RELATIONSHIP BETWEEN SDL PROCESSES AND MESSAGE SEQUENCE CHARTS

Jie Lian and Vilas Wuwongse
Computer Science and Information Management Program
School of Advanced Technologies
Asian Institute of Technology

Key words: Extended Finite State Machine, MSC, Regular Expression, SDL

Abstract: Specification and Description Language (SDL) and Message Sequence Chart (MSC), two widely used system engineering modeling languages, specify and describe different aspects of real time systems. Since MSC is always combined with SDL in system specifications and descriptions, their interrelationship is very important and deserves special attention. Two new mathematical models for precise description of SDL Processes and MSC are used to analyze their interrelationship.

1. INTRODUCTION

During the last two decades, telecommunication has employed some of the world's largest and most complex computer-based real time systems, which operate in real time with time constraint [1] and put heavy demands on system engineers. Therefore one major concern is the management of complexities, for which SDL [4, 5] and MSC [6, 7] are widely used.

SDL has been developed for two purposes of a precise specification/description of functional properties and the realization of required systems. A SDL system consists of two basic structures: Block and Process, describing system physical components and behaviors, respectively.

MSC, a popular graphical language for the visualization of selected system runs (traces) within a communication system, can also be viewed as a

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35581-8_35

T. Yongchareon et al. (eds.), *Intelligence in Networks*

special tracing language which mainly concentrates on message interchange by communicating entities and their environment [11]. MSC, combined with SDL or any other language, is now used at nearly all stages of the system development process.

Since SDL Processes and MSC can be used to describe system behaviors, they should be interrelated. A detailed analysis of this relationship will not only provide a more precise and powerful interpretation of system specifications and descriptions, but also overcome disadvantages, inherent in these models.

An analysis of this relationship demands two mathematical models. By tradition, SDL Processes are defined by Extended Finite State Machines (EFSM), which are tightly linked to Regular Expressions, whence, to start with, this paper constructs an EFSM and analyzes its relationship to Regular Expressions. Next, MSC is mathematically defined and its relationship to Regular Expressions analyzed. Finally, a relationship between SDL Processes and MSC is obtained. Detailed conversions between these two models are given in the other two related papers [8, 9].

2. AN EFSM INTERPRETATION OF SDL PROCESSES

SDL structures have the two levels of System and Block. The description at the System Level tells the designer that the system and environment interact through signal channels and that each channel is served independently by a block, that at the Block level specifies the components and the signal channels between them.

A Block may contain many Processes which describe the behavior of the system components in SDL. Since SDL Processes are defined by EFSM, their behavior is strictly sequential and concurrent behavior can only occur between Processes.

A Process yields the main dynamic description of system behavior and performs a very important role in SDL systems, since it is defined and formalized by an EFSM. The Finite State Machine (FSM) is a well-known model in computer science, which has been extended to EFSM in SDL, in order to handle inputs with data, outputs as well as tasks of systems.

Nevertheless, there do not exist a formal definition and properties of EFSM, which are used to define SDL Processes. A new version of EFSM is presented below and its properties are discussed.

2.1 The Deterministic Extended Finite State Machine (DEFSM)

An EFSM comprises a set of *states*, a set of *state transitions* which occur once an input has been chosen from an *input* set, and a set of *output and task functions*. The *initial state*, usually denoted by q_0, starts the machine. In a DEFSM, each *input* has exactly one *transition* out of each *state*, during which there is exactly one *output-task sequence* executed before the *state* changes; its textual definition employs mathematics and automata theory. The theorems, which are necessary for the construction of a DEFSM, follow.

Definition:

If M is a set of DEFSM, then the machine $m \in M$ is the seven-tuple $m=(Q_m, \Sigma_m, \Theta_m, \Psi_m, \Gamma_{m\text{-}ST}, \Gamma_{m\text{-}OT}, q_{m\text{-}0})$, where

- Q_m **is the set of** n **states** $\{q_{m\text{-}1}, q_{m\text{-}2}, \ldots q_{m\text{-}n}\}$. According to SDL usage, there exist the three *state* types: Normal, Decision and Initial. Normal State is normal states in an FMS. Decision State refers to the Decisions in an SDL Process, while Initial is a special case of the Normal State which need not receive a signal, but changes automatically to the next *state* and performs certain initializing *tasks*.
- Σ_m **is the set of inputs** $\{i_{m\text{-}1}, i_{m\text{-}2}, \ldots\}$. $In_{m\text{-}k}$ is used to denote an *input* sequence $(i_{m\text{-}1} \bullet i_{m\text{-}2} \bullet \ldots \bullet i_{m\text{-}p})$, where k is the input sequence number, p the length of $In_{m\text{-}k}$ and '•' is the sequential symbol. In SDL, there are two *input* types: Normal and Special, the former denoting the signal received from the Environment, while the latter may be TRUE or FALSE or have certain forms, which are only used in the Decision State. In fact, there do not exist TRUE and FALSE signals, emitted from the Environment; the Special Inputs are introduced, in order to construct a unified definition for the SDL Processes.
- Θ_m **is the set of outputs** $\{o_{m\text{-}1}, o_{m\text{-}2}, \ldots\}$.
- Ψ_m **is the set of tasks** $\{t_{m\text{-}1}, t_{m\text{-}2}, \ldots\}$.With the exception of Normal Task, Procedure Tasks denote procedures in SDL, which can be replaced by any EFSM with one entrance and at least one termination point. In certain special cases, it will be useful to denote the Output-Task sequence by a single graphical symbol. A dashed box will be used to denote an Output-Task sequence (Fig. 2.4).
- $\Gamma_{m\text{-}ST}$: $Q_m \times \Sigma_m \rightarrow Q_m$ **is a transition function.** Let the current state be $q_{m\text{-}k} \in Q_m$; after a new input $i_{m\text{-}p} \in \Sigma_m$ has been received, the EFSM will change to the next state $q_{m\text{-}k}' = \Gamma_{m\text{-}ST}(q_{m\text{-}k}, i_{m\text{-}p})$, where ST indicates that it is a State Transition.
- $\Gamma_{m\text{-}OT}$: $Q_m \times \Sigma_m \rightarrow (\Theta_m \cup \Psi_m)^*$ **is an Output-Task (OT) sequence function.** For each transition, defined in a transition set, let the current state be $q_{m\text{-}k} \in Q_m$; after a new input $i_{m\text{-}p} \in \Sigma_m$ has been received, the

machine will change to the next state and complete an Output-Task sequence of the form $OT_{m\text{-}k}=\Gamma_{m\text{-}OT}(q_{m\text{-}k}, i_{m\text{-}p})$, where the following condition must be true:

$$(OT_{m\text{-}k}=u_{m\text{-}1} \bullet u_{m\text{-}2} \bullet \ldots \bullet u_{m\text{-}q}) \wedge (u_{m\text{-}1}, u_{m\text{-}2}, \ldots, u_{m\text{-}q} \in (\Theta_m \cup \Psi_m)) \wedge (\exists q_{m\text{-}k}' : q_{m\text{-}k}'=\Gamma_{m\text{-}ST}(q_{m\text{-}k}, i_{m\text{-}p})).$$

$IOT_{m\text{-}j}= (i_{m\text{-}p} \bullet OT_{m\text{-}k})$ denotes an Input-Output-Task sequence for the above transition and related *OT* function. Thus, on receiving an input $i_{m\text{-}p}$, there exists for each transition an *OT* and an *IOT*. If *OT* is empty, then $IOT=i_{m\text{-}p}$.

- $q_{m\text{-}0}$ **is an initial state**, an entrance to the *EFSM m*. Its graphical symbol has been shown in the State Set definition.

Extension of a Transition

For a reception of an input sequence, a transition can be extended $\Gamma_{m\text{-}ST}: Q_m \times \Sigma_m \rightarrow Q_m$ to $\Gamma_{m\text{-}ST}: Q_m \times \Sigma_m^* \rightarrow Q_m$, i.e.,

1. $q_{m\text{-}k}= \Gamma_{m\text{-}ST}(q_{m\text{-}k}, \varepsilon)$,
2. $\Gamma_{m\text{-}ST}(q_{m\text{-}k}, In_{m\text{-}p} \bullet i_{m\text{-}n})= \Gamma_{m\text{-}ST}(\Gamma_{m\text{-}ST}(q_{m\text{-}k}, In_{m\text{-}p}), i_{m\text{-}n})$,

where (1) states that EFSM can not change state without reading an input and (2) how to find the state after reading a non-empty input sequence $In_{m\text{-}p} \bullet i_{m\text{-}n}$.

In EFSM models, there are three kinds of deterministic concepts: Determinism in transition, in the OT function and in both, respectively.

1. An EFSM is called **deterministic in Transition** (abbreviated DEFSM-Tran) if it will only admit zero or one transition from a state at the same input.
2. An EFSM is called **Deterministic in OT function**, (i.e., DEFSM in OT function, or DEFSM-OT), if it only admits zero or one OT from one state to another at the same input.
3. An EFSM is said to be **a fully deterministic EFSM** or DEFSM in full, if it is deterministic in Transition and OT function.

2.2 Non-deterministic EFSM Model

Let $Pow(Q_m)$ denote the power set of set Q_m. If the definition of transition is changed to $\Gamma_{m\text{-}ST}: Q_m \times \Sigma_m \rightarrow Pow(Q_m)$, i.e., if an EFSM allows zero, one or more transitions from a *state* to others at the same *input*, the EFSM will be called Non-deterministic in transition and abbreviated NEFSM-Tran.

If the definition of OT function is changed to $\Gamma_{m\text{-}ST}: Q_m \times \Sigma_m \times Q_m \rightarrow \{(\Theta_m \cup \Psi_m)^*, \ldots, (\Theta_m \cup \Psi_m)^*\}$, i.e., if the EFSM allows zero, one or more OT from a *state* to another at the same *input*, the EFSM will be called Non-deterministic in OT, abbreviated NEFSM-OT.

2.3 Basic Types of EFSM

In the sequel, the following abbreviations are used to denote basic types of EFSM:

- DEFSM is deterministic in its transition and OT functions, i.e., it is fully deterministic;
- NEFSM-OT is deterministic in transition and non-deterministic in the OT function.
- NEFSM-Tran is non-deterministic in the transition and deterministic in OT function.
- NEFSM-Full is non-deterministic in transition and OT functions.
- NEFSM is used to denote above three kinds of Non-deterministic EFSM.

All the SDL Processes are denoted by DEFSM-full. The reason for discussing NEFSM is when we analyze relationship between SDL Processes and MSCs, some of MSCs present non-deterministic characteristic which causes difficulty of converting MSCs to SDL Processes. So, the NEFSM is defined to give a unified relationship between them.

2.4 Languages Accepted by EFSM

An input sequence In_{m-p} is said to be accepted by an EFSM, if there exist *states* q_{m-k}, and $q_{m-k}=\Gamma_{m-ST}(q_{m-0}, In_{m-p})$, i.e., if there exists in the transition diagram a path, labeled by the *input sequence* from an *initial state* to certain *states*, then the *input sequence* is accepted by the EFSM.

The above notion of acceptance can be further extended such that an EFSM may be considered to accept not only *input*, but also Input and Output-Task Sequences. In the EFSM model with *OT* sequences, the new concept of "Behavior Sequence" (BS) is defined, in order to describe the entire behaviors of EFSM. *If m* is an EFSM and In_m an arbitrary *input sequence* of the form $(i_{m-0} \bullet i_{m-1} \bullet, \ldots, \bullet i_{m-p})$ accepted by *m*, the Behavior Sequence *BS* for In_{m-k} is defined by:

$$BS=IOT_1 \bullet IOT_2 \bullet \ldots \bullet IOT_p = (i_{m-1} \bullet OT_{m-1}) \bullet (i_{m-2} \bullet OT_{m-2}) \bullet, \ldots, \bullet (i_{m-p} \bullet OT_{m-p}),$$

where $\forall 1 \le n \le p$,

$IOT_{m-n} = i_{m-n} \bullet OT_{m-n}$,

$OT_{m-n}=\Gamma_{m-OT}(\Gamma_{m-ST}(q_{m-0}, i_{m-0} \bullet i_{m-1} \bullet, \ldots \bullet i_{m-(n-1)}), i_{m-n}, \Gamma_{m-ST}(q_{m-0}, i_{m-0} \bullet i_{m-1} \bullet, \ldots \bullet i_{m-n}))$,

and, if $n=0$, then $q_{m-0}=\Gamma_{m-ST}(q_{m-0}, \varepsilon)$ and $OT_{m-n}=\varepsilon$.

IOT is called an Input-Output-Task sequence within one transition. Due to the fact that each Behavior Sequence consists of zero, one or many *IOTs*, and each *IOT* cannot be distributed over one transition function and the relative OT function, it is also called an Axiom Behavior Sequence.

The concept of "language", accepted by FSM, is like that in automata theory; the set of all behavior sequences, accepted by an EFSM m_k, is called a language accepted by the EFSM and denoted by $L(m_k)$.

For DEFSM, there exists exactly one behavior sequence for each input sequence, while for NEFSM, there may exist one or several behavior sequences. Let $L(m)$ denote the set of all possible behavior sequences, produced by an EFSM m. Each element in $L(m)$ is denoted by $BS_{m\text{-}k}$ in the form $IOT_{m\text{-}1} \text{`} IOT_{m\text{-}2} \text{`} \ldots \text{`} IOT_{m\text{-}n}$, where k is an integer.

3. MATHEMATICAL DEFINITION OF MSC

Consider the signal exchanges among many Instances of MSC in a system and executions of certain, corresponding internal sessions. Comparing MSC'96 with MSC'92, it becomes obvious that many new language concepts have been added, including Generalized Ordering, Inline Operator Expression, Reference, High Level MSC (HMSC). In the sequel, HMSC has been ignored because it describes the top-level system structure which can not describe by SDL Processes. Meanwhile, Generalized Ordering has been ignored since in this paper only the similar function units of SDL Processes and MSC will be considered.

3.1 Symbols used for the mathematical definition of MSC

Input i : *in*(*input-name, source*), i.e.,
in(*i, env*) means that Input i comes from the Environment.
Output j : *out*(*output-name, destination*), i.e.,
out(*j, env*) means Output j is sent to Environment.
Action k : *act*(*k*), i.e.,
act(*k*) means execution of the Action k.
Reference l : *ref*(*l*), i.e.,
ref(*l*) is the MSC Reference l.
#: is an ordering symbol with the following three forms:
+: is an alternative symbol. u_1+u_2 means only one of u_1 and u_2 can be chosen, where u_1 and u_2 are one of the basic elements of MSC.
•: is a sequential order symbol. $u_1 \bullet u_2$ means that u_1 is followed by u_2.

||: is a parallel order symbol. $u_1 || u_2$ means that u_1 and u_2 are in parallel.

The priority of *the* three symbols is '||' > '•' > '+'. So $u_1||u_2 = u_1 \bullet u_2 + u_2 \bullet u_1$, and $u_1 \bullet u_2 || u_3 = u_1 \bullet (u_2||u_3) = u_1 \bullet u_2 \bullet u_3 + u_1 \bullet u_3 \bullet u_2$. The parentheses '(' and ')' can be used to indicate the priority of the sub-sequences with the highest priority.

{...}: *beginning* and end symbols of a sequence.

3.2 MSC Definition

A single Instance in MSC is a seven-tuple

$$Ins=(\Sigma_{Ins}, \Theta_{Ins}, \Psi_{Ins}, Con_{Ins}, Exp_{Ins}, Ref_{Ins}, \coprod_{Ins}),$$

where

1. Σ_{Ins} **is a set of Inputs** $\{i_{Ins-1}, i_{Ins-2}, \ldots i_{Ins-n}\}$, if the Instance has n Inputs,
2. Θ_{Ins} **is a set of Outputs** $\{o_{In-1}, o_{Ins-2}, \ldots, o_{Ins-n}\}$, if the Instance contains *n* Outputs,
3. Ψ_{Ins} **is a set of Actions** $\{a_{Ins-1}, a_{Ins-2}, \ldots, a_{Ins-n}\}$, if there are *n* Actions,
4. Con_{Ins} **is a set of Conditions** $\{con_{Ins-1}, con_{Ins-2}, \ldots, con_{Ins-n}\}$, if there are *n* Conditions.
5. Exp_{Ins} **is a set of Inline Operator Expressions** $\{exp_{Ins-1}, exp_{Ins-2}, \ldots, exp_{Ins-n}\}$, if there are *n* Inline Operator Expressions and each *exp* has the form

   exp_{Ins-i}(***Expression Keyword***, [***BS-List***], [***BS-List***]... [***BS List***]),

   where *Expression Keywords* are "alt", "par", "loop", "opt" and "exc" and *BS-List* is a Behavior Sequence.
6. Ref_{Ins} **is a Reference set** $\{ref_{Ins-1}, ref_{Ins-2}, \ldots ref_{Ins-n}\}$, if there are *n* References.
7. $\coprod_{Ins}$ **is an ordered behavior sequence** $\coprod_{Ins}=(\Sigma_{Ins} \cup \Theta_{ns} \cup \Psi_{Ins} \cup Con_{Ins} \cup Exp_{Ins})^*$, where each element in $(\Sigma_{Ins} \cup \Theta_{Ins} \cup \Psi_{Ins} \cup Con_{Ins} \cup Exp_{Ins})^*$ is connected by ordering symbols.

After dealing with all Instances of MSC, an MSC is defined by the two-tuple:

$$Msc = (I, F),$$

where

1. ***I*** **is the set of instances** $\{Ins_1, Ins_2, \ldots, Ins_n\}$.
2. ***F*** **is the ordering function of the MSC,**

$$F: \mathcal{U}_{Ins1} \times \mathcal{U}_{Ins2} \times \ldots \times \mathcal{U}_{Insn} \rightarrow \lambda(\mathcal{U}_{Ins1} \parallel \mathcal{U}_{Ins2} \parallel \ldots \parallel \mathcal{U}_{Insn}),$$

where the symbol λ is the *state operator* used to enforce the basic static requirement that a message must be sent before it is received [10].

The following examples explain these definitions further and the graphical symbols of recommendation Z.120 (MSC'96) will be used below.

3.2.1 MSC of One Instance with Sequential Inputs and Outputs

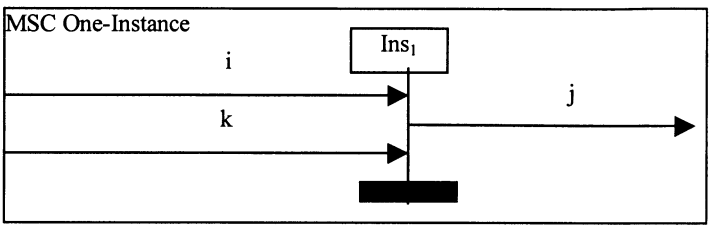

Figure 1. MSC with One Instanct

The corresponding symbolism of Fig. 1 is

$Ins = (\Sigma_{Ins}, \Theta_{Ins}, \Psi_{Ins}, Con_{Ins}, Exp_{Ins}, Ref_{Ins}, \mathcal{U}_{Ins})$,

$\Sigma_{Ins} = \{in(i,env), in(k,env)\}$, $\Theta_{Ins} = \{out(j, env)\}$, $\Psi_{Ins} = \varnothing$,

$Con_{Ins} = \varnothing$, $Exp_{Ins} = \varnothing$, $Ref_{Ins} = \varnothing$,

$\mathcal{U}_{Ins} = \{ in(i, env) \bullet out(j, env) \bullet in(k, env) \}$

MSC One-Instance = (I, F),

$I = \{Ins\}$, $F(\mathcal{U}_{Ins}) = \lambda(\mathcal{U}_{Ins}) = \{ in(i, env) \bullet out(j, env) \bullet in(k, env) \}$

3.2.2 MSC with Two Instances and Conditions

MSC Conditions (Fig. 2) can be defined in the form

$Ins_1 = (\Sigma_{Ins1}, \Theta_{Ins1}, \Psi_{Ins1}, Con_{Ins1}, Exp_{Ins1}, Ref_{Ins1}, \mathcal{U}_{Ins1})$,

$\Sigma_{Ins1} = \varnothing$, $\Theta_{Ins1} = \{out(j, Ins_2)\}$, $\Psi_{Ins1} = \varnothing$,

$Con_{Ins1} = \{c1, c2\}$, $Exp_{Ins1} = \varnothing$, $Ref_{Ins1} = \varnothing$,

$\mathcal{U}_{Ins1} = \{ c1 \bullet out(j, Ins_2) \bullet c2 \}$

$Ins_2 = (\Sigma_{Ins2}, \Theta_{Ins2}, \Psi_{Ins2}, Con_{Ins2}, Exp_{Ins2}, Ref_{Ins2}, \mathcal{U}_{Ins2})$,

$\Sigma_{Ins2} = \{in(j, Ins_1)\}$, $\Theta_{Ins2} = \varnothing$, $\Psi_{Ins2} = \varnothing$,

$Con_{Ins2} = \{c1, c2\}$, $Exp_{Ins2} = \varnothing$, $Ref_{Ins2} = \varnothing$,

$\mathcal{U}_{Ins2} = \{ c1 \bullet in(j, Ins_1) \bullet c2 \}$

MSC Condition = (I, F),

$I = \{Ins_1, Ins_2\}$

$F(\mathcal{U}_{Ins1}, \mathcal{U}_{Ins1}) = \lambda(\mathcal{U}_{Ins1} \parallel \mathcal{U}_{Ins2}) = \{ out(j, Ins_2) \bullet in(j, Ins_1) \}$.

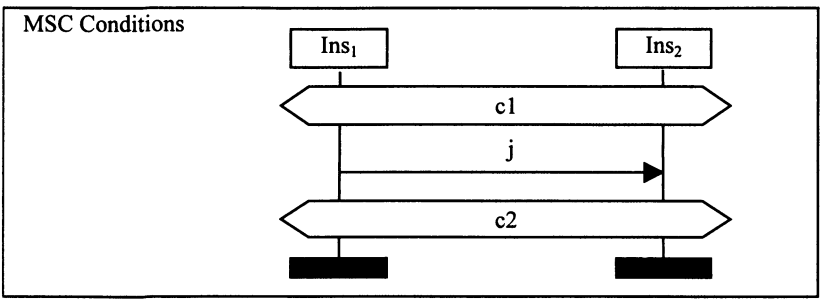

Figure 2. MSC Conditions

3.2.3 MSC with Two Instances and Inline Operator Expression

MSC Inline Expressions (Fig. 3) can be denoted in the form

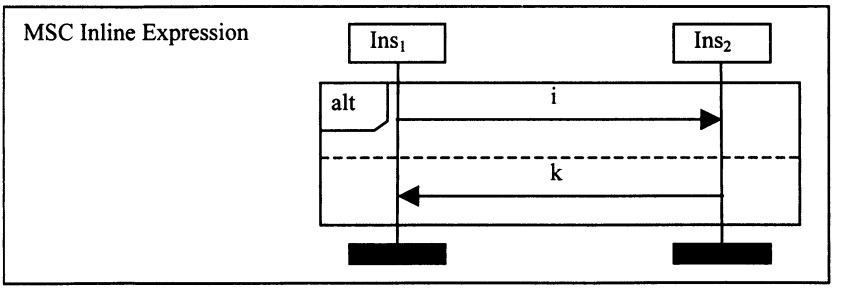

Figure 3. MSC Inline Expressions

$$Ins_1 = (\Sigma_{Ins1}, \Theta_{Ins1}, \Psi_{Ins1}, Con_{Ins1}, Exp_{Ins1}, Ref_{Ins1}, \mathcal{L}_{Ins1}),$$
$$\Sigma_{Ins1} = \{in(k, Ins_2)\}, \Theta_{Ins1} = \{out(i, Ins_2)\}, \quad \Psi_{Ins1} = \varnothing,$$
$$Con_{Ins1} = \varnothing, \quad Exp_{Ins1} = \{exp(alt, [i], [k])\}, \quad Ref_{Ins1} = \varnothing,$$
$$\mathcal{L}_{Ins1} = \{exp(alt, [i], [k])\}.$$
$$Ins_2 = (\Sigma_{Ins2}, \Theta_{Ins2}, \Psi_{Ins2}, Con_{Ins2}, Exp_{Ins2}, Ref_{Ins2}, \mathcal{L}_{Ins2}),$$
$$\Sigma_{Ins2} = \{in(i, Ins_1)\}, \Theta_{Ins2} = \{out(k, Ins_1)\}, \quad \Psi_{Ins2} = \varnothing,$$
$$Con_{Ins2} = \varnothing, \quad Exp_{Ins2} = \{exp(alt, [i], [k])\}, \quad Ref_{Ins2} = \varnothing,$$
$$\mathcal{L}_{Ins2} = \{exp(alt, [i], [k])\}.$$

MSC Inline Expression = (I, F),

$I = \{Ins_1, Ins_2\}$

$F(\mathcal{L}_{Ins1}, \mathcal{L}_{Ins2})$

$= \lambda(\mathcal{L}_{Ins1} \,||\, \mathcal{L}_{Ins2})$

$= \lambda(\{out(i, Ins_2) + in(k, Ins_2)\} \,||\, \{in(i, Ins_1) + out(k, Ins_1)\})$

$= \{out(i, Ins_2) \bullet in(i, Ins_1) + out(k, Ins_1) \bullet in(k, Ins2)\}$

3.2.4 MSC with One Action and One Reference

MSC Actions and References (Fig. 4) can be denoted in the form

$Ins = (\Sigma_{Ins}, \Theta_{Ins}, \Psi_{Ins}, Con_{Ins}, Exp_{Ins}, Ref_{Ins}, \amalg_{Ins})$,

$\Sigma_{Ins} = \{in(i, Env)\}$, $\Theta_{Ins} = \varnothing$, $\Psi_{Ins} = \{a\}$,

$Con_{Ins} = \varnothing$, $Exp_{Ins} = \varnothing$, $Ref_{Ins} = \{Ref(r)\}$,

$\amalg_{Ins} = \{ in(i, Env) \bullet Act(a) \bullet Ref(r) \}$

MSC Action and Reference $=(I, F)$,

$I = \{Ins\}$

$F(\amalg_{Ins})$

$= \lambda(\amalg_{Ins})$

$= \{ in(i, Env) \bullet Act(a) \bullet Ref(r) \}$.

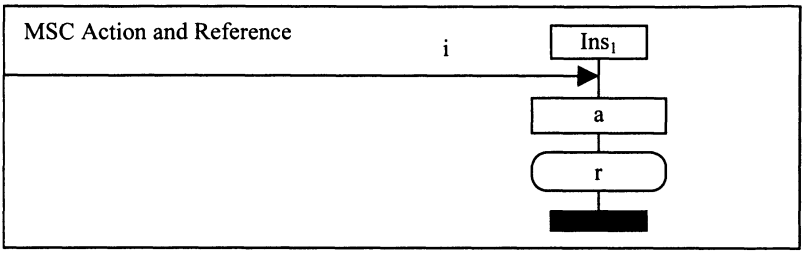

Figure 4. MSC Action and Reference

4. RELATIONSHIPS BETWEEN EFSM AND REGULAR EXPRESSIONS

This section sets up and analyzes new properties of an EFSM model, which include DEFSM, NEFSM, NEFSM with ε-move and the relationship between EFSM and Regular Expressions. Based on the definition of MSC, a relationship between it and Regular Expressions is introduced. Finally, the relationship of EFSM and MSC is determined. In order to achieve the above purposes, certain concepts and lemmas, used in these theorems, are introduced. All the lemmas and theorems are not proved due to the limited space available.

4.1 Characteristics and Properties of EFSM

The EFSM model differs from the traditional Finite State Machine (FSM) model in special characteristics and properties.

4.1.1 The Four Containment Relationships of EFSM

There exist equivalent relationships among the three kinds of NEFSM, while there does not exist an equivalence of DEFSM and NEFSM. According to the definitions of DEFSM and NEFSM, DEFSM can be viewed to be a special case of NEFSM-OT or of NEFSM-Tran. Moreover, NEFSM-OT and NEFSM-Tran are also special cases of NEFSM-Full, whence:

1. {DEFSM} $\subset$ {NEFSM-OT} $\subset$ {NEFSM-Full},
2. {DEFSM} $\subset$ {NEFSM-Tran}$\subset${NEFSM-Full},

where {DEFSM} denotes the entire set of DEFSM.

4.1.2 Equivalence of Three Kinds of NEFSM

The equivalence of NEFSM-Tran, NEFSM-OT and NEFSM-Full will be considered next. Since {NEFSM-Tran}$\subset$ {NEFSM-Full} and {NEFSM-OT} $\subset$ {NEFSM-Full}, only the following assumptions require proof:

- for any NEFSM-Tran m, there exists a NEFSM-OT m' such that $L(m)=L(m')$,
- for any NEFSM-OT m, there exists a NEFSM-Tran m' such that $L(m)=L(m')$,
- for any NEFSM-Full m, there exists a NEFSM-OT m' such that $L(m)=L(m')$.

Theorem 1: If L is a language accepted by NEFSM-Tran m, then there exists NEFSM-OT m' such that $L(m)=L(m')$.

Theorem 2: If L is a set produced by an NEFSM-OT, then there exists an NEFSM-Tran which can produce the same L.

Theorem 3: If L is a set produced by an NEFSM-Full, then there exists an NEFSM-Tran which can produce the same L.

4.2 Relationship between DEFSM and NEFSM

Unfortunately, there is not an equivalent relationship between DEFSM and NEFSM, since

Theorem 4: If L is a language, accepted by an NEFSM-OT, then no DEFSM can accept the same L.

4.3 Relationship between EFSM and Regular Expression

Based on automata theory, a Regular Expression can also define a language. By tradition, a Regular Expression *Rexp* consists only of *input* symbols. However, the EFSM model contains not only *input*, but also *output* and *task*. Thus, in the present treatment, a Regular Expression *Rexp* consists of *input*, *output* and *task* symbols; the *Behavior Sequence Set*, defined by *Rexp*, is called a language, which is denoted by *L*(*Rexp*). This language is also called a *regular set*.

According to automata theory, there exist equivalent relationships between FSM, FSM with ε-move and *regular set* [3]. The EFSM model implies equivalence of *regular sets* and the languages accepted by NEFSM, but there is not an equivalent relationship between DEFSM and Regular Expressions. The class of *L*(*m*), produced by a DEFSM *m*, is a sub-set of the class of *L*(*Rexp*), produced by the Regular Expression *Rexp*.

Due to the equivalence between any two of the three kinds of NEFSM, all kinds of them can equivalently be denoted by Regular Expressions and *vice versa*. This conclusion is proved in

Theorem 5: Equivalence of regular expressions and NEFSM.

5. RELATIONSHIP BETWEEN MSC BEHAVIOR SEQUENCE AND REXP

A MSC Behavior Sequence denotes the order of message exchange among objects of a system. All Behavior Sequences, produced by an MSC, are called a language, accepted by MSC and denoted by *L*(*MSC*).

By the definition of MSC in Section 3.2, there exist in MSC the eight basic operators: '+', '•', '||', *exp*(loop), *exp*(alt), *exp*(par), *exp*(opt), *exp*(exc). Because *exp*(alt) and *exp*(par) have the same meanings as '+' and '||', and *exp* (opt) and *exp*(exc) is being ignored here, the number of basic operators reduces to '+', '•', '||' and *exp*(loop).

If one imposes a limitation on *exp*(loop) and lets it loop to infinity, then the set of MSC Behavior Sequences becomes regular, a result which proves

Theorem 6: *Regular sets* and Behavior Sequence sets of MSC with limitation are equivalent.

6. RELATIONSHIP BETWEEN SDL PROCESSES AND MSC

Although the DEFSM does not cover all elements of SDL Processes, the main features of SDL Processes can be described by a DEFSM. Meanwhile each DEFSM can be denoted by a Regular Expression, whence all SDL Processes can be represented by Regular Expressions. Due to the equivalence of the regular and language sets of MSC, all Behavior Sequences produced by SDL Processes can be translated into MSC.

However, since the class of languages accepted by DEFSM is merely a sub-set of a regular set, there exists an equivalence of regular and language sets of MSC; thus the behavior sequences of MSC cannot always be translated into DEFSM. But each Behavior Sequence of MSC can be converted into EFSM as a result of the equivalence of EFSM and regular sets (EFSM includes DEFSM and NEFSM).

Hence, with regard to the translation from SDL Processes into MSC, it is complete and does not change the meaning of Message Exchange. When it comes to a conversion from MSC to SDL Processes, first translate MSC into EFSM, then check whether an EFSM is a DEFSM or not. If it is a DEFSM, then it is also a SDL Process. If it is not a DEFSM, a new structure must be defined, in order to translate NEFSM into SDL Processes.

7. CONCLUSIONS AND RECOMMENDATIONS

According to the equivalence of EFSM and Regular Expression as well as MSC Behavior Sequence and Regular Expression, SDL Processes have the equivalent meaning of an MSC Behavior Sequence for the basic components of these two languages. Hence SDL Processes and MSC Behavior Sequence can equivalently be translated from one into the other.

Based on this relationship, a further task is to define a set of rules to translate SDL Processes into MSC and *vice versa* [8, 9]. These translations will provide a more powerful and intuitive tool for system specification and description as well as overcome disadvantages of these two languages.

REFERENCES

[1] R. Braek and O. Haugen. (1993). *Engineering Real Time Systems--An object-oriented methodology using SDL*. Prentice Hall, Englewood Cliffs.

[2] R. Braek. (1996). SDL Basics. *Computer Networks and ISDN Systems*, Vol.28, pp.1585-1602.

[3] J. E. Hopcroft and J. D. Ullman. (1990). *Introduction to Automata Theory, Languages and Computation*. Narosa Publishing House Reprint.

[4] ITU Z.100. (1989). Recommendation Z.100, Specification and Description Language SDL. Geneva: ITU.

[5] ITU Z.100. (1993). Recommendation Z.100, Specification and Description Language SDL. Geneva: ITU.

[6] ITU-TS. (1993). Recommendation Z.120, Message Sequence Chart (MSC). *ITU-TS*, Geneva.

[7] ITU-TS. ITU-TS Recommendation Z.120, Message Sequence Chart (MSC). *ITU-TS*, Geneva. (Publication scheduled 1997).

[8] J. Lian and V. Wuwongse. (1999). Conversion from SDL Processes into MSC. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology.

[9] J. Lian and V. Wuwongse. (1999). Conversion from MSC into SDL Processes. *Technical Report*, Computer Science and Information Management Program, Asian Institute of Technology.

[10] S. Mauw. (1996). The formalization of Message Sequence Charts. *Computer Networks and ISDN Systems*, Vol. 28, pp.1643-1657.

[11] E. Rudolph, P. Graubmann and J. Grabowski. (1996). Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, Vol. 28, pp.1629-1641.