

Finite State machines to Guard-Stage-Milestone

Mike A. Marin

February 16, 2017

1 Introduction

This document presents the transformation of a Deterministic Finite State Machine (DFSM) into a Guard-Stage-Milestone (GSM) [Hul+11] type that was published in Marin, Lotriet, and van der Poll [MLv16]. This is used to convert the statechart that describe the entity life cycles in the Case Management Model and Notation (CMMN) specification [OMG14].

Solomakhin et al. [Sol+13] represented a Turing Machine as a GSM type, in order to prove that “there exists a GSM model for which verification of a propositional reachability property is undecidable”. Therefore, it is not surprising that a DFSM can be expressed using a GSM type, but we show in here that it is quite natural and elegant to express a DFSM as a GSM type. We present a transformation that faithfully rewrites a DFSM into a GSM type.

2 Implementing a DFSM using GSM

We start this section by defining a DFSM [HMu01; Sav08].

Definition 1. (*Deterministic Finite State Machine*): A DFSM is described by the 5-tuple,

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

where,

Q is the set of *states*.

Σ is the set of *input symbols*.

δ is the *state transition function* $\delta : Q \times \Sigma \rightarrow Q$.

q_0 is the *initial state* ($q_0 \in Q$).

F is the set of *final states* ($F \subseteq Q$).

Informally we can transform a DFSM $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ into a GSM type $\Gamma = \langle Att, EType, Stg, Tsk, Mst, Stry, Lcyc \rangle$ as follows,

1. The set of states Q maps to stages Stg . In GSM, each stage $S \in Stg$ has a corresponding Boolean variable in Att_{stages} . Therefore, each state $s \in Q$ becomes an $S \in Stg$, and an $s \in Att_{stages}$. For a non-active Γ all the Boolean variables in Att_{stages} are set to *false*. In an active Γ only one Boolean variable in Att_{stages} is set to *true* indicating the open stage, and all other Boolean variables in Att_{stages} are set to *false*.
2. The set of input symbols Σ maps to incoming events $EType_{inc}$. In addition, an initiating event e_0 is required to start the machine (i.e., $\delta(\emptyset, e_0) = q_0$). Each input symbol $e \in \Sigma \cup \{e_0\}$, becomes the set of incoming events $E:e \in EType_{inc}$.
3. The state transition function $\delta : Q \times \Sigma \rightarrow Q$ maps to B-steps. Each B-step moves the machine from one snapshot corresponding to a state of the machine to the next snapshot corresponding to the next state of the machine. The B-steps are implemented by the life cycle $Lcyc$ of the GSM type. For the life cycle $Lcyc$ to work the sets of $EType_{gen}$, Att_{status} , Mst , and $Stry$ need to be populated.

The state transition function $\delta(S_n, e) = S_m$, where $S_n, S_m \in Q$ (now $S_n, S_m \in Stg$), and $e \in \Sigma$ (now $E:e \in EType_{inc}$) transition from stage S_n to stage S_m when the external event e arrives. The external event e triggers a B-step that transitions the machine from one snapshot where $\{S_n = true, S_m = false\} \subseteq Att_{stages}$ to the next snapshot where $\{S_n = false, S_m = true\} \subseteq Att_{stages}$, as follows:

- (a) The achieving sentry $a \in Ach$ of milestone $m = \langle S_n, e \rangle \in Mst$ is triggered ($m \in Att_{milestones}$ becomes *true*, and event $+m = +\langle S_n, e \rangle \in EType_{gen}$ is generated).
- (b) The event $+m = +\langle S_n, e \rangle \in EType_{gen}$ triggers the terminator $t_m \in Terminators$ of stage S_n , closing stage S_n and updating the status attribute $S_n \in Att_{stages}$ to *false*.
- (c) The opening sentry $g \in Guards$ of stage S_m is also triggered, opening stage S_m ($S_m \in Att_{stages}$ becomes *true* generating an internal event $+S_m \in EType_{gen}$).
- (d) The event $+S_m$ invalidates all the milestones in stage S_m , by triggering the invalidating sentries $i \in Inv$.

The end result is that $\delta(S_n, e) = S_m$ consumes event e by closing stage S_n and opening stage S_m , resulting in the same behavior that $\delta(S_n, e) = S_m$ has in a DFMS.

The only exception to this mechanism is to start the machine, in which case the GSM implementation accepts $\delta(\emptyset, e_0) = q_0$. This is done by providing an additional guard $q_0 \in Guards$ for the initial stage q_0 , and a special terminator $t_\infty \in Terminators$ to close the final stages $S_\infty \in F$ when the machine starts.

Each snapshot is designed to contain at most one Boolean variable in Att_{stages} set to *true* indicating the active stage on the snapshot. This constraint is maintained by the sentries that are executed during the B-step. For the life cycle $Lcyc$ to implement the B-step as described, the following six types of sentries are needed:

- (a) A sentry used in the $q_0 \in Guards$ that starts the machine. This sentry condition φ must check that the machine is not executing. To verify that, the sentry checks that the status Att_{stages} of all non-terminal stages is *false*. This sentry has the following form

$$[\mathbf{on} \ E:e_0 \ \mathbf{if} \ \bigwedge_{s \in Q \cap F^c} \neg s] \quad (2.1)$$

- (b) A set of sentries to close all terminal stages, when a new execution of the machine is required. These sentries will be used for the terminators $t_\infty \in Terminators$ of all the terminal stages (corresponding to states in F).

$$\{[\mathbf{on} \ E:e_0 \ \mathbf{if} \ s] \mid e_0 \in \Sigma \wedge s \in F \wedge \delta(\emptyset, e_0) = q_0\} \quad (2.2)$$

- (c) A set of sentries used as *Guards* to open the stages in *Stg*. This sentry φ verifies that the state transition is coming from a valid previous stage.

$$\{[\mathbf{on} \ E:e \ \mathbf{if} \ \bigvee_{\delta(s,e)=y} s] \mid e \in \Sigma \wedge y \in Q\} \quad (2.3)$$

- (d) A set of sentries to trigger a milestone when an event arrives during a particular stage.

$$\{[\mathbf{on} \ E:e \ \mathbf{if} \ s] \mid e \in \Sigma \wedge s \in Q \wedge \exists x : \delta(s, e) = x\} \quad (2.4)$$

- (e) A set of sentries to terminate the stages when the milestone is achieved.





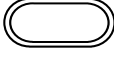
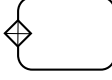




$$\{[\mathbf{on} \ +m] \mid m \in Mst\} \quad (2.5)$$

- (f) Finally, a set of sentries to invalidate the milestones when entering a stage.

$$\{[\mathbf{on} \ +s] \mid s \in Q \wedge \exists x, y : \delta(s, x) = y\} \quad (2.6)$$

4. The initial state q_0 is a stage in *Stg*. A DFSM does not define an initiating input symbol. However, in GSM all stages must have a guard, and at least one guard is needed to start the GSM type. Therefore, we use an initiating event e_0 to start the machine. To prevent an executing machine to be started while it is working, we check that the status of all non-final stages is *false* to accept the initiating event.
5. The set of final states F are stages in *Stg*. In GSM all stages must have a terminating sentry. Therefore, we will use the same initiating event e_0 , used to start the state machine, to close the final stages. Thus, a machine may be executed multiple times and each time it reaches a terminating state it stays there until the next execution.

Table 1: Mapping DFSM to GSM (synthesized by researchers)

DFSM	Description	DFSM notation	GSM notation	GSM
Q	States (nodes)			Stg, Att_{stages}
Σ	Input symbols	transition labels	external events	$EType_{inc}$
δ	State transition			B-step ($EType, Att_{status}, Mst, Stry, Submilestones, Guards, Terminators, Ach$)
q_0	Initial state ($q_0 \in Q$)			$Stg, Att_{stages}, Guard$
F	Final states ($F \subseteq Q$)			$Stg, Att_{stages}, Terminator$
	Start the machine			initial <i>Guard</i>

This informal transformation of a DFSM M into a GSM type Γ can be described by mapping the DFSM transition diagram notation into the GSM notation as described in Table 1. A DFSM transition diagram can be converted to a valid GSM diagram by just replacing the notation.

- State nodes become stage nodes in GSM.
- Input symbols used as transition labels in DFSM become external events in GSM.
- State transitions represented in DFSM as directed arcs become milestones with an event propagation and guards in GSM.
- The initial state becomes a stage with an initial guard.
- Final states become stages with terminators.

Table 1 also includes a commonly used DFSM notation to start the machine, which corresponds to a GSM starting guard. Mapping the DFSM transition diagram notation to the GSM notation and terminology shows how natural a DFSM transition diagram can be converted to a GSM diagram.

2.1 Formalization

This section formalizes the transformation of a DFSM M into a GSM type Γ that was described in the previous section. We introduce two conditions, first an initiating event is required to have a valid GSM type. Second, to avoid violating the toggle once principle

of GSM only a DFSM with no state transitions of a state into itself are transformed $(\nexists s)(\delta(s, e) = s)$.

Note that a DFSM with state transitions of a state into itself $(\exists s)(\delta(s, e) = s)$ can always be converted into a DFSM without states looping into itself, by splitting the offending state into two states and adapting the state transition function δ accordingly.

Definition 2. (*Rewriting a DFSM into a GSM type*): Given a DFSM $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ such that $(\nexists s)(\delta(s, e) = s)$, and containing an initiating event e_0 , the corresponding GSM type is defined as

$$\Gamma_M = \langle Att_M, EType_M, Stg_M, Tsk_M, Mst_M, Stry_M, Lcyc_M \rangle$$

where,

$$\begin{aligned} Att_M &= Att_{data} \cup Att_{stages} \cup Att_{milestones} \\ &= \emptyset \cup \{s \mid s \in Q\} \cup \{m \mid m \in Mst_M\} \\ &= \{s \mid s \in Q\} \cup \{m \mid m \in Mst_M\} \end{aligned}$$

$$\begin{aligned} EType_M &= EType_{inc} \cup EType_{gen} \\ &= \{E:e \mid e \in \Sigma\} \cup \{+e \mid e \in Mst_M \vee \\ &\quad (e \in Q \wedge \exists x, y : \delta(e, x) = y)\} \end{aligned}$$

$$Stg_M = \{s \mid s \in Q\}$$

$$Tsk_M = \emptyset$$

$$Mst_M = \{\langle s, e \rangle \mid s \in Q \wedge e \in \Sigma \wedge \exists x : \delta(s, e) = x\}$$

$$\begin{aligned} Stry_M &= \{[\mathbf{on} \ E:e_0 \ \mathbf{if} \ \bigwedge_{s \in Q \cap F^c} \neg s]\} \\ &\cup \{[\mathbf{on} \ E:e_0 \ \mathbf{if} \ s] \mid e_0 \in \Sigma \wedge s \in F \wedge \delta(\emptyset, e_0) = q_0\} \\ &\cup \{[\mathbf{on} \ E:e \ \mathbf{if} \ \bigvee_{\delta(s,e)=y} s] \mid e \in \Sigma \wedge y \in Q\} \\ &\cup \{[\mathbf{on} \ E:e \ \mathbf{if} \ s] \mid e \in \Sigma \wedge s \in Q \wedge \exists x : \delta(s, e) = x\} \\ &\cup \{[\mathbf{on} \ +e] \mid e \in Q \wedge \exists x, y : \delta(e, x) = y \\ &\quad \vee e \in Mst_M\} \end{aligned}$$

$$\begin{aligned} Lcyc_M &= \langle Substages_M^R, Tasks_M^R, Submilestones_M^R, Guards_M^R, \\ &\quad Terminators_M^R, Ach_M^R, Inv_M^R \rangle \end{aligned}$$

$$\text{Substages}_M^R = \emptyset$$

$$\text{Tasks}_M^R = \emptyset$$

$$\begin{aligned} \text{Submilestones}_M^R = \{ \langle s, m \rangle \mid & s \in Q \wedge m = \langle s, e \rangle \in \text{Mst}_M \\ & \wedge \exists y \in Q : \delta(s, e) = y \} \end{aligned}$$

$$\begin{aligned} \text{Guards}_M^R = \{ \langle s, g \rangle \mid & s \in Q \wedge g \in \text{Stry}_M \wedge \\ & ((g = [\mathbf{on} \ E:e \ \mathbf{if} \ \bigvee_{\delta(x,e)=s} x] \wedge x \in Q) \\ & \vee (s = q_0 \wedge g = [\mathbf{on} \ E:e_0 \ \mathbf{if} \ \bigwedge_{x \in Q \cap F^c} \neg x])) \} \end{aligned}$$

$$\begin{aligned} \text{Terminators}_M^R = \{ \langle s, t \rangle \mid & s \in Q \wedge t \in \text{Stry}_M \wedge \\ & ((t = [\mathbf{on} \ + m] \wedge m = \langle s, e \rangle \in \text{Mst}_M \\ & \wedge \exists y \in Q : \delta(s, e) = y) \\ & \vee (s \in F \wedge t = [\mathbf{on} \ E:e_0 \ \mathbf{if} \ s])) \} \end{aligned}$$

$$\begin{aligned} \text{Ach}_M^R = \{ \langle m, a \rangle \mid & m = \langle s, e \rangle \in \text{Mst}_M \wedge a \in \text{Stry}_M \\ & \wedge a = [\mathbf{on} \ E:e \ \mathbf{if} \ s] \wedge e \in \Sigma \wedge s \in Q \\ & \wedge \exists x : \delta(s, e) = x \} \end{aligned}$$

$$\begin{aligned} \text{Inv}_M^R = \{ \langle m, i \rangle \mid & m = \langle s, e \rangle \in \text{Mst}_M \wedge i \in \text{Stry}_M \\ & \wedge i = [\mathbf{on} \ + s] \} \end{aligned}$$

Definition 3. (*Language of a DFSM*): The language of a DFSM [HMU01] $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ is defined as,

$$L(M) = \{ w \mid \hat{\delta}(q_0, w) \in F \wedge w \in \Sigma^* \}$$

where,

w is a *string* of input symbols in Σ .

$\hat{\delta}$ is an *extended state transition function* for strings

$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$. Defined as

$$\hat{\delta}(q, w) = \begin{cases} q & \text{for } w = \epsilon \\ \delta(q, a) & \text{for } w = a : a \in \Sigma \\ \delta(\hat{\delta}(q, x), a) & \text{for } w = xa : x \in \Sigma^* \wedge a \in \Sigma \end{cases}$$

Definition 4. (*Equivalence between two machines*): Two machines M_1 and M_2 are equivalent if and only if they accept the same language $L(M_1) = L(M_2)$ [HMU01; Sav08]. We denote equivalence between machines as $M_1 \equiv M_2$.

Proposition 1. *if $\Gamma = \langle Att, EType, Stg, Tsk, Mst, Stry, Lcyc \rangle$ is the GSM type constructed by rewriting the DFSM $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, then $\Gamma \equiv M$.*

Proof. (Sketch) This can be proved by showing that Γ and M accept the same language, which is done by induction in two cases $L(M) \subseteq L(\Gamma)$ and $L(\Gamma) \subseteq L(M)$. In addition, we prove that at most one stage is open in each snapshot, and that Γ implements only the δ transition function and nothing else. This involves a detailed description on the order of firing of PAC rules and generated events within a B-step. The formal proof requires a rigorous treatment of B-steps and GSM operational semantics and it is beyond the scope of this paper.

For Γ and M to be equivalent they must accept the same language (see Definition 4). For Γ and M to accept the same language, they must have the same set of input symbols Σ , and the same set of terminal states F . By Definition 2:

- The set of input symbols of Γ is given by the set of external events $EType_{inc} = \{E:e \mid e \in \Sigma\}$, which is a one to one correspondence with the input symbols of M .
- The set of states of Γ corresponds to stages $Stg = \{S \mid S \in Q\}$, which is a one to one correspondence with the set of states in M , including the terminal states F and the initial state q_0 .

Now, we need to proof that $L(\Gamma) = L(M)$. We have two cases:

Case 1: input strings w accepted by M ($L(\Gamma) \supseteq L(M)$). By induction, let begin with an input string w contains a single symbol ($a \in \Sigma \wedge w = a$), and both M and Γ are in the start state or stage q_0 , and $\delta(q_0, a) = s_a$ for M . Therefore, $\hat{\delta}(q_0, a) = \delta(q_0, a) = s_a$, where $s_a \in Q$.

For Γ to be in stage q_0 , means that it is in a snapshot in which stage q_0 is open ($\{q_0 = true\} \in Att_{stages}$), and all other stages are closed. The arrival of the external event a (denoted as $E:a$) start a B-step, as described in Section 2 and Definition 2, which is designed to implement $\delta(q_0, a) = s_a$. Therefore, $\hat{\delta}(q_0, a) = \delta(q_0, a) = s_a$, where $s_a \in Stg$. Thus, both machines Γ and M move from q_0 to s_a on the arrival of a as input.

Second, let assume that w has the form $w = xa$, meaning a is the last symbol of w , and x is the rest of the w string, and both M and Γ are in an arbitrary state or stage q . We assume $\hat{\delta}(q, x) = s_x$ works for both M and Γ . Then, $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a) = s_w$ for M .

For Γ to be in stage s_x , means that it is in a snapshot in which stage s_x is open ($\{s_x = true\} \in Att_{stages}$), and all other stages are closed. The arrival of the external event a (denoted as $E:a$) start a B-step, as described in Section 2 and Definition 2, which is designed to implement $\delta(s_x, a) = s_w$. Therefore, $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a) = s_w$ for Stg .

Case 2: input strings w accepted by Γ ($L(\Gamma) \subseteq L(M)$). By induction, let begin with an input string w contains a single symbol ($a \in \Sigma \wedge w = a$), and both Γ and M are in the start stage or state q_0 , and $\delta(q_0, a) = s_a$ for Γ .

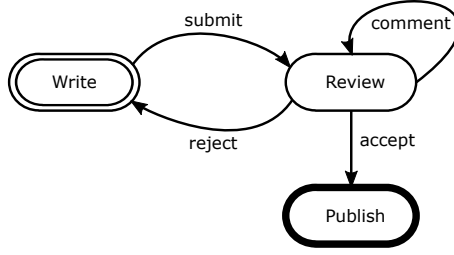


Figure 1: State transition diagram for the simple writer-review state machine

Therefore, $\hat{\delta}(q_0, a) = \delta(q_0, a) = s_a$, where $s_a \in \Gamma$; but by definition as described in Section 2 and Definition 2 $\delta(q_0, a) = s_a$ was originally defined in M . Thus, both machines Γ and M move from q_0 to s_a on the arrival of a as input.

Second, let assume that w has the form $w = xa$, and both Γ and M are in an arbitrary stage or state q . We assume $\hat{\delta}(q, x) = s_x$ works for both Γ and M . Then, $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a) = s_w$ for Γ ; but by definition as described in Section 2 and Definition 2 $\delta(s_x, a) = s_w$ was originally defined in M .

$$\therefore L(\Gamma) = L(M) \quad \Rightarrow \quad \Gamma \equiv M$$

■

3 Simple writer-review example

The state transition diagram for a DFSM $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ describing a simple writer-review state machine is shown in Figure 1.

For illustration purposes, we describe here each element of the simple writer-review state machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ as follows,

$$Q = \{\text{Write}, \text{Review}, \text{Publish}\}$$

$$\Sigma = \{\text{submit}, \text{reject}, \text{comment}, \text{accept}\}$$

$$\delta = \{\delta(\text{Write}, \text{submit}) \mapsto \text{Review}, \\ \delta(\text{Review}, \text{reject}) \mapsto \text{Write}, \\ \delta(\text{Review}, \text{comment}) \mapsto \text{Review}, \\ \delta(\text{Review}, \text{accept}) \mapsto \text{Publish}\}$$

$$q_0 = \{\text{Write}\}$$

$$F = \{\text{Publish}\}$$

This state machine violates the constrained $(\nexists s)(\delta(s, e) = s$ in Definition 2. The Review state violates the restriction, because when a comment symbol arrives it ends in the same state,

$$\delta(\text{Review}, \text{comment}) \mapsto \text{Review}$$

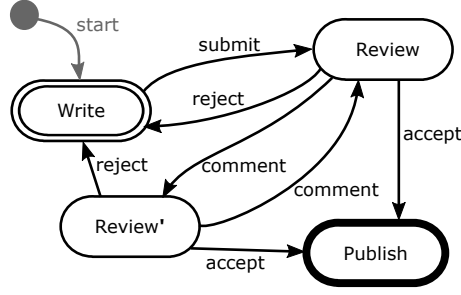


Figure 2: Expanded state transition diagram for the simple writer-review state machine

The removal of this loop requires that we introduce a new state, which we call Review'. The new state is entered when the comment input symbol arrives and the machine is in state Review. To emulate the initial machine all the incoming symbols that Review state handles must also be handled in the same way by the new Review' state.

Figure 2 shows the resulting machine, which is equivalent to the original machine by the fact that both of them can be reduced to the same DFSM. Note that we also added start as an initiating input symbol, which is a convention present in several notations. The elements of the resulting simple writer-review state machine $M' = \langle Q, \Sigma, \delta, q_0, F \rangle$ are

$$Q = \{\text{Write}, \text{Review}, \text{Review}', \text{Publish}\}$$

$$\Sigma = \{\text{start}, \text{submit}, \text{reject}, \text{comment}, \text{accept}\}$$

$$\delta = \{\delta(\emptyset, \text{start}) \mapsto \text{Write}, \\
\delta(\text{Write}, \text{submit}) \mapsto \text{Review}, \\
\delta(\text{Review}, \text{reject}) \mapsto \text{Write}, \\
\delta(\text{Review}, \text{comment}) \mapsto \text{Review}', \\
\delta(\text{Review}', \text{comment}) \mapsto \text{Review}, \\
\delta(\text{Review}', \text{reject}) \mapsto \text{Write}, \\
\delta(\text{Review}', \text{accept}) \mapsto \text{Publish}, \\
\delta(\text{Review}, \text{accept}) \mapsto \text{Publish}\}$$

$$q_0 = \{\text{Write}\}$$

$$F = \{\text{Publish}\}$$

3.1 Simple writer-review GSM version

We rewrite the simple writer-review state machine $M' = \langle Q, \Sigma, \delta, q_0, F \rangle$ in Figure 2 using Definition 2, as a GSM type. Figure 3 shows the corresponding GSM diagram. The resulting type is as follows

$$\Gamma = \langle \text{Att}, \text{EType}, \text{Stg}, \text{Tsk}, \text{Mst}, \text{Stry}, \text{Lcyc} \rangle$$

where,

$$\begin{aligned}
Att &= Att_{data} \cup Att_{stages} \cup Att_{milestones} \\
&= \emptyset \cup \{\text{Write, Review, Review}', \text{Publish}\} \\
&\cup \{\langle \text{Write, submit} \rangle, \langle \text{Review, reject} \rangle, \langle \text{Review, accept} \rangle, \\
&\quad \langle \text{Review, comment} \rangle, \langle \text{Review}', \text{reject} \rangle, \\
&\quad \langle \text{Review}', \text{accept} \rangle, \langle \text{Review}', \text{comment} \rangle\}
\end{aligned}$$

$$\begin{aligned}
EType &= EType_{inc} \cup EType_{gen} \\
&= \{\text{E:start, E:submit, E:reject, E:comment, E:accept}\} \\
&\cup \{+\text{Write, +Review, +Review}', +\langle \text{Write, submit} \rangle, \\
&\quad +\langle \text{Review, reject} \rangle, +\langle \text{Review, accept} \rangle, \\
&\quad +\langle \text{Review, comment} \rangle, +\langle \text{Review}', \text{reject} \rangle, \\
&\quad +\langle \text{Review}', \text{accept} \rangle, +\langle \text{Review}', \text{comment} \rangle\}
\end{aligned}$$

$$Stg = \{\text{Write, Review, Review}', \text{Publish}\}$$

$$Tsk = \emptyset$$

$$\begin{aligned}
Mst &= \{\langle \text{Write, submit} \rangle, \langle \text{Review, reject} \rangle, \langle \text{Review, accept} \rangle, \\
&\quad \langle \text{Review, comment} \rangle, \langle \text{Review}', \text{reject} \rangle, \\
&\quad \langle \text{Review}', \text{accept} \rangle, \langle \text{Review}', \text{comment} \rangle\}
\end{aligned}$$

$$\begin{aligned}
Stry &= \{[\mathbf{on} \text{ E:start if } \neg \text{Write} \wedge \neg \text{Review} \wedge \neg \text{Review}']\} \\
&\cup \{[\mathbf{on} \text{ E:start if Publish}]\} \\
&\cup \{[\mathbf{on} \text{ E:reject if Review}' \vee \text{Review}], \\
&\quad [\mathbf{on} \text{ E:accept if Review}' \vee \text{Review}]\} \\
&\cup \{[\mathbf{on} \text{ E:submit if Write}], [\mathbf{on} \text{ E:reject if Review}], \\
&\quad [\mathbf{on} \text{ E:accept if Review}], [\mathbf{on} \text{ E:comment if Review}], \\
&\quad [\mathbf{on} \text{ E:reject if Review}'], [\mathbf{on} \text{ E:accept if Review}'], \\
&\quad [\mathbf{on} \text{ E:comment if Review}']\} \\
&\cup \{[\mathbf{on} + \langle \text{Write, submit} \rangle], [\mathbf{on} + \langle \text{Review, reject} \rangle], \\
&\quad [\mathbf{on} + \langle \text{Review, accept} \rangle], [\mathbf{on} + \langle \text{Review, comment} \rangle], \\
&\quad [\mathbf{on} + \langle \text{Review}', \text{reject} \rangle], [\mathbf{on} + \langle \text{Review}', \text{accept} \rangle], \\
&\quad [\mathbf{on} + \langle \text{Review}', \text{comment} \rangle]\} \\
&\cup \{[\mathbf{on} + \text{Write}], [\mathbf{on} + \text{Review}], [\mathbf{on} + \text{Review}']\}
\end{aligned}$$

$$Lcyc = \langle Substages, Tasks, Submilestones, Guards, \\ Terminators, Ach, Inv \rangle$$

$$Substages^R = \emptyset$$

$$Tasks^R = \emptyset$$

$$Submilestones^R = \{ \langle Write, \langle Write, submit \rangle \rangle, \\ \langle Review, \langle Review, reject \rangle \rangle, \\ \langle Review, \langle Review, accept \rangle \rangle, \\ \langle Review, \langle Review, comment \rangle \rangle, \\ \langle Review', \langle Review', reject \rangle \rangle, \\ \langle Review', \langle Review', accept \rangle \rangle, \\ \langle Review', \langle Review', comment \rangle \rangle \}$$

$$Guards^R = \{ \langle Write, [\mathbf{on} \ E:\mathbf{start} \ \mathbf{if} \ \neg Write \ \wedge \ \neg Review \\ \wedge \ \neg Review'] \rangle, \\ \langle Write, [\mathbf{on} \ E:\mathbf{reject} \ \mathbf{if} \ Review' \ \vee \ Review] \rangle, \\ \langle Review, [\mathbf{on} \ E:\mathbf{submit} \ \mathbf{if} \ Write] \rangle, \\ \langle Review, [\mathbf{on} \ E:\mathbf{comment} \ \mathbf{if} \ Review'] \rangle, \\ \langle Review', [\mathbf{on} \ E:\mathbf{comment} \ \mathbf{if} \ Review] \rangle, \\ \langle Publish, [\mathbf{on} \ E:\mathbf{accept} \ \mathbf{if} \ Review' \ \vee \ Review] \rangle \}$$

$$Terminators^R = \{ \langle Write, [\mathbf{on} \ + \ \langle Write, submit \rangle] \rangle, \\ \langle Review, [\mathbf{on} \ + \ \langle Review, reject \rangle] \rangle, \\ \langle Review, [\mathbf{on} \ + \ \langle Review, accept \rangle] \rangle, \\ \langle Review, [\mathbf{on} \ + \ \langle Review, comment \rangle] \rangle, \\ \langle Review', [\mathbf{on} \ + \ \langle Review', reject \rangle] \rangle, \\ \langle Review', [\mathbf{on} \ + \ \langle Review', accept \rangle] \rangle, \\ \langle Review', [\mathbf{on} \ + \ \langle Review', comment \rangle] \rangle, \\ \langle Publish, [\mathbf{on} \ E:\mathbf{start} \ \mathbf{if} \ Publish] \rangle \}$$

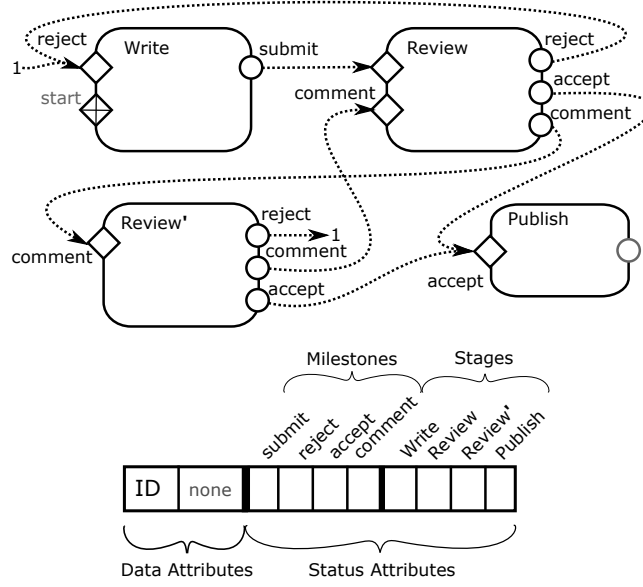


Figure 3: GSM version of the DFSM from Figure 2

$$\begin{aligned}
Ach^R = \{ & \langle \langle \text{Write}, \text{submit} \rangle, [\text{on } E : \text{submit if Write}] \rangle, \\
& \langle \langle \text{Review}, \text{reject} \rangle, [\text{on } E : \text{reject if Review}] \rangle, \\
& \langle \langle \text{Review}, \text{accept} \rangle, [\text{on } E : \text{accept if Review}] \rangle, \\
& \langle \langle \text{Review}, \text{comment} \rangle, [\text{on } E : \text{comment if Review}] \rangle, \\
& \langle \langle \text{Review}', \text{reject} \rangle, [\text{on } E : \text{reject if Review'}] \rangle, \\
& \langle \langle \text{Review}', \text{accept} \rangle, [\text{on } E : \text{accept if Review'}] \rangle, \\
& \langle \langle \text{Review}', \text{comment} \rangle, [\text{on } E : \text{comment if Review'}] \rangle \}
\end{aligned}$$

$$\begin{aligned}
Inv^R = \{ & \langle \langle \text{Write}, \text{submit} \rangle, [\text{on } + \text{Write}] \rangle, \\
& \langle \langle \text{Review}, \text{reject} \rangle, [\text{on } + \text{Review}] \rangle, \\
& \langle \langle \text{Review}, \text{accept} \rangle, [\text{on } + \text{Review}] \rangle, \\
& \langle \langle \text{Review}, \text{comment} \rangle, [\text{on } + \text{Review}] \rangle, \\
& \langle \langle \text{Review}', \text{reject} \rangle, [\text{on } + \text{Review'}] \rangle, \\
& \langle \langle \text{Review}', \text{accept} \rangle, [\text{on } + \text{Review'}] \rangle, \\
& \langle \langle \text{Review}', \text{comment} \rangle, [\text{on } + \text{Review'}] \rangle \}
\end{aligned}$$

For illustration purposes, we show the guards (*Guards*) and terminating sentries (*Terminators*) for all the stages (*Stg*) in Table 2 and the achieving (*Ach*) and invalidating sentries (*Inv*) for all the milestones (*Mst*) in Table 3. The sentry rules are enumerated from r_0 to r_{23} , as follows,

- Rule r_0 corresponds to the sentry expression defined in Equation (2.1) used in the guard of the initial stage (**Write**) which starts the machine ($q_0 \in Q$).

Table 2: GSM opening and terminating sentries for the simple writer-review

Stages (<i>Stg</i>)	Opening sentries (<i>Guards</i>)	Terminating sentries (<i>Terminators</i>)
Write	$r_0: [\mathbf{on} \text{ E:start if } \neg \text{Write} \wedge \neg \text{Review} \wedge \neg \text{Review}']$	$r_{14}: [\mathbf{on} + \langle \text{Write}, \text{submit} \rangle]$
	$r_2: [\mathbf{on} \text{ E:reject if } \text{Review}' \vee \text{Review}]$	
Review	$r_3: [\mathbf{on} \text{ E:submit if } \text{Write}]$	$r_{15}: [\mathbf{on} + \langle \text{Review}, \text{reject} \rangle]$
	$r_4: [\mathbf{on} \text{ E:comment if } \text{Review}']$	$r_{16}: [\mathbf{on} + \langle \text{Review}, \text{accept} \rangle]$
		$r_{17}: [\mathbf{on} + \langle \text{Review}, \text{comment} \rangle]$
Review'	$r_5: [\mathbf{on} \text{ E:comment if } \text{Review}]$	$r_{18}: [\mathbf{on} + \langle \text{Review}', \text{reject} \rangle]$
		$r_{19}: [\mathbf{on} + \langle \text{Review}', \text{accept} \rangle]$
		$r_{20}: [\mathbf{on} + \langle \text{Review}', \text{comment} \rangle]$
Publish	$r_6: [\mathbf{on} \text{ E:accept if } \text{Review}' \vee \text{Review}]$	$r_1: [\mathbf{on} \text{ E:start if } \text{Publish}]$

Table 3: GSM sentries for the milestones in the simple writer-review

Milestones (<i>Mst</i>)	Achieving sentries (<i>Ach</i>)	Invalidating sentries (<i>Inv</i>)
$\langle \text{Write}, \text{submit} \rangle$	$r_7: [\mathbf{on} \text{ E:submit if } \text{Write}]$	$r_{21}: [\mathbf{on} + \text{Write}]$
$\langle \text{Review}, \text{reject} \rangle$	$r_8: [\mathbf{on} \text{ E:reject if } \text{Review}]$	$r_{22}: [\mathbf{on} + \text{Review}]$
$\langle \text{Review}, \text{accept} \rangle$	$r_9: [\mathbf{on} \text{ E:accept if } \text{Review}]$	
$\langle \text{Review}, \text{comment} \rangle$	$r_{10}: [\mathbf{on} \text{ E:comment if } \text{Review}]$	
$\langle \text{Review}', \text{reject} \rangle$	$r_{11}: [\mathbf{on} \text{ E:reject if } \text{Review}']$	$r_{23}: [\mathbf{on} + \text{Review}']$
$\langle \text{Review}', \text{accept} \rangle$	$r_{12}: [\mathbf{on} \text{ E:accept if } \text{Review}']$	
$\langle \text{Review}', \text{comment} \rangle$	$r_{13}: [\mathbf{on} \text{ E:comment if } \text{Review}']$	

- Rule r_1 corresponds to the sentry set defined in Equation (2.2) used in the terminator of the terminal stages ($F \subseteq Q$, only Publish in our example).
- Rules from r_2 to r_6 corresponds to the sentry set defined in Equation (2.3) used in the guards.
- Rules from r_7 to r_{13} corresponds to the sentry set defined in Equation (2.4) used to achieve milestones.
- Rules from r_{14} to r_{20} corresponds to the sentry set defined in Equation (2.5) used in the terminators.
- Rules from r_{21} to r_{23} corresponds to the sentry set defined in Equation (2.6) used to invalidate milestones.

Figure 3 shows the resulting GSM diagram, which is visually similar to the original DFMS M' in Figure 2.

References

- [HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Addison-Wesley, 2001, p. 537 (cit. on pp. 1, 6, 7).
- [Hul+11] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. “Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles”. In: *Web Services and Formal Methods*. Ed. by M. Bravetti and T. Bultan. Vol. 6551. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 1–24. DOI: 10.1007/978-3-642-19589-1_1 (cit. on p. 1).
- [MLv16] M. A. Marin, H. Lotriet, and J. A. van der Poll. “Implementing Deterministic Finite State Machines using Guard-Stage-Milestone”. In: *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT’16)*. Johannesburg, South Africa: ACM Press, New York, USA, 2016 (cit. on p. 1).
- [OMG14] OMG. *Case Management Model and Notation, version 1.0*. Standard. Document formal/2014-05-05. OMG, 2014, p. 96 (cit. on p. 1).
- [Sav08] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. electronic edition, 2008 (cit. on pp. 1, 7).

- [Sol+13] D. Solomakhin, M. Montali, S. Tessaris, and R. de Masellis. “Verification of Artifact-Centric Systems: Decidability and Modeling Issues”. In: *Service-Oriented Computing: 11th International Conference, ICSOC 2013, Berlin, Germany, December 2-5, 2013, Proceedings*. Ed. by S. Basu, C. Pautasso, L. Zhang, and X. Fu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Verification of Artifact-Centric Systems: Decidability and Modeling Issues, pp. 252–266. DOI: [10.1007/978-3-642-45005-1_18](https://doi.org/10.1007/978-3-642-45005-1_18) (cit. on p. 1).