Mapping SDL Specification Fundamentals to Core SDL Ontology

Marina Bagić Babac and Marijan Kunštić

Original scientific paper

Abstract—**This paper gives a contribution in the efforts of Semantic web ontology development. We have developed the core ontology for Specification and Description Language (SDL), an object-oriented, formal language defined by the International Telecommunications Union - Telecommunications Standardization Sector (ITU-T) as recommendation Z.100. The language is intended for the specification of complex, event-driven, real-time, and interactive applications involving many concurrent activities that communicate using discrete signals.**

Using SDL formal model for system specification we bridge the gap between ideas in our minds and the actual implementation of the system. Being visually appealing SDL provides us with a simple tool for communication either between the software developers or between non-experts without advanced engineering skills.

In this paper we propose the ontology for the basic SDL system and process elements. We also propose a formal framework of SDL Markup Language as a medium for translating SDL model to SDL ontology.

Index Terms—**Telecommunications Process Specification, Specification and Description Language (SDL), SDL Ontology, SDL Markup Language.**

I. Introduction

Telecommunications companies are mostly focused on the activities that create added values to the services for their customers. Therefore, the business process modeling has become a major focus of attention in business analysis and information systems engineering. Process models can be used for analysis, design, simulation and automated execution of business processes. There are various software tools for modeling, simulating and execution of business processes. Also, different process modeling languages have been developed, e.g. Business Modeling Language, Event-driven Process Chains, IDEF3, Specification and Description Language, Role-Activity Diagram, Task Structures, Unified Modeling Language, etc.

The motivation for this paper lies in the authors belief that a semantically well-defined formal model for the specification of telecommunications services is a good starting point in the process of developing a telecommunications 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 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 of all, 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 which are of our particular interest. 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 telecommunications 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.

Inspired with the idea of Semantic Web [14] and triggered by Petri net ontology development [8], we have come to the idea of SDL ontology development. As the Semantic Web is the Web of data that is understandable by computers, we have proposed SDL ontology concept, but also the idea how to use this ontology without advanced knowledge of either the Semantic Web or the other techniques besides SDL language. We have proposed a formal framework of SDL Markup Language (SDL-ML) as a medium for translating SDL model to SDL ontology. Its basic elements are introduced as well as parts of .sdl XML-based file for SDL-ML system specification.

Since SDL ontology has not been defined so far, we relate our paper to those developing ontologies with similar purposes, like the one in [8], 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 [3], [5], [14] and SDL [1], [2], [10]. The basic intention of the paper is to draw attention to "translating" formal methods into the languages for the Semantic web.

The reason for choosing SDL over UML (which is currently more widely accepted) is that SDL has more formal and stronger syntax and semantics. SDL is a formal language, while UML is not so strict. Stronger requirements on syntax and semantics provide us with tools for model checking and other verifications tools and methods.

The paper is divided into three sections, the first one

Manuscript received November 11, 2009; and revised January 15, 2010.

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.

Authors are with the Faculty of Electrical Engineering and Computing, University of Zagreb, ({marina.bagic, marijan.kunstic}@fer.hr).

1845-6421/10/8190@2010 CCIS

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 Web services.

II. Specification and Description Language Modeling

Specification and Description Language (SDL) is a graphical specification language standardized by ITU (International Telecommunication Union). 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 [19].

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 [19].

The purpose of recommending Specification and Description Language (SDL) was to provide a language for unambiguous specification and description of the behaviour of telecommunication systems [10]. The specifications and descriptions using SDL are intended to be formal in the sense that it is possible to analyze and interpret them unambiguously. The terms specification and description are used with the following meaning [10]:

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 because some specifications, such as the specification of data and signals, are more naturally specified textually [1].

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.

A. 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 [17]:

- **standard** - SDL is a nonproprietary 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 informal 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.

B. SDL Architecture

The system description in SDL is divided into two parts; structural and behavioral. 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 behavioral aspects of a system; the general parameters describing properties like capacity and weight have to be described using different technique [16].

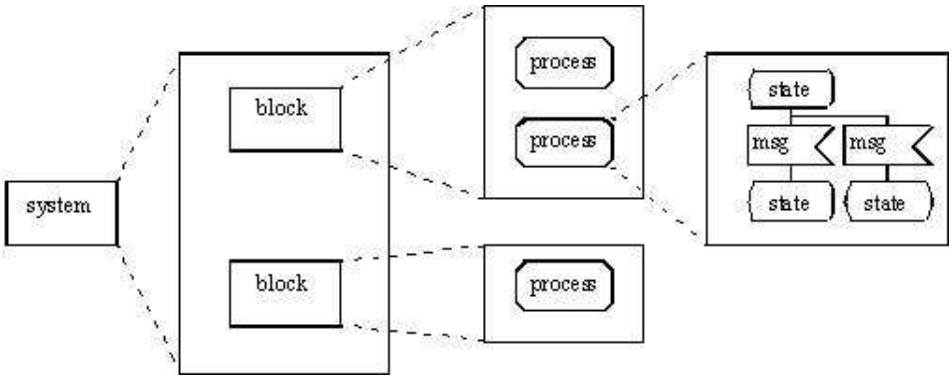

Fig. 1. SDL Architecture

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 I
SDL: SYSTEM TEXTUAL NOTATION

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.

Here is the textual notation for the block;

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

TABLE II
SDL: BLOCK TEXTUAL NOTATION

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 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 III.

process <process name>; <process declaration> <type in process specification> <process body> [endprocess[<process name>];]

TABLE III
SDL: PROCESS TEXTUAL NOTATION

Each SDL process is composed of a numerous states, even extended states (variable-based) for the prevention of state space explosion. Textual notation for the state is given in Table IV.

state <signal name>; input <signal name> [<transition>] nextstate <state name> [endstate[<state name>];]

TABLE IV
SDL: STATE TEXTUAL NOTATION

C. 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 (Figure 2). 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

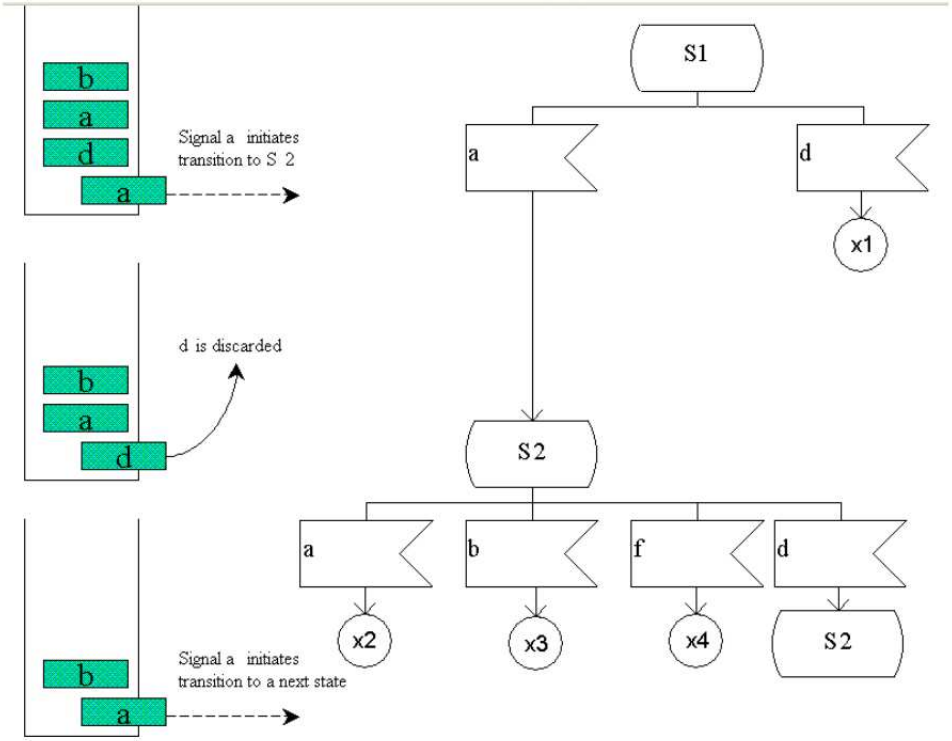

Fig. 3. SDL Input Queue for the Signals [2]

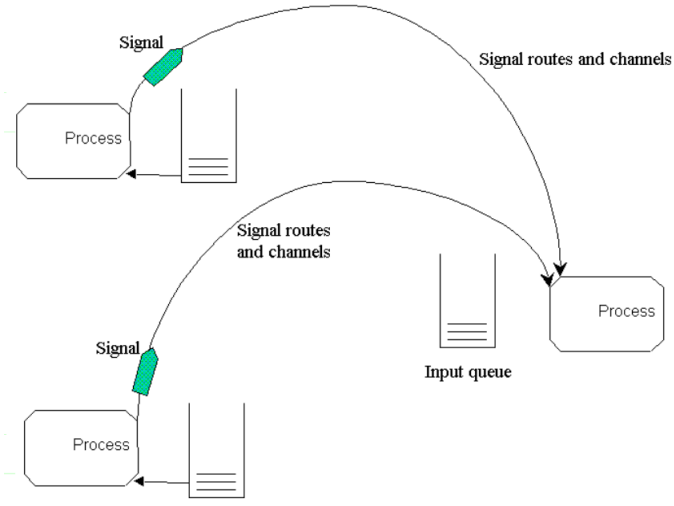

Fig. 2. SDL Communication between Processes [2]

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 [2].

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 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. Figure 3 shows an example. Discarding a signal is such a common case that there is a short-hand for this.

Let us consider 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 [2].

Timers also generate signals in the process input queue. If

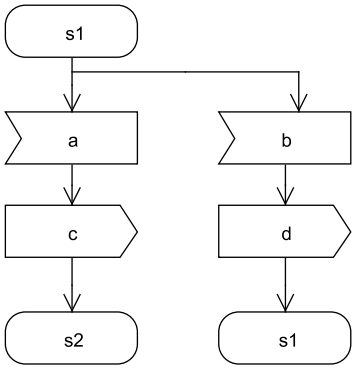

Fig. 4. SDL Process Example

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. But the timers are, at this moment, not in the scope of this paper and they will be considered as the element for extending core SDL ontology. More precisely, timers are here treated as any other signal within the process.

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

Note that in a state diagram the previous statement is always connected to the symbol upper frame and the next statement is connected to the lower frame or on the side.

D. SDL Case Study: Web Services Description

Here, we explain a case study on Web service modeling by SDL. We model the problem from [7] where Web services are developed using Tropos methodology and in [6] where process algebra was used for describing and reasoning on Web services. This case-study lies in the field of public welfare, extracted from a larger domain analysis concerning the local government of Trentino (Italy). The problem is defined as follows.

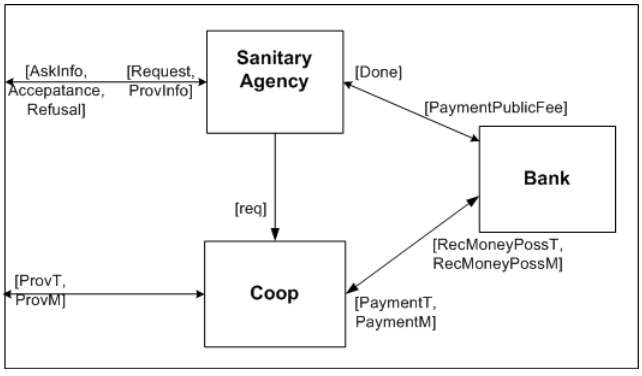

Fig. 5. Sanitary Agency System Specification

A software system aims at supporting elderly citizens in receiving sanitary assistance from the public administration. This problem involves several actors. The main actors and goals of the domain are: the Citizen that aims at being assisted; the Sanitary agency which aims at providing a fair assistance to the citizens; the TransportsInc which provides transportation services; the MealsInc which delivers meals at home; and the Bank which handles the government's finances. The dependencies and expectations that exist among these actors are, for instance, the citizen depends on the sanitary agency for being assisted, etc.

In terms of Web services this service is seen as a "client" of the other services involved in the choreography and we want these other services to be compatible with it. Intuitively, this means that this set of services has to match the behaviour of the client [6].

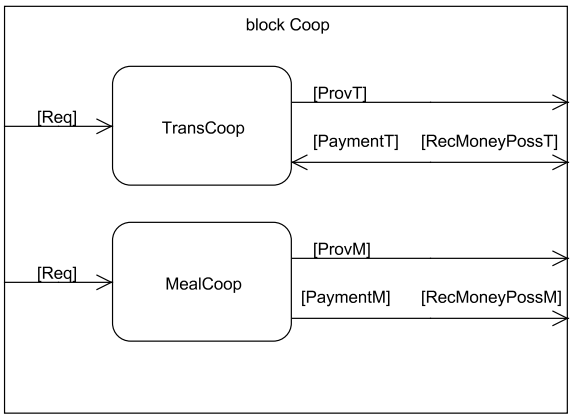

Fig. 6. Sanitary Agency System Specification

The whole system is composed of services involved when elderly people apply for a sanitary assistance: the sanitary agency satisfying requests, the transportation service, the meal delivery, the bank managing funds. We start with a view of all the processes involved in this system as well as the synchronizations between these entities.

The service is to be worked out as a citizen behaviour. A citizen posts a request, exchanges information with the agency, waits for a response, and if accepted receives a service and pays fees. This behaviour we model with signals from the environment to sanitary agency where the citizen is seen as the environment. So, this is the starting point in the development of SDL system for this Web service. SDL system is perceived as a black box at the highest level of abstraction in specifying system's behaviour.

We model graphically the system as a box containing others boxes which are the building blocks for the system. From the outside, i.e. environment, the system is seen a black box with the set of input and output signals. These signals are then distributed through the system's blocks via channels. We call these blocks Sanitary Agency, Bank and Coop (Figure 5). We specify the signals on different channels, e.g. $askInfo, accepatance, refusal$ as signals on bidirectional channel connecting block Sanitary Agency with the environment, in the direction from Sanitary Agency to the

environment. Signals $request, provInfo$ are specified on the same channel but in the opposite direction.

Let us notice here that compared to the specification of the same problem in [6], where Process Algebra is used, we do not have signal "doubling". The channel's direction specifies which are the entities receiving and which are sending the signal, while in Process Algebra there is a necessity to write the signal twice; once for the sending and once for the receiving.

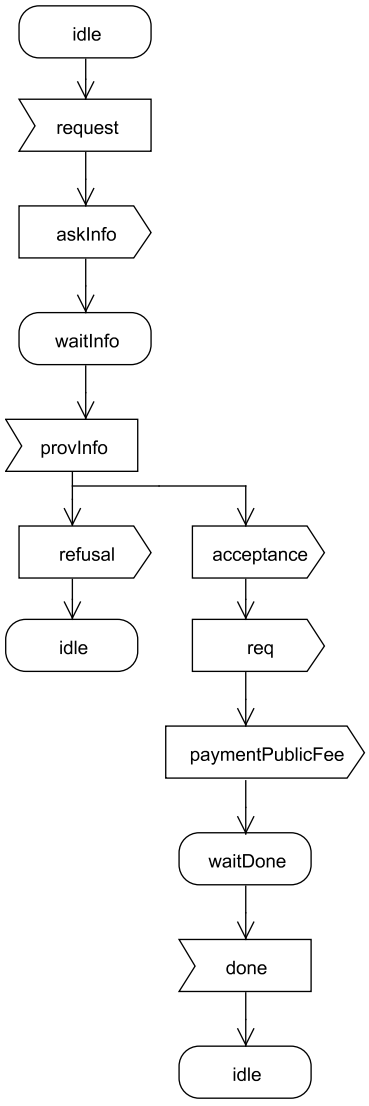

Fig. 7. Sanitary Agency Process Specification

Once the system is specified the next step is to go into more detail at the block level. According to SDL standard each block contains at least one block or process. For our Web service we have Sanitary Agency process inside the Sanitary Agency block. The same is for the Bank block which holds one Bank process. Of course, these two differ in input and output signals that are now described using routes among processes. Our Coop block contains two processes, TransCoop and MealCoop as we make difference between these two services.

After the blocks are specified, we enter the major phase of specifying the processes. We describe the process behaviour with a set of input and output signals and a set of states. These are the sufficient concepts for building our SDL ontology. At this level of development we do not need the complex data structures carried within the signals or time measurement as it is merely the signal.

The sanitary agency manages requests submitted by elderly citizens. First, it asks some information to the citizen who has posted the request. Depending on that, it sends either a refusal and waits for a new request, or the request is accepted. In the latter case, a synchronization is performed with the cooperative controller (it controls in some way both cooperatives) to order the delivery of concrete (transportation or meal) services. Then, the agency pays some public fees to the bank and waits from the bank component for a signal indicating that the transaction is completed. All these signals are specified in Figure 7 with the symbols for input and output signals and state symbol.

Then, we introduce the behaviour of the cooperative. The transportation (resp. meal) cooperative provides its service, warns the bank to start the payment, and waits for the reception of its fees. A controller, called Coop, receives a notification of the agency and launches one of the possible services. Since these two services exhibit the same behaviour, we only give one figure for this process specification (Figure 8).

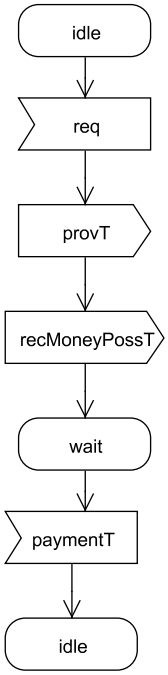

Fig. 8. Transport Cooperative Process Specification

III. SDL Ontology Core Development

In this section we give an overview of the approach for developing core SDL ontology. By core ontology we refer to the basic concepts of SDL system modeling, i.e. SDL process modeling with bare symbols for states and transitions between them. The advanced features of SDL including timers, data structures, etc. can be derived once the basic concepts were agreed upon.

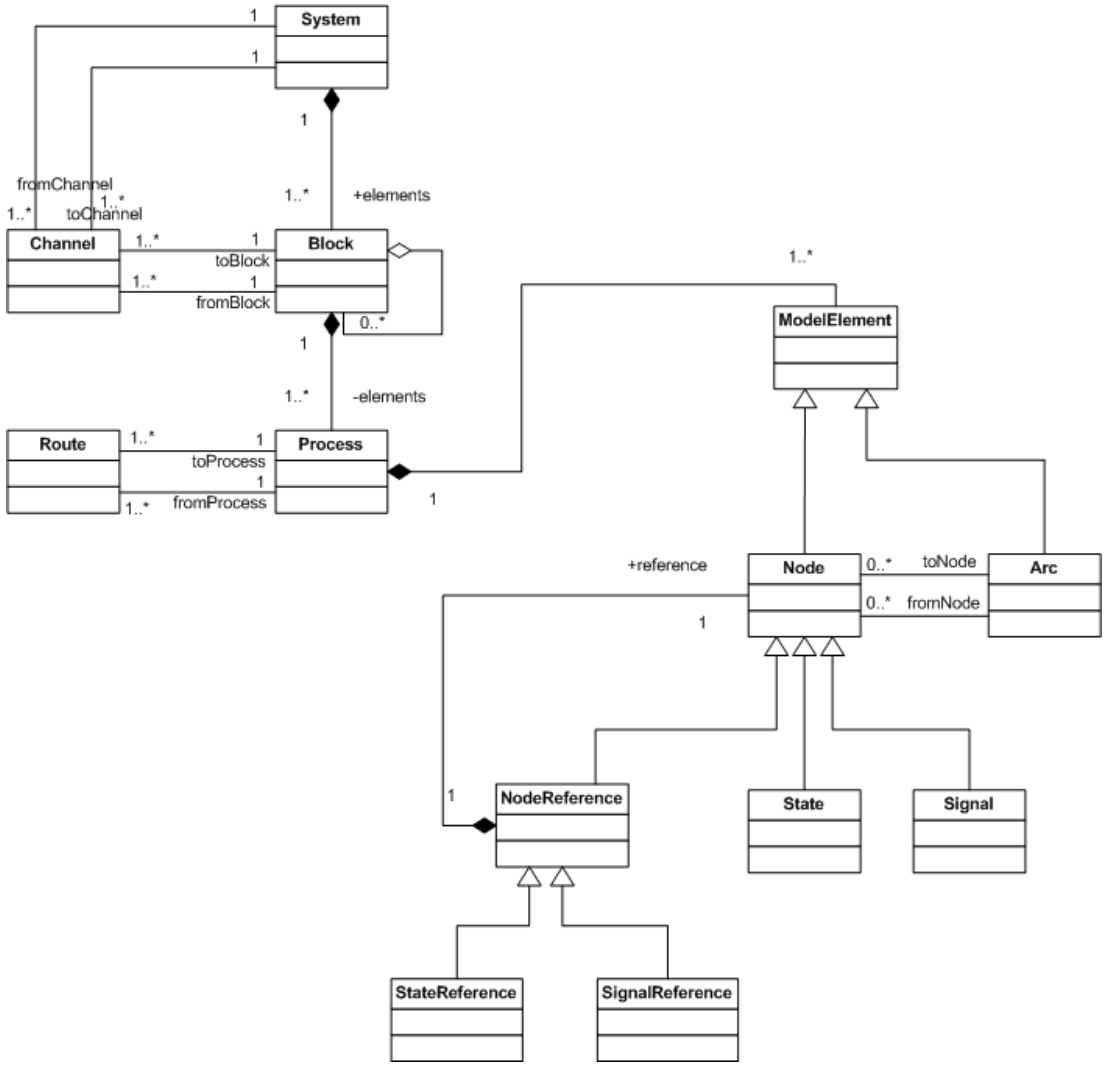

Fig. 9. SDL model ontology - UML hierarchy of core SDL concepts

Before we let the technology speak for itself, let us recall that the basic intention of this paper is to build the ontology for SDL model specification. By this we mean that 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! 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.

A. UML Ontology Representation

We start with the UML representation of the ontology and then we translate it to OWL language using Protégé Tool [15]. In order to use UML notation we follow the generally adopted convention:

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

Translating UML model to Protégé Ontology maps as follows [18];

- Each UML class is represented by one Protégé class of the same name, and with the same role (i.e., abstract or not abstract).
- These Protégé classes are arranged in an inheritance hierarchy as represented in the UML model (UML allows for multiple inheritance). Since Protégé does not support concepts such as ”interfaces”, all UML interfaces are handled as classes.
- Each attribute of a UML class is represented by an object property of a suitable type. The new object property

will have the name of the attribute, unless this name is already taken by a different object property (from another UML class) with a different type. In this case, the object property will be renamed to the format <attributeName> @ <className>, which ensures unique object property names.
- The new Protégé object properties get their multiplicity (allows multiple values or not) and the min and max cardinalities from the UML model.
- During UML export, primitive object properties (e.g., int, string and symbol) are converted to simple attributes that are attached to all UML classes where the object property is used. Non-primitive (instance) object properties are translated into UML associations, whereby inverse slots are used to create bidirectional associations.
- Metaclasses are recognized by means of the <<metaclass>> stereotype for classes.

Also, there are some known limitation not solved so far;

- Instances are not exported.
- UML associations of a higher order (e.g., ternary relationships) have not been tested yet.
- Overloaded object properties are not recognized.
- All classes from the UML model are derived from standard class, i.e. there is no support for other built-in metaclasses yet.

We have used the approach similar to [8] (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 a little complicated to has 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.

Our idea of the core SDL ontology is given by Figure 9.

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 10 and 11 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 12 and 13).

Our SDL ontology from Protégé tool is given in Figures 14 and 15.

All these translation steps are explained in more detail in the Table VII, but for full understanding of this table we suggest to read the next section. Strong SDL syntax and semantics

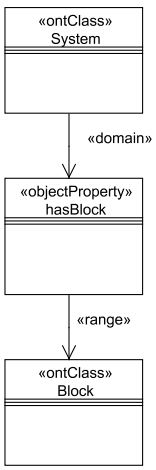

Fig. 10. System Specification for SDL ontology

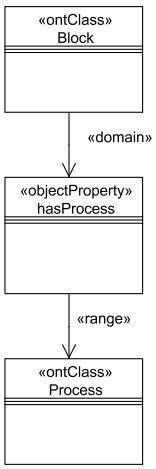

Fig. 11. Block Specification for SDL ontology

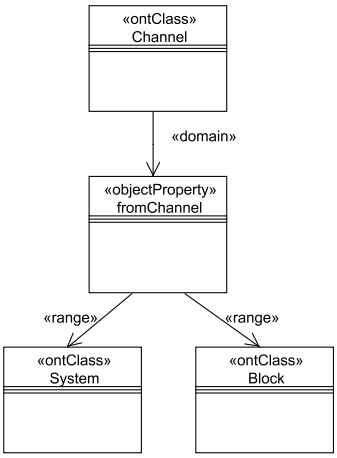

Fig. 12. Property $fromChannel$ Specification for SDL ontology

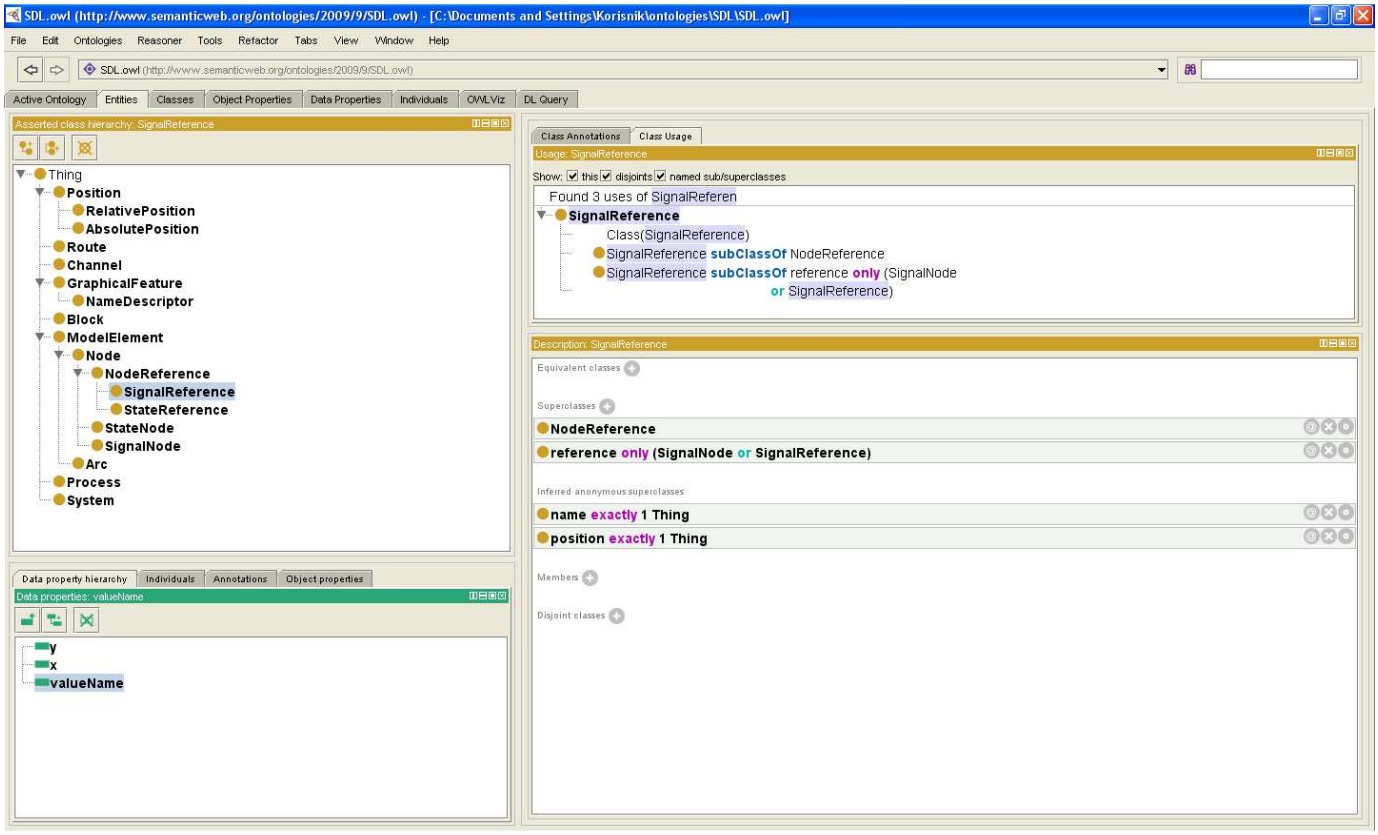

Fig. 14. SDL Ontology Hierarchy in Protégé

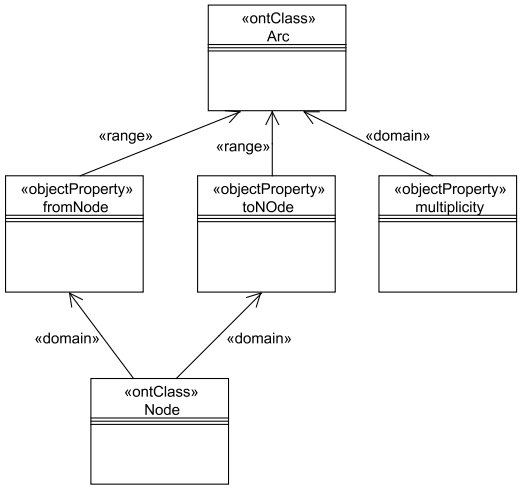

Fig. 13. Arc Specification for SDL ontology

provide us with a mechanism for writing unambiguous documents. Even more, SDL textual notation give us the idea of extending SDL notation for its XML support. Once translated to XML document, SDL file could be very easily used for different application and could be easily transported between various platforms. Its textual notation almost corresponds to XML one.

B. SDL Ontology for Sanitary Web Service

Once we have specified the SDL ontology with its core concepts, we easily provide the examples, i.e. how to use it for the system specification in terms of Semantic Web.

As both SDL and its OWL ontology are object-oriented languages, we search for the corresponding objects in 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.

More specifically, using Protégé it becomes natural to build the system from the specified ontology with a few mouse clicks in the Individuals tab of the tool.

For the system of Sanitary Web Service that we specified in previous chapters, we have the $SanitarySystem$ individual of class $System$ with $hasBlocks$ property with the range of three $Block$s, then $SanitaryAgency$ individual of class $Block$ with $hasProcess$ property with the range of one $Process$, $Bank$ individual of class $Block$ with $hasProcess$ property with the range of one $Process$, $Coop$ individual of class $Block$ with $hasProcess$ property with the range of two $Process$es. The same procedure is applied to the rest

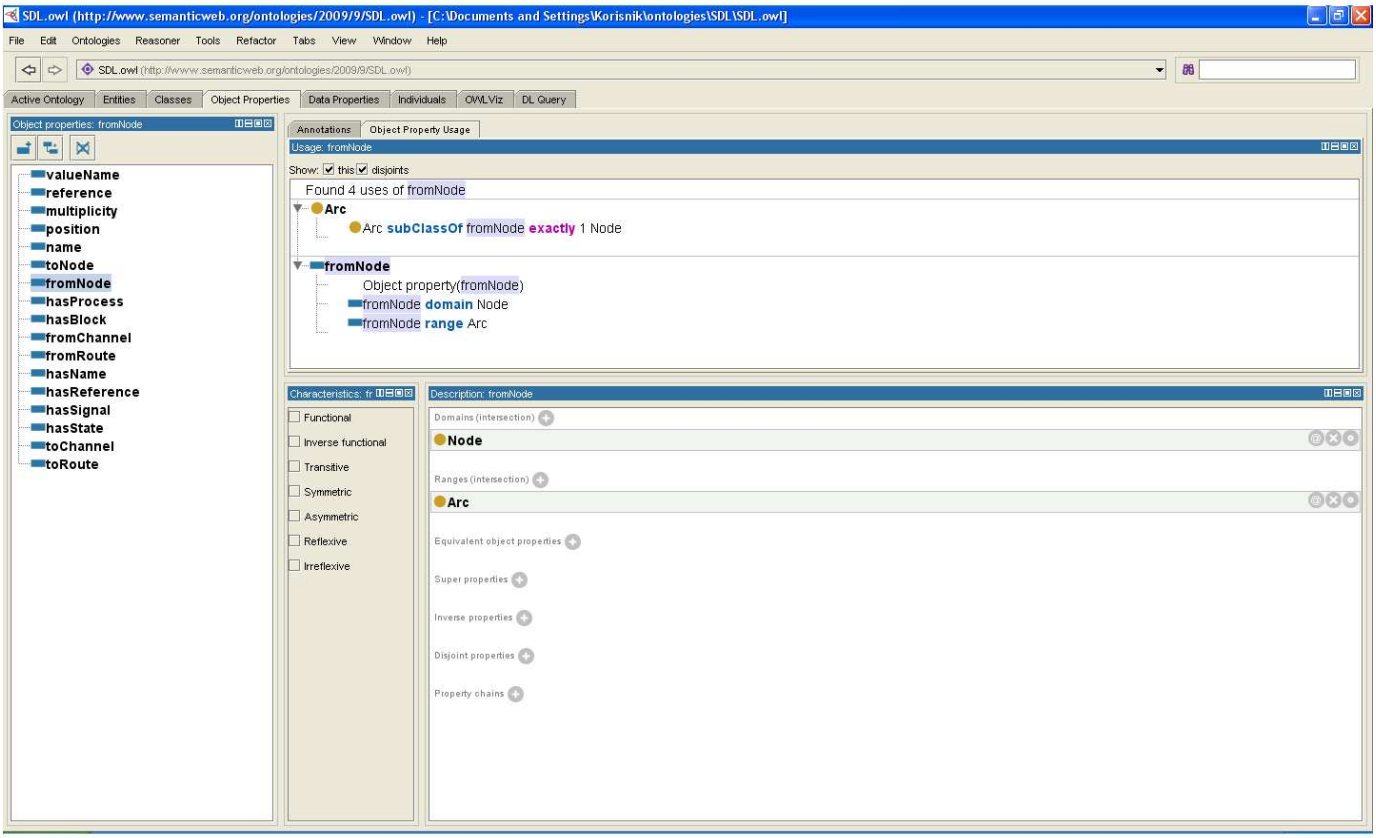

Fig. 15. Object Properties of SDL Ontology in Protégé

of the system. Each state must be specified as individual of $State$ class and each signal must be specified as individual of $Signal$ class, as well as their properties. For instance, we have $idleState$ and $waitState$ individuals for specification of Transport Cooperative states with their respective properties, then Req, $ProvT$, etc. as $Signal$ individuals for the same process. Also, all existing channels and routes must become specified with their properties.

For more details on this we suggest [15] to reader.

IV. 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 (doubled "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 the table VII we give an overview of the elements translation.

A. 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

```
<sdl:system name="SanitarySystem">
  <sdl:block name="SanitaryAgency">
    <sdl:process name="SanitaryAgency">
      <sdl:state name="Idle">
        <sdl:input>Request</input>
          <sdl:output>AskInfo</output>
            <sdl:nextstate>waitInfo</nextstate>
      </sdl:state>
      <sdl:state name="waitInfo">
        <sdl:input>ProvInfo</input>
          <sdl:output>Refusal</output>
            <sdl:nextstate>Idle</nextstate>
          <sdl:output>Acceptance</output>
            <sdl:output>Ref</output>
              <sdl:output>PaymentPublicFee</output>
                <sdl:nextstate>waitDone</nextstate>
      </sdl:state>
      <sdl:state>waitDone</nextstate>
        <sdl:input>Done</input>
          <sdl:nextstate>Idle</nextstate>
      </sdl:state>
    </sdl:process>
  </sdl:block>
  <sdl:block name="Coop">
    <sdl:process name="TransCoop">
      <sdl:state name="idle">
        <sdl:input>Req</input>
          <sdl:output>ProvT</output>
            <sdl:output>RecMoneyPossT</output>
              <sdl:nextstate>wait</nextstate>
      <sdl:state>
      <sdl:state name="wait">
        <sdl:input>PaymentT</input>
          <sdl:nextstate>idle</nextstate>
      <sdl:state>
    </sdl:process>
    <sdl:process name="MealCoop">
      <sdl:state name="idle">
        <sdl:input>Req</input>
          <sdl:output>ProvM</output>
            <sdl:output>RecMoneyPossM</output>
              <sdl:nextstate>wait</nextstate>
      <sdl:state>
      <sdl:state name="wait">
        <sdl:input>PaymentM</input>
          <sdl:nextstate>idle</nextstate>
      <sdl:state>
    </sdl:process>
  </sdl:block>
  <sdl:block name="Bank">
  </sdl:block>
</sdl:system>
```

TABLE V
SDL-ML CODE FOR BLOCKS AND PROCESSES SPECIFICATION OF SANITARY WEB SERVICE

other elements are nested into this one. It has the attribute "$name$".

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

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 "name" attribute.

Element $< sdl : state >$ defines the state element. This element contains $< input >$ and $< output >$ signal elements and the $< nextstate >$ element. It has the "name" attribute. Also, its elements are nested, i.e. "nextstate" is nested into "output", and "output" is nested into "input" element. We introduced it because of multiplicity of elements. The process can have two "output" elements and each one may refer to another "nextstate" or may be followed by another "output" element. So, we build hierarchy of these elements to preserve their order as it is not default feature of XML elements.

Element $< sdl : signal >$ defines the signal element. It has the "name" attribute.

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

B. SDL-ML Case Study: Sanitary Web Service

Here we put some of the .sdl XML file for the specification of Sanitary Web Service that we specified in previous chapters. We have divided it into two parts to make it easier to read. But the second part with channel and routes is to be put directly into the first part to make a whole. The first part in Table V specifies some of the system's blocks and processes, while the other one in Table VI focuses on communication between the blocks and processes.

V. CONCLUSION

In this paper we have given our contribution to the efforts of Semantic Web development in the sense of introduction of 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 process, as our starting point is the formalism for a telecommunications service specification. We have developed the ontology for SDL system and process specification. The ontology was developed using UML and OWL languages. UML has been used as a graphical notation for the ontology specification, which was then translated to Protégé tool to obtain OWL file. We have given the example of Web service specification to express our efforts.

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. We proposed the SDL-ML elements and also provided the example of Web service specification via SDL-ML.

Our future work naturally follows as the extending the core SDL ontology towards the time and modeling concepts. Such an extension will provide us with the fully equipment for Semantic Web reasoning of SDL models.

Another direction of future research is to make the process of SDL to OWL translation automated after bridging the gap between the inconsistencies of the two. XML technologies are only one way of doing this, using SDL-ML that we proposed. Exploring the other solutions could also contribute the future research.

```
<sdl:channel name="Ch-SA-Env">
  <sdl:from>env<sdl:from>
    <sdl:to>SanitaryAgency<sdl:to>
      <sdl:with>Request<sdl:with>
      <sdl:with>ProvInfo<sdl:with>
  <sdl:from>SanitaryAgency<sdl:from>
    <sdl:to>env<sdl:to>
      <sdl:with>AskInfo<sdl:with>
        <sdl:with>Acceptance<sdl:with>
          <sdl:with>Refusal<sdl:with>
< /sdl:channel>
<sdl:channel name="Ch-SA-Bank">
  <sdl:from>SanitaryAgency<sdl:from>
    <sdl:to>Bank<sdl:to>
      <sdl:with>PaymentPublicFee<sdl:with>
  <sdl:from>Bank<sdl:from>
    <sdl:to>SanitaryAgency<sdl:to>
      <sdl:with>Done<sdl:with>
< /sdl:channel>
<sdl:channel name="Ch-Bank-Coop">
  <sdl:from>Bank<sdl:from>
    <sdl:to>Coop<sdl:to>
      <sdl:with>PaymentT<sdl:with>
      <sdl:with>PaymentM<sdl:with>
  <sdl:from>Coop<sdl:from>
    <sdl:to>Bank<sdl:to>
      <sdl:with>RecMoneyPossT<sdl:with>
      <sdl:with>RecMoneyPossM<sdl:with>
< /sdl:channel>
<sdl:channel name="Ch-SA-Coop">
  <sdl:from>SanitaryAgency<sdl:from>
    <sdl:to>Coop<sdl:to>
      <sdl:with>Req<sdl:with>
< /sdl:channel>
<sdl:channel name="Ch-Coop-Env">
  <sdl:from>Coop<sdl:from>
    <sdl:to>Env<sdl:to>
      <sdl:with>provT<sdl:with>
      <sdl:with>provM<sdl:with>
< /sdl:channel>
<sdl:route>
<sdl:route name="Rt-TC-Env">
  <sdl:from>env<sdl:from>
    <sdl:to>TransCoop<sdl:to>
      <sdl:with>Req<sdl:with>
  <sdl:from>TransCoop<sdl:from>
    <sdl:to>env<sdl:to>
      <sdl:with>ProvT<sdl:with>
< /sdl:route>
<sdl:route>
<sdl:route name="Rt-TC-Bank">
  <sdl:from>Bank<sdl:from>
    <sdl:to>TransCoop<sdl:to>
      <sdl:with>PaymentT<sdl:with>
  <sdl:from>TransCoop<sdl:from>
    <sdl:to>Bank<sdl:to>
      <sdl:with>RecMonexPossT<sdl:with>
< /sdl:route>
```

TABLE VI
SDL-ML CODE FOR CHANNELS AND ROUTES SPECIFICATION OF SANITARY WEB SERVICE

REFERENCES

[1] J. Ellsberger, D. Hogrefe, A. Sarma, *SDL: Formal Object-Oriented Language for Communicating Systems*, Prentice Hall PTR, 1997.

[2] H. Belina, *Tutorial on SDL-88*, http://www.sdl-forum.org/sdl88tutorial/, 2007 SDL Forum Society, 1997.

[3] C. Walton, *Agency and the Semantic Web*, Oxford University Press, October 2006., ISBN 0199292485.

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

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

[6] G. Salan, L. Bordeaux, M. Schaerf, *Describing and Reasoning on Web Services using Process Algebra*, International Journal of Business Process Integration and Management 2006 - Vol. 1, No.2 pp. 116 - 128, 2006.

[7] M. Pistore, M. Roveri, P. Busetta *Requirements-Driven Verification of Web Services*, 1st International Workshop on Web Services and Formal Methods (WS-FM), 2004.

[8] D. Gašević, and V. Devedžić, *Petri net Ontology*, Knowledge-Based Systems, Elsevier, Vol. 19, Issue 4, Aug. 2006, pp. 220-234

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

[10] ITU-T Z.100, 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)*, Aug. 2002.

[11] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: *Reasoning About Knowledge* The MIT Press, Cambridge Massachusetts, London England, 2003.

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

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

[14] T. Berners-Lee, J. Hendler, O. Lassila, *The Semantic Web*, Scientific American Magazine, May 2001.

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

[16] SDL-RT, http://www.sdl-rt.org/standard/V2.2/html/SDL-RT.htm, Oct. 2009.

[17] IEC, http://www.iec.org/online/tutorials/sdl/index.asp, Oct. 2009.

[18] Protégé, http://protege.cim3.net/cgi-bin/wiki.pl?UMLBackendMapping, Oct. 2009.

[19] Cinderella SDL, http://www.cinderella.dk/, Oct. 2009.

Marina Bagić Babac works as a research assistant at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Telecommunications, where she took her degrees B.Sc, M.Sc. and Ph.D. in 2001, 2004 and 2009, respectively. Her specific field of research are formal methods for specification and verification of telecommunications systems. She is a member of IEEE Communications Society.

Marijan Kunštić works as a Professor at the University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Telecommunications, where he took his degrees B.Sc, M.Sc. and Ph.D. in 1966, 1971 and 1980, respectively. His specific field of research are formalisms and network management of telecommunications systems. He is a member of IEEE Communications Society.

SDL-ML ELEMENTS	OWL ELEMENTS
System <sdl : system name="name"> </sdl:system>	Class **System** $hasName \in [System, Literal]$ $hasSignal \in [System, Signal]$ $hasBlock \in [System, Block]$ $hasChannel \in [System, Channel]$
Block <sdl : block name="name"> </sdl:block>	Class **Block** $hasName \in [Block, Literal]$ $hasSignal \in [Block, Signal]$ $hasBlock \in [Pro, Block]$ $hasRoute \in [Channel, Route]$
Process <sdl : process name="name"> </sdl:process>	Class **Process** $hasName \in [Process, Literal]$ $hasSignal \in [Process, Signal]$ $hasState \in [Pro, Process]$
Channel <sdl : channel name="name"> <from> </from> <to> </to> <with> </with> </sdl:channel>	Class **Channel** $hasName \in [Channel, Literal]$ $fromChannel \in [Channel, Block]$ $toChannel \in [Block, Channel]$ $withSignal \in [Channel, Signal]$
Route <sdl : route name="name"> <from> </from> <to> </to> <with> </with> </sdl:route>	Class **Route** $hasName \in [Route, Literal]$ $fromRoute \in [Route, Process]$ $toRoute \in [Process, Route]$ $withSignal \in [Route, Signal]$
Signal <sdl : signal name="name" type="incoming/outgoing"> <sdl:presignal> </sdl:presignal> <sdl:nextsignal> </sdl:nextsignal> <sdl:prestate> </sdl:prestate> <sdl:nextstate> </sdl:nextstate> </sdl : signal>	Class **Signal** $hasName \in [Signal, Literal]$ $previousSignal \in [Signal, Signal]$ $nextSignal \in [Signal, Signal]$ $previousState \in [Signal, Signal]$ $nextState \in [Signal, Signal]$
State <sdl:state name="name"> <sdl:input> </sdl:input> <sdl:output> </sdl:output> <sdl:nextstate> </sdl:nextstate> </sdl:state>	Class **State** $hasName \in [Signal, Literal]$ $hasInputSignal \in [State, Signal]$ $hasOutputSignal \in [State, Signal]$ $hasNextState \in [State, State]$

TABLE VII
MAPPING SDL-ML TO OWL