18

A Comparison between the Service Addition language SAL and the ITU-T recommendation Z.120

S. Somé, R. Dssouli, J. Vaucher
Département d'Informatique et de Recherche Opérationnelle,
Université de Montréal.
C.P.6128, succ CENTRE VILLE Montreal, P.Q. Canada H3C 3J7
email: (some,dssouli,vaucher)@iro.umontreal.ca

Abstract

Message Sequence Charts describe the behavior of system components by showing the message exchange between these components and their environment. The ITU-T Recommendation Z.120 is a standard for Message Sequence Charts, however before this standardization, Message Sequence Charts have been used for many purposes. One of these uses is illustrated by the Service Addition Language SAL and the design support environment SDE. SAL is a Message Sequence Charts based language used for the full definition of systems in many large scale industrial projects. This paper provides a comparison on Z.120 and SAL descriptive power, in order to emphasize some aspects needed for the industrial usage of Message Sequence Charts.

Keywords

Comparative analyses of FDTs, Message Sequence Charts.

1 INTRODUCTION

Communication systems need careful design in order to assure their completeness and correctness. Efficient systems design is achieved by the use of formal specification languages which allow unambiguous representation of systems and which provide rules for design validation. Many specification languages have been developed and some of them like SDL (W.P.X/3, 1992), Estelle and Lotos are international standards. In SDL, systems are specified by describing the function of their control entities by Extended Finite-State Machines. SDL descriptions are often complex and it is difficult to perceive the complete behavior of the system from the set of Extended Finite State-Machines constituting them. Message Sequence Charts (MSCs) which describe system traces are a convenient and a simple way to represent a system behavior. A system global function can be described by a set of MSCs each representing a partial system behavior on describing the interactions among the system components and their environment. The Service Addition Language (SAL) (Ichikawa, 1986) describes systems by Message Sequence Charts and a full SDL

G. V. Bochmann et al. (eds.), *Formal Description Techniques VIII*

system definition is generated from such descriptions by using the System Design Environment (SDE) (Kato, 1990). Several industrial systems have been designed using SAL. One of them is an integrated key telephone system (IKTS) a PBX accommodating about 160 digital key telephones and 50 analog subscriber lines (Nishikado, 1987). Many other uses for MSC have been done by CCITT Study Groups and within industry. In order to provide an unique convention for MSCs and thus to make it possible to develop tool support for them, to exchange MSCs between different tools, to ease the mapping to and from SDL specifications and to harmonize their use, the recommendation Z.120 (ITU-T, 1993) which defines an abstract, textual graphical syntax and informal semantics for MSCs have been adopted. This paper compares SAL a language based on Message Sequence Charts and the proposed standard for Message Sequence Charts. This comparison aims at underlining how the concept of Message Sequence Charts have been adapted to an industrial context, and thus, to give a slight indication on a possible addition to Z.120.

The paper is organized as follow. Section 2 briefly reviews the recommendation Z.120, section 3 introduces the Service Addition Language, section 4 presents the comparison between SAL and Z.120, and section 5 concludes this paper.

2 THE RECOMMENDATION Z.120

The recommendation Z.120 defines a Message Sequence Chart as the description of the flow of messages exchanged between system components and their environment. It defines a textual and a graphical representation for MSCs.

```
msc basic concepts; inst Initiator:
  process ISAP_Manager_Ini, Responder;
  instance Initiator:
     process ISAP_Manager_Ini;
  condition Disconnected shared all;
     in ICONreq from env;
     out Icon to Responder;
     action setting counter;
     set T(5);
     condition wait;
     timeout T;
     out IDISind to env;
     condition disconnected;
  endinstance;
  instance Responder;
  condition Disconnected shared all;
     in ICON from Initiator;
     out ICONind to env;
     condition wait;
  endinstance;
endmsc;
```

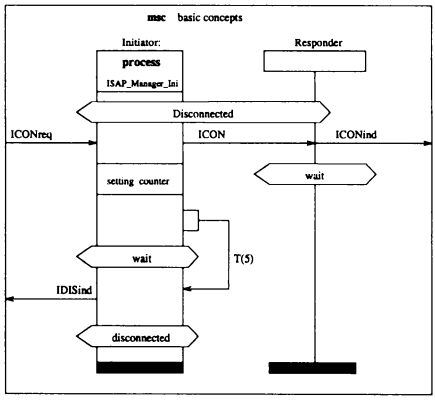

Figure 1 An example of Message Sequence Chart

A Message Sequence Chart describes a partial behavior of a system by the signal flow

between its instances, that are interacting component of the system. An instance may thus represent a process, block or service. Message Sequence Charts may also indicate actions deriving from the signals and may be structured according to global conditions. Figure 1 shows an example of MSC in Z.120 textual and graphical representations, borrowed from Z.120. This example gives a general presentation of MSCs and introduces some of Z.120 basic concepts.

An instance behavior is described as a serie of events. The different kind of events are summarized in Figure 2.

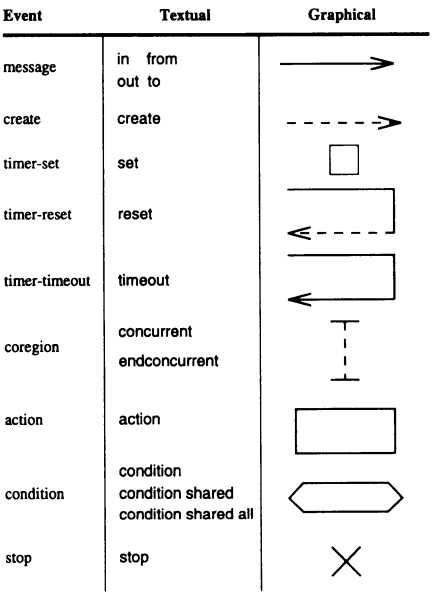

Figure 2 Keywords and symbols for events. This Figure shows the different event kind, the corresponding keywords used in the textual syntax and the symbols used in the graphical syntax to represent them.

Z.120 also defines Sub-Message Sequence Chart (*submsc*), as a way to refine instances description.

3 THE SERVICE ADDITION LANGUAGE SAL

The Service Addition Language (SAL) has been developed by the NTT Software Laboratories. Its allows service specification by message sequences. SAL descriptions are used as input by a software development environment SDE (System Design Environment), that generates SDL descriptions after making sure of their correctness. SAL is a textual lan-

guage, however there is graphical language equivalent to it, the Graphical Service Addition Language GSAL (Itoh, 1988) and SDE includes tools permitting conversion of graphical descriptions written in GSAL to SAL.

SAL basic concepts are **object**, **process**, **multilogue**, **event** and **message**.

- An *object* constitutes the representation of the system. SAL objects play a role similar to that of system/block in SDL. Objects are scope units and include other scope units like processes, sub-objects and channels. Objects are connected to their external environment by interaction points.
- A *process* is a behavioral entity of the system. Processes can be *real* or *imaginary*. A real process represents a system process generated in the SDL system description. An imaginary process describes behaviors not present at the SDL and program levels such as user behaviors. Both real and imaginary processes can be *deterministics* or *non-deterministics*.
- *Multilogues* describe (partial) behavior of objects. A multilogue consists of a set of *unilogues*. Each of these unilogues consists of events occurring within a process. Multilogues can be *named*. SAL *named* multilogues *macros*, *statics* and *sub-objects* can be called by processes using *call* and *semi-call* events.
- *Events* are internal operations done by processes. Events are classified into two categories. The first category consists of *true events* like *send*, *receive* and *primitive object call*. Other events are *pseudo events*. Pseudo events includes multilogue calls (*call* and *semi-call* events), multilogue event handling (*compound* event) and process control events (*if*, *loop*, *while*, *exit* and *null* events).
  Send and *receive* events enable processes to communicate with each other or with their environment. Communication uses channels that transmit in first in first out order.
  A *call* event is used to call a multilogue or a primitive object. Parameters can be used when the call event does not refer to statics. Macro and statics calls result in an expansion of the corresponding multilogue definition in the calling one. This expansion is done during the SAL compilation for macro calls and during the program generation for static calls. Sub-objects calls have a meaning similar to that of procedure calls in traditional programming languages.
  The *Semi-call* event is a variation of *call* event. Semi-call is available for statics and sub-objects only. The difference between these two events is their expansion mechanism. The expansion of a semi-call consists of replacing the semi-call by the named multilogue called unilogue definition corresponding to the process which has executed the semicall. In contrast, a call event is expanded by replacing it by the whole definition of the named multilogue called.
  Compound events are used to group events or unilogues into a single event. It is possible to specify choices in a compound event by using the merge operator "|".
  Process control events are an addition to SAL in order to increase the expressive power of the language and thus to ease complex system description.
- Communication between processes may result in *messages* exchange. A message being a piece of data that may be structured.

Figure 3 shows a service description in SAL, from (Ichikawa, 1991). This service is provided by an object named *MessageTransfer*.

```
object MessageTransfer()
{ p = +(pwd:password) .if(Valid
        (password))
      .loop { p = +(mes:m)
              -q(mes:m);
              q = +p(mes:m)
              -(mes:m); }
      +(Quit:) -q() { q = +p() ;};
|
p = +(pwd:password).if(Invalid
                       (password)) -r();
r = +p() -(IllegalAccess:);
}

%%
```

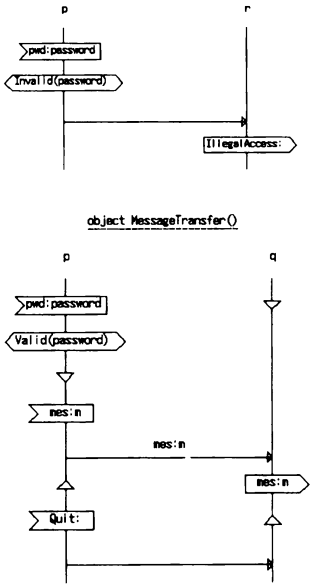

Figure 3 A message transfer service. This example shows the use of the SAL *merge* operator and also that of process control events *if* and *loop*. *Send* and *receive* events are represented respectively by *-p(mess)* and *+p(mess)* where *p* represents the second process involved in the communication, and *mess* the message exchanged.

4 COMPARISON BETWEEN SAL AND Z.120

This comparison between Z.120 and SAL is done in two parts. In the first part, SAL is examined through the concepts of Z.120, and in the second part, the concepts of SAL not present in Z.120 are analyzed.

4.1 SAL through recommendation Z.120

System description

Systems described by both Z.120 and SAL are related to SDL systems but SAL is closer to this language since Z.120 is not intended to be used only for SDL systems and there is direct generation of SDL descriptions from SAL. A system is described in Z.120 by a Message Sequence Chart document constituted by instances. Whereas in Z.120, an instance can describe any interacting entity, a SAL instance represents an SDL-process only. Blocks and systems are represented in SAL by the **object** concept.

Modularization concepts are important in describing complex systems. Z.120 solely modularization concept is *Sub-Message Sequence Chart* while SAL includes named multilogues (*Macros*, *Statics* and *Sub-objects*). These modularization concepts are not equivalents. Indeed, the expansion mechanism used for Z.120 submsc results in a complete

replacement of an instance by a submsc bound to it, and this kind of substitution is not defined in SAL. *Macros*, *statics* and *sub-objects* are expanded by adding their definition to the existing chart rather than replacing the instance which refers to them.

Instance description

Systems in both Z.120 and SAL are made by instances which interact with each other and with their environment. Instances in SAL are restricted to processes while they can represent any interacting entity in Z.120. The total time ordering of events along each instance is the same in SAL and Z.120 (in the absence of concurrent regions). In both Z.120 and SAL the essential characteristic of an instance declaration is its event description.

1. Message
   The semantics of a message event is essentially the same in SAL and Z.120 although there are some differences. In Z.120, a message must have an identification and may include a message instance name and parameters. Message identifications correspond to SDL signal names and parameters to data carried by these signals. Neither message names nor message instances are used in SAL. Also Z.120, differently from SAL, allows message overtaking by which signals may be consumed in a different order from their input.
2. Create and stop
   There is no event in SAL corresponding to Z.120 **create** used for the dynamic creation of processes. The Z.120 **stop** event corresponds to SAL event **exit**.
3. Timer
   Z.120 includes constructs for timer **set/timeout** and **set/reset**. There is no such SAL construct; instead alternative methods may be used.
4. Action
   There is a full correspondence between SAL primitive object *call* event and Z.120 *action* event. Each of these events allow the statement of internal activity by an instance.
5. Condition
   Conditions are used in Z.120 to express system states. A condition can refer to all the states of a MSC (global condition) or to a subset eventually of only one of them (non-global condition). Conditions may be used in Z.120, as syntactic separators or as composition and decomposition places, of Message Sequence Charts.
   As shown in (Ichikawa, 1987) global states can be represented in SAL using static multilogues considering that an object global state is defined by the sequence of events which occurs in the processes of this object. SAL global state representation using statics is however different from Z.120 conditions. The syntactical separation using Z.120 conditions is well reflected by SAL statics but not the composition and decomposition role. The main reason is that initial and final states can not be described using SAL statics.

4.2 SAL concepts not covered by Z.120

Concepts for process flow control, and advanced structuring concepts are defined in SAL but not in Z.120.

SAL includes events for process flow control such as **if**, **other**, **loop** and **while**. **If** is a conditional event execution, **other** may be used to specify alternatives, **while** and **loop**

are iterative execution events. For instance the *Message Transfer* example described in Figure 3 illustrates the use of some of the process flow control events.

Z.120 does not define similar events for process flow control but these events may be simulated by using conditions. Figure 4.2 shows a representation of the *Message Transfer* example in Z.120. The SAL **if** event can be represented implicitly in Z.120 by several charts

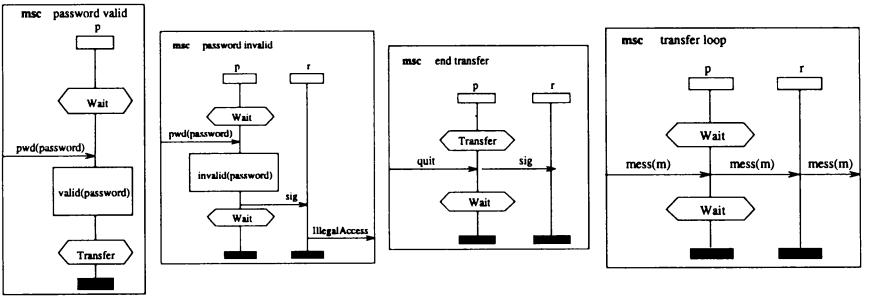

Figure 4 The Message Transfer example using Z.120

but whereas boolean conditions are formally specified in SAL, only an informal statement is possible in Z.120. The SAL repetition events can also be represented in Z.120 but not explicitly by using systems states.

The SAL alternative operator allows the specification of branches in *multilogues*. The specification of branches in Z.120 may be done only by describing several charts.

SAL named multilogue **macro** and **sub-object** parameters can be processes and a named multilogue calling process may be represented in its definition by the symbol $. This allows for the call of these multilogues by any process, the replacement of the $ sign by the calling process name being done at the expansion time by the SAL compiler. The $ sign and the possibility to have process names as parameters permits to define more reusable components in SAL. Such kind of definition is not possible in Z.120.

The absence of process flow control events and advanced structuring concepts in the Z.120, as well as the inclusion of conditions to the SAL, create a difference on the way of using them.This difference appears on comparing the Inres (Hogrefe, 1991) service in SAL (see (Somé, 1993)) and the same service description using Z.120 (W.P.X/3, 1991). Whereas descriptions in Z.120 proceed from a state based approach, SAL is a procedural language and no assumption is made about states when using it.

5 CONCLUSION

The recommendation Z.120 and the Service Addition Language (SAL) are both based on Message Sequence Charts (MSCs), and therefore the underlying concepts of these two languages are broadly the same. However many differences exist between SAL and Z.120 at the system description level. More important differences founded by this study is the inclusion of capabilities for process flow control, and structuring in SAL. These additional concepts to Message Sequence Charts have shown to be very useful for large scale industrial projects, as they allow to tackle complexity and provide tools for design

reusability. The recommendation Z.120 also includes a feature that could benefit SAL. The concept of condition, useful for composing and decomposing Message Sequence Charts. Extension of SAL and Z.120 description techniques including these useful features will certainly improve their capability for system specification.

It should be noted that Z.120 purpose goes beyond that of SAL. However using Message Sequence Charts for requirement representation and system design is becoming one of their more common uses as stressed by several system design methods (Rumbaugh, 1991),(Jacobson, 1992).

ACKNOWLEDGMENT

This work was financially supported by the Nippon Telephone and Telegraph (NTT). The authors wish to express their gratitude to Dr. H. Ichikawa, Dr. Y. Hirakawa, Dr. N. Miyake and all the other members of the Service Software research group of NTT.

REFERENCES

Hogrefe,D.(1991) *OSI formal specification case study: the Inres protocol and service, revised*, Technical Report, University of Bern.

Ichikawa,H.(1986) et al. *Protocol-Oriented Service Specification and Their Transformation into CCITT Specification and Description Language*, Trans. IECE Japan, vol E69, no 4, pp 524-535, April 1986.

Ichikawa,H.(1987) et al. *Communication Software Management with Verification and Transformation of Protocol - Oriented Specifications*, GLOBECOM'87, pp 651-655.

Ichikawa,H.(1991) et al. *SDE: Incremental Specification and Development of Communications Software*, IEEE Trans. On Computers, vol 40, no 4.

Itoh,M.(1988) et al. *A Graphic Language for Telecommunication Services Based on the Message Sequence Chart*, SIG Report on Software Engineering, vol 63, no 2.

ITU-T.(1993) *Message Sequence Chart (MSC). ITU-T Recommendation Z.120.* International Telecommunication Union.

Jacobson,I.(1992) et al. *Object-Oriented Software Engineering - A Use Case Drive Approach*, Addison-Wesley.

Kato,J.(1990) et al. *Support System for Communications Service Enhancement: SDE*, NTT REVIEW vol.2 no.2.

Nishikado,I.(1987) et al. *Integrated key telephone system with digital bus architecture*, Rev. Elec. Commun. Labs. vol 35, no 6, pp 657-664.

Rumbaugh,J.(1991) et al. *Object-Oriented Modelling and Design*, Prentice-Hall.

Somé,S.(1993) *Comparing the Service Specification language SAL and the CCITT draft recommendation Z.120 on Message Sequence Charts*, NTT Technical Report 8760.

W.P.X/3.(1992) *Recommendation Z.100 - The CCITT Specification and Description Language (SDL).* International Telegraph and Telephone Consultative Committee Geneva.

W.P.X/3.(1991) *Appendix I to Recommendation Z.100 - SDL Methodology Guidelines*, International Telegraph and Telephone Consultative Committee December 1991.