Toward Conceptual Specification of Communication Protocols

Sabah Al-Fedaghi

Computer Engineering Department, Kuwait University, Kuwait
sabah.alfedaghi@ku.edu.kw

Abstract

Communication Protocols are specified by using either formal or graphical notations. For this purpose, Specification and Description Language (SDL) is a formal language used extensively in telecommunication for development of software and hardware. Its diagrammatic version with complementary Message Sequence Chart (MSC) diagrams provides a description of system communication in the form of message flows. Still, the resultant diagrams are fragmented and lack continuity in depicting the succession of events. This paper proposes a model that can serve as a base for protocol specification. The aim is to introduce a conceptual and complete description of basic streams of flow among entities in order to identify rules of data transfer. The resultant specification is a map over which a protocol can be superimposed.

Keywords: *Message sequence diagram, specification and description language, protocol, communication, conceptual description*

1. Introduction

A protocol can be defined as a set of rules governing the exchange of data between entities [14]. Communication Protocols are specified by using either formal or graphical notations, including LOTOS (Language of Temporal Ordering Specification), ESTELLE (Extended State Transition Language), CPN (Colored Petri-Nets), and SDL (Specification and Description Language) [12, 13, 14].

SDL is used extensively in telecommunication for development of software and hardware, *e.g.*, 3G, cellular phones, switches, WLANS, and Bluetooth devices. Description of a communication system in SDL includes structure (*e.g.*, block, process), communication signal flow, behavior, and abstract data types. Giridharan [10] summarizes some of the "salient" features of SDL as follows:

- well-defined concepts
- unambiguous, clear, precise, and concise specifications
- describes the structure and behavior of systems with mathematical precision
- high degree of visual clarity, object oriented concepts, clear interfaces, and abstraction mechanisms

Nevertheless, the resultant diagrams are fragmented and lack continuity in depicting the succession of events. This paper proposes a new diagrammatic methodology for protocol specification. It is based on the notion of flow of "primitive" things in a system with six stages: creation, release, transfer, arrival, acceptance, and processing. The aim is to introduce a conceptual and complete description of basic streams of flow

among entities and stages including "crossing points" that need protocols to provide rules of transfer.

Specifically, without loss of generality, we target SDL and its complementary Message Sequence Chart (MSC) diagrams that provide a description of requirements in communication systems in the form of message flows. Our proposed conceptual description is contrasted with sample diagrammatic models developed using the SDL/MSC systems.

Next, for the sake of making this paper self-contained, we briefly review some aspects of our adopted specification methodology that have been utilized in many applications [1, 2, 3, 4].

2. Flowthing Model

The Flowthing Model (FM) is a uniform method for representing things that flow, called *flowthings*. Flow in FM refers to the exclusive (*i.e.*, being in one and only one) transformation among six states (also called stages) of transfer, process, create, release, arrive, and accept. All other states are not generic states. For example, we may have stored created flowthings, stored processed flowthings, stored received flowthings, *etc.* Flowthings can be released but not transferred (*e.g.*, the channel is down), or arrived but not accepted.

The fundamental elements of FM are as follows.

Flowthing: A thing (*e.g.*, information, material, money, data) that has the capability of being created, released, transferred, arrived, accepted, and processed while flowing within and between systems.

A flow system (referred to as flowsystem) is depicted in Figure 1, showing the internal flows of a system with the six stages and transactions among them.

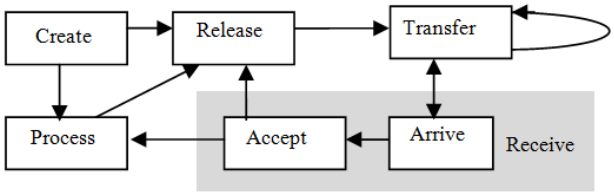


Figure 1. Flowsystem

Spheres and subspheres: Spheres and subspheres are the environments of the flowthing, such as a transistor, a battery, and a wire, which form the sphere of an electrical current, the flowthing.

Triggering: Triggering is a transformation (denoted by a dashed arrow) from one flow to another, *e.g.*, flow of electricity triggers the flow of air.

We will use *Receive* as a combined stage of *Arrive* and *Accepted* whenever arriving flowthings are always accepted.

3. Contrasting with the Specification and Description Language

In this section we provide a study case of SDL design patterns [8, 9] used for the development of distributed systems and communication protocols. "SDL design patterns

combine the traditional advantages of design patterns – reduced development effort, quality improvements, and orthogonal documentation – with the precision of a formal design language for pattern definition and pattern application" [5].

It is important to note that our aim is not to give descriptions of various notations, symbols, and shapes used in this example, which are necessary to understand the details of the given diagrammatic representation. Rather, we aim at demonstrating how extensive the efforts and fragmented the attempts to represent the problem under consideration. It is our aim to contrast these attempts with the integrated and complete FM-based diagrammatic description.

We look at the example from Webel, *et al.*, [5] that is developed to achieve reliability of a subsystem (A) that relies on data input from another subsystem (B).

> In this case, B has to be monitored in order to detect failure and to respond adequately. To provide a fail-safe or fail-operational state [6], a system must have the ability to detect system failures, which in this case can be done by using a Watchdog. This is a special component monitoring the operation of a system. The observed system has to send a periodic life-sign called Heartbeat to the Watchdog. If this life-sign fails to arrive at the Watchdog within a certain period, the Watchdog assumes a system failure and therefore moves the controlled system into a fail-safe or fail-operational state [5].

The so-called watchdog pattern describes a behavior that extends a given system. In an automatic safety device on trains, an operator must periodically press a button, and when the operator does not, the device assumes the operator is dead and stops the train, which leads the system to a fail-safe state.

Webel, *et al.*, [5] described this design problem in terms of Message Sequence Charts (MSCs) shown in Figure 2. According to ETSI [16], MSC diagrams provide a clear description of system communication in the form of message flows. MSCs and SDL descriptions should be regarded as different but complementary views of a system. SDL provides behavior descriptions of individual communicating entities, but there is no direct description of communication between several entities. By contrast, MSCs provide a clear description of system traces in the form of message flows… MSCs are … necessary in any protocol description.

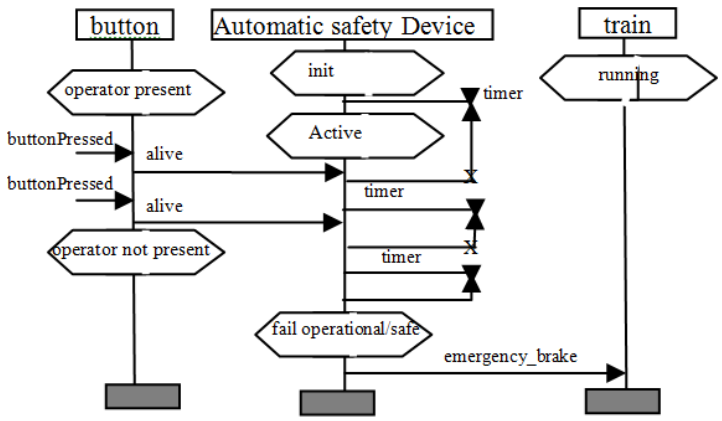


Figure 2. MSC Automatic Safety Device

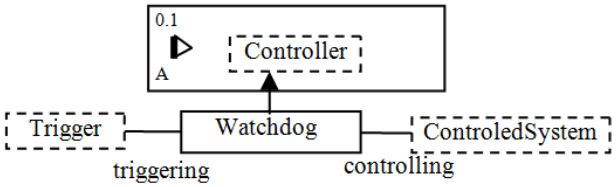


Figure 3. Structure of the Watchdog Design Solution

Further, Webel, *et al.*, [5] develop a "graphical representation of the structural aspects of the pattern's solution" (see Figure 3). This graphic representation includes:

- A trigger that periodically provides an "alive" signal
- A controller where watchdog functionality is to be added
- A controlled system

Finally, the SDL description is given as shown in Figure 4 and comprises an extended finite state machine.

Watchdog optionally refines Controller describes the watchdog functionality. The timer watchdogT is set for duration of safeInterval when triggered by a certain input from the context and restarted after a trigger (Figure centre). The duration safeInterval is the timeout interval after which the system changes to a fail-safe/fail-operational state [5].

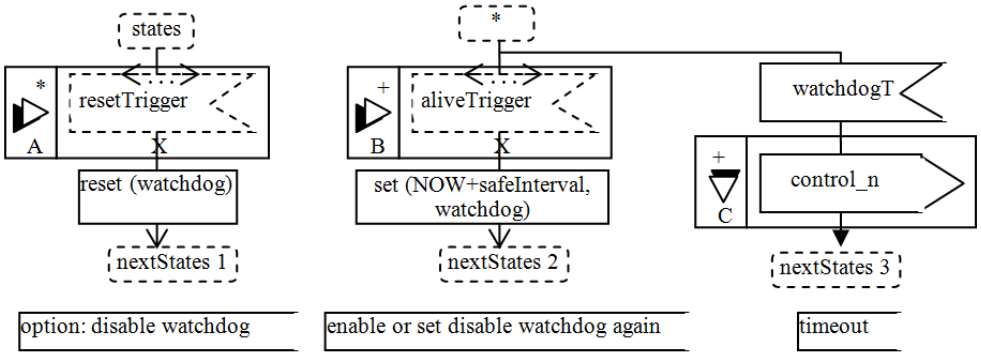


Figure 4. SDL Watchdog Design Pattern

Rather than go through the rest of the diagrams that describe various required protocols, we claim that development of a protocol for this problem can be based on a systematic and complete conceptual specification by using the FM description, as shown in Figure 5.

In Figure 5, the controller (middle box) has three sub-spheres, each represented by its rectangular box:

1. Time (*e.g.*, a clock) that creates (generates) time in the controller's context (circle 1 in Figure 5).
2. A button that when pushed triggers creation of time that flows to the watchdog (circle 2).
3. A train that represents the train's state from the controller's perspective (circle 3).

Note that, for simplicity's sake, when a subsphere has one flowsystem, we draw one rectangular box for both. For example, in Controller, the box labeled "button push" (circle 2) represents the button in the controller's sphere, and also the flowsystem "push". "Push" (pressing the button) is an action which is a flowthing that can be created, released, transferred… but, in this case, the action flowsystem includes only *create*.

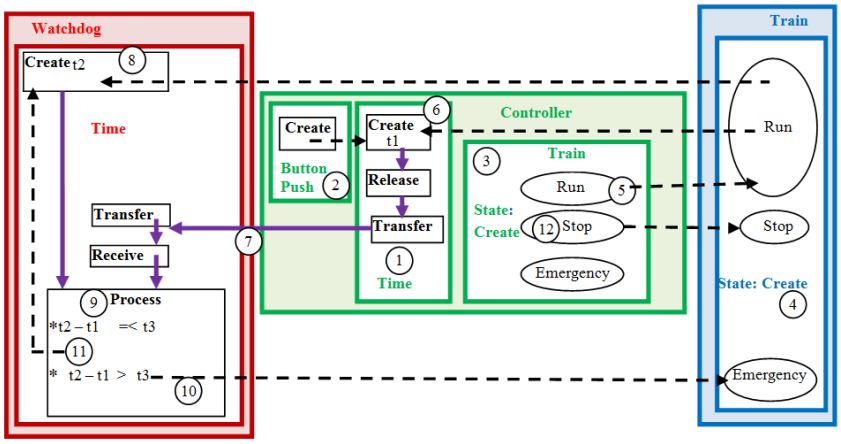


Figure 5. FM Basic Representation of the Watchdog Problem

Also, *states* are flowthings that can be created; hence, the state (of the train) sphere/flowsystem in Controller (circle 3) has three values: stopping, running, and emergency. On the right side of the figure, we see the Train's sphere (circle 4). It has one subsphere/flowsystem: *state* with values: stopping, running, and emergency. These are actual states of the real train, while inside Controller; the train's state is represented from the controller's perspective.

Assume that the train starts running after triggering by the controller (circle 5). Changing the train's condition triggers two actions:

1. Creating time t1 (circle 6) in the controller's time that flows to the Watchdog (circle 7).
2. Creating time t2 in the watchdog's time (e.g., clock – circle 8).

Both times t1 and t2 are processed (circle 9) according to the following:

(a) If the difference between t1 and t2 is greater than a certain time (t3), the emergency state of the train is triggered (circle 10).

(b) If the difference is less than or equal to t3, then t2 (the watchdog time) is triggered again (circle 11).

This process of reading the watchdog time t2 and comparing it with time t1 that flows from the controller's button pressing, continues until either it triggers emergency, or the train is stopped (circle 12).

The Basic FM diagram in Figure 5 can be complemented by different logical operators (*e.g.*, AND, OR …), synchronization symbols (forks, join,…), and constraints. It is characterized by continuity of different threads, making it possible for a tight grip on a series of superimposed protocol rules.

4. Scrutinizing Some Aspects of Message Sequence Charts

Lopez, *et al.*, [10] developed a methodology for the application of formal analysis techniques used in communication protocols to the analysis of cryptographic ones. They use a notation based on MSC, which can be translated into a generic SDL specification. Their main goal is to provide a notation for describing a formal specification for security systems. As a base for this methodology, they use the Message Sequence Chart (MSC) and its extension High-level MSC (HMSC). "With MSC we can specify elementary scenarios, and compose them to define more complex protocols" [10].

They illustrate the approach by specifying typical secure Web access to a databank portal via the Internet. Figure 6 shows the specification with two agents: "User Browser" and "Bank Portal".

> The "Bank Portal" agent has a secure web service via the HTTPS protocol, which provides authentication of the server. This is represented by an MSC reference called "https server auth" that implements the server authentication and the key exchange protocol defined in HTTPS. This MSC reference is defined in a package of standard protocols. The result of this scenario is authentication of the server and a session key called "https skey" [10].

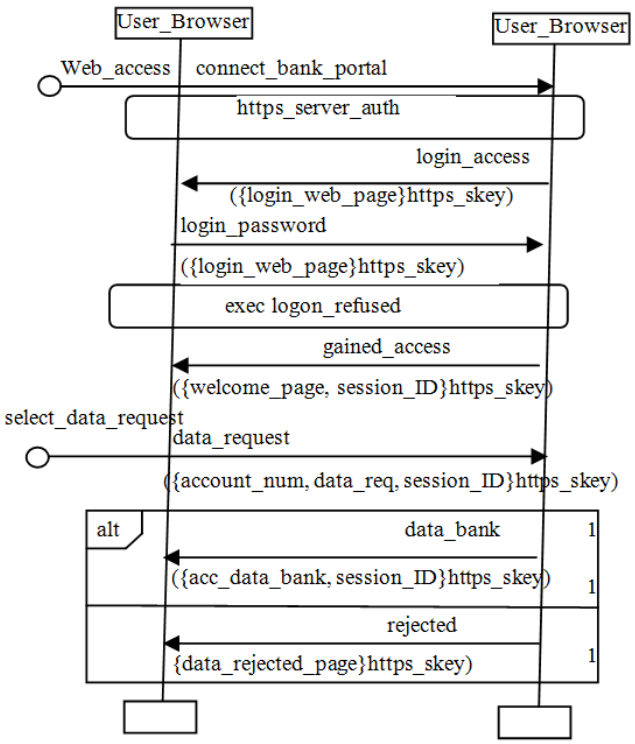


Figure 6. Security Scenario of User's Web Access to Bank (from [10])

Regardless of the details of such a typical approach, what we focus on here is the resulting conceptual picture that uses MSC with certain additional notations. Figure 6 mixes technical terminology with conceptual specification of message flow. Additionally, it is characterized by discontinuity: an arrow goes in, but it is not clear with what it connects. For example, at the bottom of Figure 6, we understand that

"data_request" flows from "User_Browser" to "User_Browser", and this leads to "rejected". It is not clear "who" rejects. It seems that the databank system is the system responsible for this. As we will see next, the FM representation clarifies the flows by specifying the life cycle of the message from the user, to the bank system, to the database system. MSC portrays such a process as discounted "shots", and the reader of the Figure must fill in between events him/herself.

By contrast, FM presents a conceptualization of reality with continuity: *logically sequential progression*. Its description is similar to a comic book, where a stream of events flows in a continuous fashion. Continuity is a necessary feature for designers. After producing a conceptual representation, a designer seeks connections through temporal continuity, causality, or some commonality such as presence in the same sphere (context).

Figure 7 shows the FM description that corresponds to Figure 6 as we understand it. The bank system involves the bank portal and a database system, as required by the succession of events that necessitates the flow of a data request to the database.

The FM specification starts with creating an access request (circle 1 in Figure 7) that flows to the bank portal (circle 2), where it is processed and sent to the authentication system (circle 3). For simplicity's sake, we do not draw the flowsystem in the authentication system; nevertheless, the reader can easily develop it because of the systematic definition of flowsystem that involves at most six stages.

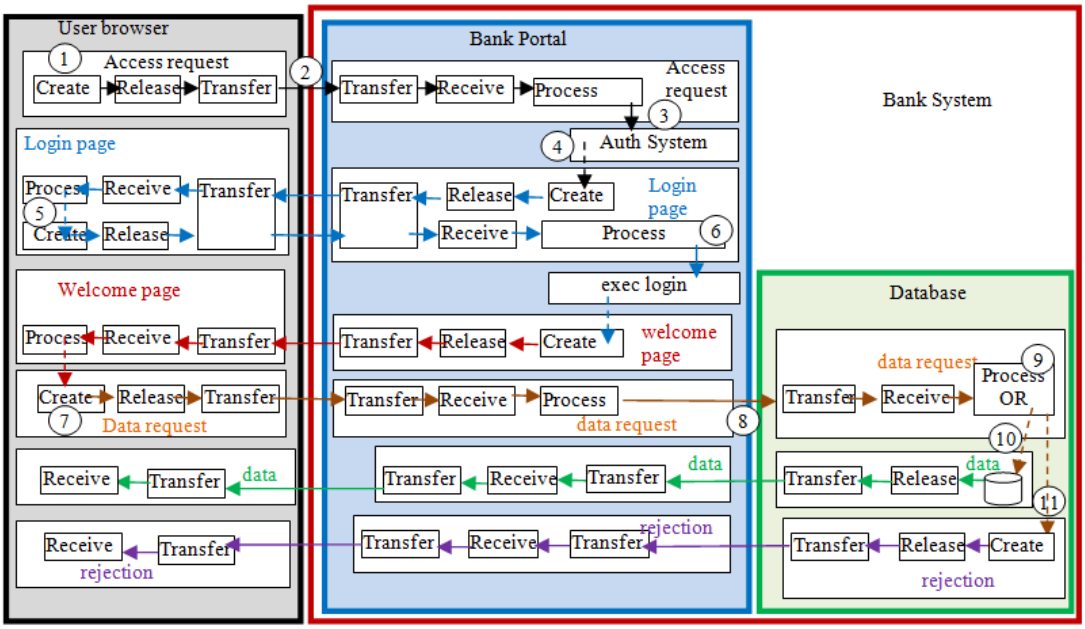


Figure 7. FM Representation that Corresponds to Figure 6

The authentication system permits (triggers – circle 4) the process of sending the login page to the user. The page is, of course, already created and stored in the system. "Creation" here means, conceptually, the appearance (emergence) of the page in the context of this request.

The Web page flows to the user, where the user generates (circle 5) a new Web page by entering information on the received page. The input information is sent to the exec login (circle 6), and so forth.

The total exchange looks like a comic-book story where continuous events unfold. Note that each type of flowthing (*e.g.*, Web access, Web page, data request, data, …) has its own flow stream that is connected with other streams by triggering.

The last flow includes a data request (circle 7) that is passed on (circle 8) to the database system where it is processed (circle 9) and then either (a) data are retrieved and sent to the user (circle 10), or (b) a rejection is created and sent to the user (circle 11).

5. Protocol Mechanisms

Different protocols are often composed of similar protocol mechanisms. In this section we explore the suitability of the FM representation for these mechanisms. One of these mechanisms is error control, where messages are sometimes lost. Protocols often utilize check sums to detect such errors. In this case an automatic repeat request mechanism usually initiates the retransmission of lost data. This mechanism relies on the transmission of an acknowledgment message by the receiver and on the detection of timeouts at the sender's end [15].

In the Stop-and-Wait scheme, the sender waits for an acknowledgment from the receiver before sending out the next data. If the sender does not receive the acknowledgment within a fixed time, it will retransmit the message. Dietterle [15] gives Figure 8 as an illustration of this type of scheme. Figure 9 shows an FM representation of the Stop-and-Wait scheme.

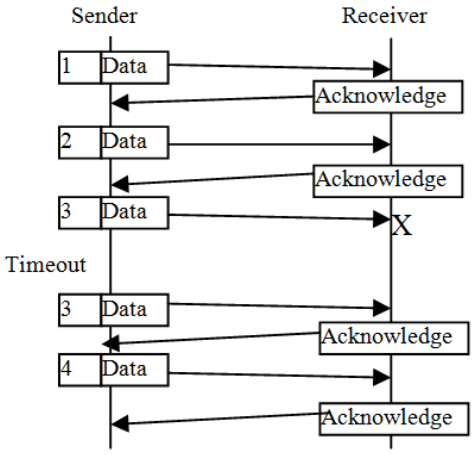


Figure 8. Illustration of Stop-and-Wait Scheme (from [15])

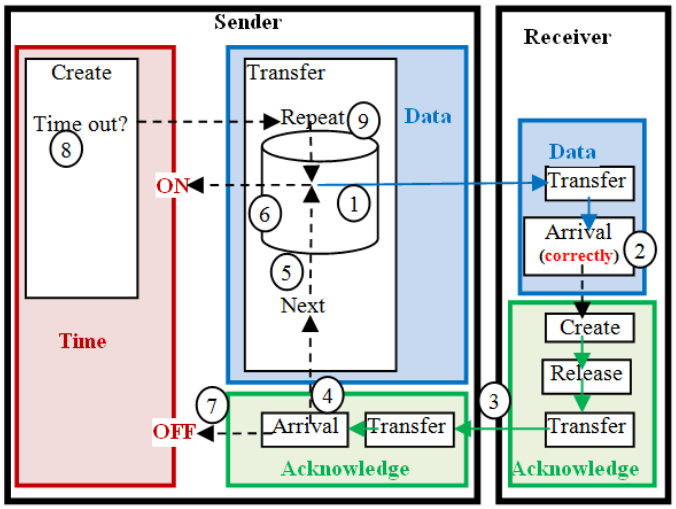


Figure 9. FM Representation of the Stop-and-Wait Scheme

Figure 9 is more extensive because it is more complete. The sender (left box) has three subspheres: data, acknowledgment, and time. In the transfer stage in the data flowsystem (again, since the subsphere has only one flowsystem, we represent the subsphere and its flowsystem with one box), data items to be transferred are retrieved (*e.g.*, from queue) and sent to the receiver (circle 1). The (correct) arrival of data at the receiver (circle 2) triggers the creation of an acknowledgment. The acknowledgment flows to the sender (circle 3), where it triggers (circle 4) the transfer of the next data item (circle 5). *Next* is a module in the transfer stage of the sender that causes the transferring of the next data item.

We notice that transferring data also triggers (activates – circle 6) a timer. We assume that the timer is initially at zero. Furthermore, the arrival of an acknowledgment sets OFF the timer (circle 7). We assume that OFF would also initialize the timer. Of course, ON and OFF can be drawn in a state flowsystem, as we did in the previous example. If the time is out (circle 8) then the module *repeat* (circle 1) in the transfer stage of data retransfers the data. The resulting description is a precise representation that is similar (in details) to electrical circuit schemata. It gives further justification for the viability of FM to be applied in protocol specifications.

Dietterle [15] also gives a diagrammatic illustration of a Go-Back-N scheme. It allows the sender to transmit data without receiving an acknowledgment up to a maximum number of not-yet-acknowledged data items. The receiver acknowledges the reception of all data items up to a sequence number given in the acknowledgment message. If a transmission error occurs and the acknowledgment from the receiver is not received in time, the sender would retransmit all data items starting from the first one that has not been acknowledged even if the receiver has already correctly received some of them [15]. Dietterle [15] presents an illustration similar to Figure 8 for this Go-Back-N scheme.

To further illustrate the FM representation, we simplify this second scheme by assuming that data items are sent in blocks of *fixed* size. If there is no acknowledgment of the arrival of the block, it is retransmitted. The resultant FM representation for this simplified Go-Back-N scheme is shown in Figure 10. It consists of the FM diagram of the previous scheme with *counters* added in the sender and receiver spheres. Assuming that the counter is initialized to zero, then each time a data item is sent, the counter is

incremented by one (circle 1). If the counter reaches some number n, then it triggers (circle 2) the timer as in the previous scheme. This means that n data items have been transferred and timing is kept from the time of transfer of the nth item. The receiver also has a counter that counts the arriving items (circle 3). If the number of correctly arrived data items is n, then:

- The counter is re-initialized to zero (circle 4).
- An acknowledgment is created (circle 5).

Note that the transfer stage in the counter has two modules, A and B, that are triggered independently. The acknowledgment then flows to the sender (circle 6). Upon arrival, the acknowledgment does the following:

- triggers sending the *next* n items (circle 7)
- triggers re-initialization of *time* to zero (circle 8)
- triggers counter re-initialization to zero (circle 9)

The whole process is then repeated.

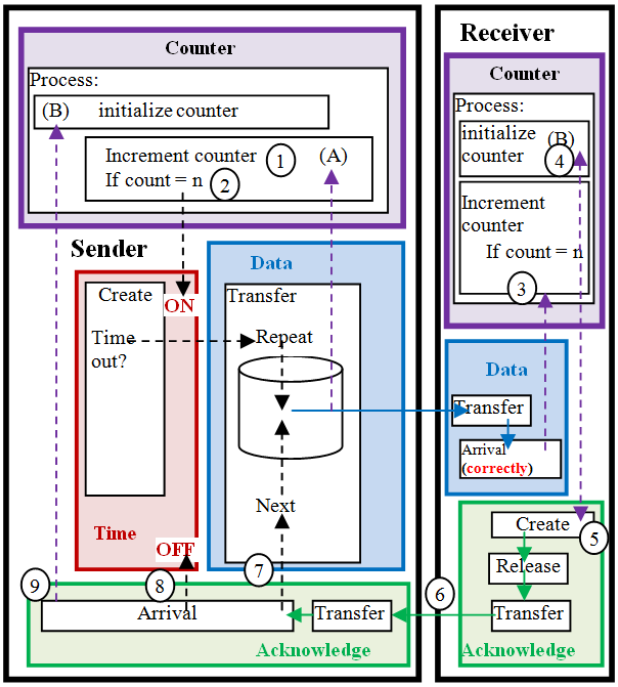


Figure 10. FM Representation of the Simplified Go-Back-N Scheme

6. Conclusion and Further Work

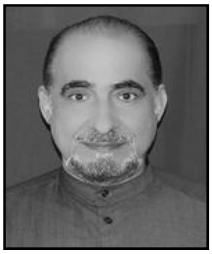
This paper proposes a new diagrammatic methodology as a base for protocol specification. The conceptual description is contrasted with SDL and its complementary Message Sequence Charts (MSC) diagrams that provide a description of requirements in communication systems. The comparison points to a viable and more systematic specification that can be achieved using the proposed method.

Two directions can be followed according to this preliminary conclusion. First, the FM description can be further formalized to serve as a conceptual description that forms a base for developing the specification of communication protocols using such tools as SDL. Additionally, it is possible to further develop FM to be a full specification and description method and language. Both of these options are currently being pursued.

References

[1] S. Al-Fedaghi, "A conceptual foundation for data loss prevention", Int. J. Digital Content Tech. Appl., vol. 5, no. 3, **(2011)**, doi:10.4156/jdcta.vol5.issue3.29, pp. 293-303.
[2] S. Al-Fedaghi, "States and conceptual modeling of software systems", Int. Rev. Comput. Software, (IRECOS), vol. 4, no. 6, **(2009)**, pp. 718-727.
[3] S. Al-Fedaghi, "System-based approach to software vulnerability", IEEE Symposium on Privacy and Security Applications (PSA-10), Minneapolis, USA **(2010)**. ftp://ftp.computer.org/press/outgoing/proceedings/SocialCom%202010/data/4211b072.pdf.
[4] S. Al-Fedaghi, "Pure conceptualization of computer programming instructions", Int. J. Adv. Comput. Tech. (IJACT), vol. 3, no. 9, **(2011)**, pp. 302 – 313.
[5] C. Webel, I. Fliege, A. Geraldy and R. Gotzhein, "Developing reliable systems with SDL design patterns and design components", In Proceedings of the Workshop on Integrated Reliability with Telecommunications and UML Languages (Rennes, France) **(2004)**, http://www.sdl-forum.org/issre04-witul/papers/witul04_developing_reliable_systems.pdf.
[6] H. Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Kluwer Academic Publisher, **(2007)**.
[7] B. Geppert, R. Gotzhein and F. Rößler, "Configuring communication protocols using SDL patterns", In SDL'97 - Time For Testing, Proceedings of the 8th SDL Forum, A. Cavalli and Q. Sara, Eds. Elsevier, Amsterdam, **(1997)**, pp. 523-538.
[8] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA **(1995)**.
[9] A. Giridharan, "SDL: A Protocol Specification Language, Department of Electrical Communication Engineering", Indian Institute of Science, Bangalore, India **(2012)**, http://pet.ece.iisc.ernet.in/course/E2223/sdl.pdf.
[10] J. Lopez, J. Ortega and J. Troya, "Applying SDL to formal analysis of security systems", In 11th International SDL Forum (Stuttgart, Germany) (SDL'03), Springer-Verlag: LNCS, vol. 2708, **(2003)**, pp. 300-317.
[11] J. Pärssinen, M. Turunen, J. Heinonen, N. von Knorring, A. Kvist and P. Jäppinen, "Protocol Engineering Concepts and Patterns using UML", **(1999)** November 25, http://edu.pegax.com/lib/exe/fetch.php?media=csa:pecp-uml.pdf.
[12] P. S. Kaliappan and H. Koenig, "An approach to synchronize UML-based design components for model-driven protocol development", In IEEE 34th Software Engineering Workshop (Limerick, Ireland, June 20-21, 2011), **(2011)**, ISBN: 978-0-7695-4627-8, pp. 27-35.
[13] S. Smith, A. Beaulieu and W. G. Phillips, "Modeling and verifying security protocols using UML 2", In IEEE International Systems Conference (SysCon), **(2011)**, pp. 72–79.
[14] W. Stallings, "Data and Computer Communications", 4th edition. MacMillan **(1995)**.
[15] D. Dietterle, "Efficient Protocol Design Flow for Embedded Systems", Brandenburg University of Technology **(2009)**, http://systems.ihp-microelectronics.com/uploads/downloads/diss_dietterle.pdf.
[16] ETSI, "Message Sequence Charts (MSC)", **(2012)**, http://www.etsi.org/WebSite/Technologies/MSC.aspx.

Author



Sabah Al-Fedaghi holds an MS and a PhD in computer science from the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, and a BS in Engineering Science from Arizona State University, Tempe. He has published two books and over 150 papers in journals and conferences on Software Engineering, Database Systems, Information Systems, Computer/information Privacy, Security and

assurance, Information Warfare, and Conceptual Modeling. He is an associate professor in the Computer Engineering Department, Kuwait University. He previously worked as a programmer at the Kuwait Oil Company and headed the Electrical and Computer Engineering Department (1991–1994) and the Computer Engineering Department (2000–2007).