Name	*gen_id1*
Kind	existence
Time type	branch
Sub1	id2.Local.Off
...	...

Fig. 5. Example: the requirements for the bottle-filling system

5 The Requirement Extraction

The task of the requirement extraction is to prompt the requirement engineer to formulate requirements expressed by patterns, because important requirements expressing the correct behavior of the system are not always explicitly defined in the technical documentation using the actions and variables of the system processes. The generated requirements use variables and events of the extracted (constructed) system; the process ontology is the input of the extraction procedure. This procedure explores the ontological description of the processes to find in the attribute values the events of interest whose satisfiability and appearance order can affect correctness. Such events are: changing the values of shared variables, sending/receiving messages, etc. The requirements may be subject-independent or subject-dependent. In any concurrent system, the following communication properties can be formulated:

– every sent message will be read (or overwritten) (*Response*);
– every message that has been read was sent before (*Precedence*);
– every guard condition for actions and communication actions must be satisfied in some system execution (*Existence* with Branching time).

For example, in Fig. 5, the right requirement for the bottle filling system *gen_id1* says that there is at least one point in at least one system execution when the bottom valve is open. Its formal semantic is CTL formula $\mathbf{EF}id2.Local.Off$.

The subject-specific requirements deal with the specifics entities of the subject domain, hence the extraction method must be customized to the subject area. For example, the typical requirements for ACS are as follows:

– the controller will send a control signal to the actuator (*Existence*);
– the actuator will send a modifying signal to the controlled object (*Existence*);
– the values captured by the sensor do not exceed its range (*Universality*).

Based on the found events, the requirements are formed as specially marked instances to populate the Requirement ontology. After population, the requirement engineer can change these instance requirements by adding eventual, time and quantitative restrictions, using the editors, or remove these requirements.

6 The Ontological Consistency

The extracted descriptions of concurrent processes and requirements may be incomplete due to insufficient information in the technical documentation. The integrity and consistency checking procedures inspect the correctness of the constructed ontology instances, i.e., the integrity of the Process and Requirement Ontologies, taking into account the default attribute values. Both ontologies are described in the OWL language of the Protégé system, with the ontology constraints formulated by axioms and SWRL-rules, hence, it is possible to use standard tools for inference ontology processing, e.g., Hermit.

For the Process Ontology, we can check the general integrity properties: definiteness of variables in processes, manipulation of only visible variables and channels, mandatory execution of any actions, the interaction with the environment through channels or shared variables, etc. For ontologies of subject domain systems, specific constraints described by axioms and SWRL-rules must also be checked. Some constraints for the ACS ontology are described in Sect. 4.

The integrity of the Requirement Ontology mainly concerns the definiteness of all attribute values for class instances. For example, an instance of class *Order* must contain two events as the values of the attributes of the ordered events. Besides the descriptive integrity, it is also necessary to check semantic integrity, since a large number of requirements are usually formulated for concurrent systems, both from technical documentation and by the Extract Requirement procedure. Formal verification of these requirements is a rather time-consuming process. The Ontological Consistency procedure executes pre-checking the simple consistency of these requirements. Since an ontology is just a declarative description of a subject domain, it is only possible to check the compatibility of requirements whose semantics do not have nested temporal operators. For more complex requirements, including, e.g., time restrictions, it is reasonable to leave consistency checking for standard formal verification tools. The following SWRL-rule restricts the inconsistent pair of requirements:

```
Proposition p holds always (Universality) vs.
                        Proposition p will be false sometime (Existence):
Occurence(?f) ^ kindPat(?f,Univ) ^ Prop(?p) ^ PatS1(?f,?p)^
Occurence(?g) ^ kindPat(?f,Exis) ^ Prop(?q) ^ PatS1(?f,?q)^
ProOp(?q,Neg) ^ ProSub1(?q,p)         -> Answ(?a)^ res(?a,Error1)
```

In [10], we describe the approach to checking requirement in detail. The Ontological Consistency procedure reports to the requirement engineer about the instances that do not satisfy the ontology constraints.

7 The Representation Modules and Editors

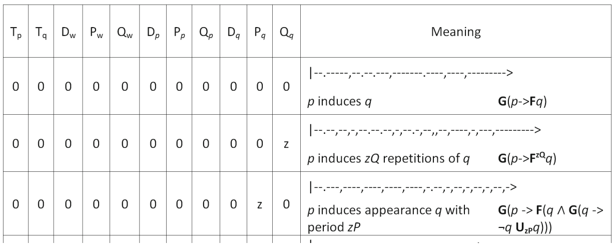
This section describes three ways of requirement representation in our system.

The Formal Semantic Representation. For using formal verification methods, requirements for concurrent systems must be presented as logic formulas. Since currently we focus on model checking, the formal semantics for instances of the Requirement ontology are expressed by formulas of temporal logics. The FSR-procedure translates the requirement instances into LTL or CTL formulas.

The translation takes several steps for determination:

1) determine whether time is branching or linear;
2) determine the requirement pattern;
3) determine the temporal/quantitative restrictions of the requirement/events;
4) compute the formula using the results of the previous steps, such that the resulting formula corresponds to the requirement without eventual constraints;
5) determine the eventual constraints;
6) compute the formula using the results of the previous steps, such that the resulting formula corresponds to the requirement with eventual constraints.

The translation grammar is highly context-sensitive. Hence, for calculating formulas, the procedure uses mainly the tables of formulas' dependence



T_p	T_q	D_w	P_w	Q_w	D_p	P_p	Q_p	D_q	P_q	Q_q	Meaning
0	0	0	0	0	0	0	0	0	0	0	|--.-----,--.--.---,-------.----,----,--------> *p* induces *q* **G**(*p*->**F***q*)
0	0	0	0	0	0	0	0	0	0	z	|--.--,--,-,--.--.--,-,--.-,--,,--,----,-,---,--------> *p* induces *zQ* repetitions of *q* $\mathbf{G}(p\text{->}\mathbf{F}^{zQ}q)$
0	0	0	0	0	0	0	0	0	z	0	|--.---,----,----,----,----,-.--,-,--,-,--,-,--,-> *p* induces appearance *q* with period *zP* $\mathbf{G}(p \text{ -> } \mathbf{F}(q \wedge \mathbf{G}(q \text{ -> } \neg q\ \mathbf{U}_{zP} q)))$

Fig. 6. A fragment of the translation table for *Response* requirements

on the restrictions and several analytic rules. The fragment of the table for *Order* requirements on Fig. 6 illustrates three variants of formal semantics of the *Response* requirement. The first columns of the table characterize the presence of duration D, periodic P, and quantitative Q restrictions on the requirement pattern itself (subscript $_w$), p-event (subscript $_p$), and q-event (subscript $_q$). The time delay of the first occurrence of p or q events using T_p or T_q, respectively, can be taken into account. The last column contains a graphical, language, and formal presentation of the *Responce* requirement for the following combination of restrictions: 1) an absence of restrictions (zero in each column), 2) the restriction on the number of repetitions of proposition q (number z in column Q_q), and 3) the restriction on the periodicity of proposition q (number z in column P_q).

The Language Representation. The requirements represented as instances of the Requirement Ontology or logic formulas are usually difficult to understand. The LR-procedure provides a natural language description of requirements. It translates instances of the Requirement Ontology into statements in natural language using a limited set of terms (e.g., “always”, “never”, “repetitions”, etc.). The translation grammar for this procedure is low context-sensitive. Hence, for formulating language expressions, the procedure uses mainly analytic rules. The language statements for the *Response* requirement are shown in Fig. 6.

The Graphical Representation. Due to the ambiguity of natural language, the language representations of requirements may be poly-semantic, and, at the same time, their unambiguous formal semantics may be hard to read. For smoothing the ambiguity of the first representation and the low readability of the second, the GR-procedure translates a requirement instance into a graphical representation which is a representative segment of a linear path (for linear time) or a fragment of a computation tree (for branching time) with the depicted events of the requirement. For this visualization, the procedure uses the tables of formulas with the restrictions and analytic rules equally. We develop a method for translating the requirements with linear time into a graphical representation in ASCII format. Examples of the translation are shown in the table in Fig. 6.

Due to the incomplete formalization of technical documentation, the correct extraction of a concurrent system and its requirements cannot be fully automatic. The requirement engineers must be provided with the editors for the data components of the verification process. For the Process and Requirement Ontologies written in the OWL-language, there exist editors, in particular, Protégé. However, for getting more visibility for the Process Ontology, we plan to develop an editor based on the semantic markup ontology [8]. It will present the processes in a tabular form as instances of classes with the corresponding constraints of the domain, and it will visualize the scheme of data flows between processes.

The instances of the Requirement Ontology can be modified by the editor that combines four editing methods depending on the representation type:

- Ontology: changing the values of class attributes within axiomatic constraints.
- Formulas: changing the syntax elements of a formula within the given patterns.
- Language: the limited natural language is used.
- Graphic: the set of patterns for events ordering on a line or in a tree is provided.

All representations should be visible in the same window (the formula and the text are usually not long). Changes in one of the representations affect the others.

8 Conclusion

In this paper, we propose an ontology-based support framework for verification of safety-critical concurrent systems. Our approach has the following advantages. First, a flexible customizing extraction/construction for systems and requirements with respect to various methods of formal verification is provided by formal semantics of ontological representation of concurrent systems and requirements. Second, a variability of verification methods becomes possible due to the customization of formal semantics defined both for the ontological representation of systems and ontological representation of requirements. Third, these formal semantics give the base for checking the integrity and consistency of systems and requirements. Simple requirement consistency checking makes applying formal verification methods easier. Fourth, comprehensible formulation of requirements in our framework is provided by using several requirement representations: the ontological representation, the formal representation as logic formulas, the representation in a limited natural language, and the graphic representation. The customizable tools for viewing, editing, and navigating over these representations help the requirement engineers to deal with requirements. Currently, we use a rule-based, ontology-driven information extraction approach [5], labelled transition systems as formal semantics for concurrent models [9], and logics LTL, CTL and their real-time extensions as formal semantics for requirements [7].

There are two main directions for future work: improving and implementing the internal components of our support framework, and customizing the variable external components.

Internal Components. Extending the Process and Requirement Ontologies with new classes will allow us to capture a wider set of formal semantics, including semantics with real numbers and probabilities. For the Requirement Ontology, we plan to introduce the dependability relation and automatic methods for detecting it. We will extend the set of requirement patterns that can be generated from the Process Ontology with possible general correctness requirements. We will study the possibilities to enrich the set of requirement patterns that can be checked for consistency with ontology means. We also will develop a semantic markup of classes of the Process Ontology for customizing it to a specific subject domain. The Requirement Extraction, Integrity and Consistence, and Representation procedures will also be implemented.

External Components. We will develop new semantics for the Process Ontology, in particular, hyperprocesses [23], abstract state machines [11], and Markov decision processes [16]. We work on designing new methods of translation to the Process Ontology from various formalisms of concurrent system descriptions (e.g., Reflex [24]), to improve our approach. We will customize our information extraction methods for important concurrent systems' subject domains, in particular, for automatic control systems. These methods will also be adopted for tables and diagrams in technical documentation.

Acknowledgment. This research has been supported by Russian Foundation for Basic Research (grant 17-07-01600), Funding State budget of the Russian Federation (IAE project No. AAAA-A17-11706061006-6), and by the BMBF project HPC2SE at WWU Muenster (Germany).

References

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 Trans. Softw. Eng. **41**(7), 620–638 (2015)
2. Clarke, E.M., Henzinger, Th.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-10575-8
3. Dwyer, M., Avrunin, G., Corbett, J.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering (ICSE-99), pp. 411–420. ACM, New York (1999)
4. Garanina, N., Sidorova, E.: Context-dependent lexical and syntactic disambiguation in ontology population. In: Proceedings of the 25th International Workshop on Concurrency, Specification and Programming (CS&P-16), pp. 101–112. Humboldt-Universitat zu Berlin, Berlin (2016)
5. Garanina, N., Sidorova, E., Bodin, E.: A multi-agent text analysis based on ontology of subject domain. In: Voronkov, A., Virbitskaite, I. (eds.) PSI 2014. LNCS, vol. 8974, pp. 102–110. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46823-4_9
6. Garanina, N., Sidorova, E., Kononenko, I., Gorlatch, S.: Using multiple semantic measures for coreference resolution in ontology population. Int. J. Comput. **16**(3), 166–176 (2017)
7. Garanina, N., Zubin, V., Lyakh, T., Gorlatch, S.: An ontology of specification patterns for verification of concurrent systems. In: Proceedings of the 17th International Conference on Intelligent Software Methodology Tools, and Techniques (SoMeT_18), pp. 515–528. IOS Press, Amsterdam (2018)

8. Garanina, N., Anureev, I., Zyubin, V.: Constructing verification-oriented domain-specific process ontologies. Syst. Inform. **14**, 19–30 (2019)
9. Garanina, N., Anureev, I., Borovikova, O.: Verification oriented process ontology. Autom. Control. Comput. Sci. **53**(7), 584–594 (2019). https://doi.org/10.3103/S0146411619070058
10. Garanina, N., Borovikova, O.: Ontological approach to checking event consistency for a set of temporal requirements. In: Proceedings of 5th International Conference on Engineering, Computer and Information Sciences, Novosibirsk, Russia. IEEE (2019)
11. Gurevich, Y.: Evolving algebras 1993: Lipari guide. In: Böorger, E. (ed.) Specification and Validation Methods. Oxford University Press, Oxford (1995)
12. Konrad, S., Cheng, B.: Real-time specification patterns. In: Proceedings of 27th International Conference on Software Engineering, pp. 372–381. ACM, New York (2005)
13. Krishnan J., Coronado P., Reed T.: SEVA: a systems engineer's virtual assistant. In: Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE-19), Palo Alto, California, USA. CEUR-WS (2019). http://ceur-ws.org/Vol-2350/paper3.pdf
14. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: RoboChart: modelling and verification of the functional behaviour of robotic applications. Softw. Syst. Model. **18**(5), 3097–3149 (2019). https://doi.org/10.1007/s10270-018-00710-z
15. Mondragón, O., Gates, A., Roach, S.: Prospec: support for elicitation and formal specification of software properties. Electron. Notes Theor. Comput. Sci. **89**(2), 67–88 (2003)
16. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
17. Salamah, S., Gates, A., Kreinovich, V.: Validated templates for specification of complex LTL formulas. J. Syst. Softw. **85**(8), 1915–1929 (2012)
18. Shanmugham, S., Roberts, C.: Application of graphical specification methodologies to manufacturing control logic development: a classification and comparison. Int. J. Comput. Integr. Manuf. **11**(2), 142–152 (2010)
19. Smith, M., Holzmann, G., Etessami, K.: Events and constraints: a graphical editor for capturing logic requirements of programs. In: Proceedings of 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, pp. 14–22. IEEE (2001)
20. Vu, A.V., Ogawa, M.: Formal semantics extraction from natural language specifications for ARM. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 465–483. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_28
21. Wong, P.Y.H., Gibbons, J.: Property specifications for workflow modelling. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 56–71. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00255-7_5
22. Yu, J., Manh, T.P., Han, J., Jin, Y., Han, Y., Wang, J.: Pattern based property specification and verification for service composition. In: Aberer, K., Peng, Z., Rundensteiner, E.A., Zhang, Y., Li, X. (eds.) WISE 2006. LNCS, vol. 4255, pp. 156–168. Springer, Heidelberg (2006). https://doi.org/10.1007/11912873_18
23. Zyubin, V.: Hyper-automaton: a model of control algorithms. In: Proceedings of Siberian Conference on Control and Communications, Tomsk, Russia, pp. 51–57. IEEE (2007)

24. Zyubin, V., Liakh, T., Rozov, A.: Reflex language: a practical notation for cyber-physical systems. Syst. Inform. **12**, 85–104 (2018)
25. Argosim. www.argosim.com. Accessed 27 Nov 2019
26. HermiT OWL Reasoner. www.hermit-reasoner.com. Accessed 27 Nov 2019
27. Model Based Systems Engineering. www.nasa.gov/consortium/ModelBasedSystems. Accessed 27 Nov 2019
28. Web Ontology Language. www.w3.org/OWL. Accessed 27 Nov 2019
29. Editor Protégé. protege.stanford.edu. Accessed 27 Nov 2019
30. IBM Rhapsody. https://www.ibm.com/se-en/marketplace/systems-design-rhapsody. Accessed 27 Nov 2019
31. SWRL: a Semantic Web Rule Language combining OWL and RuleML. www.w3.org/Submission/SWRL. Accessed 27 Nov 2019
32. Software Cost Reduction. www.nrl.navy.mil/itd/chacs/5546/SCR. Accessed 27 Nov 2019