АНАЛИЗ АЛГОРИТМА РАБОТЫ Е-СЕТЕВОГО ПЕРЕХОДА ПРИ ТРАДИЦИОННОМ И РАСПРЕДЕЛЕННОМ МОДЕЛИРОВАНИИ С ПОМОЩЬЮ АЛГЕБРЫ ВЗАИМОДЕЙСТВУЮЩИХ ПРОЦЕССОВ

В.В.Литвинов, В.В.Казимир, И.Б.Гавсиевич
Институт проблем математических машин и систем НАН Украины
E-mail: illya@ukrsoft.net

Представим традиционную систему имитационного моделирования [1], построенную на базе Е-сетей [2, 3], как множество процессов *TRANSITION*(*t*) и процесс *SCHEDULER*(*time*, *EL*) и опишем ее формально в рамках теории CSP Ч. Хоара [4, 5], где

- *TRANSITION(t)* – процесс, реализующий работу Е-сетевого перехода $t \in T$;
- *SCHEDULER(time, EL)* – процесс, реализующий работу планировщика при традиционном последовательном моделировании.

Выделим следующие события в жизненном цикле процесса TRANSITION(*t*):

- *condition* – переход начал проверять условия срабатывания;
- *conditionFail* – условия срабатывания не выполнены;
- *conditionOk* – условия срабатывания выполнены;
- *delay* – переход вычисляет время задержки;
- *activate* – начало задержки;
- *deactivate* – окончание задержки;
- *transformation* – выполнение процедуры преобразования;
- *fire* – перемещение меток в соответствии с правилами срабатывания.

Процесс *TRANSITION*(*t*) имеет следующий алфавит

$$\alpha\, TRANSITION(t) = \{\, condition.t, conditionFail.t, conditionOk.t, delay.t, activate.t, deactivate.t, transformation.t, fire.t \,\} \quad (1)$$

Формальное определение процесса *TRANSITION*(*t*) приведено ниже

$$TRANSITION(t) = condition.t \rightarrow (\, conditionFail.t \rightarrow TRANSITION(t) \mid conditionOk.t \rightarrow delay.t\ !\ \tau \rightarrow activate.t\ ?\ time \rightarrow deactivate.t \rightarrow transformation.t \rightarrow fire.t \rightarrow TRANSITION(t) \,) \quad (2)$$

Состояния процесса *TRANSITION*(*t*) показаны на рисунке 1.

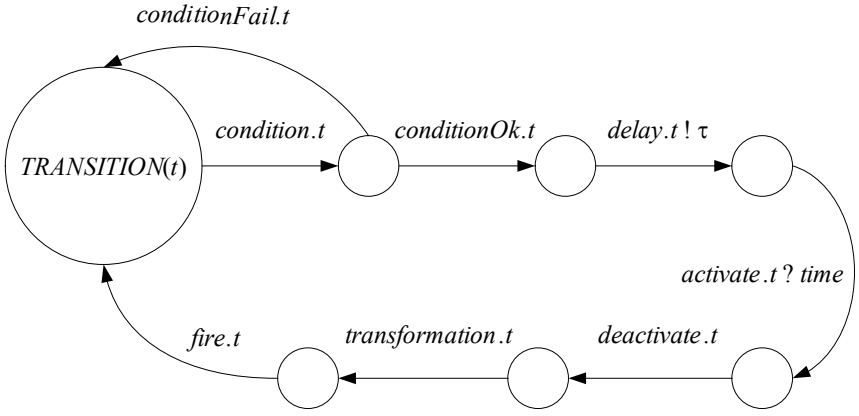

Рис. 1. Состояния процесса *TRANSITION*(*t*)

Множество параллельно работающих переходов *TRANSITIONS* можно определить

$$TRANSITIONS = \|^{t \in T} TRANSITION(t) \quad (3)$$

где каждый процесс *TRANSITION*(*t*) участвует в операции со своим алфавитом α *TRANSITION*(*t*). Поэтому алфавит процесса *TRANSITIONS* является объединением алфавитов всех *TRANSITION*(*t*)

$$\alpha\, TRANSITIONS = \cup_{t \in T}\ \alpha\, TRANSITION(t) \quad (4)$$

Выделим следующие события в жизненном цикле процесса *SCHEDULER*(*time*, *EL*):

- *condition* – переход начал проверять условия срабатывания;
- *conditionFail* – условия срабатывания перехода не выполнены;
- *conditionOk* – условия срабатывания перехода выполнены;
- *delay* – переход вычисляет время задержки;
- *activate* – начало задержки перехода;
- *nextEvent* – планировщик выбрал следующее событие;
- *deactivate* – окончание задержки перехода;

Процесс *SCHEDULER*(*time*, *EL*) имеет следующий алфавит

$$\alpha\, SCHEDULER(time, EL) = \{\ condition, conditionFail, conditionOk, delay, activate, nextEvent, deactivate\ \} \quad (5)$$

Формальное определение процесса *SCHEDULER*(*time*, *EL*) приведено ниже

$$SCHEDULER(time, EL) = SCHEDULER1(time, EL)\ \Box\ SCHEDULER2(time, EL) \quad (6)$$

$$SCHEDULER1(time, EL) = \Box_{t \notin EL}\ [\ condition.t \rightarrow (\ conditionFail.t \rightarrow SCHEDULER(time, EL) \mid conditionOk.t \rightarrow delay.t\ ?\ \tau \rightarrow activate.t\ !\ time \rightarrow SCHEDULER(time, EL \cup \{t\})\ )\ ] \quad (7)$$

$$SCHEDULER2(time, EL) = \Box_{t \in EL}\ [\ nextEvent.t \rightarrow deactivate.t \rightarrow SCHEDULER(time, EL \setminus \{t\})\ )\ ] \quad (8)$$

Состояния процесса *SCHEDULER*(*time*, *EL*) показаны на рисунке 2.

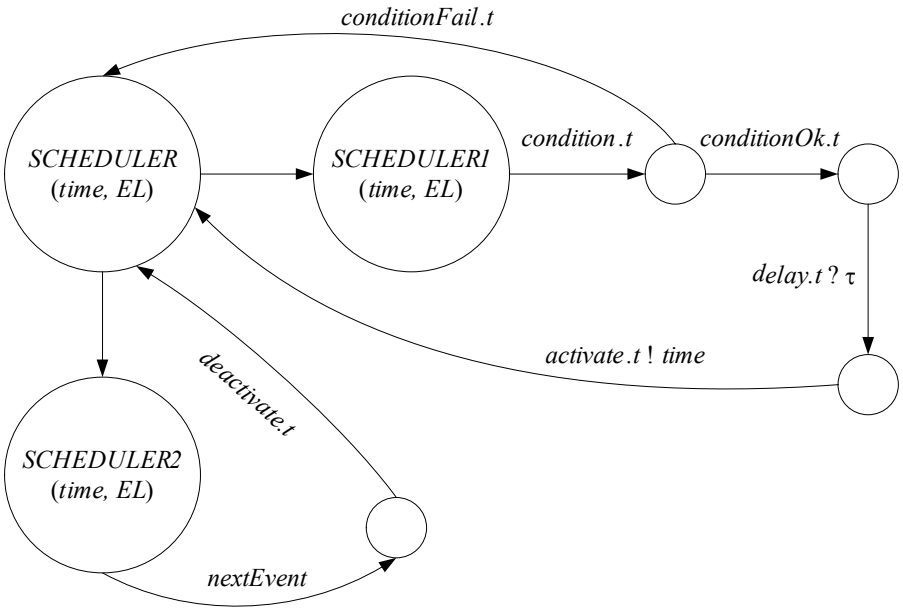

Рис. 2. Состояния процесса *SCHEDULER*(*time*, *EL*)

Запуск процесса планировщика можно определить как

$$SCHEDULE = SCHEDULER(0, \varnothing) \quad (9)$$

Алфавит процесса *SCHEDULE* совпадает с алфавитом процесса *SCHEDULER*(*time*, *EL*)

$$\alpha\, SCHEDULE = \alpha\, SCHEDULER(time, EL) \quad (10)$$

Тогда всю систему в целом можно опередить

$$SYSTEM = SCHEDULE \parallel TRANSITIONS \quad (11)$$

где процессы участвуют в операции со своими алфавитами α *SCHEDULE* и α *TRANSITIONS* соответственно.

Процесс планировщика *SCHEDULER*(*time*, *EL*) и процесс конкретного перехода *TRANSITION*(*t*) работают параллельно и синхронизируют свою работу путем одновременного участия в событиях из множества событий, полученного в результате пересечения их алфавитов.

$$\alpha\, SCHEDULER(time, EL) \cap \alpha\, TRANSITION(t) = \quad (12)$$
$$\{\, condition.t, conditionFail.t, conditionOk.t, delay.t, activate.t, deactivate.t \,\}$$

Как видно из (12) в системе существует достаточно большое количество событий планирования и синхронизации, которые связывают каждый переход и планировщик. Следовательно, параллельное и распределенное выполнения такой системы может оказаться неэффективным. В данной работе предлагается отказаться от планировщика и внести изменения в процесс, реализующий работу Е-сетевого перехода. Формальное описание процесса *MTRANSITION*(*t*), который является модификацией процесса *TRANSITION*(*t*) и следует консервативной схеме синхронизации [6 – 12] приведено ниже.

Процесса *MTRANSITION*(*t*) участвует в следующих событиях:

- *lowBound* – переход вычислил нижнюю границу ;
- *conservative* – переход начал проверку консервативного условия гарантии;
- *conservativeFail* – условие гарантии не выполняется;
- *conservativeOk* – условие гарантии выполняется;
- *condition* – переход начал проверять условия срабатывания;
- *conditionFail* – условия срабатывания не выполнены;
- *conditionOk* – условия срабатывания выполнены;
- *delay* – переход вычисляет время задержки;
- *activate* – начало задержки;
- *deactivate* – окончание задержки;
- *transformation* – выполнение процедуры преобразования;
- *fire* – перемещение меток в соответствии с правилами срабатывания;
- *nullMessages* – переход начал посылку нулевых сообщений.

Процесс *MTRANSITION*(*t*) имеет следующий алфавит

$$\alpha\, MTRANSITION(t) = \{\, lowBound.t, conservative.t, \quad (13)$$
$$conservativeFail.t, conservativeOk.t, condition.t, conditionFail.t, conditionOk.t, delay.t, activate.t, synchronize.t, deactivate.t, transformation.t, fire.t, nullMessages.t \,\}$$

Формальное определение процесса *MTRANSITION*(*t*) приведено ниже

$$MTRANSITION(t) = lowBound.t \rightarrow conservative.t \rightarrow \quad (14)$$
$$(\, conservativeFail.t \rightarrow MTRANSITION2(t)$$
$$\mid\ conservativeOk.t \rightarrow condition.t \rightarrow$$
$$(\, conditionFail.t \rightarrow MTRANSITION2(t)$$
$$\mid\ conditionOk.t \rightarrow delay.t\ !\ \tau \rightarrow$$
$$synchronize.t \rightarrow activate.t\ ?\ time \rightarrow$$

$$deactivate.t \rightarrow transformation.t \rightarrow$$
$$fire.t \rightarrow TRANSITION2(t)))$$

$$MTRANSITION2(t) = nullMessages.t \rightarrow MTRANSITION(t) \quad (15)$$

Состояния процесса *MTRANSITION*(*t*) показаны на рисунке 3.

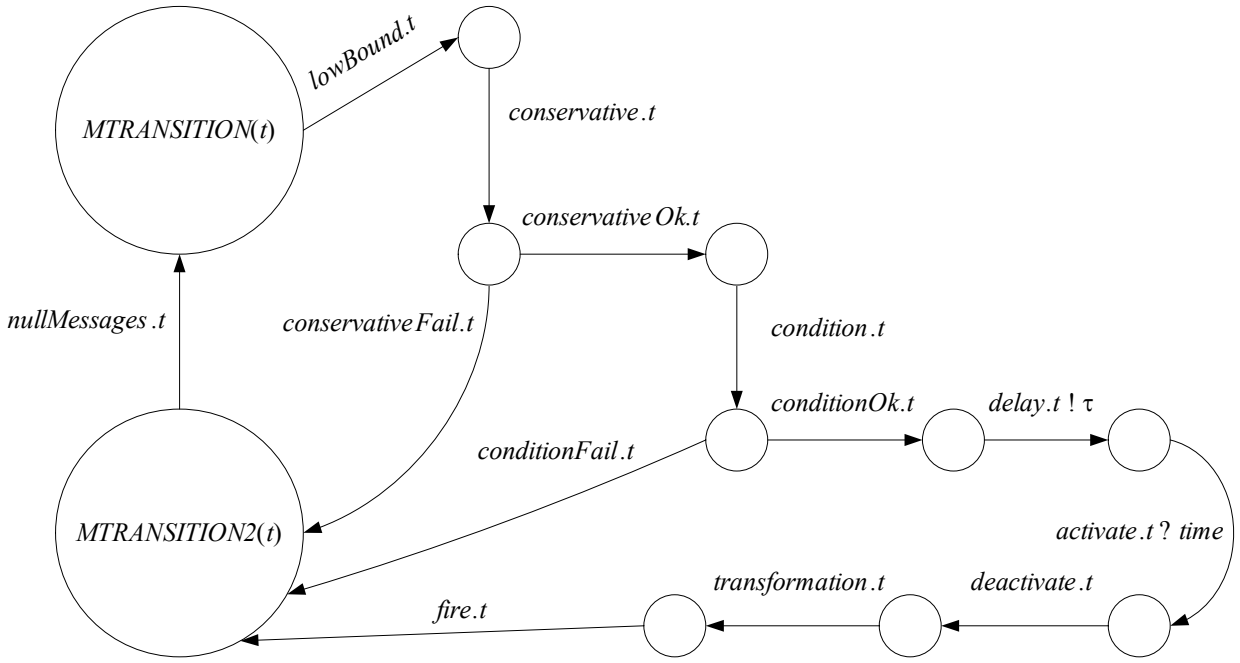

Рис. 3. Состояния процесса *MTRANSITION*(*t*)

Множество параллельно работающих переходов *MTRANSITIONS* можно определить

$$MTRANSITIONS = \|^{t \in T} MTRANSITION(t) \quad (16)$$

где каждый процесс *MTRANSITION*(*t*) участвует в операции со своим алфавитом α *MTRANSITION*(*t*). Поэтому алфавит процесса *MTRANSITIONS* является объединением алфавитов всех *MTRANSITION*(*t*)

$$\alpha\, MTRANSITIONS = \cup_{t \in T}\ \alpha\, MTRANSITION(t) \quad (17)$$

Полученные формальные описания могут быть использованы в дальнейших исследованиях, в том числе с использованием специализированный программ, ориентированных на анализ CSP-описаний.

Наиболее известной такой программой является Failures-Divergence Refinement (FDR), разработанная Formal Systems (Europe) Ltd. Она реализует 3 модели анализа: *traces*, *stable failures*, *failures/divergences*[13]. Все эти модели основаны на отношении «уточнения» (refinement). Если Q улучшает P, то это обозначается как $P \sqsubseteq Q$.

Модель traces основана на проверке отношения

$$P \sqsubseteq_T Q = traces(Q) \subseteq traces(P) \quad (18)$$

Модель failures основана на проверке отношения

$$P \sqsubseteq_F Q = failures(Q) \subseteq failures(P) \quad (19)$$

Модель Failures/Divergences основана на проверке отношения

$$P \sqsubseteq_{FD} Q = failures(Q) \subseteq failures(P) \wedge \quad (20)$$
$$divergences(Q) \subseteq divergences(P)$$

СПИСОК ЛИТЕРАТУРЫ

1. Литвинов В.В., Марьянович Т.П. Методы построения имитационных систем. – Киев: Наук. думка, 1991. – 115 с.

2. Питерсон Дж. Теория сетей Петри и моделирование систем — М.: Мир, 1984. — 264 с., ил.

3. Nutt G. J. Evaluation Nets for Computer System Performance Analysis — AFIPS FJCC: vol. 41, pt. 1, 1972. — pp. 279 - 286.

4. C. A. R. Hoare, 'Communicating Sequential Processes,' – Comm. ACM 21 (8), 666-677 (1978)

5. C. A. R. Hoare, Communicating Sequential Processes. – Prentice Hall International, 1985

6. R. E. Bryant, "Simulation of packet communications architecture computer systems," MIT-LCS-TR-188, 1977.

7. Chandy, K. M. and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. IEEE Trans. on Software Engineering, SE-5, 5 (Sept. 1979), 440---452.

8. Chandy, K. M. and Misra, J. Deadlock absence proofs for networks of communicating processes. Information Processing Lett. 9, 4, (Nov. 1979), 185-189.

9. Jayadev Misra. Distributed Discrete-Event Simulation. ACM Computing Surveys, 18(1):39--65, March 1986.

10. K. Chandy and J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Communications of the ACM, vol. 24, November 1981

11. D. A. Jefferson. Virtual Time. ACM Transactions on Programming Languages and Systems, 7(3):404--425, July 1985.

12. R. M. Fujimoto. Parallel Discrete Event Simulation. Communications of the ACM, 33(10):30--53, October 1990.

13. Failures-Divergence Refinement. FDR2 User Manual. Formal Systems (Europe) Ltd, May 2003.