Frames for protocol description*

SALLY WAGNER-DIBUZ

Central Research Institute for Physics
H—1525 Budapest, 114, P.O.B. 49., Hungary

1. Introduction

The increasing number of computer networks and their spreading application leads to the production of new computer network protocols. The specification, implementation, and test sequence generation for these protocols is a tiring work, which has to be supported by software tools. Expert system technology can be applied well in the field of communication protocols.

This paper deals with the application of frames for protocol description, which gives a formal description of the protocol. The transformation from other formal description methods into the protocol representation with frames can be given. The frame representation is a new method for protocol description, which provides a flexible and well-structured description of the protocol.

Frames are useful for representing the protocol development process too. Using a frame-based knowledge representation method data and procedural knowledge can be represented in the same way, with frames. However frames describing the protocol can be easily separated from the frames and demons representing the procedures of the protocol engineering process. This means that the protocol description with frames can be the knowledge base of an expert system for protocol engineering [WAG87], [TAR88].

In the paper examples are shown for the frame representation of protocols, and for the usage of the protocol description for the simulation of the communication between two transport entities. For demonstration the frame-based language FAIR developed in SZKI is used.

Section 2 contains a brief overview of communication protocols. In Section 3 we write about knowledge representation with frames. In Section 4 a protocol model is given, which can be represented with frames as it is described in Section 5. Section 6 summarizes the conclusions.

* Lecture presented at the 1st Finnish—Hungarian Workshop on Programming Languages and Software Tools, Szeged, Hungary, August 8—11, 1989.

2. Communication protocols

PROTOCOLS

The reference model defines seven layers in a communication network. Each entity in a layer communicates with the entities in the lower layer, with the entities in the upper layer and with the peer entities which are placed in the same layer but in other nodes. The peer entities communicate with each other according to the protocol (fig. 1.). The protocol defines the messages which can be sent during the communication, it determines the semantics of the communication and its time attributes. For the communication with the upper and the lower layers the entity uses service primitives (*SP*), and for the communication with the peer entity protocol data units (*PDU*) are used. Protocols have to be standardized so that computers produced by different producers are able to communicate with each other. There are several protocols in one layer especially in the application layer, where each application needs a new protocol.

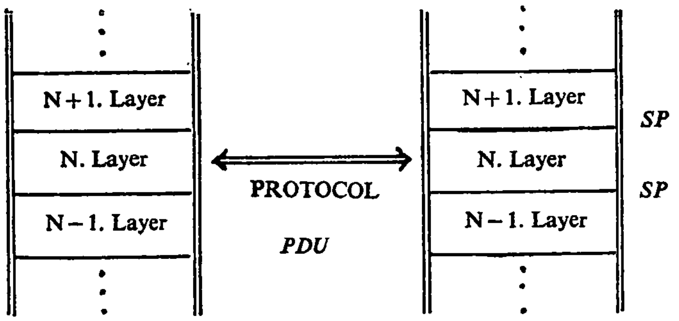

Fig. 1. The communication between the entities

PROTOCOL ENGINEERING

The protocol standard is a long and complicated description of the protocol in English. It defines everything that have to be stated about the protocol: the syntax of the messages (the bit-map), the possible responses to a given message, the parameters and their values, and the protocol mechanisms. The process in which the protocol standard is realized with an entity that is able to communicate with other entities using the protocol is called protocol engineering. (Fig. 2.) It consists of the following tasks:

— protocol specification,
— protocol verification,
— protocol implementation,
— conformance testing.

All of these activities need tiring and mostly manual work. There is a great need for software tools which support protocol engineering. Formal methods for protocol description are needed for these software tools. These formal descriptions give a more exact and clearly arranged description of the protocol, which is called the

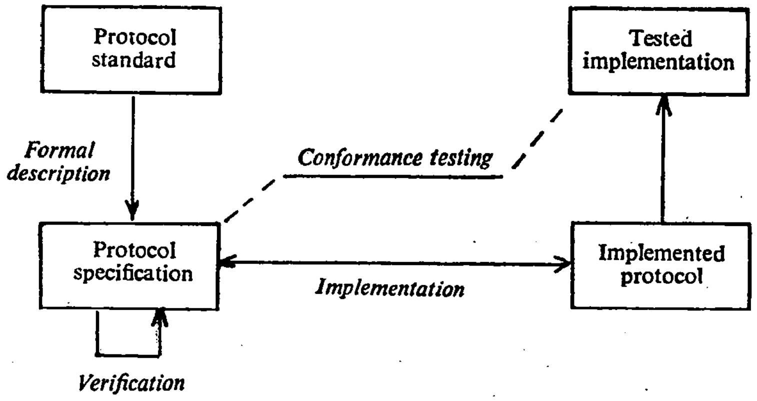

Fig. 2. Protocol engineering

protocol specification. It is easier to understand the protocol operation from the protocol specification and easier to implement the protocol in a programming language using the protocol specification. The testing of the implemented protocol is also based on the protocol specification.

3. Knowledge representation with frames

Frame—based systems usually treat information on three different levels:

— frames,
— frame—structures, and
— worlds.

Generally a frame can be considered as a structured symbolic model of a concept. Actually it is a (usually ordered) set of arguments, called slots, each of which is used to represent a property of the concept to be modelled. Every slot may have a value, which can be another frame, or an expression in some formal language.

In order to cope with the complexity of real world concepts meta—information can be associated to any part of a frame:

— to the frames,
— to the slots, and
— to the values.

The meta—information is also formalised in frames, these are called meta—frames, and the pieces of meta—information are stored in the slots of the meta—frames. This information can be used to define the range of the slot values, demons for describing the protocol activities, new frames for storing further and deeper information about the slot values, questions for filling in the slots with a given protocol etc. In this paper we use meta—frames only for describing range of values, and demons, which specify the "active" part of the program.

Frame—structures are used to represent the relations among the concepts modelled by the frames. This means that if certain slots are not found in a given frame, then search is made by the system along the relations to see if the slot and its value are contained in another frame accessible from the present one along such relations. If so then the slot and its value will be inherited by the original frame. In our example we shall use only the "is_a" relation. If two frames are in "is_a" relation it means that the first will inherit all lacking information from the other one.

Worlds are groups of frames, which can represent several things, like hypothetical alternatives, processes changing with time etc.

THE FAIR

FAIR (Frames in Artificial Intelligence Representation of knowledge) is a frame—based system integrated with MPROLOG. The form of a frame in the FAIR language is the following:

```
frame: name;
      slot_1: value_11, value_12, ..., value_1n$_1$;
      slot_2: value_21, value_22, ..., value_2n$_2$;
      ............................................................
      slot_m: value_m1, value_m2, ..., value_mn$_m$;
end.
```

The language contains several predicates for manipulating the frames. Here we enumerate only those predicates which are used in the example in Section 5:

create_frame	crf
access_value	acv
find	

The interested reader can find further details about frame—based knowledge representation in [BAR86] and in [ECS88], and about the FAIR language in [ECS 89].

4. A model for protocols

In this paper we use the abstract state machine (ASM) model of the protocol. The model is defined by seven sets:

$$(S, I, O, V, A, P, T)$$

where
S: is the finite set of states.
I: is the finite set of the incoming messages of the protocol.
O: is the finite set of the outgoing messages of the protocol.
V: is the finite set of variables, which can be divided into two distinct sets:
— local variables
— global variables.
A: is the finite set of actions, there are local actions concerning local variables, and global actions concerning all the variables.
P: is the finite set of predicates, which map the variables into the values true or false. There are three subsets of predicates:

— controlled,
— settable,
— non-controlled.

T: is the finite set of transitions, a transition is defined by the following:

$$(s, i, p, a, o, s')$$

which means that the automaton in the state "*s*" receives an input "*i*", and if predicate "*p*" is true it executes action "*a*", sends an output "*o*" and goes into state "*s'*".

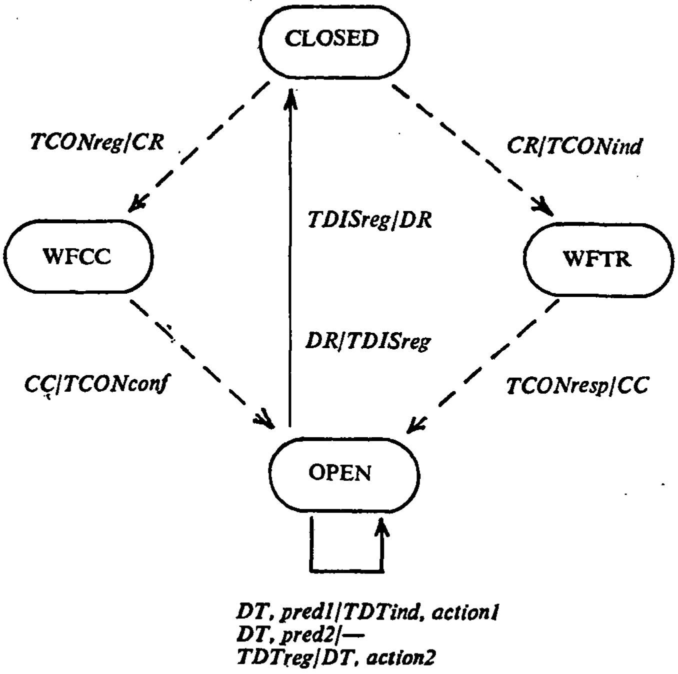

Fig. 3. Simplified state transition graph of transport protocol class 0

Fig. 3 shows the simplified state transition graph of the transport protocol class 0. Its ASM model is the following:

S = { CLOSED, WFCC, WFTR, OPEN }
I = { *TCONreq, TCONresp, TDISreq, TDTreq, CR, CC, DT, DR* }
O = { *TCONind, TCONconf, TDISind, TDTind, CR, CC, DT, DR* }
Vlocal = { SZEGM, ... }
Vglobal = { $\emptyset$ }
P = {*pred*1, *pred*2, ...},
A = {*action*1, *action*2, ...},
T = { (CLOSED, *TCONreq, CR*, WFCC), ... }

(The elements of the transitions which do not have values can be omitted.)

This is a very simple protocol, but the ASM model is capable of describing more complex protocols. The ASM model can be the basic model of the protocol in an expert system for protocol engineering. In the next section we demonstrate how this model can be stored in a frame structure.

5. A frame based approach to protocol description

Frames describing the ASM model can be divided into three groups:

— atomic,
— complex,
— pattern.

An atomic frame contains a state transition. The form of the generic frame for a transition is the following:

```
frame: transition;
     start_state: ;
     input: ;
     predicate: ;
     action: ;
     output: ;
          meta_slot: output_meta;
               if_accessed_demon: output_demon;
          meta_end
     end_state: ;
          meta_slot: end_state_meta;
               if_accessed_demon: end_state_demon;
          meta_end
end.
```

The slots of the meta—frames contain demons which are activated when the value of the slot is accessed during the operation of the program. These demons can be for instance procedures writing out the output and the end—state on the display. The frame of a transport protocol state transition is the following:

```
frame: edge5 ;
     is_a: transition;
     start_state: open;
     input: dt;
     predicate: ;
          meta_slot: ;
               if_accessed_demon: pred1;
          meta_end
     action: action1;
     output: tdtind;
     end_state: open;
end.
```

Complex frames store the information about the elements of the sets defining the ASM model. The following is the generic frame for the inputs, and the next frame represents the inputs of our transport protocol example:

```
frame: inputs;
    asp_from_upper_layer: ;
    asp_from_lower_layer: ;
    pdu: ;
end.
frame: trans_inputs;
    asp_from_upper_layer: tconreq, tconresp, tdisreq, tdtreq;
    pdu: cr, cc, dr, dt;
end.
```

The bit format of the messages, their parameters with their values, can also be represented by complex frames.

Pattern frames store test suits, or subgraphs of the state transition graph. This information is available also in the atomic frames in the description of the transitions. Such redundant information is useful in the procedures of protocol engineering.

```
frame: pattern;
    states: ;
    inputs: ;
    outputs: ;
    transitions: ;
    represented_actions: ;
    fixed_variables: ;
end.
```

This is the example of a pattern frame representing the connection establishment phase of the transport protocol:

```
frame: connection_establishment;
    is_a: pattern;
    states: closed, wfcc, open;
    inputs: tconreq, cc;
    outputs: cr, tconconf;
    transitions: edge1, edge2;
end.
```

COMMUNICATION SIMULATION

The communication of two transport protocol entities can also be described with frames. For this all the information stored about the protocol in the frames have to be put in work. This can be done by demons, which are procedures written in Prolog in our case, as the FAIR language uses PROLOG. Two frames represent the protocol entities, and one more frame represents the channel connecting the entities. They are the following:

```
frame: transport_entity_1;
    actual_state: ;
    input: ;
        meta_slot: ;
            range: tconreq, tconresp, tdisreq, tdtreq, cr, cc, dr, dt;
            if_added_demon: state_transition;
        meta_end
end.
frame: channel;
    message: ;
        meta_slot: ;
            if_added_demon: transmit;
        meta_end
end.
```

The state_transition demon is activated when the input slot of the transport_entity frame gets a new value. This demon searches through the state transition graph of the protocol (*frames edge_1,..., edge_n*) to find the transition, which the protocol presents in this part of its operation. The frame: *transport_entity_i* tells the actual state of the protocol and the message it receives from the other entity. The appropriate transition can be found using this information. If such a frame is found, then its predicate has to be examined. If it succeeds, the demon in the action slot of the frame representing the transition is run, and the output message and the new state of the protocol can be found in the frame. If the examined predicate doesn't succeed, the search through the transitions is continued, as there must be a transition which has the same start_state, input, and which predicate succeeds. The following PROLOG statement is the activity part of the *state_transition* demon:

```
state_transition_activity: —
    acv(actual_entity,name;N),
    acv(N,input,I),
    acv(N,actual_state,S),
    find(Frame,[start_state, input, predicate], [S, I, succeed]),
    run(Frame, action),
    acv(Frame, output, O),
    acv(Frame, end_state, E),
    crf(N, actual_state, E).
```

6. Conclusions

In this paper we described a method for representing a protocol in a knowledge base. The frame—based language can be used in a natural manner for specification of protocol data units and the communication itself. This kind of protocol specification is:

— Comprehensible even by non-expert user.
— Executable directly by machines.
— It can be used as a protocol description for protocol engineering methods.

The examples given in the paper show some further advantages:

— Every aspect of the protocol can be described as deeply in the hierarchy of frames as it is needed by the user.

— The data types and the control mechanisms of the protocol can be defined by frames in an identical way.

— It gives a runnable description of the protocol, the operation of the protocol can be easily simulated and demonstrated.

References

[BAR86] Barr, A., Feigenbaum, E. A., The handbook of artificial intelligence I—III., Heuris-TECH Press, Stanford, 1986.

[ECS88] Ecsedi-Tóth, P., A frame-based approach to protocol engineering, Technical report of SZKI, 1988.

[ECS89] Ecsedi-Tóth, P., Wagner, P. A., "FAIR, a frame-based system integrated with MProlog", Proc. of Expert Systems Conf. Visegrad, 1989.

[TAR88] Tarnay, K., Wagner-Dibuz, S., "A protocol consultant", Proc. of DECUS Europe Symposium, Cannes, Sept. 1988.

[WAG87] Wagner-Dibuz, S., Protocol consultant, an expert system for protocol engineering, KFKI report 28—M, Budapest, 1987.