УДК 681.3.06, 519.8

В.В. ЛИТВИНОВ, В.В. КАЗИМИР, І.Б. ГАВСІЄВИЧ

АЛГОРИТМ ПАРАЛЕЛЬНОГО ВИКОНАННЯ ТА СИНХРОНІЗАЦІЇ Е-МЕРЕЖІ

Abstract: *The paper is devoted to theoretical and practical problems of performing E-nets imitation models with using conservative approach. The E-nets transition's algorithm and sequential scheduler's algorithm were formalized. The parallel processes of transitions and scheduler were marked out and described on Hoare's CSP language.*
Key words: *distributed simulation, parallel program synchronization, E-nets, conservative approach, CSP.*

Анотація: *В статті розглядаються принципи паралельного виконання Е-мережевих імітаційних моделей на основі процесо-орієнтованої парадигми та створення алгоритму синхронізації паралельних ділянок в межах консервативного підходу з застосуванням методу запобігання взаємних блокувань на базі NULL-повідомлень. Для досягнення поставленої мети формалізовано алгоритми роботи Е-мережевого переходу та планувальника при традиційному послідовному моделюванні, виділено паралельні процеси переходів і планувальника, а також розроблено їхній формальний опис за допомогою процесної алгебри CSP Т. Хоара.*
Ключові слова: *розподілене імітаційне моделювання, синхронізація паралельних програм, Е-мережі, консервативна схема, CSP.*

Аннотация: *В статье рассматриваются принципы параллельного выполнения Е-сетевых имитационных моделей на основе процессо-ориентированной парадигмы и построения алгоритма синхронизации параллельных участков в рамках консервативного подхода с применением метода предотвращения взаимных блокировок на базе NULL-сообщений. Для достижения поставленной цели формализовано алгоритм работы Е-сетевого перехода и планировщика при традиционном последовательном моделировании, выделены параллельные процессы переходов и планировщика, а также разработано их формальное описание с помощью процессной алгебры CSP Т. Хоара.*
Ключевые слова: *распределенное имитационное моделирование, синхронизация параллельных программ, Е-сети, консервативная схема, CSP.*

1. Введення

Сучасні тенденції в області інформаційних технологій виносять на порядок денний питання створення розподілених систем імітаційного моделювання (РСІМ), здатних реалізувати додаткові переваги моделювання як методу дослідження складних систем [1, 2].

При створенні РСІМ необхідно враховувати:

– цільову апаратно-програмну платформу;

– математичну основу РСІМ;

– схему синхронізації паралельних ділянок;

– засоби реалізації РСІМ.

У роботі будемо орієнтуватися на розподілені апаратні платформи, що складаються із звичайних персональних комп'ютерів, об'єднаних у локальні мережі, а також на MPI-кластери. Для формалізованого опису структури й процесу функціонування в РСІМ будемо використовувати математичний апарат Е-мереж [3, 4]. В роботі була обрана базова консервативна схема синхронізації виконання паралельних процесів на базі NULL-повідомлень [5, 6]. Технологічною базою побудови РСІМ послужить специфікація CORBA [7].

Метою даної статті є розробка принципів паралельного виконання Е-мережевих імітаційних моделей на основі процесо-орієнтованої парадигми та створення алгоритму синхронізації паралельних ділянок у межах консервативного підходу з застосуванням методу запобігання взаємних блокувань на базі NULL-повідомлень.

2. Формальне визначення Е-мережі

Формально Е-мережу EN можливо визначити п'ятіркою:

$$EN = \langle P, T, Pre, Post, \mu_0 \rangle, \quad (1)$$

де P – кінцева непуста множина позицій;

T – кінцева непуста множина переходів; множини переходів і позицій не перетинаються, $T \cap P = \varnothing$;

$Pre: T \cup P \to \{0,1\}$ – вхідна функція; $Pre(t, p) = 1$ – означає, що існує дуга, яка веде з позиції $p \in P$ до переходу $t \in T$; $Pre(t, p) = 0$ – означає, що такої дуги не існує. Тоді $IN(t) = \{ p \in P \mid Pre(t, p) = 1 \}$ – множина всіх вхідних позицій переходу $t \in T$;

$Post: T \cup P \to \{0,1\}$ – вихідна функція; $Post(t, p) = 1$ – означає, що існує дуга, яка веде від переходу $t \in T$ у позицію $p \in P$; $Post(t, p) = 0$ – означає, що такої дуги не існує. Тоді $OUT(t) = \{ p \in P \mid Post(t, p) = 1 \}$ – множина всіх вихідних позицій переходу $t \in T$;

$\mu: P \to \{0,1\}$ – функція розмітки, що визначає маркування або стан позиції; $\mu(p) = 0$ – означає, що позиція $p \in P$ вільна (не містить мітку); $\mu(p) = 1$ – означає, що позиція зайнята (містить метку); μ_0 – початкова розмітка мережі.

В даній роботі розглянемо базисний набір переходів, який визначається множиною типів $D = \{\text{"T"}, \text{"F"}, \text{"J"}, \text{"X"}, \text{"Y"}\}$.

Будь-який перехід $t \in T$ можна описати:

$$p = \begin{cases} \langle d, \tau, z \rangle, & \text{для переходів типу "T", "F" або "J"} \\ \langle d, \tau, z, rx \rangle, & \text{для переходу типа "X"} \\ \langle d, \tau, z, ry \rangle, & \text{для переходу "Y",} \end{cases} \quad (2)$$

де $d \in D$ – тип переходу;

$\tau \in \Re^+$ – час затримки на переході;

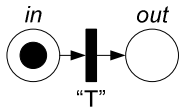
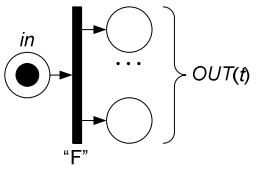
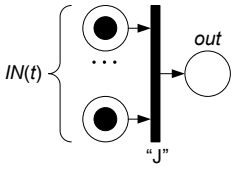
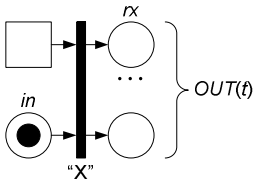
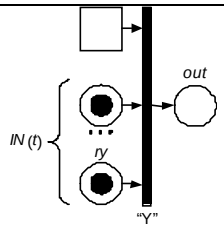
z – процедура перетворення атрибутів;

rx, ry – результат обчислення керуючої процедури. Результат обчислення може бути невизначеним – FAIL або належати множині вхідних (для переходу типу "Y") чи вихідних (для переходу типу "X") позицій, $rx \in OUT(t) \cup \{\text{FAIL}\}$, $ry \in IN(t) \cup \{\text{FAIL}\}$.

Функція $Enabled: T \to \{0,1\}$ визначає, чи виконані умови спрацьовування переходу $t \in T$; $Enabled(t) = 1$ – означає, що умови спрацьовування виконані й перехід готовий до спрацьовування; $Enabled(t) = 0$ – означає, що умови спрацьовування не виконані.

Визначення функції $Enabled(t)$ залежно від типу переходу наведено в табл. 1.

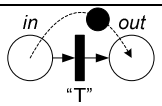
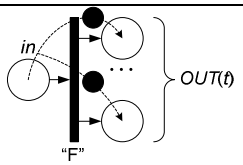
Таблиця 1. Визначення функції $Enabled(t)$ залежно від типу переходу

	$Enabled(t)=\begin{cases}1, & \text{якщо } [\mu(in)=1]\wedge[\mu(out)=0]\\ 0, & \text{інакше,}\end{cases}$ де $in\in IN(t),\ \ out\in OUT(t)$
	$Enabled(t)=\begin{cases}1, & \text{якщо } [\mu(in)=1]\wedge \\ & \wedge[\forall\, out\in OUT(t)\colon \mu(out)=0]\\ 0, & \text{інакше,}\end{cases}$ де $in\in IN(t)$
	$Enabled(t)=\begin{cases}1, & \text{якщо } [\forall\, in\in IN(t)\colon \mu(in)=1]\wedge \\ & \wedge[\mu(out)=0]\\ 0, & \text{інакше,}\end{cases}$ де $out\in OUT(t)$
	$Enabled(t)=\begin{cases}1, & \text{якщо } [\mu(in)=1]\wedge \\ & \wedge[rx\neq \mathrm{FAIL}]\wedge[\mu(rx)=0]\\ 0, & \text{інакше,}\end{cases}$ де $in\in IN(t),\ \ rx\in OUT(t)$
	$Enabled(t)=\begin{cases}1, & \text{якщо } [ry\neq \mathrm{FAIL}]\wedge[\mu(ry)=1]\wedge \\ & \wedge[\mu(out)=0]\\ 0, & \text{інакше,}\end{cases}$ де $ry\in IN(t),\ \ out\in OUT(t)$

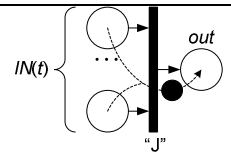
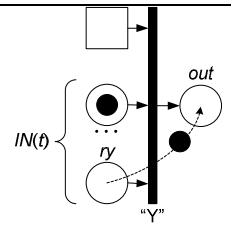
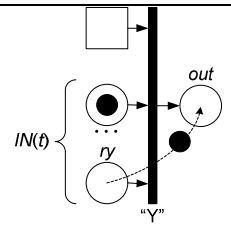
При спрацьовуванні Е-мережевого переходу відбувається переміщення міток із вхідних позицій у вихідні за правилами, визначеними для різних типів переходів.

Зміна розмітки $\mu(p)$ у нову розмітку $\mu'(p),\ \forall\, p\in[IN(t)\cup OUT(t)]$ при спрацьовуванні переходу $t\in T$ залежно від його типу, наведені в табл. 2.

Таблиця 2. Зміна розмітки мережі при спрацьовуванні переходу

	$\mu'(in)=0,$ $\mu'(out)=1$
	$\mu'(in)=0,$ $\forall\, out\in OUT(t)\colon \mu'(out)=1$

Продовження табл.2

	$\mu'(in)=0,$ $\mu'(rx)=1,$ $\forall\, out \in [OUT(t)\setminus\{rx\}]: \mu'(out)=\mu(out)$
	$\mu'(in)=0,$ $\mu'(rx)=1,$ $\forall\, out \in [OUT(t)\setminus\{rx\}]: \mu'(out)=\mu(out)$
	$\mu'(ry)=0,$ $\forall\, in \in [IN(t)\setminus\{ry\}]: \mu'(in)=\mu(in),$ $\mu'(out)=1$

Алгоритм роботи E-мережевого переходу можна описати блок-схемою, що наведена на рис. 1.

3. Планування в традиційній послідовній системі імітаційного моделювання

Алгоритм роботи планувальника, що керує системою імітаційного моделювання, яка дозволяє досліджувати моделі, описані за допомогою апарату E-мереж, приведений на рис. 2 – 4.

На блок-схемах були використовуються такі позначення:

– $time$ – поточний модельний час;

– EL – список подій; $\langle t, time\rangle$ – подія закінчення затримки переходу $t \in T$, запланована на момент часу $time$ (таким чином, у даному алгоритмі список подій містить затримані переходи);

– $\pi_k V$ – проекція вектора V на k-у вісь; якщо W – множина векторів однакової довжини, то $\pi_k W$ – множина проекцій усіх векторів з W на k-у вісь $-\pi_k W = \{\pi_k w \,|\, w \in W\}$;

– $SCHEDULER1$ – підпрограма, яка з множини пасивних (що не перебувають у стані затримки) переходів виділяє ті, в яких виконані умови спрацьовування; обчислює час затримки на переході; формує події та додає їх у список подій;

– $SCHEDULER2$ – підпрограма, яка з множини затриманих переходів виділяє ті, в яких відбулися події закінчення затримки, оновлює список подій і дозволяє спрацьовування переходу.

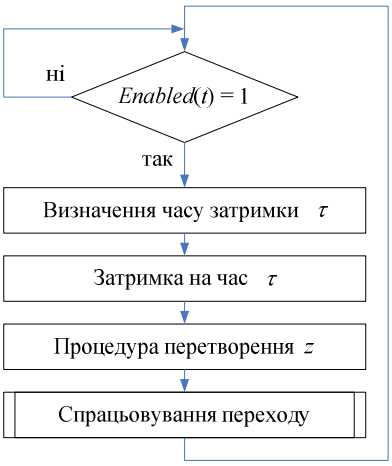


Рис. 1. Алгоритм роботи переходу $t \in T$

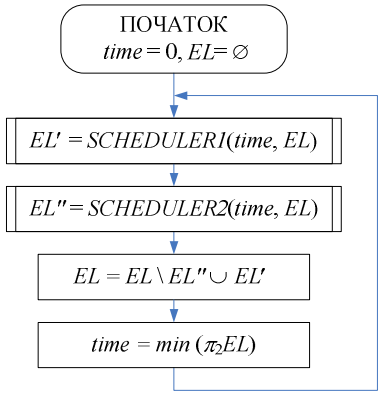


Рис. 2. Алгоритм роботи планувальника

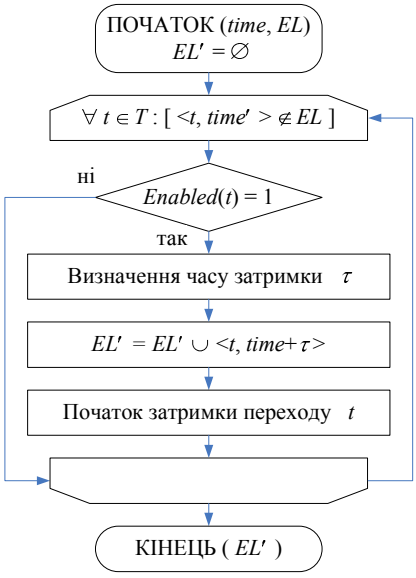


Рис. 3. Алгоритм підпрограми $SCHEDULER1$

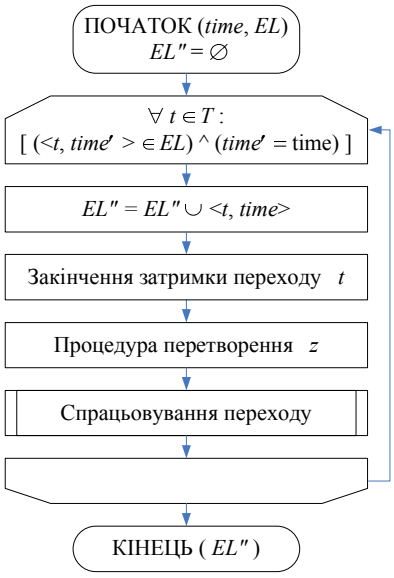


Рис. 4. Алгоритм підпрограми $SCHEDULER2$

4. Організація паралельного виконання імітаційних моделей

Представимо традиційну систему імітаційного моделювання, побудовану на базі Е-мереж, як множину процесів $TRANSITION(t)$ та процес $SCHEDULER(time,EL)$ і опишемо її формально в межах теорії CSP [8], де

– $TRANSITION(t)$ – процес, реалізуючий роботу Е-мережевого переходу $t \in T$;

– $SCHEDULER(time,EL)$ – процес, реалізуючий роботу планувальника.

Процес $TRANSITION(t)$ має такий алфавіт:

$$\alpha TRANSITION(t) = \{condition.t, conditionFail.t, conditionOk.t, delay.t, activate.t, deactivate.t, transformation.t, fire.t\}. \quad (3)$$

Формальне визначення процесу $TRANSITION(t)$ приведено нижче:

$$TRANSITION(t) = condition.t \rightarrow \quad (4)$$
$$(conditionFail.t \rightarrow TRANSITION(t)$$
$$\mid conditionOk.t \rightarrow delay.t\,!\tau \rightarrow$$
$$activate.t\,?\,time \rightarrow deactivate.t \rightarrow$$
$$transformation.t \rightarrow fire.t \rightarrow TRANSITION(t)).$$

Стани процесу $TRANSITION(t)$ показані на рис. 5.

Множину паралельно працюючих переходів $TRANSITIONS$ можна визначити:

$$TRANSITIONS = \|^{t \in T}\ TRANSITION(t). \quad (5)$$

Процес $SCHEDULER(time,EL)$ має такий алфавіт:

$$\alpha SCHEDULER(time,EL) = \{condition,conditionFail, \quad (6)$$
$$conditionOk,delay,activate,nextEvent,deactivate\}.$$

Формальне визначення процесу $SCHEDULER(time,EL)$ приведено нижче:

$$SCHEDULER(time,EL) = SCHEDULER1(time,EL)\ \square \quad (7)$$
$$SCHEDULER2(time,EL);$$
$$SCHEDULER1(time,EL) = |_{t \notin EL}\ [condition.t \rightarrow \quad (8)$$
$$(conditionFail.t \rightarrow SCHEDULER(time,EL)$$
$$\mid conditionOk.t \rightarrow delay.t\,?\,\tau \rightarrow$$
$$activate.t\,!time \rightarrow SCHEDULER(time,EL \cup \{t\}))];$$
$$SCHEDULER2(time,EL) = |_{t \in EL}\ [nextEvent.t \rightarrow \quad (9)$$
$$deactivate.t \rightarrow SCHEDULER(time,EL \setminus \{t\})].$$

Стани процесу $SCHEDULER(time,EL)$ показані на рис. 6.

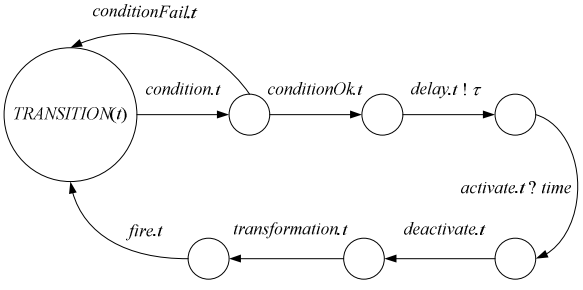


Рис. 5. Стани процесу $TRANSITION(t)$

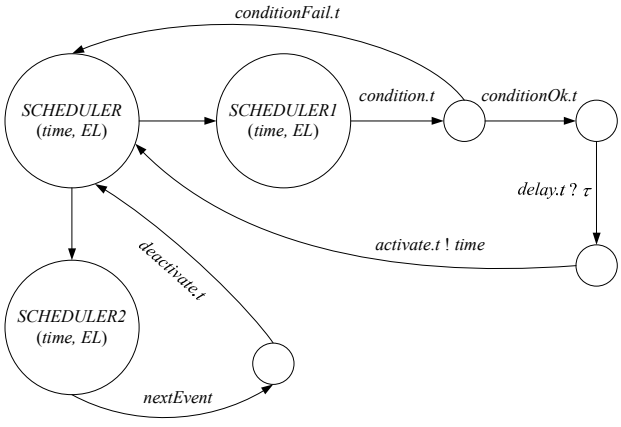


Рис.6. Стани процесу $SCHEDULER(time,EL)$

Запуск процесу планувальника можна визначити як

$$SCHEDULE = SCHEDULER(0,\varnothing). \tag{10}$$

Тоді всю систему в цілому можна визначити:

$$SYSTEM = SCHEDULE \| TRANSITIONS. \tag{11}$$

Процес планувальника $SCHEDULER(time,EL)$ і процес окремого переходу $TRANSITION(t)$ працюють паралельно і синхронізують свою роботу шляхом одночасної участі в подіях з множини подій, яка отримана в результаті перетину їхніх алфавітів:

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

Використання традиційного планувальника, заснованого на глобальному списку подій при описаній вище організації паралельного виконання, не є ефективним через велику кількість повідомлень між планувальником і окремим переходом.

У даній роботі пропонується не використовувати глобальне керування модельним часом, а реалізувати планування в кожному переході локально і розробити спеціальні механізми синхронізації модельного часу між переходами.

5. Модифікації Е-мереж

На основі консервативної схеми синхронізації та методу запобігання взаємних блокувань на базі NULL-повідомлень розробимо модифікації Е-мереж для синхронізації виконання паралельних ділянок.

1. Введемо функцію Lvt – локальний модельний час переходу $t \in T$:

$$Lvt : T \to \Re^+. \tag{13}$$

2. Введемо функцію Ts – часова відмітка останньої події позиції $p \in P$: одержання мітки, видалення мітки, одержання NULL-повідомлення (див. далі):

$$Ts : P \to \Re^+. \tag{14}$$

3. Введемо функцію $LowBound : T \to \Re^+$ – нижня часова границя з усіх вхідних повідомлень (повідомлень, отриманих від вхідних позицій) переходу $t \in T$:

$$LowBound(t) = \min[Ts(in)], \quad \forall in \in IN(t). \tag{15}$$

4. Розіб'ємо множину вхідних позицій $IN(t)$ переходу $t \in T$ на дві непересічні підмножини – вхідних вільних $IN_F(t)$ і вхідних $IN_B(t)$ зайнятих позицій:

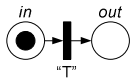
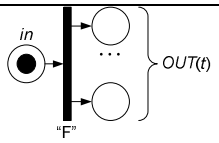
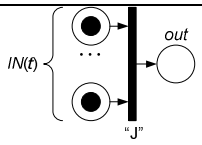
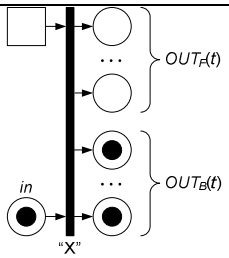
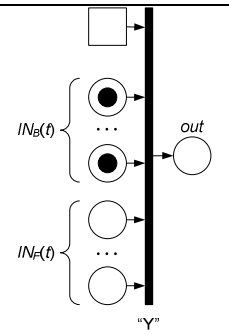
$$IN_B(t) = \{p \in IN(t) \mid \mu(p) = 1\},$$
$$IN_F(t) = \{p \in IN(t) \mid \mu(p) = 0\}, \tag{16}$$
$$IN_B(t) \cap IN_F(t) = \varnothing, \quad IN_B(t) \cup IN_F(t) = IN(t).$$

5. Розіб'ємо множину вихідних позицій $OUT(t)$ переходу $t \in T$ на дві непересічні підмножини – вихідних вільних $OUT_F(t)$ і вихідних $OUT_B(t)$ зайнятих позицій:

$$OUT_B(t) = \{ p \in OUT(t) \mid \mu(p) = 1 \},$$
$$OUT_F(t) = \{ p \in OUT(t) \mid \mu(p) = 0 \}, \qquad (17)$$
$$OUT_B(t) \cap OUT_F(t) = \varnothing, \quad OUT_B(t) \cup OUT_F(t) = OUT(t).$$

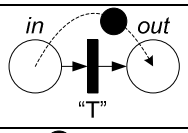
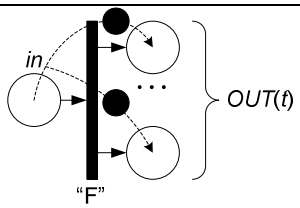
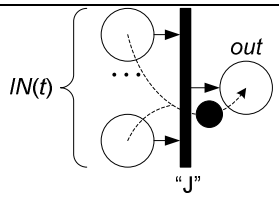
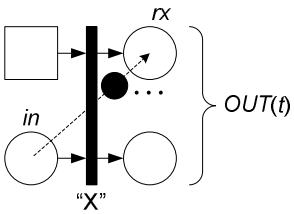
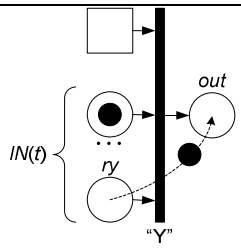
6. Введемо функцію $Conservative : T \to \{0,1\}$ – умова гарантії відсутності наведених помилок (causality errors – помилки, що можуть виникати у разі паралельного імітаційного моделювання) у майбутньому для переходу $t \in T$; $Conservative(t) = 1$ – означає, що умова виконується; $Conservative(t) = 0$ – означає, що умова не виконується. Визначення функції $Conservative(t)$ залежно від типу переходу наведено в табл. 3.

Таблиця 3. Визначення функції $Conservative(t)$ залежно від типу переходу

	$Conservative(t) = 1$
	$Conservative(t) = 1$
	$Conservative(t) = 1$
	$Conservative(t) = \begin{cases} 1, & \text{якщо } [\forall out \in OUT_B(t): Ts(out) > MIN_{OF}(t)] \\ 0, & \text{інакше,} \end{cases}$ де $MIN_{OF}(t) = \min[Ts(out)], \quad \forall out \in OUT_F(t)$
	$Conservative(t) = \begin{cases} 1, & \text{якщо } [\forall in \in IN_F(t): Ts(in) > MIN_{IB}(t)] \\ 0, & \text{інакше,} \end{cases}$ де $MIN_{IB}(t) = \min[Ts(in)], \quad \forall in \in IN_B(t)$

7. Введемо функцію $ActivateTS: T \to \Re^+$ – момент часу початку затримки переходу $t \in T$. Визначення функції $ActivateTS(t)$ залежно від типу переходу приведено в табл. 4.

Таблиця 4. Визначення функції $ActivateTS(t)$ залежно від типу переходу

	$ActivateTS(t) = \max[Ts(in), Ts(out)]$, де $in \in IN(t)$, $out \in OUT(t)$
	$ActivateTS(t) = \max(Ts(in), [\max(Ts(out)), \forall out \in OUT(t)])$, де $in \in IN(t)$
	$ActivateTS(t) = \max([\max(Ts(in)), \forall in \in IN(t)], Ts(out))$, де $out \in OUT(t)$
	$ActivateTS(t) = \max[Ts(in), Ts(rx)]$, де $in \in IN(t)$, $rx \in OUT(t)$
	$ActivateTS(t) = \max[Ts(ry), Ts(out)]$, де $ry \in IN(t)$, $out \in OUT(t)$

8. Внесемо зміну в спрацьовування переходу таким чином, щоб при зміні маркування позиції $p \in P$ обновлялася її часова відмітка $Ts(p)$. Зміна розмітки $\mu(p)$ у нову розмітку $\mu'(p)$, а також зміна часової відмітки $Ts(p)$ на нову часову відмітку $Ts'(p)$, $\forall p \in [IN(t) \cup OUT(t)]$ при спрацьовуванні переходу $t \in T$ залежно від його типу наведені в табл. 5.

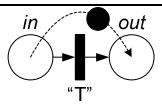
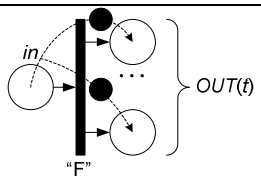
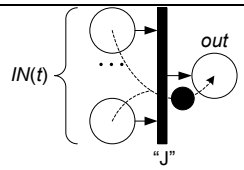
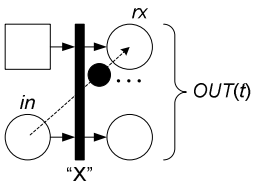
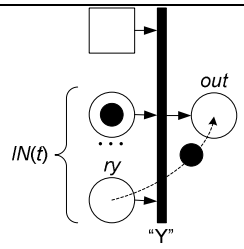
9. Для запобігання взаємних блокувань (deadlock) введемо процедуру посилання нульових (NULL) повідомлень, яку формально можна описати:

$$\forall in \in IN_B(t): [\mu'(in) = \mu(in), Ts'(in) = Lvt(t)]; \quad (18)$$

$$\forall out \in OUT_F(t): [\mu'(out) = \mu(out), Ts'(out) = Lvt(t)].$$

10. Модифікований алгоритм роботи Е-мережевого переходу $t \in T$ можна описати блок-схемою, що приведена на рис. 7.

Таблиця 5. Зміни розмітки мережі та часових відміток позицій при спрацьовуванні переходу залежно від його типу

	$\mu'(in)=0, \quad Ts'(in)=Lvt(t);$ $\mu'(out)=1, \quad Ts'(out)=Lvt(t)$
	$\mu'(in)=0, \quad Ts'(in)=Lvt(t);$ $\forall out \in OUT(t): [\, \mu'(out)=1, Ts'(out)=Lvt(t)\,]$
	$\forall in \in IN(t): [\, \mu'(in)=0, Ts'(in)=Lvt(t)\,];$ $\mu'(out)=1, \quad Ts'(out)=Lvt(t)$
	$\mu'(in)=0, \quad Ts'(in)=Lvt(t);$ $\mu'(rx)=1, \quad Ts'(rx)=Lvt(t);$ $\forall out \in [\, OUT(t) \setminus \{rx\}\,]:$ $[\, \mu'(out)=\mu(out), Ts'(out)=Ts(t)\,]$
	$\mu'(ry)=0, \quad Ts'(ry)=Lvt(t);$ $\forall in \in [\, IN(t) \setminus \{ry\}\,]:$ $[\, \mu'(in)=\mu(in), Ts'(in)=Ts(in)\,];$ $\mu'(out)=1, \quad Ts'(out)=Lvt(t)$

Тепер уся система складається з множини процесів $MTRANSITION(t)$, реалізуючих роботу модифікованого Е-мережевого переходу $t \in T$. Цей процес має алфавіт:

$$\alpha\, MTRANSITION(t) = \{ lowBound.t, conservative.t, \quad (19)$$
$$conservativeFail.t, conservativeOk.t, condition.t,$$
$$conditionFail.t, conditionOk.t, delay.t, activate.t,$$
$$deactivate.t, transformation.t, fire.t, nullMessages.t \ \}.$$

Формальне визначення процесу $MTRANSITION(t)$ наведено нижче:

$$MTRANSITION(t) = lowBound.t \to conservative.t \to \quad (20)$$
$$(\, conservativeFail.t \to MTRANSITION2(t)$$
$$\mid conservativeOk.t \to condition.t \to$$
$$(conditionFail.t \to MTRANSITION2(t)$$
$$\mid conditionOk.t \to delay.t\,!\,\tau \to$$
$$activate.t\,?\,time \to deactivate.t \to$$

$$transformation.t \rightarrow fire.t \rightarrow MTRANSITION2(t));$$

$$MTRANSITION2(t) = nullMessages.t \rightarrow MTRANSITION(t). \quad (21)$$

Стани процесу $MTRANSITION(t)$ наведені на рис. 8.

Множину паралельно працюючих переходів $MTRANSITIONS$ можна визначити:

$$MTRANSITIONS = \|^{t \in T} MTRANSITION(t). \quad (22)$$

Розроблені у роботі формальні описи на CSP також можуть бути використані для аналізу і верифікації за допомогою спеціалізованих програм, таких як Process Behavior Exploration (ProBE) та Failures-Divergence Refinement (FDR), розроблених Formal Systems (Europe) Ltd [9].

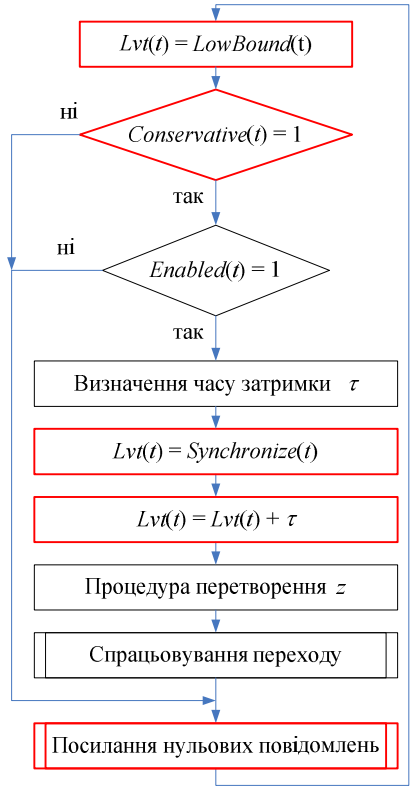


Рис. 7. Модифікований алгоритм роботи Е-мережевого переходу $t \in T$

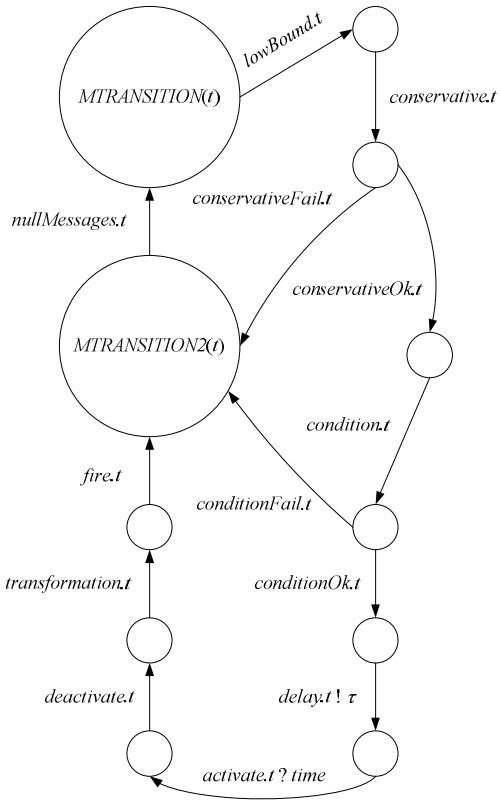


Рис. 8. Стани процесу MTRANSITION(t)

6. Висновок

В роботі розроблені принципи паралельного виконання Е-мережевих імітаційних моделей на основі процесо-орієнтованої парадигми та створення алгоритму синхронізації паралельних ділянок в межах консервативного підходу з застосуванням методу запобігання взаємних блокувань на базі NULL-повідомлень.

Для досягнення поставленої мети формалізовано алгоритми роботи Е-мережевого переходу та планувальника при традиційному послідовному моделюванні, виділено паралельні процеси переходів і планувальника, а також розроблено їхній формальний опис за допомогою процесної алгебри CSP Т. Хоара.

Алгоритм паралельного виконання Е-мережевих моделей може бути реалізований в інших розподілених системах імітаційного моделювання. Формальні описи паралельних процесів переходів і планувальника мовою CSP можуть бути застосовані в подальших розробках і верифікації паралельних програм.

СПИСОК ЛІТЕРАТУРИ

1. Литвинов В.В., Марьянович Т.П. Методы построения имитационных систем. – Киев: Наукова думка, 1991. – 115 с.
2. Гусев В.В., Марьянович Т.П., Пепеляев В.А. Основные принципы разработки системы технологической поддержки распределенного моделирования // Проблемы программирования. – 2000. – № 1–2. Спец.выпуск. – С. 620–625.
3. Казимир В.В., Демшевська Н.В., Азарова А.О. Мова специфікацій імітаційного моделювання та методика її застосування // Вісник Вінницького політехнічного інституту. – 2000. – № 1. – С. 67–71.
4. Казимир В.В. Моделирование синтетического окружения для реактивных систем // Математичне моделювання. – 2003. – № 2(10). – С. 24–32.
5. Bryant R.E. Simulation of Packet Communication Architecture Computer Systems. Computer Science Laboratory. – Cambridge, Massachusetts: Massachusetts Institute of Technology, 1977. – 120 p.
6. Chandy K.M., Misra J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs // IEEE Transactions on Software Engineering. – 1978. – N 5. – P. 440–452.
7. Object Management Group, The Common Object Request Broker: Architecture and Specification. – Object Management Group, 2000. – 948 p.
8. Hoare C. Communicating sequential processes // Comm. ACM. – 1978. – N 21. – P. 666–677.
9. Roscoe A.W. The Theory and Practice of Concurrency. – Prentice Hall, 1997. – 450 p.