International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol. 7, No. 1, pp. 23 – 44, 2010

TOWARDS SDL ONTOLOGY

Marina Bagić Babac and Marijan Kunštić*

University of Zagreb, Faculty of Electrical Engineering and Computing
HR-10000 Unska 3, Zagreb, Croatia
marina.bagic@fer.hr, marijan.kunstic@fer.hr

Web In this paper we have developed the ontology for Specification and Description Language (SDL), an object-oriented, formal language defined by ITU-T as recommendation Z.100. The language is intended for the specification of real-time complex and concurrent applications that communicate using discrete signals. SDL formal model bridges the gap between ideas in our minds and the actual implementation of the system. It facilitates communication between the software developers and also non-experts without advanced engineering skills.We use SDL Markup Language, an XML-based language, as a medium for translating SDL model to SDL ontology.

Keywords: Specification and Description Language (SDL); SDL Ontology; SDL Markup Language.

1. Introduction

The motivation for this paper lies in the authors belief that a semantically welldefined formal model for the specification of a system is a good starting point in the process of developing a communications software. Specifications are easily defined via process languages such as process algebra or another (graphical or not, formal or informal) languages or frameworks which rely on Finite State Machine (FSM) model. These are, but not limited to, CCS, CSP, UML, different dialects of Petri nets (coloured, timed, object-oriented), etc. We have chosen SDL, Specification and Description Language for a few reasons. First, it is a standardized language by the International Telecommunications Union - Telecommunications Standardization Sector (ITU-T) as Z.100 recommendation and is defined for the specification and description of telecommunications services and systems. Then, it has both graphical and textual notation enabling the non-experts in the engineering and software development to easily follow the workflow of a communications processes.

Furthermore, we find SDL suitable not only for the software specification and design phase, but also for the simulation and testing of implemented system. As it relies on finite state automata, we find it easily translated to process algebra languages which have a strong support for model checking.

*This work was carried out within research project 036-0362027-1640 ”Knowledge-based network and service management”, supported by the Ministry of Science, Education and Sports of the Republic of Croatia.

In this paper we have developed the ontology for SDL. We have described step by step conversion from SDL system specification elements to SDL ontology elements. SDL ontology itself is a contribution to the efforts of Semantic Web development. Analyzing SDL system elements we extract the relations among them, and writing them explicitly we build SDL ontology. We have expressed it in both UML and OWL languages (using Protégé tool). We have also proposed a formal framework of SDL Markup Language (SDL-ML) as a medium for translating SDL model to SDL ontology. So, the benefit of this paper goes to SDL users with a wide range of different application areas, besides the telecommunications one. Any FSM-based system can be well-defined using SDL.

Since SDL ontology has not been defined so far, we relate our paper to those developing ontologies with similar purposes, like the one in [Gašević, D. et al. (2006)], where Petri net ontology was developed. Petri nets are yet another formal and specification language with wide range of usage. We have developed our SDL ontology with the similar development steps, e.g. starting from UML model through OWL mapping to comparison of XML-based languages for Petri nets or SDL models. Another related work is mostly concerned with Semantic Web [Walton, C. (2006)], [Mika, P. (2007)], [Berners-Lee, T. et al. (2001)] and SDL [Ellsberger, J et al. (1997)], [Belina, H. (1997)], [ITU-T (2002)].

The paper is divided into three sections, the first one describing SDL and its major characteristics, then the section describing SDL ontology with UML and OWL, and third, the idea of SDL ontology translation to OWL. In the end we provide the SDL example of INRES protocol which implements a reliable, connection-oriented data transfer service between two users. The protocol operates above a medium that offers an unreliable data transfer service.

2. Specification and Description Language Modeling

Specification and Description Language (SDL) is a graphical specification language standardized by ITU-T (International Telecommunication Union - Telecommunications Standardization Sector). SDL, defined in Z.100, has been evolving since the first recommendation in 1980. Every fourth year an updated revision of the language standard has been adopted. In 1992 Object Oriented features were included in SDL. The standard from 1996, called SDL-96, introduced only minor updates. The current standard is SDL-2000, and it introduces a number of new features, including exception handling, a new data model, and composite states.

Although SDL is widely used in the telecommunications field, it is not designed specifically for describing telecommunications services. Rather SDL is a general purpose specification language for communication systems and embedded systems. The graphical notation, the formal semantics, and object-oriented concepts makes SDL a powerful and versatile language both for systems specification and their implementation [Belina, H. (1997)].

The purpose of recommending SDL was to provide a language for unambiguous

specification and description of the behaviour of telecommunication systems [ITU-T (2002)]. The specifications and descriptions using SDL are intended to be formal in the sense that it is possible to analyse and interpret them unambiguously. The terms specification and description are used with the following meaning [ITU-T (2002)]:

(1) a specification of a system is the description of its required behaviour;
(2) a description of a system is the description of its actual behaviour; that is, its implementation.

The SDL language supports two equivalent notations. In addition to the graphical notation (SDL-GR), the textual notation (SDL-PR) is standardized. The textual notation SDL-PR uses the textual syntax only. The graphical notation SDL-GR not only has graphical components, but also some textual parts that are identical with the textual representation SDL-PR. This is beacuse some specifications, such as the specification of data and signals, are more naturally specified textually [Ellsberger, J et al. (1997)].

The basis for description of behaviour in SDL is communicating extended finite state machines, represented by processes. A process consists of a number of states and a number of transitions connecting the states. Communication between processes is done by signal exchange. Signals can be exchanged between two processes in a system or between a process and the environment of the system. The remote procedure and remote variable paradigms for information exchange between entities in an SDL system are also supported.

2.1. *SDL Characteristics*

SDL is a design and implementation language dedicated to advanced technical systems (i.e., real-time systems, distributed systems, and generic event-driven systems where parallel activities and communication are involved). Typical application areas are high- and low-level telecom systems, aerospace systems, and distributed or highly complex mission-critical systems.

SDL has a set of specialized characteristics that distinguishes it from other technologies [IEC (2009)]:

- **standard** - SDL is a non-proprietary internationally standardized language (ITU-T standard Z.100).
- **formal** - SDL is a formal language ensuring precision, consistency, and clarity in the design that is crucial for mission-critical applications (e.g., most technical systems).
- **graphical and symbol-based** SDL is a graphical and symbol-based language providing clarity and ease of use. An SDL design is both an implementation and its own documentation.
- **object-oriented** (OO) - SDL is a fully OO language supporting encapsulation, polymorphism, and dynamic binding. Moreover, SDL extends the traditional

data-oriented OO class concept by customizing it for technical applications and introducing OO concepts for active objects (e.g. systems, blocks, and state machines).

- **highly testable** - SDL has a high degree of testability as a result of its formalism for parallelism, interfaces, communication, and time. The quality and speed improvements are dramatic compared to traditional nonformal design techniques.
- **portable, scalable, and open**; SDL implementations are independent of cross compilers, operating systems, processors, interprocess communication mechanisms, and distribution methods. A single SDL implementation can be used for many different target architectures and configurations.
- **highly reusable** - SDL provides a high degree of reuse. Because of visual clarity, testability, OO concepts, clear interfaces, and abstraction mechanisms, SDL design has a much higher degree of reusability than any other type of design or implementation.
- **efficient** - The formalism and the level of abstraction that is provided by SDL make it possible to apply sophisticated optimization techniques for cross-compilation.

2.2. *SDL Architecture*

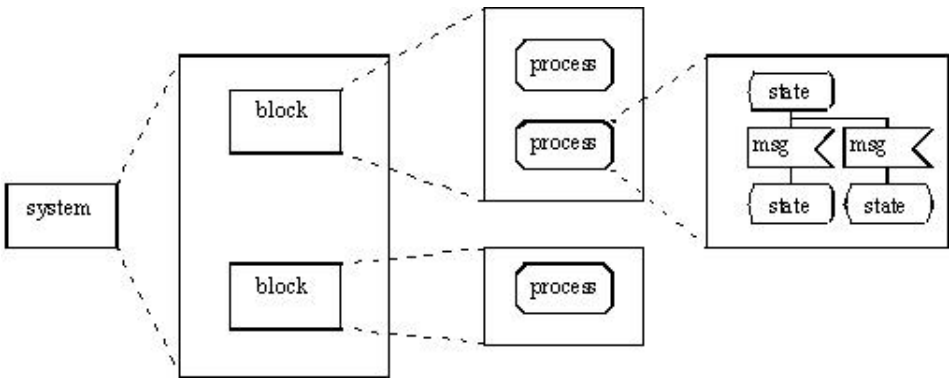

Fig. 1. SDL Architecture

The system description in SDL is divided into two parts; structural and behavioural. Structural part refers to describing system as a black box with interaction to the environment. It contains blocks or agents which contain processes. So, the SDL hierarchy is given as system - block - process sequence which enable the developers to develop the system in top-down manner instead of more exhausting bottom-up approach (Figure 1).

A system specification, in a broad sense, is the specification of both the behaviour and a set of general parameters of the system. However, SDL is intended to specify the behavioural aspects of a system; the general parameters describing properties like capacity and weight have to be described using different technique [SDL-RT (2009)].

2.2.1. *System*

The overall design is called the system and everything that is outside the system is called the environment. There is no specific graphical representation for the system but the block representation can be used if needed.

The corresponding textual notation for the system is given as follows;

system <system name>;
<system declarations>
<type in system specification>
<block interaction>
endsystem[<system name>];

Table 1. SDL: System Textual Notation

2.2.2. *Blocks*

A block (or an agent) is an element in the system structure. There are two kinds of agents: blocks (meaning that a block can contain block) and processes. A system is the outermost block. A block is a structuring element that does not imply any physical implementation on the target. A block can be further decomposed in blocks and so on allowing to handle large systems. A block symbol is a solid rectangle with its name in it.

When the SDL system is decomposed down to the simplest block, the way the block fulfils its functionality is described with processes. A lowest level block can be composed of one or several processes. To avoid having blocks with only one process it is allowed to mix together blocks and processes at the same level e.g. in the same block. A process symbol is a rectangle with cut corners with its name in it.

In Table 2 there is the textual notation for the block.

2.2.3. *Process*

A process is basically the code that will be executed. It is a finite state machine based task and has an implicit message queue to receive messages. It is possible to have several instances of the same process running independently. The number of

block <block name>; <block declarations> <type in block specification> <process interaction> <channel to route connections> endblock[<block name>];

Table 2. SDL: Block Textual Notation

instances present when the system starts and the maximum number of instances are declared between parenthesis after the name of the process. The full syntax in the process symbol is given in Table 3.

process <process name>; input <number of instances at startup> [<maximum number of instances>] <type in process specification> [endprocess[<process name>];]

Table 3. SDL: Process Textual Notation

2.3. *SDL Communication*

Every process instance has its own input message queue to receive the messages listed in the channels, which normally acts on a First In First Out (FIFO) basis. Any signal arriving at the process and belonging to its so-called complete valid input signal set is put into the input queue. In fact the complete valid input signal set defines those signals that the process is prepared to accept and it is not allowed for any other signals to be sent to the process. For an output to contain a signal that is delivered to the process, the signal must be mentioned on the communication paths leading to the process, or in an input of the process. This is because the complete valid input signal set is derived from this information. Processes in the environment are required to behave also as SDL processes, so the environment also only outputs signals that a process can receive [Belina, H. (1997)].

A process description is based on an extended finite state machine. A process state determines which behavior the process will have when receiving a specific stimulation. A transition is the code between two states. The process can be hanging on its message queue or a semaphore or running e.g. executing code.

Signals can be received in the input queue at any time, regardless of whether the process is in a state or interpreting a transition. When a state is entered or

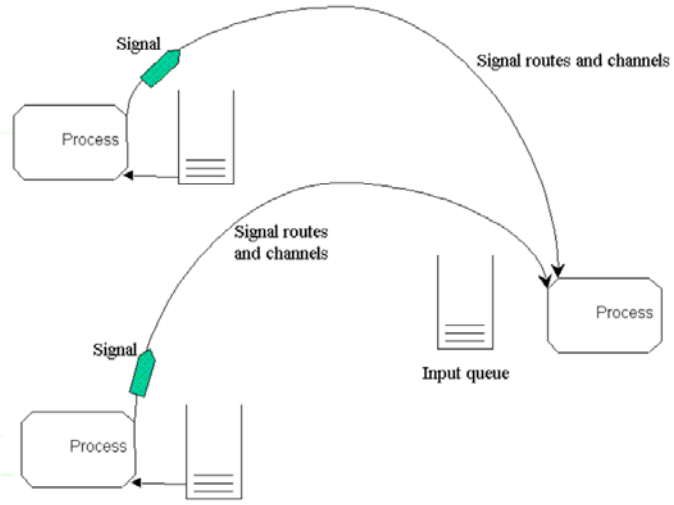

Fig. 2. SDL Communication between Processes

while a process is in a state and receives a signal, the input queue is examined to see if there are any signals that are not saved for that state. If there are only signals that are saved for that state in the input queue, nothing happens and the process remains in the state.

If in a given state the input queue is not empty and there are signals for that state that are not saved, the first such signal (in FIFO order) is removed from the queue (it is consumed), and initiates a transition. If the transition simply leads back to the same state with no other action, the signal is effectively discarded. Discarding a signal is such a common case that there is a short-hand for this. Let us consider

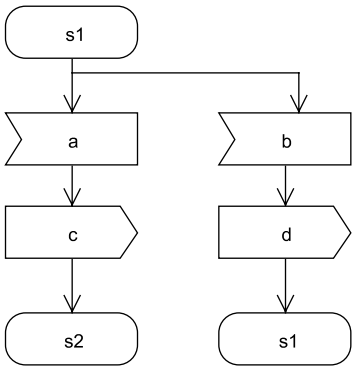

Fig. 3. SDL Process Example

the process of figure 3 in state S1. The input queue contains the signals of type *a, d, a,* and *b*, in the that order. The signals of type *a* and *d* can initiate a transition. A signal of type *a* is first in the input queue. It is removed from the queue, and the process performs the transition to state S2. Now the signal of type *d* is first in the queue. Since the transition for *d* leads directly back to S2 with no action, it is effectively discarded. The next signal is of type *a*, which can initiate a transition from state S2 to some other state [Belina, H. (1997)].

Timers also generate signals in the process input queue. If timer T is activated to the expiration time x, at time x a signal of type T is put into the input queue of the process. If at time x the input queue already contained signals of type a, b and c, these will stay in the input queue before signal T. Any signals that arrive after time x (such as the signals of type d and c) are placed after the signal of type T.

SDL processes run concurrently; depending on the target hardware the behavior might be slightly different. But messages and semaphores are there to handle process synchronization so the final behavior should be independent

3. SDL Ontology Core Development

In this section we give an overview of the approach for developing SDL ontology. Let us recall that the basic intention of this paper is to build the ontology for SDL model specification. We try to express the relations among SDL constructs. The ontology can be expressed and depicted in various modeling techniques and by different tools. However, the meaning of the relations among concepts should remain the same. Basic concepts of SDL are explained in previous section. These are the system, the block and the process and their detailed respective structures and relations among them and their structural elements.

The relations are what it is really about because the Semantic Web is about relations. As the search engines today do not "understand" the relations among different notions, we need a better Web - a Semantic Web which will put each notion into context and provide us with better information once we need it. Therefore, the main issue here it put things into their proper context, i.e. define the relations among the concepts.

3.1. *UML Ontology Representation*

We start with the UML representation of SDL ontology (Figure 4) and then we translate it to OWL language [Antoniou, G. et al. (2008)] using Protégé Tool [Horridge, M. (2009)]. In order to use UML notation we follow the generally adopted convention [Gašević, D. et al. (2006)]:

- Each real concept is described by UML class notation.
- Each synthetic concept is depicted as abstract UML class.
- Relations are described by UML associations (as well as aggregation and composition).

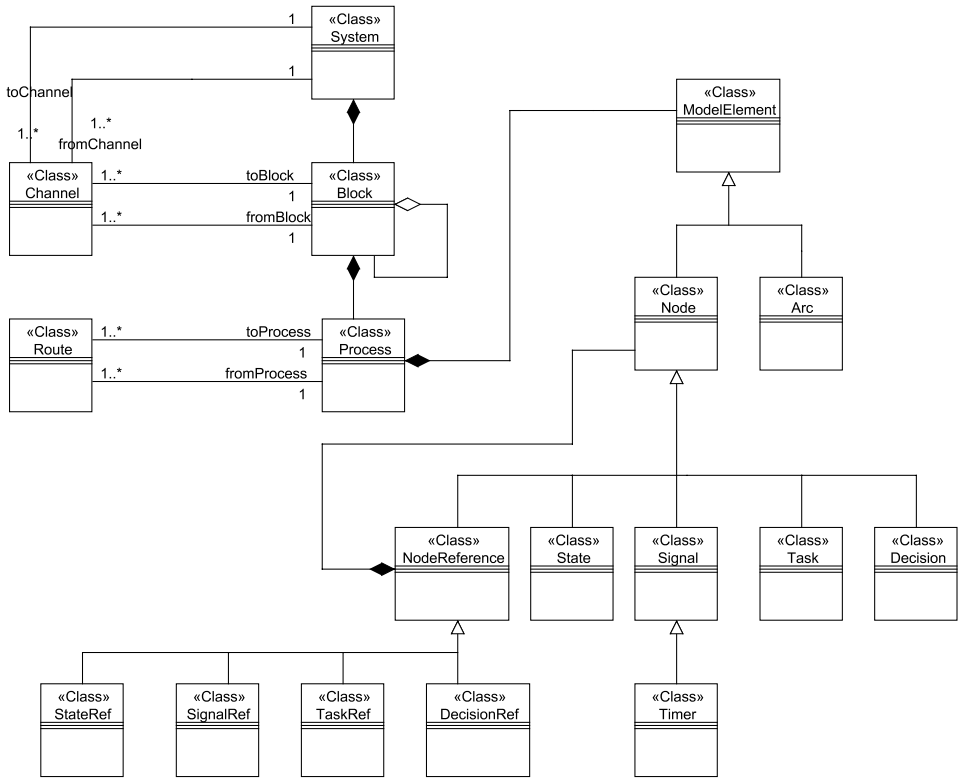

Fig. 4. SDL model ontology - UML hierarchy of core SDL concepts

- Inheritance is depicted as standard UML inheritance relation , i.e. generalization and specialization.

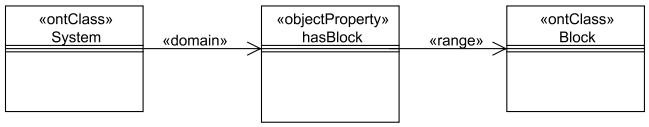

Fig. 5. System Specification for SDL ontology

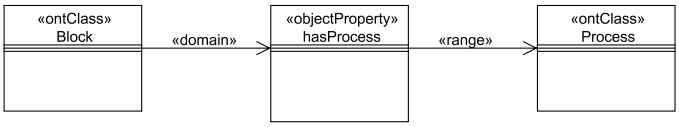

Fig. 6. Block Specification for SDL ontology

We have used the approach similar to [Gašević, D. et al. (2006)] (development of Petri net ontology) where the authors started with root element called *ModelElement* because the UML metamodel uses the same name for its root class and this element is the parent for all elements. However, SDL and Petri nets have a slight difference in modeling hierarchy. While Petri net's module is yet another Petri net, and it is just a reference to it, SDL hierarchy contains another kinds of elements. For instance, block reference may refer to another block, but it may also refer to a process specification. Therefore, it is not trivial to have only one parent element for the whole SDL net and we distinguish hierarchy elements, i.e. system contains blocks, block contains processes, etc. We describe these relationships via OWL object properties using their domains and ranges.

Inherited elements from the Node Class are State, Signal, Task and Decision Classses, as well as NodeReferences to all of these. They are the structural components of the process. They are disjoint classes which means that they have no intersection. Timer Class is inherited from Signal Class which means that a signal may be a timer.

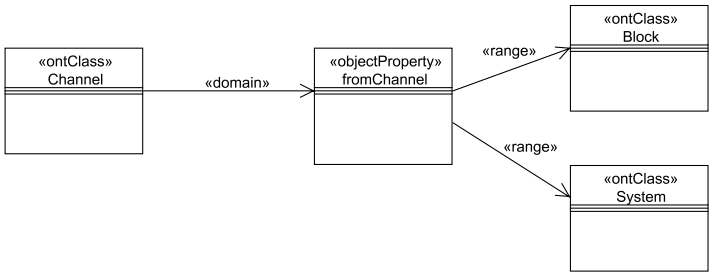

Fig. 7. Property $fromChannel$ Specification for SDL ontology

Each SDL model consists of exactly one system. Each system contains at least one block and each block contains at least one process. Both of these statements are depicted in the Figures 5 and 6 where the first one states that the concept of system is related to the concept of Block via the attribute *hasBlock*, where we additionally specify the multiplicity indicating the minimum number of these relations for each object of System or Block type. Similarly, we define the properties for Channel and Route entities, $fromChannel$ and *toChannel*, $fromRoute$ and *toRoute* (Figures 7

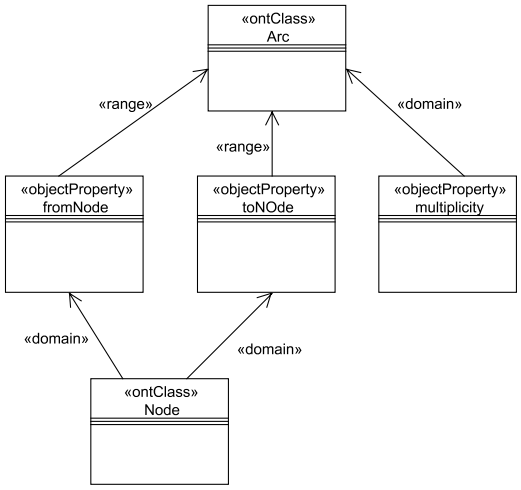

Fig. 8. Arc Specification for SDL ontology

and 8).

Our SDL ontology from Protégé tool is given in Figure 9.

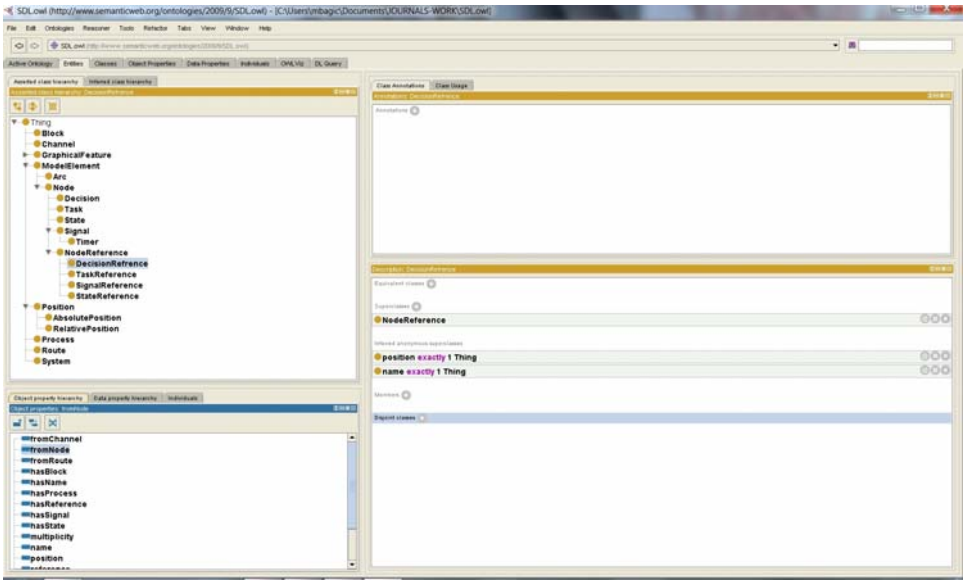

Fig. 9. SDL Ontology Hierarchy in Protégé

As both SDL and its OWL ontology are object-oriented languages, we relate the corresponding elements of both. This way SDL objects are mapped to OWL objects or individuals (Individual is another name for Instance in ontology terminology). So, each SDL system is mapped to OWL system, each SDL block is mapped to OWL block, each SDL signal is mapped to OWL signal. But, when it comes to relations, things get a little more complicated. As the relations in OWL are described via properties, and there are no explicit properties in SDL, we translate SDL hierarchy and its structural elements to corresponding OWL entities and their properties.

4. SDL Markup Language

As we have defined the SDL ontology, we are triggered to impose it to the both SDL and non-SDL users. Our intention is to save their time and energy in the process of using it. Therefore, we believe that a step toward the automated translation from SDL specification to SDL Ontology should be developed. In this paper we outline this idea by introducing the fundamentals for SDL Markup Language (SDL-ML).

Although this combination (double "language" within the name) might sound a little peculiar at first, it actually has a quite logical explanation. As we find SDL a good language for the specification of telecommunications systems as well as the others systems and processes, we believe it would be useful to create a markup language to support SDL for communication with another systems not supporting SDL. As the markup language is actually the description language about things of general meaning, SDL-ML is meant to be the language about the language. The markup language to describe another description language.

SDL-ML is meant to be a SDL interchange format that is independent of specific tools and platforms. Moreover, the interchange format needs to support extensions of SDL. Since this is our first proposal of this language, we shall leave the liberties to change or add some of the elements or their attributes in future. However, this main set of element should retain its semantics. On the other hand, the main problem for automating the process of conversion of SDL or SDL-ML to OWL lies in the hierarchy structure of the both. In SDL (or SDL-ML) the hierarchy exists among the elements in the sense of their structure, i.e. a system holds the blocks, blocks hold another blocks or processes, etc. This is well explained by UML composition relation because the block does not exist without its system, or the process does not exist without its block and system. However, OWL hierarchy is given by inheritance concept, so the verbatim conversion is not possible. In Table 4 we give an overview of the elements translation.

4.1. *SDL-ML Elements*

We naturally use SDL-PR textual notation for the syntax of SDL-ML. We chose the "$xmlns : sdl$" as the namespace for our .sdl XML file.

Element $< sdl : system >$ defines the system element which is the root element for the whole specification, i.e. all other elements are nested into this one. It has

the attribute "*id*" for unique identification.

Element $< sdl : block >$ defines the block element which contains processes. This element is the parent element of the process elements. It has "id" attribute for unique identification.

Element $< sdl : process >$ defines the process element which contains states and signals. This element is the parent element of the set of state and signal elements. It has the "id" attribute for unique identification.

Element $< sdl : state >$ defines the state element. It has the "id" attribute for unique identification. Its nested elements are $< sdl : pre >$ and $< sdl : post >$ denoting the related elements to the state elements, its preceding and subsequent elements.

Element $< sdl : signal >$ defines the signal element. It has the "id" attribute. It has the same nested elements as the $< sdl : state >$ element.

Element $< sdl : channel >$ defines channel to connect block elements within the system. It has the "id" attribute. It also holds nested elements $< from >$, $< to >$ and $< with >$ to specify the elements that it is connected to.

Element $< sdl : route >$ defines route to connect process elements within the specified block. It has the "id" attribute. It also holds nested elements $< from >$, $< to >$ and $< with >$ to specify the elements that it is connected to.

Element $< sdl : task >$ defines the task element. It has the "id" attribute. It has the same nested elements as the $< sdl : state >$ element for defining relations with its preceding and subsequent elements. It also has $< sdl : variable >$ and $< sdl : expression >$ elements denoting the specific task of the process.

Element $< sdl : decision >$ defines the decision element. It has the "id" attribute. It has the same nested elements as the $< sdl : state >$ element for defining relations with its preceding and subsequent elements. It also has $< sdl : question >$ and $< sdl : answer >$ elements denoting the specific decision of the process.

5. SDL Example of INRES Protocol

In this section we give the example of the SDL (SDL-ML, SDL ontology) system specification. We have chosen INRES protocol (Figure 10), and we provide SDL and SDL-ML specification for the Initiator entity (Figure 11) of INRES protocol.

The INRES (Initiator-Responder) protocol contains many OSI concepts and is therefore very suitable for illustrative purposes. The INRES protocol implements a reliable, connection-oriented data transfer service between two users. The protocol operates above a medium that offers an unreliable data transfer service. The INRES service is not symmetrical: it offers only one way data transmission from an initiating process to a responding process [Luukkainen, M. et al. (1998)].

The following service primitives are used for the communication between user and provider [Ellsberger, J et al. (1997)]:

- ICONreq, request of a connection by Initiator-user
- ICONind, indication of a connection by the provider

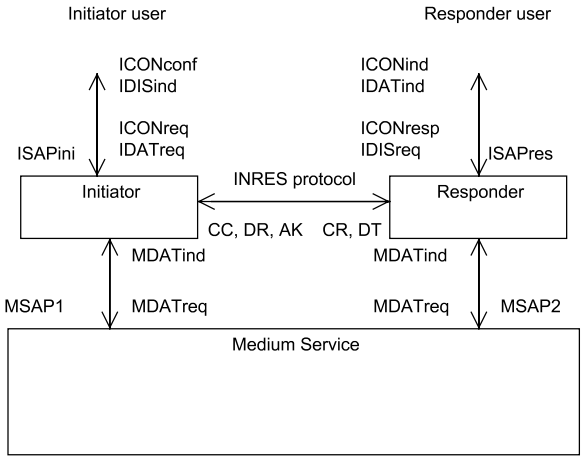

Fig. 10. The INRES Protocol

- ICONresp, response to a connection attempt by Responder-user
- ICONconf, confirmation of a connection by the provider
- IDATreq, data from the Initiator-user to the provider
- IDATind, data from the provider to the Responder-user
- IDISreq, request of a disconnection by the Responder-user
- IDISind, indication of disconnection by the provider

The protocol entities communicate by exchange of the protocol data units CR (connection establishment), CC (connection confirmation), DT (data transfer), AK (acknowledgement) and DR (disconnection).

Here we put some of the .sdl XML file for the specification of Initiator in INRES protocol. The first part in Table 5 specifies the Initiator block and process , while the other one in Table 6 focuses on communication channels between the blocks in INRES protocol. Each SDL element is represented with its corresponding .sdl XML element, which maps also to corresponding OWL class. Once the SDL ontology has been developed, it is easily used in Protégé tool for entities or objects specification of real-time systems.

The same is for the rest of the protocol, but we find Initiator specification sufficient to represent the ideas of SDL-ML code.

6. Conclusion

In this paper we have given our contribution to the efforts of Semantic Web development introducing a new ontology for concurrent and distributed systems specification. Particularly, we have chosen Specification and Description Language, a ITU-T standard Z.100 for specifying and describing telecommunications systems. We have developed SDL ontology using UML and OWL languages. UML has been used as

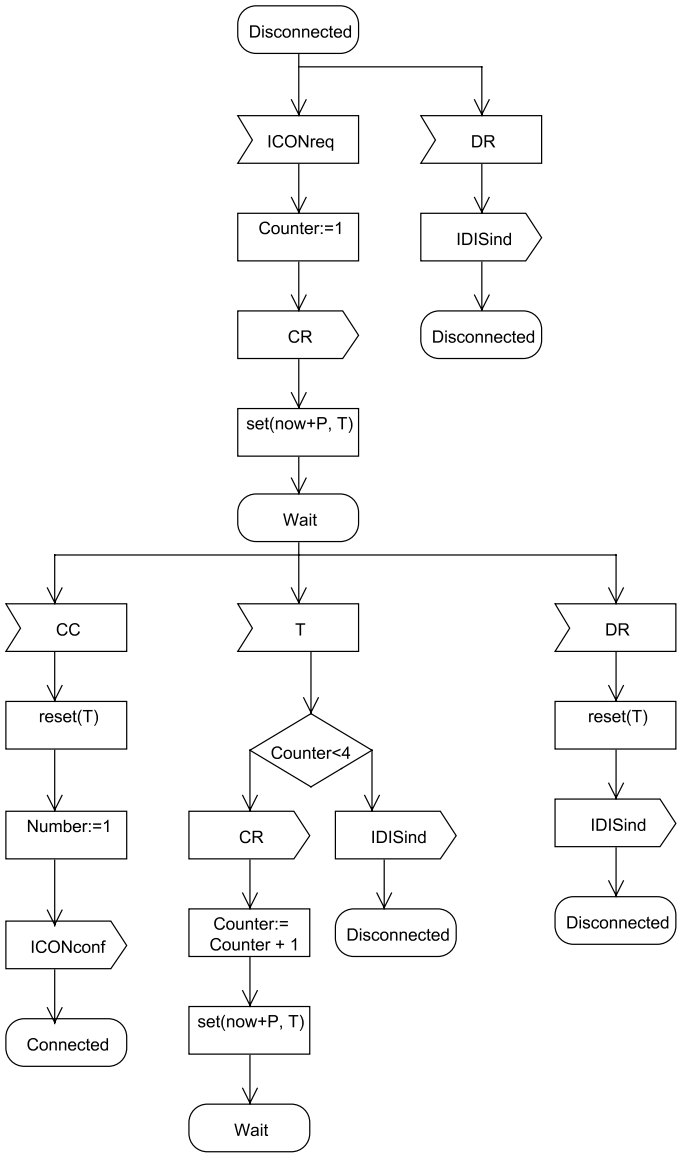

Fig. 11. The Initiator in INRES Protocol

a graphical notation for the ontology specification, which was then translated to Protégé tool to obtain OWL file. We have given the example of INRES protocol specification.

Besides the SDL Ontology we have suggested the development of SDL Markup Language. The idea is to find the mechanism of mapping SDL to OWL directly, i.e.

mapping SDL system specification to its SDL ontology specification.

References

Ellsberger, J., Hogrefe, D., Sarma, A. (1997). SDL: Formal Object-Oriented Language for Communicating Systems, Prentice Hall PTR

Belina, H. (2007). Tutorial on SDL-88, 1997., http://www.sdl-forum.org/sdl88tutorial/, SDL Forum Society

Walton, C. (2006). Agency and the Semantic Web, Oxford University Press, ISBN 0199292485.

Antoniou, G. and Van Harmelen, F. (2008). A semantic web primer. Second edition. Cambridge, MA: The MIT Press, xxi, 264 pp. ISBN: 978-0-262-01242-3.

Mika, P. (2007). Social Networks and the Semantic Web, Series: Semantic Web and Beyond, Vol. 5, XIV, 234 p. 74 illus., Hardcover, ISBN: 978-0-387-71000-6

Gašević, D. and Devedžić, V. (2006). Petri net Ontology, Knowledge-Based Systems, Elsevier, Vol. 19, Issue 4, 220–234.

Gašević, D. (2004). Petri net Ontology, PhD dissertation (in Serbian), Department of Information Systems and Technologies, FON School of Business Administration, University of Belgrade

ITU-T Z.100. (2002). Telecommunication standardization sector of ITU, Series ZSERIES Z: Languages and General Software Aspects for Telecommunications Systems, Formal description techniques (FDT) Specification and Description Language (SDL)

Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y. (2003). Reasoning About Knowledge, The MIT Press, Cambridge Massachusetts, London England

Huth, M.R.A., Ryan, M.D. (2000). Logic in Computer Science: Modelling and reasoning about systems, Cambridge University Press, Cambridge, England, UK

Shadbolt, N., Berners-Lee, T., Hall, W. (2006). The Semantic Web Revisited, IEEE Intelligent Systems 21(3), 96–101

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web, *Scientific American Magazine*

Horridge, M. (2009). A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, Edition 1.2, University of Manchester

Luukkainen, M. and Ahtiainen. (1998). A. Compositional Verification of SDL Descriptions. Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC SAM'98, Berlin, Germany

IEC. (2009). Specification and Description Language (SDL), Definition and Overview, http://www.iec.org/online/tutorials/sdl/index.asp

SDL-RT. (2009). Specification and Description Language - real-time, http://www.sdl-rt.org/standard/V2.2/html/SDL-RT.htm

SDL-ML ELEMENTS	**OWL ELEMENTS**
System <sdl : system id="id"> </sdl:system>	Class **System** $hasID \in [System, Literal]$ $hasSignal \in [System, Signal]$ $hasBlock \in [System, Block]$ $hasChannel \in [System, Channel]$
Block <sdl : block id="id"> </sdl:block>	Class **Block** $hasID \in [Block, Literal]$ $hasSignal \in [Block, Signal]$ $hasBlock \in [Pro, Block]$ $hasRoute \in [Channel, Route]$
Process <sdl : process id="id"> </sdl:process>	Class **Process** $hasID \in [Process, Literal]$ $hasSignal \in [Process, Signal]$ $hasState \in [Pro, Process]$
Channel <sdl : channel id="id"> <from> </from> <to> </to> <with> </with> </sdl:channel>	Class **Channel** $hasID \in [Channel, Literal]$ $fromChannel \in [Channel, Block]$ $toChannel \in [Block, Channel]$ $withSignal \in [Channel, Signal]$
Route <sdl : route id="id"> <from> </from> <to> </to> <with> </with> </sdl:route>	Class **Route** $hasID \in [Route, Literal]$ $fromRoute \in [Route, Process]$ $toRoute \in [Process, Route]$ $withSignal \in [Route, Signal]$
Signal <sdl : signal id="id" type="incoming/outgoing/timer"> <sdl:pre> </sdl:pre> <sdl:post> </sdl:post> </sdl : signal>	Class **Signal** $hasID \in [Signal, Literal]$ $hasType \in [Signal, Literal]$ $hasPre \in [Signal, Node]$ $hasPost \in [Signal, Node]$
State <sdl:state id="id"> <sdl:pre> </sdl:pre> <sdl:post> </sdl:post> </sdl:state>	Class **State** $hasID \in [Signal, Literal]$ $hasPre \in [State, Node]$ $hasPost \in [State, Node]$
Task <sdl:task name="name"> <sdl:variable> </sdl:variable> <sdl:expression> </sdl:expression> <sdl:pre> </sdl:pre> <sdl:post> </sdl:post> </sdl:task>	Class **Task** $hasName \in [Task, Literal]$ $hasVariable \in [Task, Literal]$ $hasExpression \in [Task, Literal]$ $hasPre \in [Task, Node]$ $hasPost \in [Task, Node]$

Decision	Class **Decision**
<sdl:decision name="name">	$hasName \in [Decision, Literal]$
<sdl:question> </sdl:question>	$hasVariable \in [Decision, Literal]$
<sdl:pre> </sdl:pre>	$hasPre \in [Decision, Node]$
<sdl:answer> </sdl:answer>	$hasExpression \in [Decision, Literal]$
<sdl:post> </sdl:post>	$hasPost \in [Decision, Node]$
</sdl:decision>	

Table 4. Mapping SDL-ML to OWL

```
<sdl:system name="INRES">
  <sdl:block id="Initiator">
    <sdl:process id="Initiator">
      <sdl:state id="Disconnected">
        <sdl:post name="signal">ICONreq</post>
        <sdl:post name="signal">DR</post>
      </sdl:state>
      <sdl:signal id="ICONreq" type="incoming">
        <sdl:pre name="state">Disconnected</pre>
        <sdl:post name="task">Counter:=1</post>
      </sdl:signal>
      <sdl:signal id="DR" type="incoming">
        <sdl:pre name="state">Disconnected</pre>
        <sdl:post name="signal">IDISind</post>
      </sdl:signal>
      <sdl:task id="Counter">
        <sdl:variable>Counter</sdl:variable>
          <sdl:expression>Counter:=1</sdl:expression>
        <sdl:pre name="signal">ICONreq</pre>
        <sdl:post name="signal">CR</post>
      </sdl:task>
      <sdl:signal id="IDISind" type="outgoing">
        <sdl:pre name="signal">DR</pre>
        <sdl:post name="state">Disconnected</post>
      </sdl:signal>
      <sdl:signal id="CR" type="outgoing">
        <sdl:pre name="task">Counter:=1</pre>
        <sdl:post name="task">set(now+P,T)</post>
      </sdl:signal>
      <sdl:task id="setT">
        <sdl:variable>T</sdl:variable>
          <sdl:expression>set(now+P,T)</sdl:expression>
        <sdl:pre name="signal">CR</pre>
        <sdl:post name="state">Wait</post>
      </sdl:task>
      <sdl:state id="Wait">
        <sdl:pre name="task" name="setT">set(now+P,T)</post>
        <sdl:post name="signal">CC</post>
        <sdl:post name="signal">T</post>
        <sdl:post name="signal">DR</post>
      </sdl:state>
```

```
<sdl:signal id="CC" type="incoming">
  <sdl:pre name="state">Wait</pre>
  <sdl:post name="task">reset(T)</post>
</ /sdl:signal>
<sdl:signal id="T" type="timer">
  <sdl:pre name="state">Wait</pre>
  <sdl:post name="decision">Counter¡4</post>
</ /sdl:signal>
<sdl:signal id="DR" type="incoming">
  <sdl:pre name="state">Wait</pre>
  <sdl:post name="task">reset(T)</post>
</ /sdl:signal>
<sdl:task id="resetT">
  <sdl:variable>T</sdl:variable>
    <sdl:expression>reset(T)</sdl:expression>
  <sdl:pre name="signal">CC</pre>
  <sdl:post name="task">Number:=1</post>
</ /sdl:task>
<sdl:task id="resetT">
  <sdl:variable>T</sdl:variable>
    <sdl:expression>reset(T)</sdl:expression>
  <sdl:pre name="signal">DR</pre>
  <sdl:post name="signal">IDISind</post>
</ /sdl:task>
<sdl:task id="number">
  <sdl:variable>Number</sdl:variable>
    <sdl:expression>Number:=1</sdl:expression>
  <sdl:pre name="task" name="resetT">reset(T)</pre>
  <sdl:post name="signal">ICONconf</post>
</ /sdl:task>
<sdl:signal id="ICONconf" type="outgoing">
  <sdl:pre name="task" name="number">Number:=1</pre>
  <sdl:post name="state">Connected</post>
</ /sdl:signal>
<sdl:state id="Connected">
  <sdl:pre name="signal">ICONconf</post>
</ /sdl:state>
<sdl:signal id="IDISind" type="outgoing">
  <sdl:pre name="resetT">reset(T)</pre>
  <sdl:post name="state">Disconnected</post>
</ /sdl:signal>
```

```
        <sdl:state id="Disconnected">
          <sdl:pre name="signal">IDISind</pre>
        </sdl:state>
        <sdl:decision id="counter">
          <sdl:pre name="signal" type="timer">T</pre>
          <sdl:question>Counter<4</pre>
          <sdl:answer>true</answer>
            <sdl:post name="signal" type="outgoing">CR</post>
          <sdl:answer>false</answer>
            <sdl:post name="signal" type="outgoing">IDISind</post>
        </sdl:decision>
        <sdl:signal id="CR" type="outgoing">
          <sdl:pre name="decision" name="counter">Counter<4</pre>
          <sdl:post name="task">Counter:=Counter+1</post>
        </sdl:signal>
        <sdl:task id="counter">
          <sdl:variable>Counter</sdl:variable>
            <sdl:expression>Counter:=Counter+1</sdl:expression>
          <sdl:pre name="signal" type="outgoing">CR</pre>
          <sdl:post name="task">set(now+P,T)</post>
        </sdl:task>
        <sdl:task id="setT">
          <sdl:variable>T</sdl:variable>
            <sdl:expression>set(now+P,T)</sdl:expression>
          <sdl:pre name="task">Counter:=Counter+1</pre>
          <sdl:post name="state">Wait</post>
        </sdl:task>
      </sdl:process>
    </sdl:block>
</sdl:system>
```

Table 5. SDL-ML Code for Initiator Block and Process in INRES Protocol

```
<sdl:channel name="Ch-Initiator-Responder">
  <sdl:from>Initiator<sdl:from>
    <sdl:to>Responder<sdl:to>
      <sdl:with>CR<sdl:with>
      <sdl:with>DT<sdl:with>
  <sdl:from>Responder<sdl:from>
    <sdl:to>Initiator<sdl:to>
      <sdl:with>CC<sdl:with>
      <sdl:with>DR<sdl:with>
      <sdl:with>DK<sdl:with>
</sdl:channel>
<sdl:channel name="Ch-Initiator-MediumService">
  <sdl:from>Initiator<sdl:from>
    <sdl:to>MediumService<sdl:to>
      <sdl:with>MDATreq<sdl:with>
      <sdl:with>MSAP1<sdl:with>
  <sdl:from>MediumService<sdl:from>
    <sdl:to>Initiator<sdl:to>
      <sdl:with>MDATind<sdl:with>
</sdl:channel>
<sdl:channel name="Ch-Initiator-Env">
  <sdl:from>Initiator<sdl:from>
    <sdl:to>Environment<sdl:to>
      <sdl:with>ICONconf<sdl:with>
      <sdl:with>IDISind<sdl:with>
  <sdl:from>Environment<sdl:from>
    <sdl:to>Initiator<sdl:to>
      <sdl:with>ICONreq<sdl:with>
      <sdl:with>IDATreq<sdl:with>
      <sdl:with>ISAPini<sdl:with>
</sdl:channel>
```

Table 6. SDL-ML Code for Channels Specification of INRES Protocol