UDC 004.41
Original Scientific Paper

Describing Layered Communication Architecture in SDL Markup Language

Marina Bagić Babac *marina.bagic@fer.hr*
Unversity of Zagreb
Faculty of Electrical Engineering and Computing

Marijan Kunštić *marijan.kunstic@fer.hr*
Unversity of Zagreb
Faculty of Electrical Engineering and Computing

Dragan Jevtić *dragan.jevtic@fer.hr*
Unversity of Zagreb
Faculty of Electrical Engineering and Computing

Abstract

Using Specification and Description Language (SDL) as a formal language for specification of requirements for the complex, real-time and distributed systems involving many concurrent activities, we have come to the idea of making the language independent of platforms and operating systems which may use it. Shortly, we have developed markup version of the SDL language, and named it SDL Markup Language. It is an XML-based version of the SDL-PR (Phrase Representation), an SDL textual notation. We use the language to specify the complex communications protocols, which are used in wide range of layered architectures. We provide the specification from the INRES protocol in SDL-ML.

Keywords: SDL, SDL Markup Language, XML, Finite State Machine

1. 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 [3], [4]. One of the most popular, UML (Unified Modeling Language) has the wide application range, however it remains informal in some of its aspects, which might not be satisfying for some special purposes. We have chosen Specification and Description Language (SDL), a standardized language by ITU-T. Its purpose is the specification and description of complex, event-driven, real-time and interactive applications involving many concurrent activities that communicate using discrete signals.

Moreover, in this paper we present SDL Markup Language, our idea of XML based SDL language. We think of it as an interchange SDL format that is independent of specific tools and platforms. Moreover, the interchange format needs to support extensions of SDL. The beauty of the approach is in its simplicity, reusability and the wide range of usage. Any application that is using XML for data interchange can use the SDL-ML specification.

The related work is mostly the literature on SDL [1], [2], [5], [8], [10], finite state machines communication [6], [7], and layered communication architecture examples [12]. Since no such language has been developed so far, we can only outline that our inspiration comes from the similar trials like PNML (Petri Net Markup Language) that was developed for the same purpose for Petri net platform independent interchange format [13].

After the introduction, the paper shortly explains major ideas form SDL architecture and its elements. Then, using SDL machine processable format, it introduces the ideas of SDL-ML language. In the end, we provide the example of INRES protocol from the literature, an example of layered communication architecture and we model the Initiator part in SDL and SDL-ML.

2. Specification and Description Language

SDL language supports two equivalent notations, graphical (SDL-GR) and textual (SDL-PR). The textual notation SDL-PR (Phrase Representation) 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 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. 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 another 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 process is a finite state machine (FSM) based task and has an implicit message queue to receive messages. It is possible to have several instances of the same process running independently.

2.1 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.

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 [8].

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;

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 fulfills its functionality is described with processes. A lowest level block can be composed

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

Table 1: SDL: System Textual Notation

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 2: SDL: Block Textual Notation

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 3.

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

Table 3: 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 4.

2.2 SDL Communication

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

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

Table 4: SDL: State Textual Notation

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.

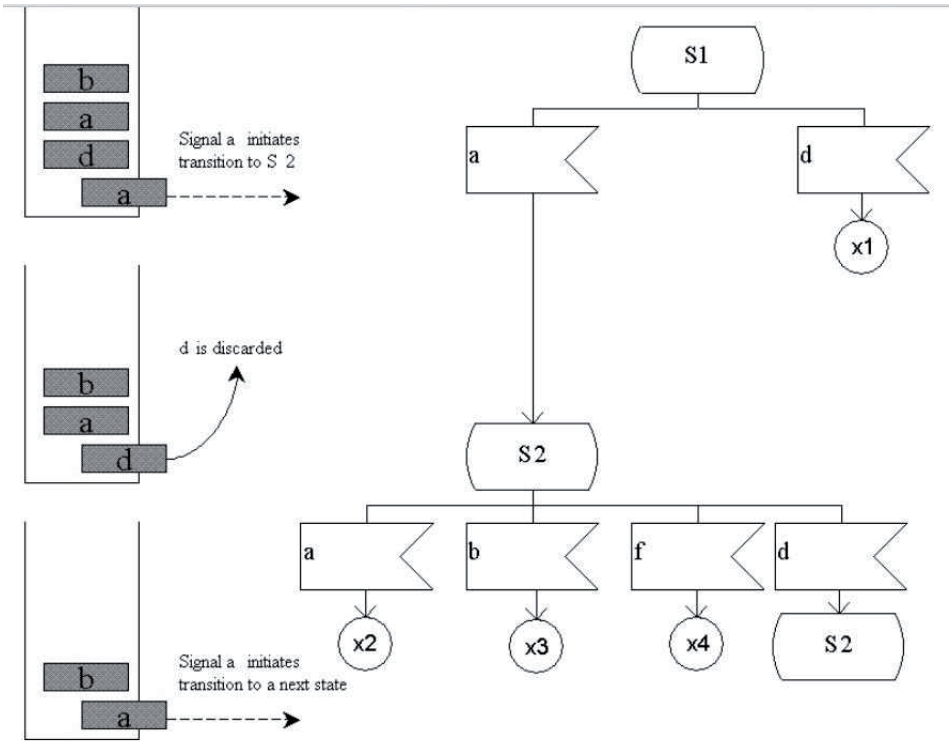

Figure 1: SDL Input Queue for the Signals

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 1 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 1 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 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 Markup Language

In this paper we outline this idea of SDL Markup Language (SDL-ML). SDL-ML is meant to be an interchange format that is independent of specific tools and platforms. Moreover, the interchange format needs to support extensions of SDL. 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.

3.1 SDL-ML Elements

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" which stands for identification.

Element $< sdl : block >$ defines the block element which contains processes. This element is the parent element of the process elements. It has the "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 with the "id" attribute for unique identification. If it is set to "Any" it denotes a special state, e.g. any state ("*" in graphical notation). Its child elements are $< sdl : pre >$ and $< sdl : post >$ to define the related elements of the state, the preceding and subsequent ones.

Element $< sdl : signal >$ defines the signal element with the "id" attribute for unique identification. It has the same child elements as the $< sdl : state >$ element. A child element is also $< sdl : data >$ with attribute of "type" denoting data type.

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.

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.

Elements $< sdl : pre >$ and $< sdl : post >$ have an attribute "type" to denote what kind of a symbols the parent element is connected to (or related to). These are, for instance, "signal", "state", etc.

3.2 SDL-ML Application

The motivation for introduction of SDL Markup Language is to obtain an XML-based interchange format for SDL documents. Due to various tools there is a need for unified modeling of SDL systems. SDL-ML applications are software programs that process and manipulate data using XML technologies. Further more, once we have an XML-based SDL system description, we have a variety of different target languages to translate our specification into. For instance, using XSLT we transform our SDL-ML specification into any XML or non-XML language, if needed, which enriches our opportunities for facing another applications or user interfaces. There are already hundreds of serious applications of XML languages. SDL strength against another modeling languages and techniques is its strong definition and formality. What is out of SDL scope of modeling cannot be a subject to implementation with programming languages. SDL processes are therefore programmable and verified by the modeling itself. Based on SDL-ML specification a programmer is able to implement a system in object-oriented languages (e.g. Java, C++, etc.) easily. And the field of application is wide; concurrent and distributed real time systems and applications where communication via message exchange is needed. We provide a detailed insight into the communication protocol named INRES in the next section.

4. SDL Example of INRES Protocol

The layering in the communication architecture can be compared with abstraction levels following the principle of information hiding: together with the lower layers, every layer renders a service to the next higher layer. The means by which this service is rendered is hidden from the higher layer. The higher layer needs no information about implementation details of the lower layers, but can rely on a defined service [11].

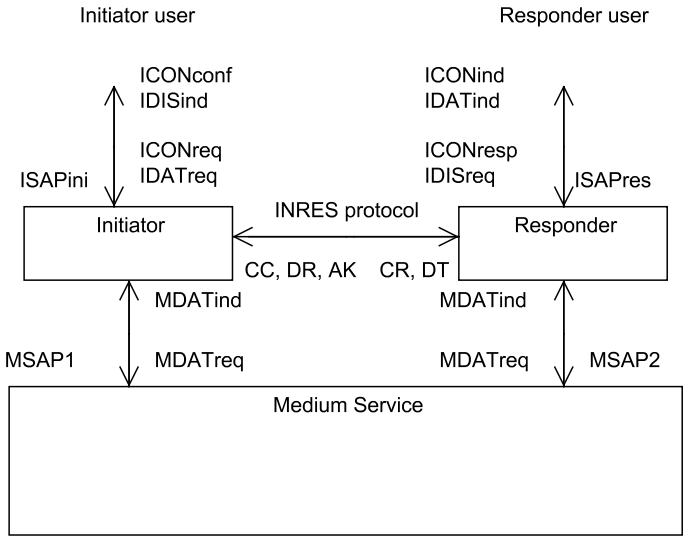

Figure 2: The overall architecture of INRES Protocol

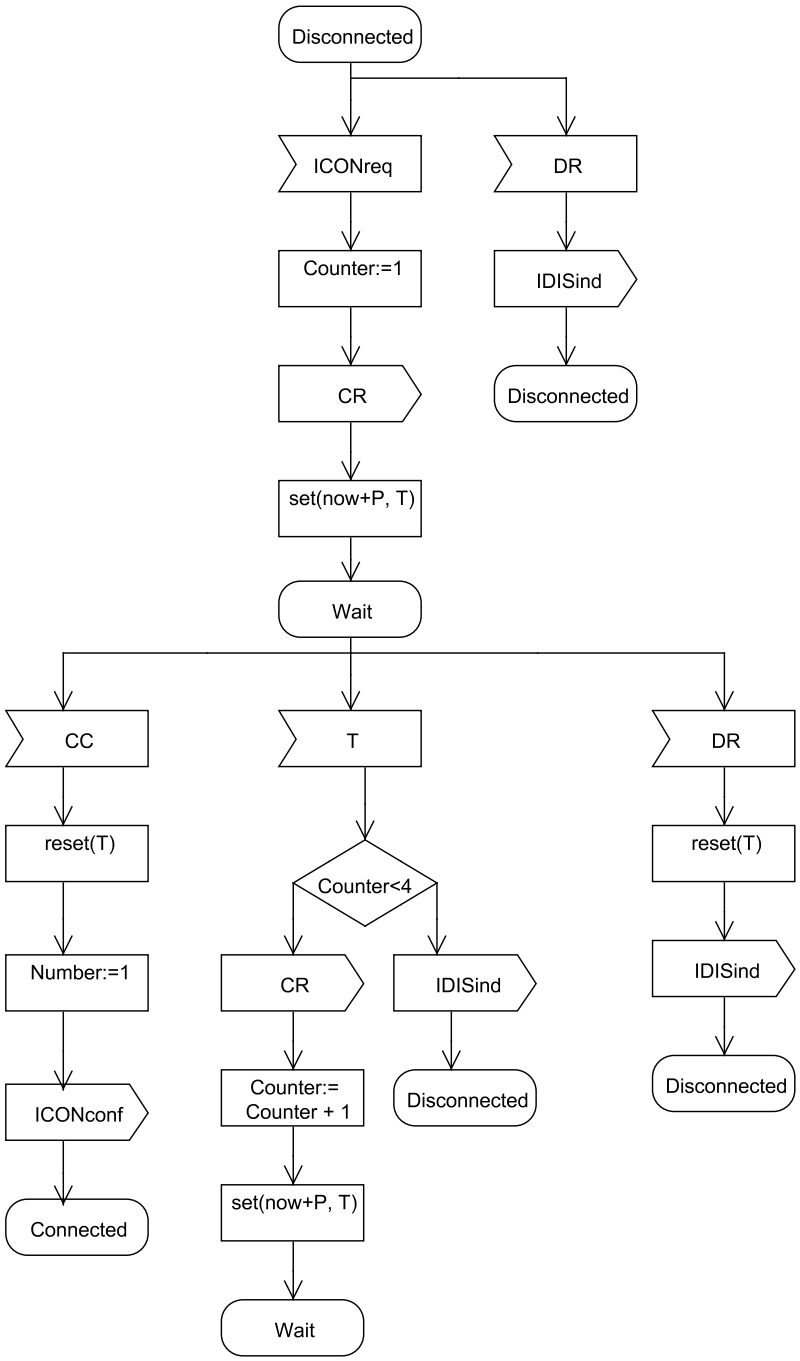

Figure 3: The Initiator in INRES Protocol (1)

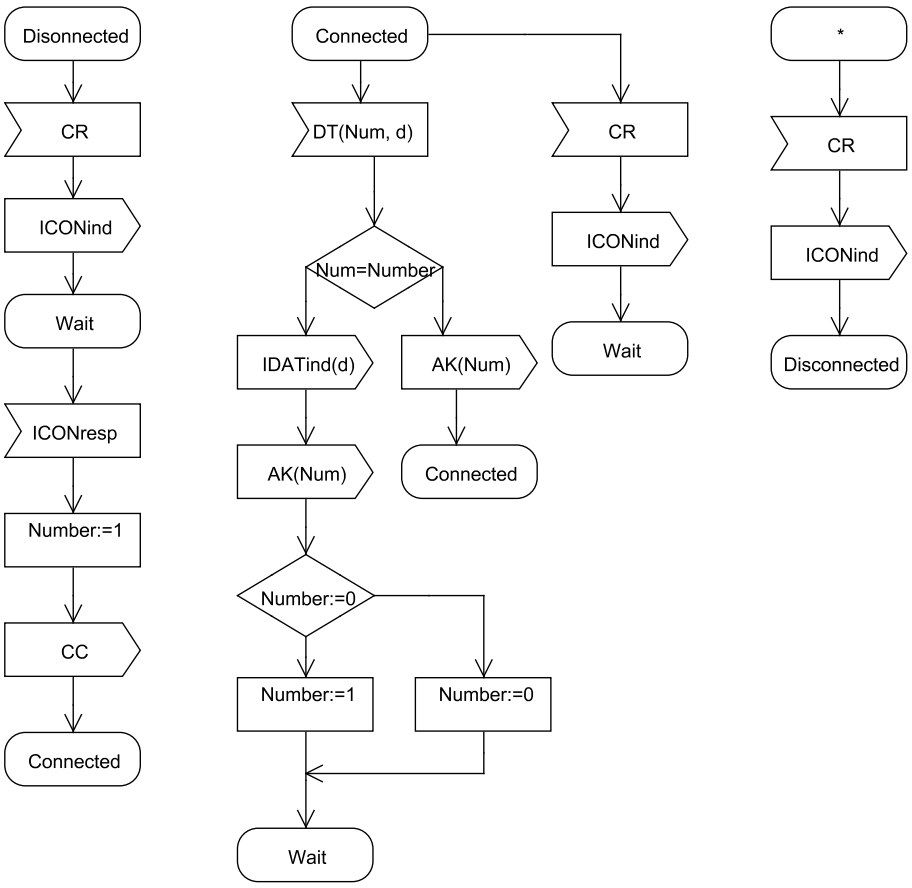

Figure 4: The Initiator in INRES Protocol (2)

In this section we give the example of the SDL (SDL-ML) system specification. We have chosen INRES protocol, and we provide SDL and SDL-ML specification for the Initiator entity 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 [12].

INRES Protocol renders a connection-oriented service to its users with the aid of Medium service (Figure 2), which can be used for unreliable transmission of data units. The figure expresses the interchange of messages which are explained by its meaning (not the structure) as we are interested in abstract modeling neglecting the details of implementation of the protocol. We have three blocks in the figure communicating via message passing and each message carries the potion of relevant information to the protocol interchange. Two of the blocks represent the communicating subjects; initiator and the responder, and the third one is the service of their use.

Before describing the protocol in SDL vocabulary, let us first explain details of messages to be exchanged between the Initiator and Responder. They are used below in the specification of INRES Initiator states and signals (Figures 3 and 4). Both of the communication entities are represented as a single SDL process, and each is assumed to be in an initial stated before performing an action. An action is usually the message receipt or message sending. An Initiator is assumed to be in "Disconnected" state at the system startup and three of the incoming messages are expected to trigger the Initiator to change its state. These are DR, ICONreq and CR, which can also "come" when the Initiator enters any of its states.

The following service primitives are used for the communication between user and provider [1]. Their names in SDL graphical symbols (shown in figures and tables below) denote their purpose (input or output signals, states, etc.):

- ICONreq, request of a connection by Initiator-user
- ICONind, indication of a connection by the provider
- 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 (Initiator and Responder) communicate by exchange of the protocol data units (PDUs) with respective service primitives (SPs): CR (connection establishment - SPs: ICONreq, ICONind), CC (connection confirmation - SPs: ICONresp, ICONconf), DT (data transfer - SPs: IDATreq, IDATind), AK (acknowledgment) and DR (disconnection - SPs: IDISreq, IDISind).

The communication takes place in three distinct phases: the connection establishment phase, the data transmission phase and the disconnection phase.

A **connection establishment** is initiated by the Initiator with an ICONreq and it sends a CR to the entity Responder. Responder answers with CC or DR. In case of CC, Initiator issues an ICONconf to its user and the data phase can be entered. If Initiator receives nothing at all within 5 seconds, CR is transmitted again. If, after four attempts, still nothing is received by Initiator, it enters the disconnection phase. If Responder receives a CR from Initiator, the Responder gets an ICONind. The user can respond with ICONresp or IDISreq. ICONresp indicates its willingness to accept the connection, Responder thereafter sends a CC to Initiator and the data transmission phase is entered. Upon receipt of an IDISreq, Responder enters the disconnection phase [1].

Data transmission phase. If the Initiator-user of the entity issues an IDATreq, the Initiator sends a DT to the Responder and is then ready to receive another IDATreq from the user. After having sent a DT to Responder, Initiator waits for 5 seconds for a respective acknowledgment AK. Then the DT is sent again. After four unsuccessful transmissions, Initiator enters the disconnection phase. If Responder receives a CR, it enters the connection establishment phase, and, on receipt of an IDISreq, it enters the disconnection phase.

Disconnection phase An IDISreq from the Responder-user results in the sending of a DR by the Responder. Subsequently, Responder can receive another connection establishment attempt CR from Initiator. At the Initiator, the DR results in an IDISind sent by the Initiator to its user. An IDISind is also sent to the user after DT or CR has been sent unsuccessfully to the Responder. It is now possible to establish a new connection.

As the protocol has been explained, we refer to Figures 3 and 4, where the protocol is interpreted graphically, in SDL-GR version. Both figures represent the set of Initiators states and input and output signals. We have the initial state, which is called "Disconnected", and is expecting one of the three signals: DR, ICONreq and CR. If DR arrivals first, Initiator responds by sending IDISind signal and remains in the same initial state "Disconnected". If ICONreq comes first, Initiator sets Counter to one and sends CR signal, then starts the timer and waits for the response (state "Wait"). While waiting, Initiator can either receive the timer expiration, or CC or DR signals. If CC comes first, it resets the timer, sets Number to one, send ICONref and finally connects (state "Connected"). If DR comes, then Initiator resets the timer, sends IDISind and returns to the initial state. Else, if timer expires first, it resets it if no four timers expired yet, or send IDISind and returns to the initial state (Figure 3).

If in the initial state, Initiator receives CR, it sends ICONind and waits for ICONresp (state "Wait"). When it comes, it sets Number to one, sends CC and connects. When Initiator is connected (state "Connected"), it can either receive signal CR or DT(Num, d). If it receives CR first, it sends ICONind and waits (or goes to the initial state if receives CR in any of other states). Else, if DT(Num, d) comes, it evaluates Num and sends either AK(Num) and remains in the same states, or sends IDATind(d), AK(Num), evaluates Number and waits (Figure 4).

Here we put some of the .sdl XML file for the specification of Initiator in INRES protocol. The first part in Tables 5, 6 and 7 specifies the Initiator block and process, while the other one in Table 7 focuses on communication channels between the blocks in INRES protocol. Each SDL element is represented with its corresponding .sdl XML element.

Table 5 specifies the system which is given the name "INRES", the block with id "Initiator" and also the process with the same id. These are not in the name collision since they use the different element types (the block and the process are two different elements). Here the same message exchange from the Figures 3 and 4 is repeated, but in an XML

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

Table 5: SDL-ML Code for Initiator Specification in INRES Protocol (1)

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

Table 6: SDL-ML Code for Initiator Specification in INRES Protocol (2)

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

Table 7: SDL-ML Code for Initiator Specification in INRES Protocol (3)

based version, using elements from the previous section. Each signal is now represented as element node in SDL-ML and each element is given its predecessors and adherents. Let us take an example of ICONreq signal. Its attribute is specified as "incoming" denoting the direction of the signal (coming into the process, not outgoing). Its predecessor is state symbol "Disconnected" and its adherent is task symbol "Counter:=1".

Table 6 continues the specification. Among others, it specifies the timer with signal id="T" and type="timer". Its predecessor is the waiting state ("Wait"), and its adherent is decision symbol verifying if Counter<4. It also specifies task of resetting the timer with id="resetT" and expression "reset(T)" denoting the SDL task for timer reset. Its predecessor is CC signal, and its adherent is task symbol of "Numer:=1" task.

Table 7 finishes the Initiator's process SDL-ML specification. It specifies the "Disconnected" state with its predecessor IDISind signal. Then, it specifies the decision with id="counter" questioning if "Counter<4" and sending signals according to its answers, either CR or IDISind. It also performs the task "counter" with its expression "Counter := Counter+1" with predecessor outgoing signal CR and the adherent task, setting the timer "set(now+P,T)"

```
<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 8: SDL-ML Code for Channels Specification of INRES Protocol

Finally, Table 8 specifies the communication channels and routes between the INRES blocks and processes. Channel "Ch-Initiator-Responder" is bidirectional and we specify the signals flow in both directions. From Initiator to Responder we have the signals CR and DT, and signals CC, CR and DK in the opposite direction. Channel "Ch-Initiator-MediumService" is also bidirectional and specifies signals MDATreq i MSAP1 from Initiator to MediumService, and MDATind in the opposite direction. Channel "Ch-Initiator-Env" is the one connecting the system to its environment.

5. Conclusion

The layering in the communication architecture can be compared with abstraction levels following the principle of information hiding: together with the lower layers, every layer renders a service to the next higher layer. The means by which this service is rendered is hidden from the higher layer. The higher layer needs no information about implementation details of the lower layers, but can rely on a defined service.

In this paper we have chosen Specification and Description Language, a ITU-T standard Z.100 for specifying and describing telecommunications processes from layered architectures, as our starting point is the process of formalizing a telecommunications service specification. We have developed elements for SDL Markup Language based on SDL-PR (Phrase Representation) version of the language and XML syntax. SDL-ML applications are software programs that process and manipulate data using XML technologies. Further more, once we have an XML-based SDL system description, we have a variety of different target languages to translate our specification into.

The approach is illustrated by the example of INRES Protocol, a layered communication protocol for reliable, connection-oriented data transfer service between two users. The future work is to extend the basic SDL-ML concepts and to introduce more details in the existing concepts of tasks, decisions and timer control at the level of process modeling. At the system level, data parameters of the signals are to be introduced with more details in data type specification . Another direction is to explore the spectrum of different application areas for formal system specification based on SDL.

Acknowledgments

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.

References

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

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

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

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

[5] 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.

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

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

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

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

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

[11] Parnas, D. On the criteria to be used to decompose systems into modules, *Comm. ACM*, vol. 15, 1972.

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

[13] Weber, M.; Kindler, E. The Petri Net Markup Language, Vol. *Petri Net Technology for Communication Based Systems*, LNCS series "Advances in Petri Nets", Apr. 2002.