

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

Natalia Garanina^{1,2,3}, Igor Anureev^{1,2}, Elena Sidorova^{1,3},
Vladimir Zyubin^{2,3}, and Sergei Gorlatch⁴

¹ A.P. Ershov Institute of Informatics Systems, Novosibirsk, Russia
{garanina, anureev, lena}@iis.nsk.su

² Institute of Automation and Electrometry, Novosibirsk, Russia
zyubin@iae.nsk.su

³ Novosibirsk State University, Russia

⁴ University of Muenster, 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 that automatically extracts a concurrent system’s requirements from the technical documentation and formally verifies 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 support are ontological descriptions of the verified system and its requirements. We describe the methods used by our system modules 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 formal verification of concurrent systems. Our verification is based on patterns, because many requirements on real-world systems have recurring formulations with similar properties. We rely on the well-proven methods of model checking for formal verification.

Systems for supporting the development and verification of requirements based on patterns are an active topic of research [4,16,17,19,20,21]. 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 [21] does not yet use all benefits of the ontological knowledge representation.

Our envisaged advantage over the state-of-the-art approaches is that our system supports a user-friendly, integrated strategy for the quality assurance of concurrent systems using a single tool that extracts, corrects, and verifies models, together with textual explanation and visualization of requirements. In particular, our support system offers the following functionality: 1) constructing the model of a concurrent system and the system requirements by extracting information about them from technical documentation, 2) generating typical requirements from the internal description of this extracted model, 3) representing the extracted requirements in a mathematical, linguistic and graphical manner, 4) editing these different representations, 5) checking the integrity and consistency of the model's and requirements' representations, 6) choosing a suitable verifier for checking the model requirements, and 7) translating the model representation into the input language of the selected verifier. In comparison to the state-of-the-art support systems, our generation of requirements both from technical documentation and from the internal description of the concurrent system model is automatic, 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 by the editors of our system.

An important feature of our approach is the intensive use of ontologies for representing knowledge about concurrent systems and their requirements. First, we use ontologies in the information extraction process and, second, all system modules (except a verifier) use ontology instances and relations between them. The ontological representation allows us to realize the following useful functions: 1) specializing concurrent system descriptions for a particular subject domain by using auxiliary ontology axioms and rules; 2) checking the integrity of a concurrent system description; 3) checking the integrity and consistency of a requirements' description; and 4) naturally supporting the term consistency between the description of a system and system's requirements.

The following Section 2 sketches our system for supporting formal system verification as a whole. Section 3 outlines our ontology-based methods used in the module of information extraction and illustrates them with a use case of a bottle-filling system. In Section 4 defines the Requirement and Process Ontologies used for the internal description of systems and requirements. Sections 5–8 describe the program modules of our system and the methods they use. Section 9 discusses our current system's limitations that we plan to address in future work.

2 The System for Supporting Formal Verification

Fig. 1 presents an overview of our envisaged system for supporting formal verification of concurrent systems that automatically extracts and generates system requirements. The key data components of the system are Process Ontology [7] and Requirement Ontology based on patterns [11] (shown in bold boxes in Fig. 1). We use ontologies for the internal representation of concurrent systems and requirements: an ontology is a convenient way to systematize knowledge, which facilitates formulating and checking non-trivial consistency properties. Besides, there are several well-developed tools for creating, editing and checking ontologies [1,3]. In our case, the contents of ontologies are ontological descriptions of a particular concurrent system model and the requirements it must meet.

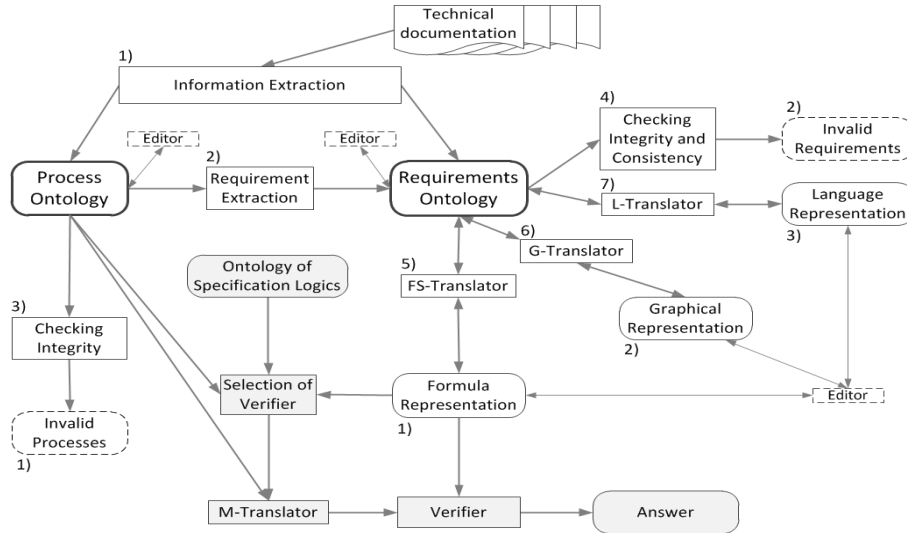


Fig. 1. The System for Supporting Formal Verification

System descriptions are extracted from corpus of technical documentation by the module of information extraction [9,8,10] (box (1) in Fig. 1). The requirements extracted from documentation can be extended with the descriptions of requirements, automatically generated from the ontological description of the system by the module of requirement extraction (box (2)). The descriptions of the system model and the requirements are the basis for formal verification.

To verify a system, it is necessary to 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 system uses the special verification algorithms for specific patterns. In this paper, we do not consider the verification part of our system; it is painted grey in Fig. 1.

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. The integrity and consistency checking modules verify the correctness of the constructed instances of the Process Ontology (box (3) in Fig. 1) and the Requirement Ontology (box (4)). 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 modules are sets of incorrectly constructed entities of the considered concurrent system, as well as incorrectly formulated or inconsistent requirements (dashed rounded boxes (1) and (2) in Fig. 1).

A key feature of our ontologies is their formally defined semantics, which enables formal verification. The Formal Semantic Translator (box (5) in Fig. 1) generates the formulas of temporal logics for the instances of the Requirement Ontology. Its output describes the requirements in the input language of a verification tool. Dealing with requirements involves not only their formal semantics, but also their representations both in natural language and in graphical form. Two translation modules (boxes (6) and (7) in Fig. 1) produce the language and graphical representations of requirements: they are important, because 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. This correction can use the editors for ontology representations, as well as the editors for formal, language and graphical representations (shown in dashed boxes in Fig. 1). In the following sections, we describe in more detail the main ontologies and the non-grey modules in Fig. 1.

3 The Information Extraction Module

Fig. 2 shows the general scheme of our information extraction module that takes technical documentation as input and searches for concepts and relations to populate the Process and Requirement Ontologies described below. We use the rule-based multi-agent approach to implement this module [9].

The process of information extraction includes the preliminary (lexical) step and the main (ontological) step. At the lexical step, the module 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

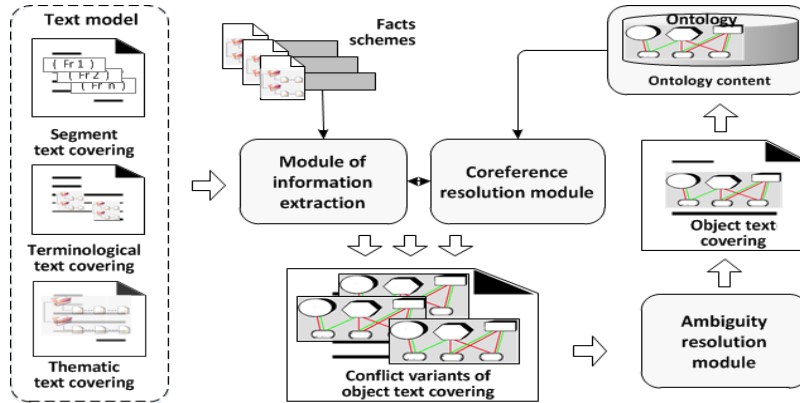


Fig. 2. The Information Extraction and Ontology Population Module

covering is based on the thematic classification methods. At the lexical step, the module constructs objects that represent instances of concepts and relations of the domain ontology from the lexical objects. Our module 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 [8] and co-reference resolution [10] 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 view 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 module 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 also applied to tables or TD schemas containing 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 **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.

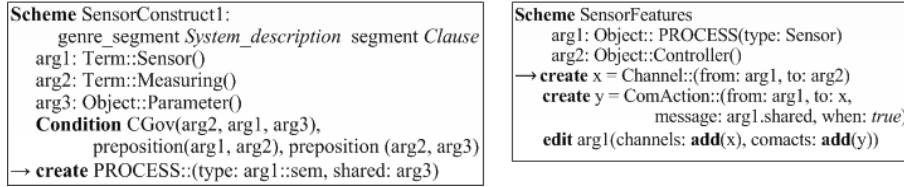


Fig. 3. Example: The lexico-syntactic patterns for extracting objects for “sensors”

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 [3] with the OWL language [2] and the SWRL language [13]. 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 [1].

The Process Ontology. The Process Ontology [7] 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. The Process Ontology has its formal semantics as a labelled transition system.

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)
```

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	<i>id1</i>	Name	<i>Cont</i>	Name	<i>id2</i>
Type	SENSOR	Type	controller	Type	ACTUATOR
Local	-	Local	<i>Sens_1; Do_1: {On, Off}, ...</i>	Local	<i>On, Off:Bool; Cont...</i>
Shared	fluid level	Shared	-	Shared	-
Actions	-	Actions	<i>when true: Cont_id1 ? Sens_1; ...</i>	Actions	<i>when true: Cont_id2 ? Cont; when Cont=Off : Off=true; ...</i>
Channels	<i>Cont_id1</i>	Channels	<i>Cont_id1, Cont_id2, ...</i>	Channels	<i>Cont_id2, Obj_id2, ...</i>
ComActs	<i>when true : Con_id1 ! fluid level</i>	ComActs	<i>when Sens_1 < EMPTY : Cont_id2 ! Off; ...</i>	ComActs	<i>when Off : Obj_id2 ! Stop; ...</i>

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 [5,15]. In these semantics, patterns are formulated using temporal operators, and events are 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 (such as reachability of some states of the system), 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 from natural language text is a content of a certain ontology of a subject domain. The content of the ontology is unique for each individual corpus of technical documentation.

Our Requirement Ontology [11] organizes the existing systems of specification patterns [6,14] into unified structure and contains 11 classes and 14 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-module 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 $\mathbf{G}(id3.Local.On \rightarrow id2.Local.Off \wedge id4.Local.Off)$, 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	<i>id1</i>	Name	<i>gen_id1</i>
Kind	universality	Kind	existence
Time type	linear	Time type	branch
Sub1	$id3.Local.On \rightarrow id2.Local.Off \ \& \ id4.Local.Off$	Sub1	<i>id2.Local.Off</i>
...

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

5 The Requirement Extraction Module

The task of this module 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 module. This module 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 module 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 module formulates the requirements as specially marked instances of the Requirement ontology, and populates the ontology with them. After that, the requirement engineer can change these instance requirements by adding eventual, time and quantitative restrictions, using the editors described below, or remove these requirements.

6 The Integrity and Consistency Modules

The extracted descriptions of concurrent processes and requirements may be incomplete due to insufficient information in the technical documentation. The integrity and consistency modules check 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 Section 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 module. Formal verification of these requirements is a rather time-consuming process. The Consistency part of the module is used to pre-check the simple consistency of these requirements. Since an ontology is just a declarative description of a subject area, 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, a special add-in is required. In this case, it is reasonable to leave compatibility checking for standard formal verification tools. The following SWRL-rule restricts the incompatible pair of requirements: *Proposition p holds always (Universality) vs.*

Proposition p will be false sometime (Existence):

```

Occurrence(?f)^kindPat(?f,Univ)^Prop(?p)^PatS1(?f,?p)^
Occurrence(?g)^kindPat(?f,Exis)^Prop(?q)^PatS1(?f,?q)^
ProOp(?q,Neg)^ProSub1(?q,p) -> Answ(?a)^res(?a,Error1)

```

The modules report to the requirement engineer about the instances that do not satisfy the ontology constraints, and he can correct them in the system editors.

7 The Representation Modules

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

The Formal Semantic Module. For using formal verification methods, requirements for concurrent systems must be presented as formulas of some logic. Since we focus on model checking (e.g., SPIN verification tool [12]), the formal semantics for instances of the Requirement ontology are expressed by formulas of temporal logics. The FST-module 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 module uses mainly the tables of formulas' dependence on the restrictions and several analytic rules. The fragment of the table for *Order* re-

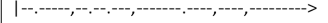
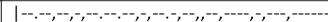

T_p	T_q	D_w	P_w	Q_w	D_p	P_p	Q_p	D_q	P_q	Q_q	Meaning
0	0	0	0	0	0	0	0	0	0	0	 p induces q $\mathbf{G}(p \rightarrow \mathbf{F}q)$
0	0	0	0	0	0	0	0	0	0	z	 p induces zQ repetitions of q $\mathbf{G}(p \rightarrow \mathbf{F}^z q)$
0	0	0	0	0	0	0	0	0	z	0	 p induces appearance q with period zP $\mathbf{G}(p \rightarrow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

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

quirements 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 *Response* 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 Translation Module. The requirements represented as instances of the Requirement Ontology or logic formulas are usually difficult to understand. The LT-module 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 module is low context-sensitive. Hence, for formulating language expressions, the module uses mainly analytic rules. The language statements for the *Response* requirement are shown in Fig. 6.

The Graphic Translation Module. 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 GT-module translates a requirement instance into a graphic 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 module uses the tables of formulas dependence on 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.

8 The Editors

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 a special editor based on the semantic markup ontology. 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:

- Ontological representation: the values of class attributes can be changed following the axiomatic constraints.
- Language representation: the limited natural language is used.
- Graphic representation: the set of patterns for events ordering on a straight line or in a tree is provided.
- Formal representation: the syntax elements of a formula can be changed within the specified patterns.

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.

9 Conclusion

In this paper, we propose an ontology-based support for verification of concurrent systems. Our approach has the following advantages. First, it uses the universal ontological representation of concurrent systems and the requirements on them, which systematizes the system data and requirements and makes possible the integrity and consistence checking. Second, our support provides several representations of concurrent systems and requirements: the initial representation in natural language as technical documentation, the ontological representation, the formal representation as logic formulas, the representation in a limited natural language, and the graphic representation. The system tools for viewing, editing, and navigating over these representations help the requirement engineers to deal with requirements. Third, the formal semantics for ontologies of concurrent systems and requirements are used in formal methods for the verification of concurrent systems to ensure their correctness and quality.

There are several directions for future work. Currently, we have developed Process and Requirement Ontologies, their formal semantics and ontology axioms and rules for them. 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. We work on designing new methods of translation to Process Ontology from various formalisms of concurrent system descriptions (e.g., Reflex [22]), to improve our approach. The information extraction methods are already partially implemented; we will specialize them for important concurrent systems' subject domains, in particular for automatic control systems. The Requirement Extraction, Integrity and Consistence, and Representation modules will also be implemented with the enriched sets of requirement patterns.

Acknowledgment. This research has been supported by Russian Foundation for Basic Research (grant 17-07-01600), and by the BMBF project HPC2SE at WWU Muenster (Germany).

References

1. Hermit OWL Reasoner. <http://www.hermit-reasoner.com/>
2. OWL Web Ontology Language Overview. <https://www.w3.org/TR/owl-features/>
3. Protégé: A free, open-source ontology editor and framework for building intelligent systems. <https://protege.stanford.edu/>
4. Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. *IEEE Transactions on Software Engineering* **41**(7), 620–638 (Jul 2015). <https://doi.org/10/f3nth4>
5. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. MIT Press (1999)
6. Dwyer, M., Avrunin, G., Corbett, J.: Patterns in property specifications for finite-state verification. In: *Proc. of the 21st Int. Conference on Software Engineering*. pp. 411–420. ICSE '99, ACM, New York, NY, USA (1999). <https://doi.org/10/fp54nc>

7. Garanina, N., Anureev, I.: Verification oriented process ontology. *Automatic Control and Computer Sciences* **53**(7) (2019), to appear
8. Garanina, N., Sidorova, E.: Context-dependent lexical and syntactic disambiguation in ontology population. In: *Proc. of the 25th International Workshop on Concurrency, Specification and Programming (CS&P)*. pp. 101–112. Humboldt-Universität zu Berlin (2016)
9. Garanina, N., Sidorova, E., Bodin, E.: A multi-agent text analysis based on ontology of subject domain. In: Voronkov, A., Virbitskaite, I. (eds.) *Perspectives of System Informatics*. LNCS, vol. 8974, pp. 102–110. Springer (2015)
10. Garanina, N., Sidorova, E., Kononenko, I., Gorlatch, S.: Using multiple semantic measures for coreference resolution in ontology population. *International Journal of Computing* **16**(3), 166–176 (2017)
11. Garanina, N., Zubin, V., Lyakh, T., Gorlatch, S.: An ontology of specification patterns for verification of concurrent systems. In: *Proc. of the 17th Intern. Conf. SoMeT-18*. *Frontiers in Artificial Intelligence and Applications*, vol. 303, pp. 515–528. IOS Press, Amsterdam (2018). <https://doi.org/10/c87f>
12. Holzmann, G.: *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, Boston, 1 edition edn. (2003)
13. Horrocks, I., et al.: SWRL: a Semantic Web Rule Language combining OWL and RuleML. <https://www.w3.org/Submission/SWRL/>
14. Konrad, S., Cheng, B.: Real-time specification patterns. In: *Proc. of 27th Intern. Conf. on Software Engineering (ICSE)*. pp. 372–381 (May 2005). <https://doi.org/10/bzk6x5>
15. Koymans, R.: Specifying real-time properties with metric temporal logic. *Real-Time Systems* **2**(4), 255–299 (1990). <https://doi.org/10/fmg8jj>
16. Mondragón, O., Gates, A., Roach, S.: Prospec: Support for elicitation and formal specification of software properties. *ENTCS* **89**(2), 67–88 (Oct 2003). <https://doi.org/10/fv94fk>
17. Salamah, S., Gates, A., Kreinovich, V.: Validated templates for specification of complex LTL formulas. *Journal of Systems and Software* **85**(8), 1915–1929 (2012). <https://doi.org/10/c87s>
18. Shanmugham, S., Roberts, C.: Application of graphical specification methodologies to manufacturing control logic development: A classification and comparison. *Int. J. of Computer Integrated Manufacturing* **11**(2), 142–152 (2010). <https://doi.org/10/df3jsw>
19. Smith, M., Holzmann, G., Etesami, K.: Events and constraints: A graphical editor for capturing logic requirements of programs. In: *Proc. of 5th IEEE International Symposium on Requirements Engineering*. pp. 14–22. IEEE, Toronto, Ontario, Canada (27). <https://doi.org/10/bw6r64>
20. Wong, P., Gibbons, J.: Property specifications for workflow modelling. In: Leuschel, M., Wehrheim, H. (eds.) *Proc. of 7th International Conference “Integrated Formal Methods” (IFM)*. LNCS, vol. 5423, pp. 56–71. Springer Berlin Heidelberg, Düsseldorf, Germany (Feb 2009). <https://doi.org/10/cr25w9>
21. Yu, J., et al.: Pattern based property specification and verification for service composition. In: *Proc. of 7th International Conference on Web Information Systems Engineering*. LNCS, vol. 4255, pp. 156–168. Springer Berlin Heidelberg, Wuhan, China (Oct 2006). <https://doi.org/10/drqf85>
22. Zyubin, V., Liakh, T., Rozov, A.: Reflex language: A practical notation for cyber-physical systems. *System informatics* **12**, 85–104 (2018). <https://doi.org/10/c9b3>