J. Software Engineering & Applications, **2008, 1: 13-19**
Published Online December 2008 in SciRes (www.SciRP.org/journal/jsea)

Scientific Research Publishing

Designing and Verifying Communication Protocols Using Model Driven Architecture and Spin Model Checker

Prabhu Shankar Kaliappan, Hartmut Koenig

Chair Computer Networks and Communication Systems, Brandenburg Technical University, Cottbus, Germany
Email: {psk, koenig}@informatik.tu-cottbus.de

Received November 21st, 2008; revised November 26th, 2008; accepted November 29th, 2008.

ABSTRACT

*The need of communication protocols in today's environment increases as much as the network explores. Many new kinds of protocols, e.g. for information sharing, security, etc., are being developed day-to-day which often leads to rapid, premature developments. Many protocols have not scaled to satisfy important properties like deadlock and livelock freedom, since MDA focuses on the rapid development rather than on the quality of the developed models. In order to fix the above, we introduce a 2-Phase strategy based on the UML state machine and sequence diagram. The state machine is converted into PROMELA code as a protocol model and its properties are derived from the sequence diagram as Linear Temporal Logic (*LTL*) through automation. The PROMELA code is interpreted through the SPIN model checker, which helps to simulate the behavior of protocol. Later the automated LTL properties are supplemented to the SPIN for the verification of protocol properties. The results are compared with the developed UML model and SPIN simulated model. Our test results impress the designer to verify the expected results with the system design and to identify the errors which are unnoticed during the design phase.*

Keywords**:** *UML Modeling, Communication Protocols, Protocol Verification, SPIN Tool*

1. Introduction

Due to the huge complexity of modern software systems, it is required to specify precisely what a software component should do and how it should behave [1]. If the final implementation deviates from the expected behavior, then the use of the developed component may fail. This also applies for the development of communicating protocols as they are merely implemented in the software. Currently, most of the protocols are developed through the natural, informal language because it is easy to understand. Special languages known as formal description Techniques (FDTs) have been developed for an unambiguous specification of the software. FDTs distinguish from programming languages by having a formal semantics. Programming languages, such as Java or C++, have only a formally defined syntax. In order to back-up such languages, the *Unified Modeling Language* 2 (UML 2) [2] is a collection of semi-formal standard notations and concepts for modeling the software systems at different stages and views during their development.

The development process is supported by the *Model Driven Architecture* (MDA) concept [3], which is initiated from the Object Management Group (OMG). The UML semantics is described in natural English language which includes semantic variation points that leave some semantics issues deliberately open. This desirable property represents a drawback from the verification point of view. To cope with the above problem we propose a 2-phase strategy (see Figure 1). In the first phase, we model the behavior view by UML state charts and activity diagrams. Next they are translated as a combination of state charts with the semantics of activity diagrams into PROMELA (*PROcess MEta LAnguage*) [4]. In the second phase, we design the communication view using UML sequence and timing diagrams. The model properties are translated into a temporal logic and imported together with the PROMELA code into the model checker SPIN (*Simple Promela INterpreter*) [5] for verification. Furthermore, we illustrate the importance of UML in developing and SPIN in verifying the communication protocols through our approach.

The paper is organized as follows. In section 2 we give a short overview of related work. Section 3 illustrates the MDA approach applied to the development of communication protocols. Section 4 presents our 2-phase design and verification strategy using a case study as example. Some final remarks and an outlook on future work which concludes the paper.

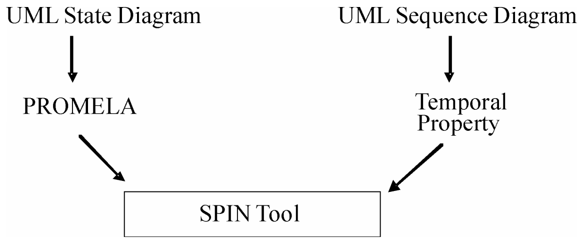


Figure 1. 2-phase strategy

2. Related Works

An approach for the formal verification of UML diagrams, such as class, state and communication diagrams, is presented in [6]. The approach applies an object oriented language, called the *Maude*, for verifying the static and dynamic features of object oriented specifications. Maude is based on rewrite logic. According to [7], there is no proof of correctness (due to the missing UML semantics), when a UML model is translated into PROMELA. To overcome this drawback the static and dynamic verification is carried out individually and integrated into the final validation stage. The verification of the UML class and activity diagrams is illustrated for a simple protocol in [8,9]. The activity diagrams are converted into an FSM (based on behaviors). Thereafter the FSM is converted into PROMELA through an intermediate language. Most of the above specified approaches illustrate how to verify the UML state diagrams. The open issue is how to specify and verify communication protocol properties in detail. According to our concern, the protocols can be efficiently developed if they are verified simultaneously while modeling. In order to fulfill the concern, we specify and verify the protocol properties in the Platform Independent Model (PIM) and the Platform Specific Model (PSM) independently.

3. Architecture Template for Communication Protocols

3.1 Model Driven Architecture

Model driven architecture is an approach to software development based on the modeling and automated mapping of models. MDA has divided its components into two important parts, namely PIM and PSM, which are discussed in detail further as basis.

The *Platform Independent Model* is a model with a high level of abstraction that is independent of any implementation technology [10]. A modeling language capable of generating all the required artifacts such as the Unified Modeling Language is required at this level. According to [3], the PIM provides two basic advantages. First, the person responsible for defining the functionality do not have to take any platform details into the consideration while modeling, which gives the designer a freedom to concentrate and focus only on the logical rule. Second, since the functionality is pure from any implementation details, it is easier to produce implementations on different platforms. The PIM is stored in the Meta Object Facility (MOF) and serves as the input to the mapping step which will produce a *Platform Specific Model*. The PSM's can be described in one of two ways: 1) using UML diagrams (class, sequence, activity etc.) or 2) using interface definitions in a concrete implementation technology (IDL, XML, Java etc), but in both cases the behavior and constraints are specified using a formal notation (UML diagrams) or an informal notation (natural language). Automated tools will be used to map the platform independent models onto the specific platforms. The final step takes PSM as an input to produce the implementation for a particular platform using a transformation tool.

3.2 Communication Protocols

A communication is carried out between a sender and a receiver over a physical medium using an authorized service provider. The service is provided by means of communicating entities. These entities are active objects exchanging messages with their environment. The service users interact with the entities by exchanging service primitives through *service access points* (SAPs). Each SAP is uniquely mapped to an entity which handles the primitives and maps them on *protocol primitives* or protocol data units (PDUs), respectively, that are send to the peer entity. The exchange of the protocol primitives is based on rules which are specified by means of a *communication protocol.* A communication protocol describes the interacting behavior of the entities by specifying the timely sequence of the protocol primitives exchanged. Furthermore, the format (syntax) and the meaning (semantics) of the messages are defined.

3.3 MDA and Communication Protocols

The following template for the design of communication protocols consists of three components, namely: the model designer, the model mapper, and the system generator (see Figure 2). These are illustrated with respect to PIM, PSM, and the code generator in the following.

1) Model Designer

The model designer has the task to model the proposed system based on the requirement specification. The modeling is carried out by means of the UML, the *Meta Object Facility* (MOF) for the data repository, and the *Object Constraint Language* (OCL) for the external semantics. The hardware and software may be modeled together or separately. Further on these models are combined by the model integrator (*integrated model*) with the help of external semantics (supplied through OCL), which can be introduced automatically or manually. The advantage of designing hardware and software models independently is that both of them are not considered about the dependency. This gives the developer the freedom to focus on system design rather than on programming details. When considered to the protocol development, the service layer and protocol layer are independently developed in this phase.

2) Model Mapper

The *model mapper* maps the PIM to PSM by means of an appropriate domain specifier. It consists of three different components: the *Domain Specifier* for specifying the target domain, *Transformation Rules*, i.e. a modified Query View Transformation (QVT) [11] is a standard set of rules to map the UML profile to the particular domain, and (preferably) *UML profiles* for the specification of appropriate models (say protocols). The possible input of the model mapper is UML and the output will be of XML Metadata Interchange (XMI). The transformation process is carried out by an appropriate transformative algorithm which reads the required model (UML profile for communication systems) and applies the QVT rules. The possible outcome of the model mapper is the UML profile based specification models. The transformation method is not strict with the communication system profiles, based on the requirement the profile can be chosen from the repository.

3) Model Checker and Model Verifier

The model checker is used to validate the structural behaviors of the developed models. The semantics of PIM are not much validated in this phase because the PIM illustrates only the logical solution to the particular problem. Hence, the structural behaviors are independently verified and combined by the integrated model. *The model verifier* checks the logic after model mapping. In completion of the model mapper phase, the model verifier is introduced to check the static and dynamic behaviors of the mapped model. The verification results from the PIM and PSM are matched by comparing both of the results. Here, the SPIN tool is used along with formal verification techniques to check the behavior of PIM and PSM.

4) System Generator

Finally the code generation is carried after a successful mapping of the model to a particular platform. The target code, such as C++, Java, .Net or SystemC, can be generated by the development tool including the appropriate library files and plug-ins. With help of XMI, which is the (preferable) output code from the previous phase, the code is generated automatically. The generated code is validated thereafter by testing.

By addressing the advantages in the above template, we can consider the top down and bottom up development as

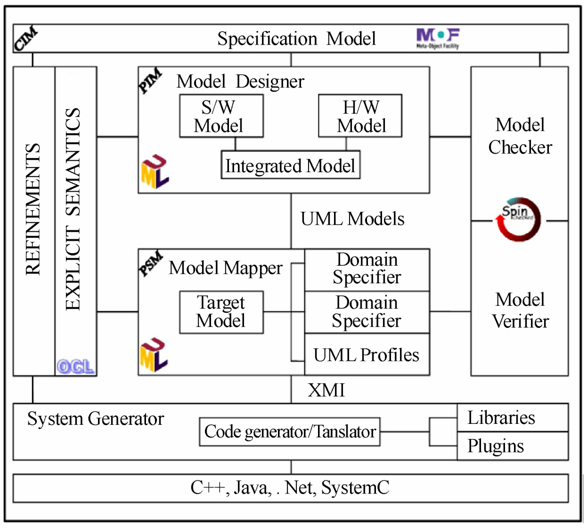


Figure 2. Template for protocol development

Top down

- Development is from the scratch and to the target code.
- Step by step process, which can be easily debugged or traced.
- Deviation / Refinement are possible at any cost of time.

Bottom up

- Development is from the code and to the specification model.
- Due to the generalized conversion of the XMI, any tool is capable for the conversion of platform independent models.

By the above, the complexity and the development code is systematically reduced with the proposed template.

4. Design and Verification of Communication Protocols

Communication protocols can be distinguished in two different viewpoints: the behavior and the communication oriented one. They can be matched with the UML models as illustrated in the Table 1. The further discussion is based on the above template for protocol development, i.e. we illustrate how the protocol is designed and verified through this template.

4.1 Model Designer

To illustrate the work flow of our method, we use an example case study of the *eXample Data Transfer* (XDT) protocol [12] which is being used as teaching protocol. XDT works on a distributed environment to transfer large files over an unreliable media using the *go back N* principle. The XDT protocol description consists of a service specification and a protocol specification which both include a data format specification. The connection establishment uses a two-way handshake and assumes that the XDT receiver always accepts new connections. The sender makes an initiative for transmission to the receiver by means of an XDATrequ service primitive. The new connection is indicated by an XDATind primitive. The protocol indicates the successful connection set up to the sender by XDATconf.

After this, the data are transferred by means of a DT message. However in certain cases, the service provider may not preserve the order of the data units. In this case, the ABO data unit is initialized to abort the connection.

Table 1. Comparison of protocol and UML viewpoints

Protocol Viewpoints	UML Design Viewpoints
Behavior oriented	***Behavior design***
What are the behaviors of each communicating entity?	What should happen in the system?
Communication oriented	***Interaction design***
What is the concrete communication exchange between the entities?	What is the control flow of the data?

This is indicated to the users by a XABORTind service primitive. XBREAKind is initialized to stop the transmission for a certain period, if the *go back N* data buffer is full. The end of transmission is indicated by setting the parameter *eom* in the final data unit of XDATrequ and XDATind primitives. The connection is released implicitly, indicated by an XDISind primitive at the sender and the receiver side, after successfully transmitting the last data unit. The further explanation of the XDT protocol can be found in [12].

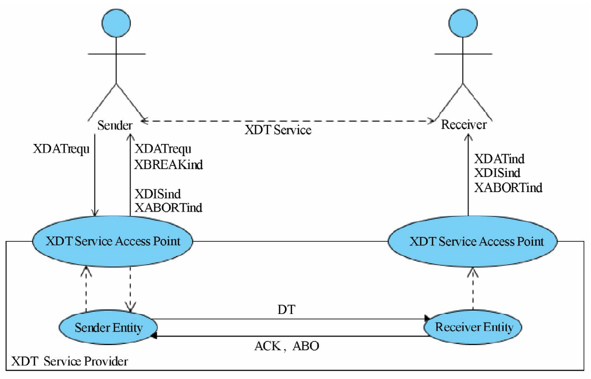
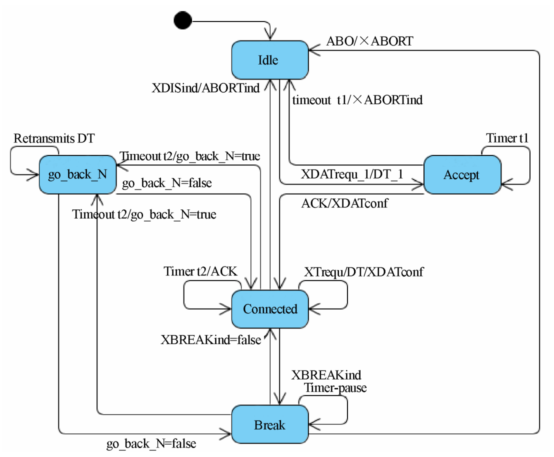
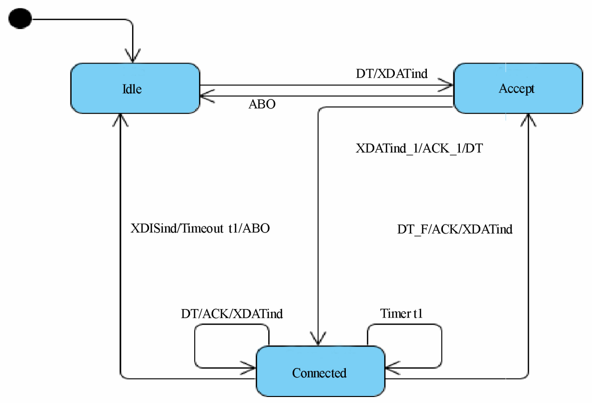


Figure 3. Use case diagram for XDT protocol



(a)



(b)

Figure 4. (a) State machine for XDT protocol-sender; (b)State machine for XDT protocol-receiver

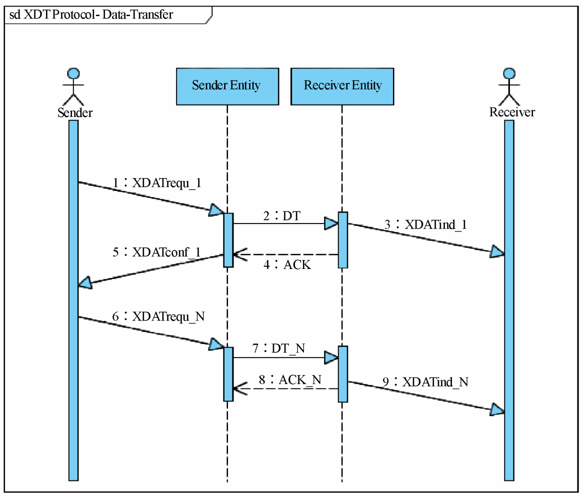


Figure 5. Sequence diagram for XDT data transfer

As a first phase we design the behavior view point by UML use case diagram (see Figure 3) to identify the entities, activity diagrams for the static behaviors, and state machine diagrams (see Figure 4(a), 4(b)) for the dynamic behaviors. Figure 3 i.e. the use case diagram visualize the developer to identify the possible service (XDATrequ, XDATind, XDATConf, XABORTind, XBREAKind, XDISind) and protocol (DT, ACK, ABO) primitives of the protocol. The activity diagrams are used to determine the internal behaviors of the protocol (in which only the semantics are specified). The state machines in Figure 4(a) and 4(b) are the core part for the development. They determine the external behaviors of the protocol by combining the service and protocol primitives. Figure 4(a) and 4(b) represent the sender and receiver part respectively.

As a second phase, we further use the behavior viewpoint as a base and design the communication viewpoint through the sequence (see Figure 5) and timeline diagram to identify the control flow. Figure 5 represents the dynamic behavior of the data transfer state (i.e. connected state in the Figure 4(a) and 4(b)) of the protocol. The same kind of sequence diagram is modeled for all states of the XDT protocol. These sequence diagrams are used further for verifying the protocols.

4.2 Model Checker

To ensure the quality of the developed protocol through the template, the protocol properties (see Table 2) like deadlock, livelock freedom are considered for evaluation. In further we consider our two phase mechanism for verifying these protocol properties.

Phase 1: We retrieve the behavioral viewpoints through the UML use case and activity diagrams from the earlier stage. Later these models are translated into the PROMELA via the UML state machine, where the SPIN tool interprets the code. The difference between our approach and others is the following. We use the state machine diagram as a base for the PROMELA translation, and the semantics from the activity diagrams are added to specify the protocol properties. Since the UML is a semantic-less language, we use the activity diagram as a semantic for the UML state machine model, which is a major advantage. Instead of using external semantics in PIM, the internal semantics makes less complexity and easy usage. The translated PROMELA code is shown in the Figure 6. The protocol entities are described through the keyword *proctype* and the states with *progress*. The code resembles like a C code which is easy to interpret the model. Reference [4] for complete syntax of the PROMELA.

The SPIN model checker executes the PROMELA code and the verification result is produced. The result ensures the quality of the protocol properties like deadlock, livelock, code coverage through its behavior.

Phase 2: To confirm the data flow properties like liveness, the UML sequence diagram is retrieved from the earlier stage and it is converted into a Linear Temporal Logic (LTL) [13]. The LTLs are mathematical annotated formulae to make statements on a linearly progressing time. Since, it is difficult to convert all the UML sequence properties into an LTL; we use another technique known as *Protocol Predictor* (*PP*). It identifies the best case criteria in the sequence diagram and marks the event through a unique identifier, e.g. PP:1. The Protocol Predictor is an automated algorithm for UML sequence diagram. It reads the sequence diagram and maintains a periodic log for all service and protocol primitives. The Protocol Predictor has a pre-defined common rules like, the data should be transferred only after a proper acknowledgment; the sequence number should be verified periodically etc. Based on these rules, the algorithm generates the LTL property for the required protocol. In our case, consider that the protocol is working efficiently by transferring the data with sequence number to the receiver. Here we can predict that the sequence number from the sender and receiver should be equal at any time. To do so, we consider the existing LTL property from SPIN as $\square ((p) \Rightarrow (\lozenge q))$ with PP:1 and shown in the following code.

Table 2. Communication protocol properties

Condition	Properties
Absence of Deadlock	The system never enters a state that cannot be left due to a missing or occupied resource
Absence of Livelock	The system never enters cycles that cannot be left due to a missing or occupied resource.
Code Coverage	Each statement defined in the system can potentially be executed.
Liveliness	Each state of the system can be reached from the initial state.
Robustness	The system can react to unexpected, unusual or missing events.
Termination	The final state or an idle state for cyclic systems can always be reached.
Recovery from Failures	The system can recover to a normal state within a limited time after an error has occurred.

```
PP:1
# define p (Data[sequ].sequ == S_N)      /* Sender Sequence
number */
# define q (Data[sequ].sequ ==    R_N)    /* Receiver Sequence
number */
/*if p becomes true at one state, q should become true at least
once;
Here by assigning if p (sequence number) is true in Sender,
then q (sequence number) should be true in Receiver */
never {       /* !([ □ ((p) ⇒    (◊q)))    */
Start_S:    if
      :: (! (q) && (p)) → goto accept_S
      :: (1) → goto Start_S ; fi;
accept_S: if    :: (! (q)) → goto accept_S; fi;    }
```

The idea behind the conversion is that; instead of identifying the worst cases in the communication protocol, we look for the failure of best cases (successful data transmission) which results in identifying the worst cases. This is due to the probability of identifying the worst cases is very less than the probability of best cases. By means of this LTL, it is easy to identify the failure cases like the possibility that sender becomes true and thereafter the receiver remains false forever (or) the possibility that sender becomes *false* before the receiver becomes *true*. Further this code is imported as a supple mentary data to the PROMELA code through the SPIN tool for verification. The SPIN model checker validates whether the property holds or not. By investigating this type of combination from the sequence diagram, it is determined that an error-free model is designed. The final result is obtained by transferring five sample protocol primitives from the sender to receiver entity in the SPIN tool. The tool simulates the PROMELA code as a graphical state chart (see Figure 7) to identify the dynamic behaviors and verifies the defined (PP:1) protocol property simultaneously. The verification output from the SPIN tool is shown in Figure 8 with the number of depth reached, state and transition explored. Figure 8 illustrates that no deadlock, livelock is detected in the verification and the five protocol primitives are transferred successfully. The designed model (see Figure 5) is been compared with the SPIN simulated model (see Figure 7). The data transfer phase (second iteration of the Figure 7) is matched perfectly with the designed model. This ensures that the design model is verified for the correctness properties. The advanced LTL property verification represents the model is checked for the protocol properties.

5. Final Remarks

We have discussed about the need of model driven architecture in designing a protocol for dependable systems and the importance of verification. From the above discussion, it is well understood that the combination of MDA technique and the SPIN tool is a reasonable match for the communication protocol development. MDA has the advantage of rapid system development and the SPIN provides a powerful verification mechanism. Since it is an example consideration, the implementation and the

```
active proctype Sender_Entity()
 /* Sender Protocol Specification */
{
progress_phase_connect_s:
             XS_XR!Data[1];
accept_Sender:
if
::XR_XS?Ack[1] -> goto Transfer
::else -> goto progress_phase_connect_s;
fi;
Transfer:
atomic {
 progress_phase_Data_Transfer_s:
If
::(! go_back_N) && (! B_break) ->
sequ = sequ + 1; XS_XR!Data[sequ];
.....................
fi; }
end_Sender_Entity: }
```

```
active proctype Receiver_Entity()
 /* Receiver Protocol Specification */
{
progress_connect_r:
if
::XS_XR?Data[1] -> goto
progress_Data_Transfer_r
::else -> goto progress_connect_r
fi;
progress_Data_Transfer_r:
if
::XS_XR?Data[sequ] ->
………………
fi;
end_Receiver_Entity:
}
```

Figure 6. Promela code for XDT protocol

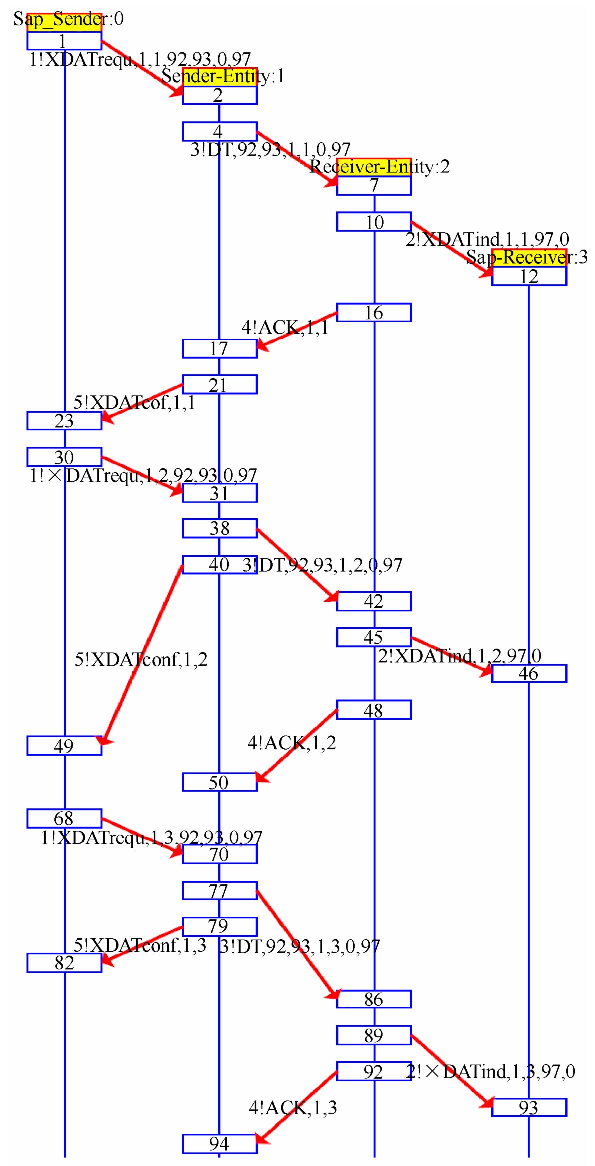


Figure 7. Message sequence chart from SPIN simulation

transformation is carried out manually to test the efficiency of the template. The design and the simulation phase are correlated among each other and the

```
(Spin Version 5.1.4-27 January 2008)
            +Partial Order Reduction

Full statespace search for:
            never claim      +
            assertion violations + (if within scope of claim)
            non-progress cycles+(fairness enabled)
            invalid end states-(disabled by never claim)

state-vector 692 byte, depth reached 149, errors:0
   3816 states, stored (8976 visited)
   9673 states, matched
  18649 transitions (=visited+matched)
   6286 atomic steps
hash conflicts:     2(resolved)
```

Figure 8. Result obtained from the SPIN tool

effectiveness was measured with the UML sequence diagram and the SPIN chart. As a shot term vision, the architecture template and verification strategy has developed on the basis of the MDA approach with the PIM as example implementation.

The further work of the proposed research is to build an automated architecture template for communication protocols. The pitfalls in the existing MDA approach like explicit semantics with standard specifications will be incorporated by proper solutions. It is also planned to develop UML components for the communication protocols. The basic behavior of the protocols will be pre-defined as a component through sequence diagram. Later the sequence diagram will be used in the rapid development as drag-an-drop. Since, we focus to develop a common approach; the same can be used in any protocol development. As a long term vision, the implementation of the developed architecture will be carried out with a real-time peer-to-peer intrusion detection protocol from design to deployment stage.

REFERENCES

[1] C. Werner, “UML profile for communicating systems,” Ph.D thesis, University of Göttingen, Department of Computer Science, 2006.

[2] Unified Modeling Language, The official homepage of UML, Object Management Group. http.//www.uml.org.

[3] Model Driven Architecture: A Technical Perspective, Architecture Board MDA Drafting Team, Document Number ab/2001-02-04, ftp://ftp.omg.org/ pub/docs/ab/ 01-02-04.pdf, Object Management Group, February 2001.

[4] Process Meta Language. http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html.

[5] G. J. Holzmann, “The model checker SPIN,” IEEE Transactions on Software Engineering, 23 (1997) 5: pp. 279–295, 1997.

[6] M. Farid, G. Patrice, and B. Mourad, “Verifying UML diagrams with model checking: A rewriting logic based approach,” Seventh International Conference on Quality Software (QSIC 2007), pp. 356–362, 2007.

[7] S. Wuwei, C. Kevinon, and H. James, “A toolset for supporting UML static and dynamic model checking,” 26th Annual International Computer Software and Applications Conference, 2002.

[8] B. Prasanta, “Automated translation of UML models of architectures for verification and simulation using SPIN.,” 14th IEEE International Conference on Automated Software Engineering (ASE’99), pp. 102–109, 1999.

[9] S. W. Vitus and J. Padget, “Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach,” 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS’04), pp. 337–346, 2004.

[10] A. Kleppe, J. Warmer, and W. Bast, “MDA Explained—The Model Driven Architecture: Practice and Promise,” Addison-Wesley, 2003.

[11] Query, Views, Transformations: A Specification document. http://www.omg.org/technology/documents/modeling spec catalog.htm.

[12] eXample Data Transfer (XDT) Protocol. http://www.protocol-engineering.tu-cottbus.de/index_xdt.htm

[13] E. M. Clarke, O. Grumberg, and D. Peled, “Model checking,” MIT Press, 1999.