



Georg-August-Universität
Göttingen
Institut für Informatik

ISSN 1611–1044
Nummer IFI–TB–2004–002

Technischer Bericht

Modeling Soft State Protocols with SDL

Xiaoming Fu and Dieter Hogrefe

**Technische Berichte
des Instituts für Informatik
an der Georg-August-Universität Göttingen**

August 2004

Georg-August-Universität Göttingen
Institut für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 14

Fax +49 (5 51) 39-1 44 15

Email office@informatik.uni-goettingen.de

WWW www.ifi.informatik.uni-goettingen.de

Modeling Soft State Protocols with SDL

Xiaoming Fu and Dieter Hogrefe
Telematics Group, University of Göttingen
Lotzestr. 16-18, 37083 Göttingen, Germany
{fu,hogrefe}@cs.uni-goettingen.de

Abstract—Soft state enables new services to packet-switching networks by introducing a type of state in the network nodes which is refreshed by periodical messages otherwise expire. System designers build protocols that implement soft state concepts based on intuition or on high-level explanations believe that the design is “better” than hard state and soft state implementations should be robust, reliable and interoperable. As states in the network nodes are critical for both applications the and network infrastructure, the operations of soft state protocols, which tend to be designed more and more complex, need to be error-free and deadlock-free. Thus, verification, formal analysis and validation of these protocols become a vital task. In this paper we utilize formal techniques, specifically, Specification and Description Language (SDL) and Message Sequence Chart (MSC), for modeling, analysis and validation of general soft state protocols. We propose a general architecture of state management systems and find several points through the SDL/MSC modeling which may enrich the design, modeling and evaluation of real soft state protocols: 1) modeling these protocols using these techniques is feasible, 2) it can be possible to use these techniques to identify possible design errors and deadlocks/livelocks, which may be caused by imprecise informal specifications of these protocols.

Keywords: Soft State, Hard State, SDL, MSC, Protocol Design, Formal Modeling

I. INTRODUCTION

In communication networks, there is a need to maintain certain information (“state”) in network nodes, associated with endpoint-generated sessions or calls. For example, ATM switches maintain information about VCs such as bandwidth allocation and VCI/VPI input-output mapping. As state generally reflects some requirements of an end-to-end session/call to the traversed nodes, it needs to be maintained properly, especially when network “conditions” change (e.g., some link or node fails, or the traversing route changes).

The state maintained by the network can be categorized as *hard state* and *soft state*. Hard state is installed in nodes upon receipt of a setup message and is removed only upon receipt of an explicit tear-down message. It is vital for the state installer to know when the state has been installed or removed, and ensure that install and removal are performed only once. Furthermore, since hard state remains installed unless explicitly removed, there needs to be a mechanism to remove orphan state that after the state installer has crashed or departed without removing state. In contrast, “soft state” refers to certain non-permanent control state in network nodes which will be expired unless refreshed. Since soft state will

eventually expire, this approach does not require explicit removal or a mechanism for removing orphan state.

For packet-switching networks such as the Internet, a conventional thought was that state information would be only needed for end systems. As stated in the fundamental end-to-end principle [1], “An end-to-end protocol design should not rely on the maintenance of state (i.e. information about the state of the end-to-end communication) inside the network. Such state should be maintained only in the endpoints, in such a way that the state can only be destroyed when the endpoint itself breaks.” This “end-to-end communications should not to have state inside the network” idea, however, has hardly been realized in real networks. For example, routing protocols such as BGP, OSPF, RIP and IS-IS, ATM signaling protocol (Q.2931) and early Internet signaling protocol (ST-II) [2] use hard state in the network. In addition to hard state, soft state was also introduced into the Internet. After being applied in the Resource Reservation Protocol (RSVP) [3], [4], which allows establishing QoS resource reservation state in the network nodes along the path for end-to-end communications in IP networks, the soft state paradigm has been followed by many others such as the Real-Time Control Protocol (RTCP) [5], Protocol Independent Multicast (PIM) [6], the Session Initiation Protocol (SIP) [7] and the Cross-Application Signaling Protocol (CASP) [8], [9]. By the use of state – either hard state or soft state – inside the network, protocols can provide certain enhanced services for end-to-end communications. Note both hard state and soft state can be installed either in intermediate nodes or end hosts only, or both. Due to the nature of state, unlike other types of protocols, state management protocols often requires extremely complex and powerful mechanisms to ensure that the state is perfectly synchronized and up-to-date (otherwise things could be evil). With the informal, text-based IETF specifications, operations of these protocols tend to be error-prone. For example, a recent report [10] showed numerous problems or unexpected behaviors in the specification and implementation of TCP [11], the dominating end-to-end transport protocol which uses hard state in end hosts. With the ever-increasing number of soft state protocols and the increase of their complexity, unfortunately, the risk of design and implementation errors for soft state protocols increases. Therefore, it becomes vital to ensure the correctness of state management operations in protocol specifications.

The ways to discern and correct state management operations can be classified into two basic groups. The most natural way, namely the empirical approach, is to study

protocol behaviors with an actual implementation (or a prototype). Another possibility follows a model-based approach in which protocol behaviors may be studied with a model of the protocol. The empirical approach is only effective for examining standard, ordinary behaviors of a protocol, whereas model-based approaches, based on simulation or analytical models, can be more effective in determining possible errors in the design. Applying the model-based approach to state management systems can, unlike the empirical (“trial-and-error”) way of debugging and correcting these specifications, avoid excessive problems (and costs!) in standards and implementations. Furthermore, as soft state protocols involves two or more network nodes, each requiring to independently maintain several timers and soft state associated with an end-to-end session, any real soft state protocol is rather complex and difficult to be analyzed through implementations. Moreover, they are generally specified informally and imprecisely. Therefore, model-based approaches are preferred.

In this paper we follow a model-based approach to study the basic functionalities and aliveness properties of general soft state systems, using the Specification and Description Language (SDL) [12] and the Message Sequence Charts (MSCs) [13]. As successfully demonstrated in experiences in modeling other communication protocols (e.g., [14]), SDL and MSCs are efficient tools for such tedious tasks. By presenting a general architecture for state management systems and modeling one representative system with SDL/MSC, we demonstrate the feasibility of applying this approach in the modeling of soft state protocols, analyzing basic properties of them and advocating a potential way of improving protocol descriptions. Current work is limited to the general state management system rather than a real protocol; a few minor weaknesses still remain in the tools. In this paper we focus on the functionality modeling and validation of different variants of soft state protocols and hard state protocols described in [15], including verification of (the absence of) deadlocks and livelocks. Our goal here is not to model a real soft state protocol such as RSVP or CASP, or hard state protocols like ST-II or TCP, but rather to model general soft state protocols in order to capture and verify the essential concepts and general functionalities of interest. There are other, non-functional aspects of these protocols, such as complexity and performance, but they are beyond the scope of this paper.

After shortly reviewing the related work in Section II, we present a general architecture of soft state protocols in Section III, followed by SDL models of different soft state variants in Section III and a verification of the models (Section V). We summarize our modeling experiences and outline future work in Section VI.

II. RELATED WORK

A. Studies on Soft State Protocols

System designers argue soft state is “better” than hard state, and using soft state the handling of network condition changes is “easy” [16], [17]. However, these claims have been more based on intuitive, high-level thoughts and explanations,

instead of formal, exhaustive modeling and analysis. In contrast to the original expectations, soft state protocols being developed so far are still far from being simple, especially when coupled with channel reliability, multicast sessions or traffic control models.

Soft state protocols developed so far can be categorized into two types: end-to-end protocols and hop-by-hop protocols. The former only involves certain type of state in an end-to-end way, without bothering any other nodes in between; examples of this type include RTP and SIP. Hop-by-hop protocols, such as RSVP and CASP, on the other hand, involve state in one or more router(s) in between in addition to state in the communicating ends. The latter is more representative and more comprehensive to demonstrate the soft state operations so we choose this as the example for general discussions of soft state.

Given the particular importance of soft state protocols, there have been recently a few efforts on their modeling and analysis. Raman and McCanne [17] presented a model for the soft state notion based on Jackson queueing networks; a performance study of hard state and soft state signaling protocols was performed by Ji *et al.* [15]. However, more detailed formal modeling and validation is still missing.

B. An Introduction to SDL and MSC

As described earlier, SDL and MSCs are two potential techniques for modeling soft state protocols. In SDL, a system is divided into building blocks that communicate using channels. Blocks are composed of processes. Processes (within a block) are connected using signal routes. Each process is an extended finite state machine, which has its own infinite queue and is assumed to operate independently from all other processes. MSCs are another valuable description technique for visualizing and specifying inter-system, asynchronous component interaction. MSCs’ strength lies in their ability to describe communication between cooperating processes. Each process is represented as an identifier and has a process life line extends downward. There are arrows representing messages passed from a sending to a receiving process. Messages not starting or ending at a process life line are exchanged with users, be they human or mechanical (so-called “the environment”).

Detailed descriptions of SDL and MSC can be found in [12], [13], [18] and an example of modeling systems using the combination of SDL and MSC can be found in [14]. Modern SDL development tools like Telelogic Tau SDT also support verification and validation based on developed models.

III. A GENERAL ARCHITECTURE OF STATE MANAGEMENT SYSTEMS

As described by Ji *et al.* [15], in contrast to hard state (HS) protocols, soft state (SS) protocols can be further classified into 4 variants: ‘pure’ soft state (pSS), soft state with explicit removal (SS+ER), soft state with reliable trigger (SS+RT) and soft state with reliable trigger/removal (SS+RTR). In this section, we present a simple abstraction and typical operations

for all these state management protocols (both soft state and hard state), as well as possible problems that can occur in their operations.

We begin our modeling with a generic architecture for state management systems as shown in Fig. 1, which covers two services and one protocol as below:

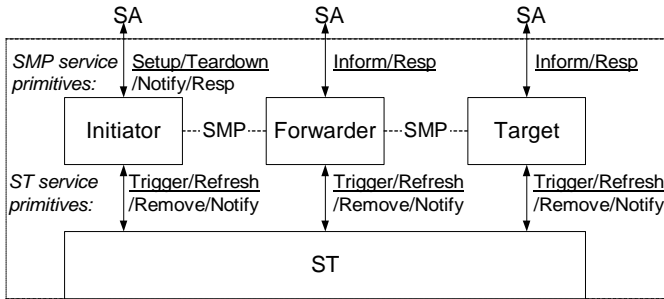


Fig. 1. A general architecture of state management systems

- The state message transport (ST) service, which transmits state messages over the lossy channel.
- The (soft or hard) state management (SM) service, which is the service rendered by the state management protocol to the state application (SA).
- The state management protocol (SMP), which manages state in the network nodes.

A. Expected Behavior of State Management Protocols

Behaviors of a state management protocol are determined by timers and state messages used by the three types of protocol entities, namely the state initiator, forwarder and target. *Timers and state messages:* An important feature of soft state systems is timers. In a soft state system, there could be three types of timers dealing with state management: the *state timer* which will expire the state unless refreshed, the *refresh timer* which triggers periodical refreshes, and the *retransmission timer* which triggers periodical retransmission of trigger or removal messages (SS+RT or SS+RTR). In HS systems there are only retransmission timers for state trigger and removal messages, while state timers and refresh timers are necessary for SS systems for operating state refresh messages. The existence or absence of these timers and different operations for state messages in a state management protocol determines which protocol type it belongs to.

Protocol behavior: We present the Message Sequence Chart (MSC) description (shown in Fig. 2) to illustrate the messages and mechanisms of various state management protocols. The communications between the three types of entities are realized through several service primitives (Setup, Trigger, Teardown, and optionally, Refresh, Resp, Notify and Remove). For the simplicity purposes we omit the Inform and Resp primitives since they generally do not change state information.

The protocol communication takes place in three possible, distinct phases:

- *State Setup:* This phase is initialized by SA in the entity Initiator with a *Setup*. Initiator can thereafter issue a *Trigger* towards the entity Target. Upon the receipt of Trigger, every Forwarder entity creates a state (which is associated with a state timer for soft state protocols), and then forwards the Trigger message on. When Target receives the Trigger, it creates a state (which is associated with a state timer for soft state protocols). If HS, SS+RT or SS+RTR is used, additionally Target issues back a *Notify* to Initiator when it receives a Trigger. In this case, Initiator starts a retransmission timer before it issues Trigger. If it does not receive a Notify after the Retransmission timer expires, Trigger is transmitted again. After repeating certain times, the retransmission stops and Initiator returns to the initial state.
- *State Maintenance:* This phase is only used in soft state protocols. Upon the expiration of the refresh timer, Initiator sends a *Refresh*¹ towards Target and restart the timer. Any forwarder receives the Refresh checks whether corresponding state already exists: if yes, refreshes the state timer, otherwise recovers a state together with a state timer. Then it forwards the Refresh on towards the Target. Upon receipt of the Refresh, similar to Forwarder, Target recovers state or just refreshes the state timer. If no Refresh is received in a Forwarder or Target before the state timer expires, state will be removed.
- *State Teardown:* In pSS or SS+RT, there is no such phase; Initiator just does nothing and state in all the other nodes will expire when their state timers time out. In other state management protocols, a *Remove* is issued by the Initiator towards the Target to remove all state and associated timers (if exist). Additionally, if HS or SS+RTR is used, after Target receives Remove and removes its state (and timers), a *Notify* is sent back to the Initiator. Initiator in a HS or SS+RTR system follows a way similar to the retransmission in state setup phase if no Notify is received within a given time.

B. Possible Problems in State Management Operations

The most obvious problem that occurs in state management protocols is failure to install or remove state correctly. In addition to this, there are timing considerations to be taken account, since a protocol of this kind needs to be able to react to timer events and receipts of state messages appropriately. An installed state can become invalid either due to the receipt of a removal message or state timer expires. The latter can occur either because a state refresh or notify message gets lost during its transmission, or as a result of state initiator gets crashed.

There are other potential problems with state management protocols. For example, there can be infinite state management

¹Refreshes described here are limited to be originated from Initiator. Some SS protocols (for example, RSVP) further allow more reactive operations upon network condition changes, where Refresh messages can also be initiated an intermediate node. This behavior is not covered in this paper.

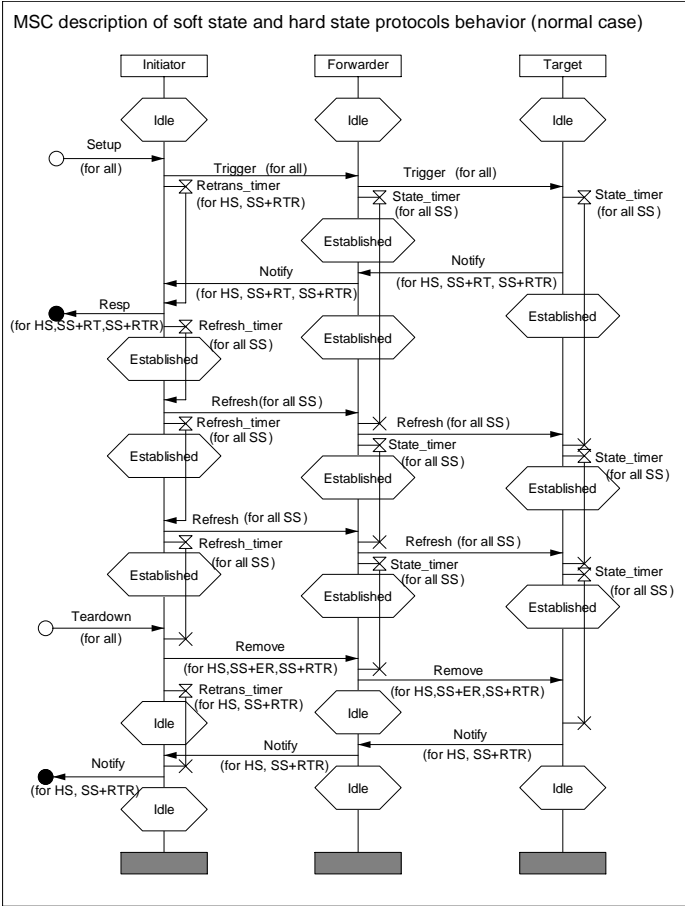


Fig. 2. MSC for various state management protocols operation (normal case)

loops, i.e., state messages enter a transmission circle somewhere between an initiator and a target. This kind of loops can result in two possible errors:

- **Deadlocks:** a state management system goes into a state without possibilities to enter another state. A deadlock is most often caused by the fact, which one process waits for a message from a second one which in reverse waits for a message from the first one; both wait and nothing happens. In pSS, deadlocks are theoretically impossible, as it only allows refresh and expiration operations for a state, and there is no resource contention. However, we cannot exclude this possibility in more comprehensive state management protocols, due to more complex synchronization and/or notification mechanisms.
- **Livelocks:** a state management system into an endless loop consisting of a set of states but cannot jump out to make the progress towards a next expected behavior.

These are very undesirable since there appears to be a valid state management behavior in any individual node, but in reality it cannot further deliver other desired messages or jump out of certain running state(s). For example, the following operation could be possible in the original description of

SS+RT [15].

Example 1. A deadlock behavior in SS+RT. A brief explanation of such a scenario is as follows:

- 1) Initiator initially sends a Trigger message to Target through Forwarder. The link between Forwarder and Target then suddenly goes down.
- 2) The Trigger message is lost before reaching Target, and Initiator cannot receive a desired Notify message which acknowledges the success of state installation along the path.
- 3) After the retransmission timer expires, Initiator resends a Trigger towards Target.
- 4) Again the Trigger is lost, and Initiator still thinks it is just due to random loss and retransmits the Trigger.
- 5) If there is no specification about which conditions to stop sending Triggers, the result is a deadlock in which case Initiator expects a Notify (but cannot receive it) after sending out a Trigger while Responder expects a Trigger but (but cannot receive it).

This error is somewhat easy to detect and fix. However, sometimes such flaws can be very subtle, so that even senior designers cannot detect them in protocol specifications, particularly in IETF informal, text-based specifications for complex operations of state management protocols, which were designed typically without formal modeling, validation and verification. For example, the inadvert synchronization problem [19] was noticed as an important issue for periodical soft state systems. In real specifications of some soft state protocols, for example RTCP [5] and RSVP [4], designers have tried to avoid this problem by setting the refresh timers to be varied randomly (e.g., over the range [0.5, 1.5] times the calculated interval). However, true randomness in a real implementation is hard to achieve and moreover, as these intervals are sometimes not mandatory requirements in protocol specifications (e.g., as a “should” requirement in RSVP), typically default fixed values (30 seconds in the case of RSVP [20]) are used instead in practice for implementation simplicity. All these still contribute to the potential synchronization problem of soft state management.

IV. MODELING SOFT STATE PROTOCOLS WITH SDL

A. Key Modeling Issues and the System Model

Without loss of generality, we chose the most comprehensive state management protocol, SS+RTR in a hop-hop manner, as the target for modeling. Other variants can be easily derived from this model by removing certain messages, timers, state transitions and/or behaviors.

As our work is still preliminary, we here only model the key concepts of state manage protocols, not to every detail. Address of each node is represented by a pre-assigned integer, 1, 2, 3, respectively. The state message format is also simplified as shown in Fig. 3(a). There are further issues vital for the modeling process:

- 1) How do we model ST, so as to allow state messages (generally from Initiator to Target) be visible for Forwarder?

- 2) How do we model the lossy channel, which can be of a given loss rate?
- 3) How do we model the duration (start and end) of a session state? This also naturally reflects the system model, namely which information should be made inside the system, and which needs to be put in the environments.

Fig. 3 shows the SMP system model for SS+RTR. We assume the SMP system model to be composed of three nodes: Initiator, Forwarder and Target, each of which is represented by a process. Furthermore, an additional process ST is used to transmit SS messages from Initiator towards Target, or reverse. Both 4 processes forms the only block *SM* of the system.

B. ST Modeling

ST is modeled as Fig. 4. It can be used for all variants of state management protocols. Here we use random Abstract Data Type (ADT) to simulate a given loss rate of the link. Note the transport service should determine the direction (forward or backward) according to the type of the message it receives. A more comprehensive ST can have different loss rates in different links and this feature is to be added in the next step.

C. SMP Entities Modeling

Fig. 5 (Initiator), Fig. 6 (Forwarder) and Fig. 7 (Target) show the detailed models for each of the three SMP entities.

In order to model the duration of a session state, the Initiator process communicates with environments through an SA-SMP interface. When it receives an ASetup with certain state information data, it assigns a new session identifier (sid) and installs a soft state locally, then normally goes through the state setup and maintenance phases. When Initiator receives an ATearDown from the SA-SMP interface, it enters into the teardown phase. For simplicity sid is currently set as a fixed value. To avoid confusion different notification messages have to be identified: Notify1 for notifying the success of state setup, Notify2 for notifying state timer expiration in Target, and Notify3 for notifying the success of state teardown.

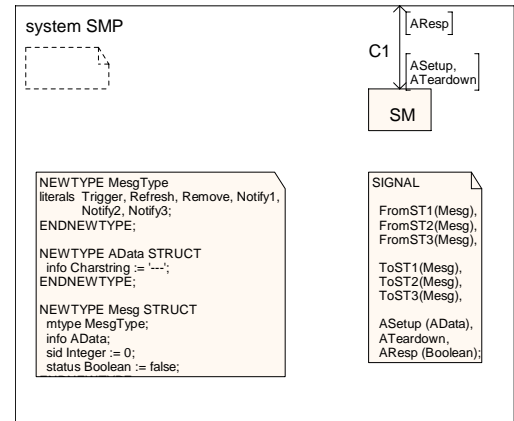
The Forwarder process first listens to ST input. If a Trigger or Refresh message arrives, it installs a soft state locally and forwards the message on. Expiration of a local state in Forwarder only removes itself.

The Target process installs – or refreshes, if exists – a soft state locally upon receipt of a Trigger or Refresh message. When receiving a Trigger or Remove message, Target needs to notify Initiator about it. Expiration of a local state in Target also accompanies a Notify2 to trigger a teardown.

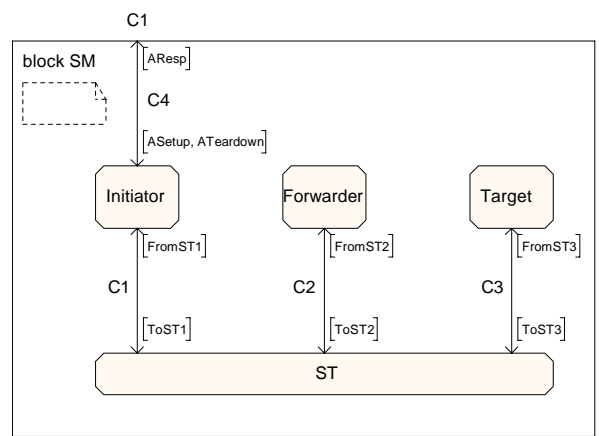
V. MODEL VERIFICATION AND VALIDATION

The finite state machines derived from the SDL model is shown in Fig. 8.

Since state management protocols involve distributed timing and message interaction, their correctness is hard to verify using an informal description. We use the Tau integrated package SDT 4.4 which includes support for SPIN, to verify the SDL model of the SS+RTR protocol against deadlocks, unspecified receptions, livelocks and unreachable states. In



(a) System SMP



(b) Block SM

Fig. 3. Description of the SDL System and Block

order to verify the model, we have chosen the following scenario case study to validate our design against some of the specific properties of the modeled SS+RTR protocol:

- 1) Establish a session between Initiator and Target.
- 2) Stop the state message transmission between Forwarder and Target.
- 3) Change the ST loss rate between Forwarder and Target to different values between 0 and 1.
- 4) Stop the Target process.
- 5) Let Initiator teardown the session.

With the above scenario, we have covered all ST service primitives and SMP service primitives as well as all important scenarios, but not all possible scenarios. Therefore, after checking the scenario, we have used Tau validator to validate all possible walk algorithms.

We generated a number of MSCs for this scenario case study to check the protocol functionality at each stage of the simulation. For the SDL models designed in Section III,

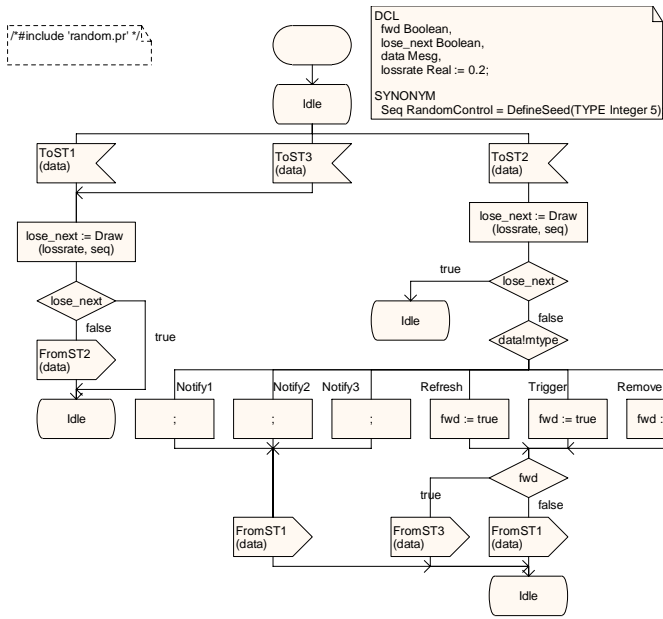


Fig. 4. SDL Model for State Message Transport

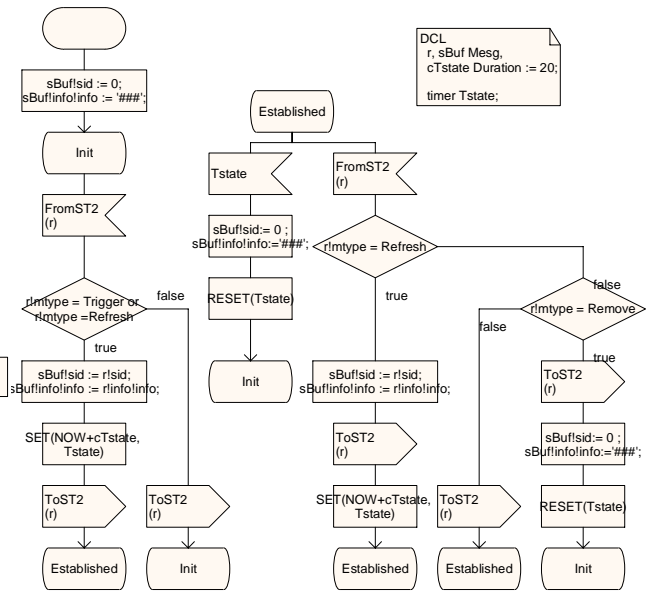


Fig. 6. SDL Model for Forwarder (SS+RTR)

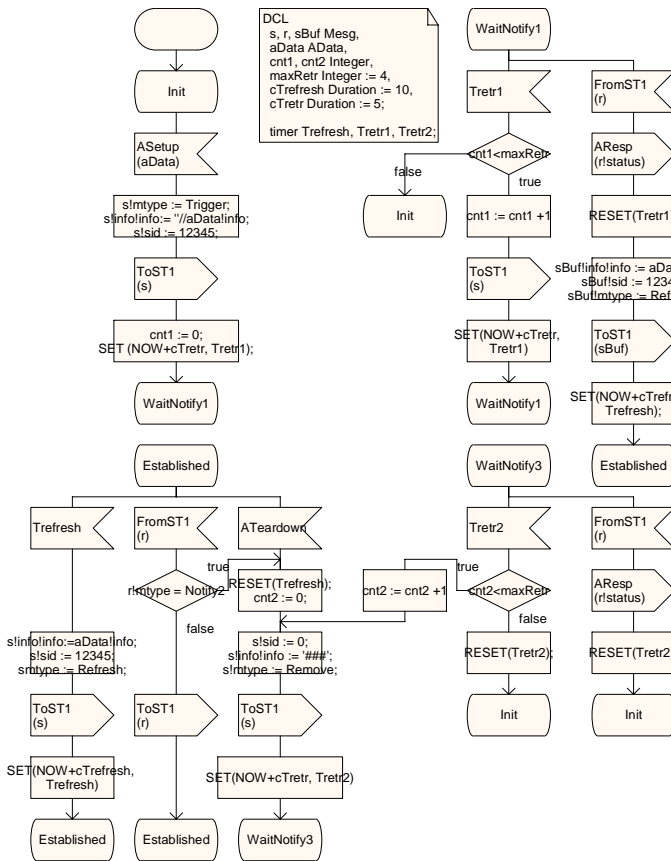


Fig. 5. SDL Model for Initiator (SS+RTR)

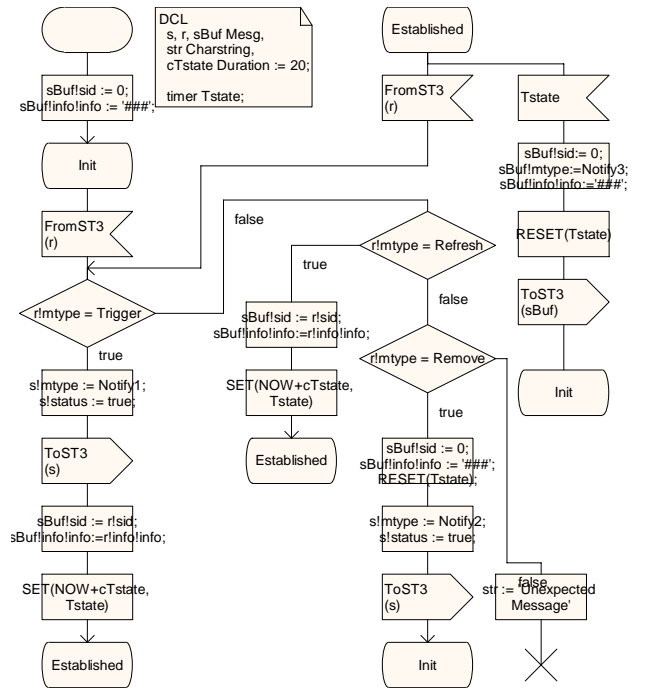


Fig. 7. SDL Model for Target (SS+RTR)

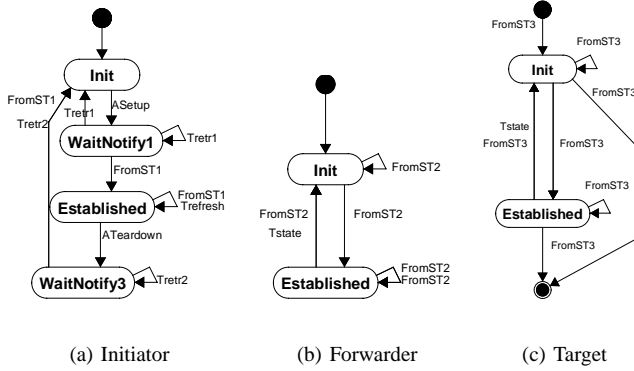


Fig. 8. State Diagram of a Soft State Protocol (SS+RTR)

MSCs shown in Fig. 9-11 represent some interesting protocol behaviors. Through a comparison with the general description, we are able to refine the abstract system and make more concrete descriptions.

Through this procedure, we have found some modeling errors as well as flaws in the original description, including the missing default behavior for some message types, and unreachable states. One experience we learnt was that an imprecise informal specification can bring deadlocks and livelocks (for example, the retransmission counters and refresh durations – either explicit or implicit – are missing in many text-based IETF specifications); through verification and validation of the SDL models, they could be avoided. As an example we corrected a flaw in the original description by adding description on retransmission counters.

Some other results show that in the first version of the models Notify1, Notify2 and Notify3 messages received by Forwarder were not processed but consumed; they need to be sent back to ST. Also, misused values of timers can exclude Initiator from entering the Established state. We also come to the following conclusion: by adding reliable trigger and explicit removal to soft state protocols, the usage of state (reflected as network resources especially memories) can be more efficient. This can be explained as, for example for the reliable explicit removal case, if a user tries to remove a state, but the teardown message is lost during transmission, the state will remain in place until it times out after a relatively long time. Since state typically implies enhanced services for end-to-end communications, holding a state incurs costs, thus this will impose users to pay for the extra time that has been spent waiting for the state expiration.

With the developed model, we have verified these aspects against the description of SS+RTR, and improve it.

VI. SUMMARY AND FUTURE WORK

Given the fundamental importance of soft state protocols, it is vital to guarantee their behaviors are specified correctly. We have generalized and modeled behaviors for different variants of soft state communications with the aid of formal techniques SDL and MSC. On the other hand, we found that due to the

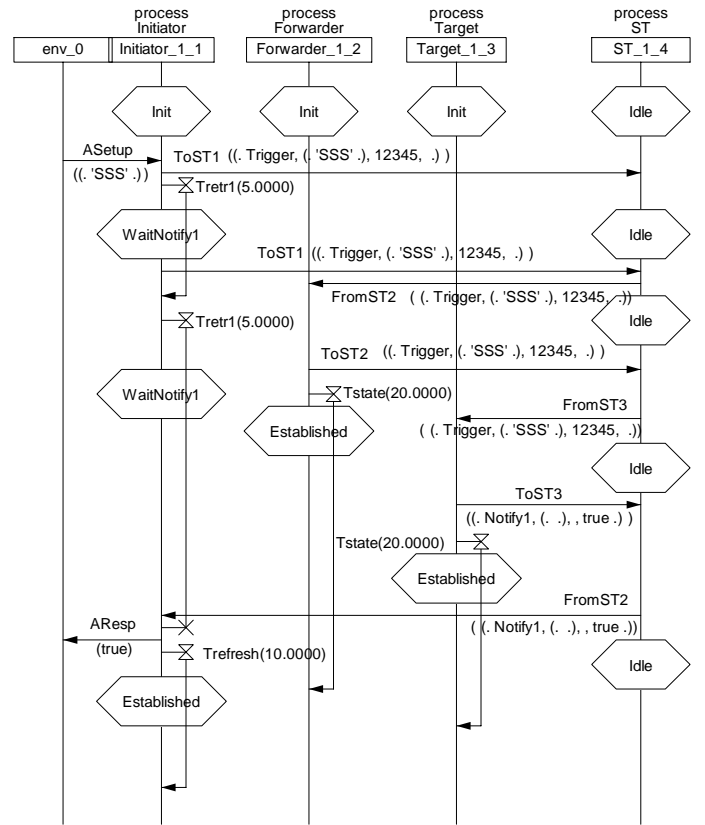


Fig. 9. An MSC for State Setup Phase(SS+RTR)

utilized tool's weakness of inaccurate timing, more realistic performance evaluation has to rely on tools with stronger real-time support. Nevertheless, the powerful modeling language has turned out to be of great help for efficient designing and engineering models of soft state communications, and in particular functionality behaviors (through checking for possible deadlocks and livelocks). We intend to extend our model to support dynamic process creation to handle soft state requests. Currently we are modeling the generic soft state signaling protocol CASP which relies on these existing transport services and a discovery component. Next we would like to study soft state protocols' robustness in handling node failure, mobility and route change cases using formal techniques.

ACKNOWLEDGEMENT

The authors would like to acknowledge Milan Zoric, Sebastian Mueller and Michael Ebner for their helpful suggestions.

REFERENCES

- [1] B. Carpenter, "Architectural principles of the Internet," Internet Engineering Task Force, RFC 1958, June 1996.
- [2] L. Delgrossi and L. Berger, "Internet stream protocol version 2 (ST2) protocol specification - version ST2+," RFC 1819, Aug. 1995.

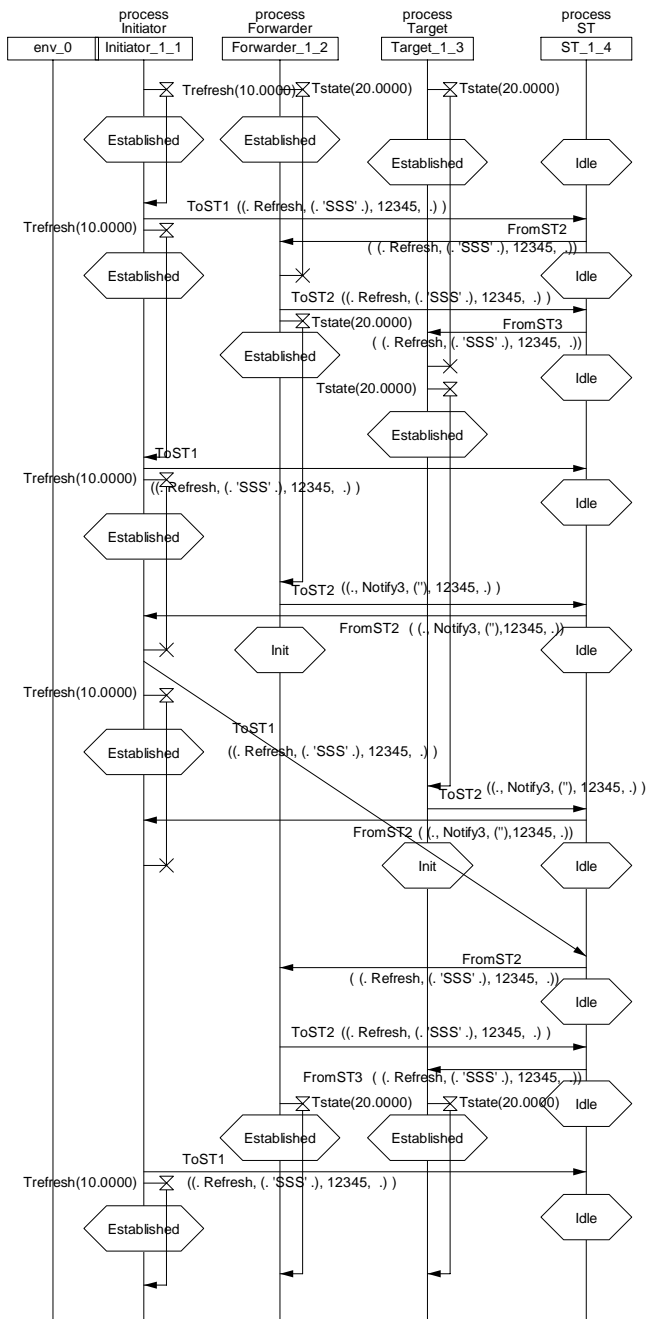


Fig. 10. An MSC for State Maintenance Phase (SS+RTR)

[3] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new Resource ReSerVation Protocol," *IEEE Network*, vol. 7, no. 5, pp. 8–18, Sept. 1993.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – version 1 functional specification," RFC 2205, Sept. 1997.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," RFC 1889, Jan. 1996.

[6] D. Estrin, D. Farinacci, et al., "Protocol independent multicast-sparse mode (PIM-SM): protocol specification," RFC 2362, June 1998.

[7] J. Rosenberg, H. Schulzrinne, et al., "SIP: session initiation protocol," RFC 3261, June 2002.

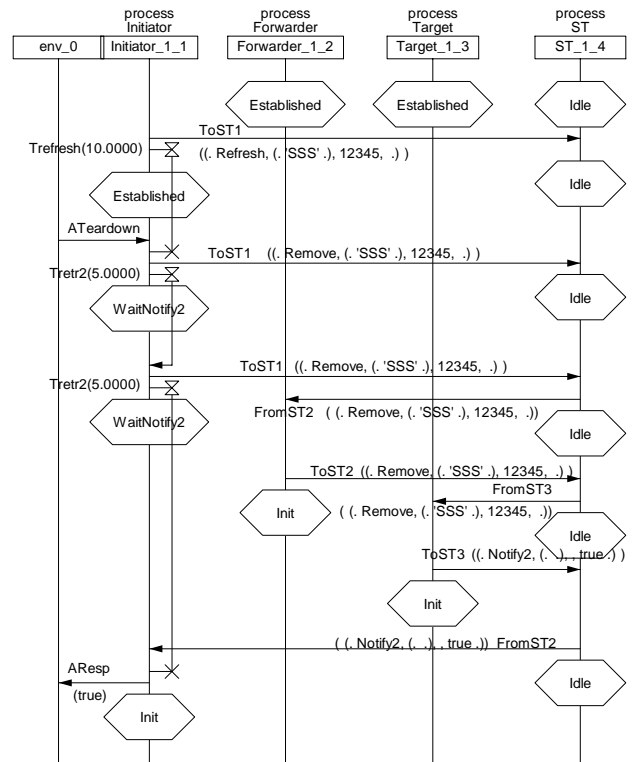


Fig. 11. An MSC for State Teardown Phase (SS+RTR)

[8] H. Schulzrinne, H. Tschofenig, X. Fu, and A. McDonald, "CASP – Cross-Application Signaling Protocol," Internet draft (draft-schulzrinne-schulzrinne-casp-01), work in progress, Mar. 2003.

[9] X. Fu, D. Hogrefe, and S. Willert, "Implementation and evaluation of the cross-application signaling protocol (CASP)," in *Proc. of ICNP 2004*, Berlin, Germany, Oct. 2004.

[10] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, and J. Semke, "Known TCP implementation problems," Internet Engineering Task Force, RFC 2525, Mar. 1999.

[11] J. Postel, "Transmission Control Protocol," RFC 793, Sept. 1981.

[12] *ITU-T Recommendation Z.100 – Specification and Description Language (SDL)*, 1999.

[13] *ITU-T Recommendation Z.120 – Message Sequence Chart (MSC)*, 1999.

[14] P. Schaible and R. Gotzhein, "View-based animation of communication protocols in design and in operation," *Computer Networks*, vol. 40, no. 5, pp. 621–638, 2002.

[15] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A comparison of hard-state and soft-state signaling protocols," in *Proc. of SIGCOMM 2003*, Karlsruhe, Germany, Aug. 2003.

[16] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, "Scalable timers for soft state protocols," in *INFOCOM'97*, Kobe, Japan, Apr. 1997.

[17] S. Raman and S. McCanne, "A model, analysis, and protocol framework for soft state-based communication," in *Proc. of SIGCOMM 1999*, Cambridge, MA, Sept. 1999.

[18] J. Ellsberger, D. Hogrefe, and A. Sarma, *SDL – Object Oriented Language for Communication Systems*. Prentice Hall, 1997.

[19] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 122–136, 1994.

[20] P. Pan and H. Schulzrinne, "Staged refresh timers for RSVP," in *Proc. of 2nd Global Internet Conference*, Phoenix, AZ, Nov. 1997.