UDC 004.451.24, 004.451.62

V. MEZHUIEV, E. VERHULST

MODELLING OPENCOMRTOS TASKS INTERACTION

Abstract. *The model of tasks interaction in Open Communication Real Time Operation System (OpenComRTOS) is developed and discussed in the paper. The main feature of the proposed approach is the using the intermediate hub entity for decoupling interacting tasks. Different temporal semantics (waiting, non waiting and waiting timeout) of tasks synchronization mechanism is introduced. Emerging the effects (i.e. synchronisation or the absence of it) in the case when the tasks actions have different temporal semantics is analysed. The different approaches for expanding the hub model with using TLA and Hoare triplets are proposed.*
Key words: *real time operation system, tasks interaction, temporal semantics of synchronization.*

Анотація. *У статті розроблюється й обговорюється модель взаємодії задач в операційній системі реального часу OpenComRTOS. Головна особливість запропонованого підходу – використання сутності синхронізації Hub як проміжної ланки в механізмі взаємодії задач. Проаналізована різна часова семантика процесу синхронізації задач (очікування, неочікування, очікування протягом періоду часу). Розглянуто виникнення ефекту синхронізації у випадку, коли дії задач мають різну часову семантику. Запропоновані різні підходи для розширення моделі Hub з використанням TLA (Temporal Logic of Actions) і трійок Hoare (Hoare triplets).*
Ключові слова: *операційна система реального часу, взаємодія задач, часова семантика синхронізації.*

Аннотация. *В статье разрабатывается и обсуждается модель взаимодействия задач в операционной системе реального времени OpenComRTOS. Главная особенность предложенного подхода – использование сущности синхронизации Hub как промежуточного звена в механизме взаимодействия задач. Проанализирована различная временная семантика процесса синхронизации задач (ожидание, неожидание, ожидание в течение периода времени). Рассмотрено возникновение эффекта синхронизации в случае, когда действия задач имеют различную временную семантику. Предложены различные подходы для расширения модели Hub с использованием TLA (Temporal Logic of Actions) и троек Hoare (Hoare triplets).*
***Ключевые слова*:** *операционная система реального времени, взаимодействие задач, временная семантика синхронизации.*

1. Introduction

Our previous papers discussed the rationale for developing OpenComRTOS – the concurrent real-time operation system in the context of interacting entities modelling paradigm [1, 2]. In OpenComRTOS tasks represent processing entities and hubs interaction entities. Tasks can only interact through hubs, while the hub entities implement the interactions.

The benefit of this approach is that the hubs decouple the individual tasks. While a task is an active entity, a hub is a passive entity with a predefined behaviour that mediates between tasks. By decoupling we mean that a task does not know about the other task it interacts with. During an interaction a copy of the internal state of a task is passed and this protects the private state of the task. The mechanism is also independent of the location of the tasks, allowing the transparent parallel processing. In programming terms, because pointers are only valid locally, they must not be passed through a hub. OpenComRTOS does not impose any restrictions of passing pointers; it is the responsibility of the developer to ensure their validity.

All this results in turning an OpenComRTOS task into a component. By combining components it is possible to create more complex systems glued together by the hub-interactions. In other words the interacting entities modelling paradigm gives us the compositional process view. To use a component, we don't need to have access to its internal state - it is sufficient to know its interfaces (i.e. the protocols it obeys to).

This view was first formalized in Hoare's process algebra of Communicating Sequential Processes (CSP) [3]. In CSP a system is composed of Processes and Channels. A process executes a possibly

infinite number of sequential steps. The sequences of individual steps (called traces of processes) are separated by channel communications. A channel communication can be seen as the simplest form of interaction. It is fully synchronous and when processes synchronize on a channel, data can be transferred over it. This mechanism is very powerful and provides a mechanism for formal construction and verification of large systems. However, the CSP channel communication is simple in its semantics. CSP channels are also time-agnostic although later on this was remedied with the formulation of Timed CSP [4].

The hubs generalize the functionality of a CSP channel. The basic functionality of a hub is synchronization between tasks, just like in CSP. A hub however synchronizes between tasks using a boolean guard that can be a lot more complex. The resulting behaviour in CSP is to allow each process to continue. This basic behaviour is also present in the hub concept. The difference is that a hub can be specialized because its state space is allowed to be user defined. Specific hub types (i.e. Event, Semaphore, FIFO, Resource etc.) define a well known superset of the basic CSP semantics. This also allows a hub to act as an intermediate for a larger number of tasks whereas a CSP channel is a point-to-point connection between two processes only. Note that in CSP the behaviour of a hub can be achieved as well, by introducing a special intermediate process between tasks.

An important advantage of the hub is that the interaction behaviour can be customized. This allows the application designer to express the system in a way which matches its intended behaviour. E.g. the OpenComRTOS kernel provides standard support for blocking, nonblocking, blocking with a timeout or fully asynchronous interaction. Application specific hubs can be created by customizing the synchronization and action functions, and this is possible without having to modify the kernel itself. It is also supported by the OpenComRTOS Metamodel, used for code generation and for the visual development of applications. This is very different from traditional approaches, which either requires a complete rebuilding of the kernel, or the creation of a middleware layer that emulates the required behaviour using standardized kernel services.

Using guards before actions makes it amenable to formal reasoning, e.g. by using TLA (Temporal Logic of Actions) [5]. The hub structure consists of logical proposition (the synchronization condition) and the synchronization action. The synchronization action is invoked once the synchronization condition is true. We expand the hub model by decomposing the synchronization proposition into the pre and the post conditions. In this paper we also will show the analogy between Hoare triples {P}C{Q} and the hub.

Section 2 of this paper expands the semantics of the CSP interprocess communication by creating a model of task interactions via intermediate entities (hubs). A sequence of such interactions creates a protocol for intertask interaction. Section 3 explores the time properties of the proposed task interaction model. Section 4 further learns behaviour of the hub and expands its model by analyses analogy with TLA specification and Hoare triplets. The conclusions and references sections finalize the paper.

2. Modelling Task Interaction

Dividing a system into entities and interactions is not new. It represents the natural way of human thinking. Accordingly, existing modelling techniques emphasize the use of objects and their relationships. Currently, the most prominent examples are Object Oriented Design (OOD) and Object Oriented Programming

(OOP) [6]. The common object model, that encapsulates the internal properties and methods, however, pays less attention to the interactions between such objects and as result its view is fairly static.

Embracing interacting entities as an architectural modelling approach is much better at expressing the dynamic interactions between objects. By using the term interaction we express that tasks take an action in a mutual way, which is often considered just a side effect of their communication. But the mutual way of actions does not mean that tasks perform actions in cooperation toward some predefined goal. This is the difference with agent-based approach.

Interaction takes place when two or more entities have an effect upon one another. So, the basic idea of interaction consists of a two-way effect, as opposed to a one-way causal effect. Fig. 1 presents such an interaction scheme.

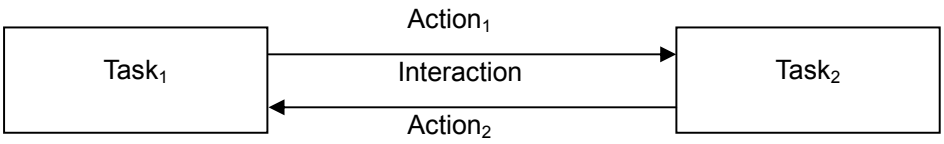

Fig. 1. Graphical model of interaction

From this simple graphical model follows, that an interaction should incorporate at least two[1] mutually linked actions in opposite directions, named here $Action_1$ and $Action_2$.

As was previously mentioned, the idea of the task interaction model of OpenComRTOS centres on the Hub, put in between $Task_1$ and $Task_2$.

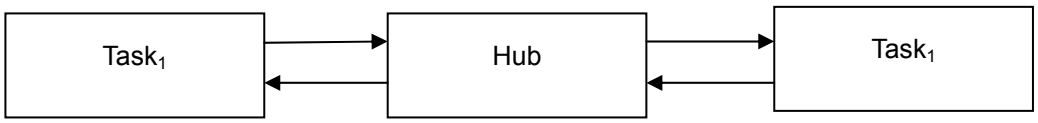

Fig. 2. Interaction via intermediate entity

In this model, an interaction between tasks is composed of two sub-interactions between each task and the intermediate hub entity. The sub-interaction between a task and a hub also consists of two complementary interactions, reflecting the layered implementation. At the top level an interaction composed of a send-receive pair of packets. Each of these send-receive actions results in an acknowledgement, expressing success or failure of the interaction (called return values in programming). This is essential feature of proposed approach in the context of embedded real-time systems. Note, that in different theoretical approaches the return values of services are not always considered or it is assumed, that the interaction is always successful.

To simplify the analysis, we ignore these sub-interactions and only consider the top level task interactions. Such a task interaction consists of 2 symmetric actions, which we call Put- and Get-actions. While these terms are reminders of the actual packets, that are interchanged in the implementation, no interaction order is implicitly defined in the names.

[1] A hub however supports N-N task interactions scheme.

An interaction is hence divided into a matching pair of Put (that we will call P) and Get (that we call G) actions leading to an effect S, i.e. the synchronization between tasks. Note that we do not state anything regarding actions timing and order, nor any guards attached to these actions. We only express that the mutual actions between two tasks are mandatory for synchronization.

$$(P \wedge G) \vee (G \wedge P) \rightarrow S, \quad (1)$$

where $\wedge$ represents the logical AND which symbolizes that both actions must have happen for the synchronization S take place.

We use here the concept of action correspondingly to the TLA definition of Lesley Lamport [5, p.16]: “an action is an ordinary mathematical formula, except that it contains primed as well as unprimed variables… An action is true or false on a step.”

Equation (1) follows from one of the design principles of OpenComRTOS – the symmetry of the synchronization mechanism. The symmetry principle results from using a synchronization entity in between tasks. This principle is used to check the properties of OpenComRTOS applications – i.e. for tasks synchronization the equation (1) must be satisfied.

Tasks in a concurrent system have equal rights for communication in the system. Thus by default, there is no master task, which causes effects upon other tasks. A task sending a packet to the hub knows nothing about the other task interacting with the hub, i.e. there is any cause-effect relationship between the interacting tasks.

The model, defined in Equation (1), states that only the mutual actions of two tasks lead to its synchronization, and that’s why we classify it as the interaction.

An important case to be considered occurs when only one action (put or get) happens. If an external observer detects only one action, then the synchronization has started, but it is not finished. In this case when using waiting / blocking services, the entity, which exhibited the action, is prevented from making progress. In other words, the task, that executed the action, is blocked.

Equations (2) and (3) model the scenario when Task_1 executes the waiting action P_1, but Task_2 did not yet exhibit action G_2. This result in the Task_1 is being blocked. Similarly, Task_2 is blocked after it has executed waiting G_2 and Task_1 did not exhibit P_1.

$$P_1 \wedge \neg G_2 \rightarrow Task_1 \ blocked, \quad (2)$$

$$\neg P_1 \wedge G_2 \rightarrow Task_2 \ blocked. \quad (3)$$

Note, that equations (2) and (3) reflect the usual semantics of processes communication on a blocking CSP channel[2].

A blocked task becomes unblocked if the corresponding action occurs. Corresponding means here that the type of the actions should be opposite: if first action is of the Put type, then it follows that second action must be of the Get type. Synchronization is the only way to allow blocked tasks to make progress again.

[2] Note, in OpenComRTOS tasks nether become blocking when using non waiting services.

3. Timing properties of Task Interactions

Equation (1), states that only mutual actions of tasks lead to synchronization. In this section we discuss the meaning of the term mutual with respect to time.

OpenComRTOS implements 3 possible time related semantics for the interactions over a hub: Waiting (W), Non Waiting (NW) and Waiting with Timeout (WT). Lets define the effects (i.e. synchronization or the absence of it) in the case when the sub-actions have different timing properties.

In general the semantics of task interactions depends on the type of Hub and boundary conditions (e.g. a FIFO list size). To simplify the discussion this section will use the concept of synchronization in CSP sense, i.e. as the synchronous data transfer between two tasks. The OpenComRTOS entity, closest to a CSP channel, is the port hub[3] .

In general, OpenComRTOS has 9 possible cases of the time semantics of two tasks interaction.

Table 1. Cases of the time semantics of two tasks interaction in OpenComRTOS

Time semantics	W	WT	NW
W	symmetry	<W, WT>	<W, NW>
WT	<WT, W>	symmetry	<WT, NW>
NW	<NW, W>	<NW, WT>	symmetry

In case of interactions are belong to the first quadrant of the Table 1 (i.e. to W semantics) synchronization is not depending on time. The symmetry cases, represented as diagonal of the Table 1, occur when both tasks use the same type of services (i.e. NW and NW, WT and WT, W and W). The property of such “symmetry in time” interactions is that synchronization does not depend on the relative order of the interactions.

In case of using of 1) <W, WT>, 2) <W, NT>, 3) <WT, NW> sequences the synchronization depends on the interactions order. Temporal logic operators should be used to express the time semantics of the synchronization mechanism in such a case.

1) If W action happened before WT action, the synchronization occurs in any case.

If WT action happened first, to have synchronization the W action should be done inside the time interval T.

2) If W action happened before NW action, the synchronization occurs in any case.

If NW action happened first, the synchronization will not occur.

3) If WT action happened before NW action, to have synchronization the NW interaction should be done inside the time interval T.

If NW action happened before WT action, the synchronization will not occur.

Note, in formulation of all these statements we ignore the hub buffering capability.

Such basic sequences of interactions can be applied to analyse much more complex interactions by means of the basic sequences composition. For this purpose such basic sequences of interactions should be formulated as temporal formulas in TLA sense. The next state of the system will depend on result of previous interactions (leading to synchronization or blocking the application system).

[3] The port hub can be considered as the partial case of FIFO with list size equal 1.

Let first formalize and consider the time properties of the symmetry cases of OpenComRTOS synchronization semantics.

NW task interaction has the following semantics:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(t_1 = t_2\right) \rightarrow S\,. \tag{4}$$

Equation (4) defines that, in case of NW services, we have synchronization S if and only if both tasks exhibit their respective actions, P and G, at the same time[4].

Therefore <NW, NW> synchronization works under no circumstance – NW services will fail because of the sequential execution on a CPU (Von Neumann machine). The kernel, where the hub is located, strictly serialises the access to the hub, therefore the hub sees no two NW-packets at any time, and thus the NW synchronisation at the hub is not possible.

The equation (4) is only true when one of actions (of Put or Get type) was buffered in the waiting list of a hub. E.g. we can get packet from FIFO using NW semantics if it was already put in the FIFO using NW semantics. So the NW synchronisation semantics is valid when the hub does buffering the interactions. It expresses the fact, that task synchronisation is a secondary effect from successful completion of the sub-interaction between a hub and a task and not directly between tasks.

Hence, the only safe semantics of interactions is the waiting one. Therefore, waiting semantics are the usual way for task synchronization implementations (as e.g. in CSP). The NW services can be considered as a way to check whether or not synchronization is possible and when the hub has the buffering support.

WT task interaction (synchronization) has the following semantics:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(\left|t_2 - t_1\right| < T\right) \rightarrow S\,, \tag{5}[5]$$

where T stands for timeout.

Equation (5) defines that, with WT services the synchronization occurs when the time gap between actions P and G is less than the interval T. Such formula can be applied to NW services, with condition that $T = 0$.[6]

Leading to synchronization S the W task interaction has the following semantics:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(\left|t_2 - t_1\right| < \infty\right) \rightarrow S\,. \tag{6}$$

For the case of using W services, the time, during which task synchronization can happen, is infinite. So, equation (6) can be simplified to equation (1), i.e. there is no real time dependency – the synchronization occurs in case of mutual actions irrespectively of its time. This type of synchronization here is the same as for a CSP channel.

Fully asynchronous interactions are also possible in OpenComRTOS, although such interactions are really delayed synchronous interactions. The semantics of asynchronous in time services can be defined as 'shoot and forget' semantics. Real systems always have limited resources, but asynchronous interactions require an unlimited number of resources (e.g. for data storage). This imposes serious

[4] The discrete time t, which values define the timestamps of actions P and G, is used in (4) and further.

[5] NW and W can be considered as the partial case of WT with $T = 0$ and $T = \infty$.

[6] Actually the communication delay puts a minimum value on the timeout values that are valid to be meaningful.

limitations on the verifiability of applications. Therefore, asynchronous interactions must be restricted. Because no formal analysis was done on this aspect, we postpone the discussion on asynchronous interactions to a later time.

The decoupling functionality of a hub provides another way to implement asynchronous behaviour. E.g. for a FIFO, we will have waiting semantics only if the FIFO is full (the limit of the FIFO list size is reached) and we have a sending task; or if the FIFO is empty (FIFO list size is zero) and we have a receiving task.

$$\left(P \wedge Count\left(FIFO\right) = Size\right) \vee \left(G \wedge Count\left(FIFO\right) = 0\right) \rightarrow Wait\,. \tag{7}$$

Hence, the normal behaviour of such a service is asynchronous, switching to a synchronous behaviour only when the FIFO is full (or empty). This behaviour is desirable because it allows to limit the use of resources.

So for FIFO for all types of interactions (i.e. W, NW, WT) the synchronization takes place irrespectively of the time if the formula (7) is false.

$$\neg\left(P \wedge Count\left(FIFO\right) = Size\right) \vee \left(G \wedge Count\left(FIFO\right) = 0\right) \rightarrow S\,. \tag{8}$$

The similar properties can be also formulated for Event[7] and Semaphore functionality, which also exhibit an asynchronous behaviour.

In case of task synchronization on an event and on a semaphore hub the order of actions following is important. Synchronization on an event hub takes only place, if the first in time action is Put (RaiseEvent), followed by Get (TestEvent), with the assumption, that the initial value of an event boolean flag (isSet) is false. First the Put-operation sets the flag isSet to true, next the Get-operation sets the flag to false again. Both tasks can continue immediately after their interaction with the Hub.

Therefore, the depicted general time semantics of task interactions depends on the type of hub and boundary conditions (as e.g. of the event isSet boolean flag value or the FIFO list size).

It was also decided not to implement a hub with the functionality of a “pipe” (as found e.g. in UNIX systems) because the pipe service assumes an infinite buffer to be available, resulting in a system failure, if the receiving side gets blocked.

A hub is intended for decoupling different tasks (e.g. $Task_1$ and $Task_2$). At the same time the $Task_1$ can e.g. get own packet from the FIFO, which was put into the FIFO before.

While describing the time semantics of the synchronization mechanisms, we implicitly use the assumption, that synchronisation can only occur at the same hub:

$$Hub\left(P\right) = Hub\left(G\right). \tag{9}$$

Equation (9) is the property, which describes the space semantics of the synchronization mechanism. In a distributed concurrent system we can find the same time and space semantics as we find for synchronization of entities in the real world.

4. Models of Hub

Lets express the hub structure in terms of the interacting entities systems grammar[8]:

[7] Event is partial case of Semaphore with the maximum value of the counter equal 1.

[8] This model of hub allow us draw a parallel with the concept of Guarded Atomic Action (GAA) [9].

Synchronization Predicate (1) AND

Synchronization Action (1) AND

State (1-N)

This conceptual model says: a hub has one Synchronisation Predicate, one Synchronisation Action and is always in one of its N States.

In a condensed form:

$$Hub \overset{def}{=} Predicate \wedge Action \wedge State. \tag{10}$$

The TLA language [5] was chosen as the base for OpenComRTOS formal specification. The definition (10) of the hub corresponds to the TLA specification (11), which is composed from guards and actions [5].

$$A_0 = Guard_0 \wedge Action_0$$

…

$$A_{j-1} = Guard_{j-1} \wedge Action_{j-1}. \tag{11}$$

TLA defines the next state (Next) of system as the result of the logical or operation between all possible guarded actions.

$$Next = A_0 \vee A_1 \vee ... \vee A_{j-1}\ . \tag{12}$$

This allows us to define the protocol of the OpenComRTOS task interactions as a sequence of guarded actions, taking place in the hubs.

The model of a concurrent program in OpenComRTOS can be defined as a state machine, executing certain actions only and only if specific synchronization conditions are met. The synchronisation condition also includes the time properties, considered in the section 3 of the paper.

The approach taken describes the behaviour of the program system in terms of changing its internal state. The program state we consider as the union of tasks and hubs states subsets. There is a sequential ordering of tasks actions $< A_1,\ A_2,..., A_n >$ which causes the updates of the hub states $< H_1,\ H_2,..., H_n >$. Only a complementary pair of actions leads to a synchronisation, so an interaction can be seen as a mechanism, which causes a state transition of the involved tasks. In case of synchronisation the state of the tasks changes from “waiting” to “ready”, allowing the tasks proceed further. Therefore, sequential ordering of task interactions $< I_1,\ I_2,..., I_n >$ updated the program states $< P_1,\ P_2,..., P_n >$.

Checking specified properties of the state is the main method to prove correctness of a program in axiomatic verification. E.g. predicates on data are placed in different points of the program and remain as invariants in the process of its execution. Relations between predicates are set by axioms of the programming language. Such entry/exit predicates describe the behaviour of a program in an alternative form.

The first logical system, developed for verifying computer programs, was the Floyd and Hoare logic [7, 8]. The idea of Hoare logic is to put predicates at the beginning of a program block. Axioms of

Hoare logic define such input predicates as sufficient conditions, which guarantee that the successful execution of the program block results in the specified postconditions.

Hoare defines axiomatic semantics of a programming language, as a system of assumptions, whereby Hoare triples are used as base elements. Program operators are transformed into predicates and the validity of a program tree is reduced to the validity of a set of lemmas, not containing program operators.

The next step in the hub model development is to consider it in terms of Hoare triplets [8]. These triplets refer to the following relationship:

$$\{P\}C\{Q\}\}, \tag{13}$$

where P is called the precondition, Q – postcondition assertions[9] and C is the command.

This needs expansion of the hub model, by adding the necessary Pre- and Postconditions. Synchronization precondition of hub pertains to the state before an action, enabling it.

$$P\,recondition \wedge Action \rightarrow Postcondition\,. \tag{14}$$

As a concrete example let's consider the test semaphore functionality.

Preconditions:

– At least one waiter task on reader's side AND;

– Semaphore counter $>= 1$.

Actions:

– Set state of task in Active;

– Decrement semaphore counter.

Postconditions:

– Waiter task runnable again AND;

– Semaphore counter decremented and $>= 0$.

Adding the postcondition predicate to the hub definition is a step towards formal program validation techniques[10] and can be used to implement runtime checks in the OpenComRTOS[11].

5. Conclusions

This paper discussed OpenComRTOS as a process oriented programming paradigm. The feature and benefit of the proposed approach is generalization of the different forms of synchronization methods into the unique hub concept. The hub allows the application designer to express the synchronization mechanism as best matching the behaviour of a system, e.g. by using different forms of time synchronization semantics. Building the hub on a formal base links programming techniques with formal methods and therefore offers the possibility of designing formally proven programs.

[9] Assertions are formulas in predicate logic.

[10] The testing cannot prove the program validness and just can show program incorrectness.

[11] Note, most model assertions are related to the correctness of the program design.

REFERENCE

1. Verhulst E. An Industrial Case: Pitfalls and Benefits of Applying Formal Methods to the Development of a Network-Centric RTOS / E. Verhulst, G. Jong, V. Mezhuyev // Lecture Notes in Computer Science. FM 2008: Formal Methods. – Heidelberg: Springer Berlin, 2008. – P. 411 – 418.
2. Verhulst E. OpenComRTOS: A Runtime Environment for Interacting Entities / E. Verhulst, V. Mezhuyev, Bernhard H.C. Sputh [et al.]; P. Welch, H. Roebbers, T. Announced (eds.) // Communicating Process Architectures 2009. – IOS Press. – 2009. – P. 173 – 184.
3. Hoare C. A. R. Communicating Sequential Processes. Published by Prentice Hall / C.A.R. Hoare // International Series in Computer Science. – 1985. – P. 276.
4. Ouaknine J. Timed CSP: A Retrospective / J. Ouaknine, S. Schneider // Electronic Notes in Theoretical Computer Science. – 2006. – Vol. 162 (1). – P. 273 – 276.
5. Lamport L. Specifying systems: the TLA+ language and tools for hardware and software engineers / Lamport L. // Addison-Wesley, Boston. – 2002. – 364 p.
6. Booch G. Object-Oriented Analysis and Design with Applications. Addison-Wesley / Booch G. – 2007. – 608 p.
7. Floyd R.W. Assigning meanings to programs / R.W. Floyd; J.T. Schwartz (ed.) // Proc. of Symposium on Applied Mathematics. – 1967. – Vol. 19. – P. 19 – 32.
8. Hoare C.A.R. An Axiomatic Basis for Computer Programming / C.A.R. Hoare // Communications of the ACM. – Vol. 12, N 10. – P. 576 – 583.
9. Rosenband D. Modular Scheduling of Guarded Atomic Actions / D. Rosenband, Arvind // Proc. of the 41st Design Automation Conference (DAC'04) (San Diego, June 7–11, 2004). – San Diego, California, USA, 2004. – P. 8.

Стаття надійшла до редакції 13.10.2009