УДК 004.414.23.042

I.D. Perepelytsya, G.M. Zholtkevych

V.N. Karazin Kharkiv National University, Kharkiv

ON SOME CLASS OF MATHEMATICAL MODELS FOR STATIC ANALYSIS OF CRITICAL-MISSION ASYNCHRONOUS SYSTEMS

Abstract. *A mathematical model of asynchronous software system is considered in the paper. This model bases on the notion abstract finite pre-machine which generalizes the notion abstract finite automaton. In contrast to generally accepted models the model proposed in the paper makes possible to specify more complex system behaviour than it is provided by finite automata models. Specifically, live-lock anomaly can be specified using the notion pre-machine. Authors adduce the criterion of live-lock existence and illustrate it by example.*

Keywords: *critical-mission software, asynchronous software system, static analysis, live-lock, queue explosion*

1. Introduction

Nowadays nature of a developing cycle for information processing systems (IPS) has changed essentially. Now practically there exist no software projects that start from scratch. As a rule modern development processes for IPS are processes of systems reengineering. Therefore, architectural solutions for modern IPS should provide flexibility of such systems without loss of their integrity.

As known, integrity for software system is supported by interaction of flows. Each of the flows belongs to only one of two classes: the class of control flows and the class of data flows [1].

Control flows are realised by transfer of control from one software component to another. This mechanism results in high coupling of software components [2]. High coupling opposes extracting a software component for its modification and testing outside of the system. So to increase flexibility of the system one can try to decrease system coupling.

To realise this idea developers use architectural styles that consider a system as a complex of independent components with an asynchronous interaction (see, for example event-driven architecture [3], message-driven architecture [4], data-driven architecture [5], service-oriented architecture [6] and so on). Usage this approach provides coupling decreasing but increasing complexity of data flows between components is payment for this.

Let's stress that systems of all enumerated above kinds include architectural layers which realise the event-driven strategy.

Therefore development models and methods for static analysis of event-driven systems is actual problem in the field of software engineering.

Developers of critical-mission software systems are especially in need of software tools for studying the behaviour of such systems and ensure the trustworthiness of software development processes, so this paper deals with some theoretical and applied questions of developing software tools for static analysis of components of asynchronous systems.

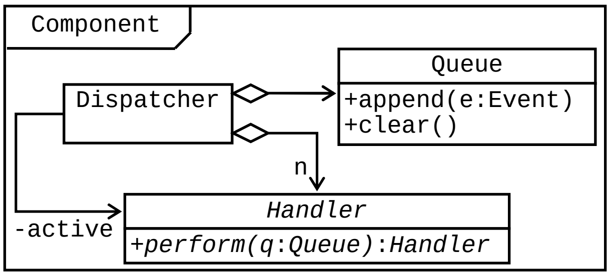

Fig. 1. Static model of a software component

2. Modelling components of asynchronous software systems

In this section we specify the generalized static model of a software component in the UML notation [7, 8] (cf. fig. 1). This model specification bases on the suggestion that the component has asynchronous type of

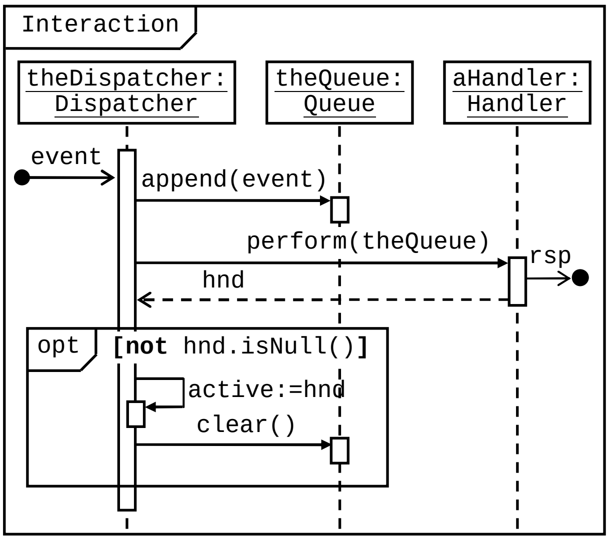

Fig. 2. Dynamic model of a software component

interaction with other parts of the software system.

So, each component of the such kind has the next principal units: the dispatcher, the queue, and handlers. Their responsibilities and intaraction is shown in fig. 2, namely,

- `theDispatcher` provides collaboration between old units;
- `theQueue` is used for accumulating received events;
- objects named `aHandler` provide handling of events queue.

Methods which are referenced on in fig. 1 should be realized to support the next functionality:

- `Queue::append(e:Event)` appends an event into the events queue;
- `Handler::perform(q:Queue)` provides specific handling of the events queue;
- `Queue::clear()` clears the events queue after successful handling.

3. Pre-machine as mathematical model of software component

Mathematical model of the software component with described behaviour was introduced in [10]. Detailed mathematical background one can find in [11]. In [11] authors refined this model.

We shall use the next notion.

For sets X and Y by $[X \to Y]$ the set of total maps from X into Y is denoted, and by $(X \to Y)$ the set of partial maps from X into Y is denoted.

In the last case we shall use the notation $f(x) = \varnothing$ to denote that x is not lie in the domain of f and the notation $f(x) \neq \varnothing$ to denote that x lies in the domain of f.

<u>Definition 1.</u> Let Q be a finite set of states, Σ be a finite alphabet of events, T be a transition function which is an element of $\left(Q \times \Sigma^* \to Q\right)$, then the triple $\mathbf{P} = (Q, \Sigma, T)$ is called an abstract finite pre-machine if the next conditions hold:

1) $\varnothing \neq T(q,\varepsilon) = q$ for all $q \in Q$;
2) for any $q \in Q$, $u, v \in \Sigma^*$ from $T(q,u) \neq \varnothing$ and $T(T(q,u),v) \neq \varnothing$ it follows that $T(q,w) \neq \varnothing$ and $T(q,w) = T(T(q,u),v)$ where $w = uv$;
3) for any $q \in Q$, $u, v \in \Sigma^*$ and $w = uv$ from $T(q,u) \neq \varnothing$ and $T(q,w) \neq \varnothing$ it follows that $T(T(q,u),v) \neq \varnothing$ and $T(T(q,u),v) = T(q,w)$.

With a finite pre-machine $\mathbf{P} = (Q, \Sigma, T)$ one can associate the next languages over the alphabet Σ:

- for $q \in Q$ the language $\mathrm{Out}(q)$ is the set of words $w \in \Sigma^+$ such that $T(q,w) \neq \varnothing$ and $T(q,u) = \varnothing$ if $u \in \Sigma^+$ and $w = uv$ for some $v \in \Sigma^+$;
- $\mathrm{Jump}(q,q') = \{ w \in \mathrm{Out}(q) \mid T(q,w) = q' \}$.

As shown in [11], these languages play a significant part to study pre-machines.

Each among these languages is a prefix code.

This property is the basic for such languages.

<u>Example 1.</u> Consider a system of two processes (P_1 and P_2) that concurrent try to use some critical resource with exclusive access. To balance processes' access to the resource assigning dynamic priorities is used. Let r_1 (r_2) be an event “Process P_1 (P_2) has requested access to the resource”. To decide either process P1 or process P2 has priority we use the next solver which is presented in fig. 3.

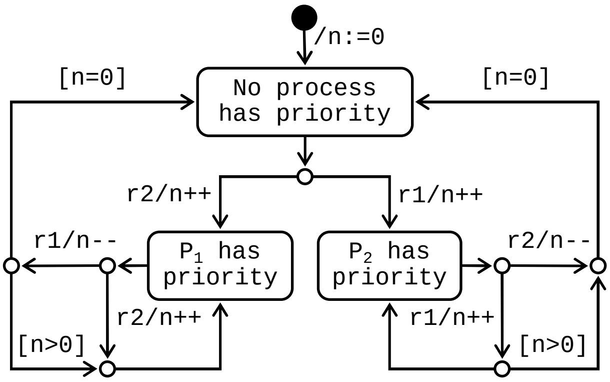

Fig. 3. Component for assignment of priorities

Let’s describe this state/chart machine [7, 8] by such finite pre-machine:

$Q = \{q_0, q_1, q_2\}$, where

q_0 is the state “no process has priority”;

q_1 is thee state “ P_1 has priority”;

q_2 is thee state “ P_2 has priority”.

$\Sigma = \{r_1, r_2\}$, where

r_1 is the event “Process P_1 has requested access to the resource”;

r_2 is the event “Process P_2 has requested access to the resource”.

$\mathrm{Jump}(q_0, q_1) = r_2$;

$\mathrm{Jump}(q_0, q_2) = r_1$;

$$\mathrm{Jump}(q_1, q_0) = \\ \left\{ w \in \Sigma^+ \mid s_w^{r_1}(w) = s_w^{r_2}(w), \\ s_k^{r_1}(w) < s_k^{r_2}(w), k = 1, \ldots, |w| - 1 \right\};$$

$$\mathrm{Jump}(q_2, q_0) = \\ \left\{ w \in \Sigma^+ \mid s_w^{r_2}(w) = s_w^{r_2}(w), \\ s_k^{r_2}(w) < s_k^{r_2}(w), k = 1, \ldots, |w| - 1 \right\},$$

where $s_k^r(w)$ is equal to a number of detecting the event r among the first k events.

As one can check all languages:

- $\mathrm{Out}(q_0) = \mathrm{Jump}(q_0, q_1) \cup \mathrm{Jump}(q_0, q_2)$,
- $\mathrm{Out}(q_1) = \mathrm{Jump}(q_1, q_0)$,
- $\mathrm{Out}(q_2) = \mathrm{Jump}(q_2, q_0)$

are prefix codes.

4. Pre-machine's live-lock recognizing

Generalisation finite machines, based on formalising model of a data accumulation process, permits to model anomalies which are classified as live-locks.

In our case live-lock is a condition when a process cannot leave the current state because it receives additional external requests for service during handling the current queue of requests. Live-lock leads to the effect of queue's explosion. It means that the queue of unserved requests is increasing unbounded.

Let's give the formal definitions.

We consider some finite pre-machine $\mathbf{P} = (Q, \Sigma, T)$ where as usual by Q denote its state set, by Σ denote its alphabet, and by $T \in (Q \times \Sigma \to Q)$ denote its transition function.

For each $q \in Q$ denote by $\mathrm{Out}(q)$ the set of all words $w \in \Sigma^+$ such that

1) $T(q, w)$ is defined;

2) $T(q, u)$ is undefined if $u \in \Sigma^+$ and $w = uv$ for some $v \in \Sigma^+$.

By Σ^ω denote the set of all right-side infinite sequences.

For the word $w \in \Sigma^+$ by $C(w)$ denote the subset of Σ^ω such that $\alpha \in \Sigma^\omega$ is a member of $C(w)$ if and only if for some $\beta \in \Sigma^\omega$ the equality $\alpha = w\beta$ holds.

It's evident that $C(w_1) \cap C(w_2) = \varnothing$ if and only if w_1 is not non-empty prefix of w_2 and vice-versa.

In the case when w_1 is some non-empty prefix of w_2 we have the inclusion: $C(w_2) \subset C(w_1)$.

Definition 2. Let $q \in Q$ then we shall say that live-lock is possible in state q if there exists $\alpha \in \Sigma^\omega$ such that for any $w \in \Sigma^+$ which is a prefix of α condition (4.1) holds:

$$T(q, w) \text{ is undefined.} \qquad (1)$$

Really, under conditions of Definition 2 there exists a sequence of events, namely the sequence α, such that any its initial part cannot ensure any possibility for the system to go out from the state q.

Theorem 1. Let $\mathbf{P} = (Q, \Sigma, T)$ be a finite pre-machine with Q as the state set, Σ as the alphabet, and $T \in (Q \times \Sigma \to Q)$ as the transition function. $\mathbf{P}$ has no live-lock in a state $q \in Q$ if and only if the next conditions hold:

1) the set $\mathrm{Out}(q)$ is finite;

2) equality (2) holds

$$\sum_{w \in \mathrm{Out}(q)} \frac{1}{2^{|w|}} = 1. \qquad (2)$$

We don't know any elementary proof of Theorem 1. Its proof, which we have found, is simple and smart, but it bases on knowledge of some facts from general topology (cf. for example [9]).

Example 2. This example deals with live-lock analysis of the finite pre-machine which described in Example 1.

Easy to see that the state q_0 satisfies the hypothesis of Theorem 1, so live-lock is impossible in this state. But in the states q_1 and q_2 live-lock is possible. For example the sequences r_2^ω and r_1^ω satisfy to Definition 2 for states q_1 and q_2 respectively.

In this case to evaluate probability of live-lock in states q_1 and q_2 is very interest. Consider the case of the state q_1.

To do it note that for each word $w \in \{r_1, r_2\}^+$ we can associate uniquely the finite sequence $x_1, \ldots, x_n$ where n is length of the word w, $x_i = 1$ if the i-th event in w equals r_2 and $x_i = -1$ $x_i = 1$ if the i-th event in w equals r_1. So $s_k = x_1 + \ldots + x_k$ equals counter value in the state q_1, when $k = 1, \ldots, |w|$. It is evidently, that for $k = 1, \ldots, |w|$ inequalities $s_k > 0$ provides that the pre-machine is in the state q_1. Using technique from [13, ch. 3, § 1] one can obtain the next asymptotic estimate

$$\Pr(W_n(q_1)) \sim \frac{1}{\sqrt{2\pi n}}, \qquad n \to \infty, \qquad (3)$$

where $W_n(q_1)$ is the condition "the pre-machine is in the state q_1 for the last n steps".

The case of the state q_2 gives the same estimate:

$$\Pr(W_n(q_2)) \sim \frac{1}{\sqrt{2\pi n}}, \qquad n \to \infty, \qquad (4)$$

where $W_n(q_2)$ is the condition "the pre-machine is in the state q_2 for the last n steps".

So probability of live-lock equals zero.

From formulae (3) and (4), unfortunately, it is follows that mean pre-machine residence time in each of the states q_1 and q_2 is not bounded.

5. Conclusion

Problems of mathematical modelling of the important class of software systems, namely, asynchronous software systems were considered in the paper.

Authors generalized the commonly accepted that based on the notion abstract finite machine and used the notion abstract finite pre-machine.

The example of using pre-machines for modelling some program component, the solver for assignment priority, is described in the paper.

This generalization permits to model such anomaly as live-lock.

Theorem 1 establishes the criterion for the absence of live-lock in the pre-machine state.

For the pre-machine from the example live-lock possibility was analysed. For states in which live-lock is possible probability of it was estimated. Authors obtained the exact estimate. It has ensured to establish that mean residence time in the state does not exist.

Bibliography

1. Wirth N. Algorithms + Data Structures = Programs / N. Wirth. – Prentice-Hall, 1975. – 366 p.

2. Faison T. Event-Based Programming: Taking Events to the Limits / T. Faison. – Apress, 2006. – 670 p.

3. Chandy M.K. Event-Driven Applications: Costs, Benefits and Design Approaches / M.K. Chandy // Gartner Application Integration and Web Services Summit, San Diego, CA, June 2006. – San Diego, CA: California Institute of Technology, 2006.

4. Curry E. Message-Oriented Middleware / E. Curry // Middleware for Communications, ed. Q.H. Mahmoud. – Chichester, England: Wiley & Sons, 2004. – P. 1 – 28.

5. Treleaven P.C. Data-Driven and Demand-Driven Computer Architecture / P.C. Treleaven, D.R. Brownbridge, R.P. Hopkins. – J. ACM Comp. Surv. – V. 14, N 1, 1982. – P. 93 – 143.

6. Bell M. Service-Oriented Modeling: Service Analysis, Design, and Architecture / M. Bell. – Hoboken, NJ: Wiley & Sons, 2008. – 366 p.

7. OMG Unified Modeling Language™ (OMG UML), Infrastructure. – Version 2.3. – http://www.omg.org/spec/UML/2.3/ Infrastructure/PDF.

8. OMG Unified Modeling Language™ (OMG UML), Superstructure Version 2.3. – http://www.omg.org/spec/UML/2.3/ Superstructure/PDF.

9. Kelley J.L. General topology / J.L. Kelley. – Princeton, NJ: D. Van Nostrand Company, Inc., 1957. – 423 p.

10. Novikov B. Pre-automata as Mathematical Models of Event Flows Recognisers / B. Novikov, I. Perepelytsya, G. Zholtkevych // V. Ermolayev et al. (eds.) Proc. 7-th Int. Conf. ICTERI 2011, Kherson, Ukraine, May 4-7, 2011. – CEUR-WS.org/Vol-716, ISSN 1613-0073, 2011. – P. 41 – 50.

11. Dokuchaev M. Partial actions and automata / M. Dokuchaev, B. Novikov, G. Zholtkevych. – Algebra and Discrete Mathematics. − 2011. – V. 11, No 2. – P. 51 – 63.

12. Novikov B. Derivatives Series of Finite State Pre-Machines / B. Novikov, I. Perepelytsya, G. Zholtkevych // Specification and Verification of Hybrid Systems. – Proc 1st Int. Seminar, Kyiv, Ukraine, October 10 – 12, 2011. – T. Shevchenko Nat. Univ. in Kyiv, Paul Sabatier Univ. Toulouse, State Found for Fund. Res. Ukraine, 2011. - P. 40 - 50.

13.Feller W. An introduction to probability theory and its applications, 3rd ed., Vol. 1 / W. Feller. – New York Chichester Brisbane Toronto: John Wiley & Sons, 1970. – 493 p.

Рецензент: д-р техн.. наук, проф. Б.М. Конорев, провідний науковий співробітник, Сертифікаційний центр АСУ, Державна інспекція ядерного регулювання України, Харків

Автор: ЖОЛТКЕВИЧ Григорій Миколайович
Харківський національний університет імені В.Н. Каразіна, Харків, доктор технічних наук, професор, декан механіко-математичного факультету, завідувач кафедри теоретичної та прикладної інформатики.
Роб. тел. – 707-53-25, E-mail: g.zholtkevych@gmail.com

Автор: ПЕРЕПЕЛИЦЯ Іван Дмитрович
Харківський національний університет імені В.Н. Каразіна, Харків, аспірант кафедри теоретичної та прикладної інформатики.
Роб. тел. – 707-55-35, E-mail: ivanperepelytsya@gmail.com

Про один клас математичних моделей статичного аналізу асинхронних систем критичного призначення

Г.М. Жолткевич, І.Д. Перепелиця

У статті розглянуто математичну модель асинхронних програмних систем. Ця модель спирається на поняття скінченої абстрактної перед-машини, яке узагальнює поняття абстрактного скінченого автомату. На відміну від загальноприйнятих моделей модель, запропонована в роботі, дозволяє описувати більш складну поведінку системи в порівнянні з моделями скінчених автоматів. Зокрема, аномалія «активний тупик» може бути промодельована в термінах скінченої перед-машини. Автори наводять критерій виникнення активного тупику та ілюструють його застосування на прикладі.

Ключові слова: програмне забезпечення критичного призначення, асинхронні програмні системи, статичний аналіз, активний тупик, вибух черги

Об одном классе математических моделей статического анализа асинхронных систем критического назначения

Г.Н. Жолткевич, И.Д. Перепелица

В статье рассмотрена математическая модель асинхронных программных систем. Эта модель опирается на понятие абстрактной конечной пред-машины, которое обобщает понятие абстрактного конечного автомата. В отличие от общепринятых моделей модель, предложенная в работе, позволяет описывать более сложное поведение системы по сравнению с моделями конечных автоматов. В частности, аномалия «активный тупик» может быть промоделирована в терминах конечной пред-машины. Авторы приводят критерий возникновения активного тупика и иллюстрируют его применение на примере.

Ключевые слова: программное обеспечение критического назначения, асинхронные программные системы, статический анализ, активный тупик, взрыв очереди