Modeling Tasks Synchronization in the Fault-Tolerant Cyber-Physical Systems

Vitaliy Mezhuyev¹, Kamal Z. Zamli¹, Mohamed Ariff Ameedeen¹, Horacio Pérez-Sánchez², Ravi Samikannu³

¹Faculty of Computer Systems and Software Engineering
University Malaysia Pahang, Gambang, Malaysia
E-mail: mejuev@ukr.net, kamalz@ump.edu.my, mohamedariff@ump.edu.my

²Computer Science Department, Catholic University of Murcia, Murcia, Spain, E-mail: hperez@ucam.edu

³Faculty of Electrical Engineering, Selvam college of technology, India, E-mail: drraviee@gmail.com

1. INTRODUCTION

Cyber Physical Systems (CPS) are systems that contain both computational and physical elements with the close integration of physical processes and information processing [1]. The design of CPS should provide the new types of abstractions, applicable for modeling both the physical domains and the corresponding software. As multiple authors noticed, standard abstractions do not work here [2].

In a previous work we built a model of software tasks interaction in the real-time and concurrent operation environment [3; 4]. Tasks can only interact through synchronization objects, which decouple the individual tasks. In this approach a task does not know about other task it communicates with. Throughout communication, a copy of an internal task variables is passed, which allows to protect the private state of a task. The approach is not linked with physical location of the software tasks, which allows to implement parallel processing in more clear way.

In this paper we expand the proposed modeling approach for CPS design. This allows us to express heterogeneous semantics of interlinked computational and physical domains. For the design of CPS becomes important not only temporal, but geometrical properties of a space, where CPS is located. Using different types of guarded actions allows to conjoin abstractions for modeling physical and software parts of a CPS. At the same time, the approach allows us to express and guarantee fault-tolerant behavior of a CPS being designed.

Section 2 of this paper discusses the mathematical base of the proposed modeling approach. Section 3 builds a model of tasks synchronization in a CPS. Section 4 formalize and learn the timing properties of the proposed model. Section 5 expands the proposed model of tasks synchronization in terms of Hoare triplets. The future work, conclusion and references sections finalize the paper.

2. MATHEMATICAL BASE

The model of software tasks interaction in computer science was initially developed as algebra of Communicating Sequential Processes (CSP) by Hoare [5]. In his approach a software system comprises of processes and communication channels. Each process has finite or infinite number of sequential steps of execution. The sequences of process’ steps are split by communications via channels. A communication is fully synchronous and is used for data transfer when processes synchronize on a channel. CSP approach is very strong and gives a way for robust development and verification of concurrent software systems. Though, the communication via channel in CSP has very simple semantics, e.g. it is not take into account properties of time (although later it was elaborated in so called timed CSP [6]).

In this paper we expand CSP by development of the model of tasks interaction through intermediate synchronization objects. The proposed model expands the functionality of a CSP channel. The idea of this model is to formalize semantics of tasks synchronization scenario, like we have in CSP. The model also allows synchronization between software tasks using a logical guard, containing pre- and post- conditions. The idea of synchronization in CSP is to permit each communicating process to continue. We model this behavior, with the modification, that logic of synchronization can be customized.

The approach allows a CPS designer to model the system in a way, that matches its required properties and behavior. E.g. it is possible to express nonblocking, blocking, blocking with a timeout or asynchronous scenarios. A sequence of tasks interactions defines a synchronization protocol. The idea is to create application specific protocols by customizing the synchronization condition and action functions, which next to be transformed into fault tolerant executable code of the CPS being designed.

Using logical conditions before actions allows application of formal methods, for example Temporal Logic of Actions (TLA) [7]. Lets express the structure of synchronization protocol of software tasks in a CPS. A synchronization object has a Guard (synchronization condition) and synchronization Action. This definition corresponds to the structure of TLA specification.
\[ A_1 = \text{Guard}_1 \land \text{Action}_1 \]
\[ \ldots \]
\[ A_N = \text{Guard}_N \land \text{Action}_N \]  

(1)

The next state \((\text{Next})\) of a system is a logical formula, defined as a composition by logical OR of all guarded actions

\[ \text{Next} = A_0 \lor A_1 \lor \ldots \lor A_N \]  

(2)

where \(N\) is a general amount of guarded actions. Following the approach we define the protocol of tasks synchronization as a sequence of guarded actions, which take place in a CPS. This model also has similarity with Guarded Atomic Action (GAA) approach, used in parallel processing [8]. In Section 5 we will expand the proposed synchronization model by means of decomposition of logical guard into pre- and post-conditions.

3. MODELING TASK INTERACTION IN A CPS

Let’s represent a model of a CPS as interacting via synchronization objects software tasks in the physical environment (see fig. 1).

So task interaction consists of two mutual and symmetric actions, which we will call Put (\(P\)) and Get (\(G\)) actions. Here we do not suppose about actions order and timing, and not classify guards for the actions. For the moment, we show the property that the mutual actions between two tasks are obligatory for their synchronization \(S\)

\[ S = \exists \text{Task}_1 \neq \text{Task}_2, \text{Sync}: \]

\[ (\text{Put(}\text{Task}_1, \text{Sync}) \land \text{Get(}\text{Task}_2, \text{Sync})) \]

\[ \lor (\text{Put(}\text{Task}_2, \text{Sync}) \land \text{Get(}\text{Task}_1, \text{Sync})) \]  

(3)

An interaction is decomposed into a corresponding pair of \(P\) and \(G\) actions of two different tasks, which leads to an effect \(S\) (representing synchronization of the tasks). We can rewrite (3) in more simple form

\[ S = (P_1 \land G_2) \lor (P_2 \land G_1) \]  

(4)

Note, that decomposition of a system into objects and interactions is not new idea and just reflects the usual way of thinking. Therefore, a lot of existing modeling approaches accentuate the use of objects and their relationships. As most important practice the Object Oriented Analyses and Design can be mentioned [9].

Modeling interactions of objects in a CPS allow us more explicitly express the dynamics of a system. By using the concept “interaction” we present the fact, that software tasks make actions in a mutual way, which is commonly considered just as a side effect of their communication. An interaction occur when two or more objects have mutual effect upon each other. Consequently, the main idea of interaction is modelling two-way effect, as contrast to one-way causal relation. Thus, an interaction should incorporate at least two mutually linked actions in opposite directions (Action\(_1\) and Action\(_2\) in Fig. 1).

To guarantee the fault-tolerance of tasks interaction in a CPS, a synchronization object Sync is put in between Task\(_1\) and Task\(_2\).

In proposed approach, an interaction among tasks comprises two complementary actions between each task and an intermediate synchronization object.

Fig. 1. Model of tasks interaction in a CPS

logical AND indicates here, that both actions must occur for synchronization of the tasks. The idea of an action corresponds to its definition of Lamport a “TLA action is an ordinary mathematical formula, which can be true or false on a step” [7, p.16].

Equation (4) reflects the symmetry of the scenario of synchronization, resulted from putting a synchronization object between tasks. This general principle is needed for validation of the properties of a CPS, first of all, fault tolerance of tasks synchronization.

Let’s consider a case when only one action (Put or Get) of an interaction happens. If we detect only one action, this indicates that synchronization mechanism has started, however, not finished. In case of using waiting services it prevents the task, which have started a synchronization action, from making further progress. Thus, the task, which first executes the synchronization action, becomes blocked.

Equation (5) represents the waiting scenario \(W\), when first task executes the action \(P_1\), but second task did not yet exhibit
action $G_2$. This results that first task becomes blocked. Likewise, second task becomes blocked after execution of $G_2$, in the case then first task did not exhibit $P_1$:

$$W = (P_1 \land \neg G_1) \lor (\neg P_1 \land G_2)$$ (5)

Equations (4) and (5) reflect the typical scenario of tasks synchronization using a CSP channel. Communicating on a channel process becomes blocked and unblocked only if occurs the matching action. Matching here denotes that the nature of the actions: if the first action is Put, when the second action should be Get, and vice versa. Synchronization scenario (3) gives the possibility to blocked tasks to process again.

4. TIMING PROPERTIES OF TASKS INTERACTION

In previous section we have considered the property that only mutual tasks actions can lead to their synchronization (4). Lets discuss this property regarding time. We will consider three possible timing semantics of tasks interactions: Non Waiting (NW), Waiting (W), and Waiting Timeout (WT). Lets analyze the emergence of synchronization effect in the case of tasks actions having different timing properties.

NW synchronization of tasks is described by the following scenario (the time $t$ defines the timestamps of tasks actions $P$ and $G$):

$$S_{NW} = (P_{t=1} \land G_{t=2}) \land (t_1 = t_2)$$ (6)

Equation (6) illustrates that, in the case of using NW services, synchronization $S$ take place if only both tasks show their corresponding actions at the same moment of time. From it follows, that sequential execution of the actions flow (NW, NW) on a Von Neumann type of machine results in no synchronization. This is caused by strict serialization of the access to the synchronization object on a Single Processor (SP) machine, therefore no two NW-requests available at one time.

Equation (6) is only true when we use multiprocessor (MP) architecture or when one of the actions (Put or Get) is buffered by a waiting list of a synchronization object. E.g. a user can take data from a FIFO by NW action if it was already put in the FIFO previously by NW action (even by the same task). Thus, the NW synchronization semantics is only valid when we buffer interactions or we use MP system, which typical example is a CPS.

The only possible semantics of synchronization on a SP computer is waiting one. NW scenario we can use here as a method to check, if synchronization of tasks is possible (i.e. if there is a task on other side of a communication channel).

WT synchronization of tasks is described by the following scenario:

$$S_{WT} = (P_{t=1} \land G_{t=2}) \land (|t_2 - t_1| < T)$$ (7)

where $T$ designates timeout.

Equation (7) shows, that using WT actions the synchronization takes place if the time interval between $P$ and $G$ is smaller than $T$. This formula allows to generalize NW services, if we consider time equals to zero $T = 0$.

Resulting in synchronization $S_W$ waiting tasks interaction is described by the following scenario:

$$S_{W} = (P_{t=1} \land G_{t=2}) \land (|t_2 - t_1| < \infty)$$ (8)

Thus, in case of using waiting actions, the time, in which synchronization of tasks should occur, can be infinite. Formula (8) can be simplified to (4), having no dependency on time—the synchronization takes place when there are two mutual actions irrespectively of their time. As we already mentioned, this synchronization scenario is typical for a classical CSP channel.

From (7) follows that NW and W scenarios can be considered as the partial case of WT, substituting corresponding $T = 0$ and $T = \infty$.

Note, at formulation of timing semantics of interaction mechanisms, we suppose that synchronization (Put and Get actions) can only occur at the same object Sync.

Considered basic sequences of tasks interactions we apply for analyses of more complex synchronization scenarios by composition of these basic structures, expressed as temporal formulas. The next state of a modeled CPS is a result of all previous interactions of composing CPS software tasks.

Equation (8) describes synchronous semantics, while asynchronous interactions are also possible. Note, a CPS (like any other embedded in physical world system) always has limited resources, while implementation of asynchronous scenarios is based on using unrestricted amount of resources. This puts serious constraints on applicability of asynchronous services for CPS design.

If a synchronization object has a buffer, it makes possible another approach to implement asynchronous behavior. For example, FIFO becomes waiting only if the buffer of it is full and there is sending (Put) task; other possibility if the buffer of FIFO is empty and there is requesting (Get) task

$$W = (P \land FIFO.Count = Size) \lor (G \land FIFO.Count = 0)$$ (9)

Therefore, the normal behavior of such type of synchronization objects is an asynchronous, swapping to a synchronous behavior only in boundary conditions (e.g. when the FIFO buffer is full or becomes empty). Implementation of such services in CPS is crucial because leads to minimization of used resources.

Thus, for a FIFO for all considered types of scenarios - NW, W, WT - the behavior is asynchronous if the formula (9) is false. Related properties can be also considered for other samples of synchronization objects ( semaphore, mutex etc.). Also, the approach allows to synchronize for any amount of interacting tasks, while a channel in CSP is a point-to-point link only between two processes.

In case of synchronization on a mutex becomes important the order of task actions. Synchronization on a mutex can only occur if the first action is lock, followed by unlock, with the assumption, that initially mutex was not locked. First, the Put-action locks mutex, next the Get-action unlocks it. Both interacting via mutex tasks can continue immediately after their synchronization.
5. EXPANDING THE MODEL OF TASKS SYNCHRONIZATION

Let's consider the model of a concurrent program in a CPS as a Finite State Machine (FSM), executing needed actions if specific synchronization conditions are true. For CPS, the synchronization condition should include both the timing properties, discussed in previous section of the paper, as also properties of physical space (we considered in [12]).

The FSM describes the behavior of a CPS in terms of concepts of a state and a state transition. Lets consider a program (software part of a CPS) state as a combination of states of tasks and synchronization objects. Sequence of tasks $<T_1, T_2 \ldots T_N>$ actions causes the updates of states of synchronization objects $<S_1, S_2 \ldots S_M>$ ($N, M$ – are amount of tasks and synchronization objects correspondingly).

Only a matching pair of tasks actions leads to their synchronization, so exactly interaction causes a state transition of the engaged tasks. In the case of synchronization the state of involved tasks changes from “waiting” to “ready”, allowing the tasks to proceed further. Thus, we can say, that sequence of task interactions $<I_1, I_2 \ldots I_P>$ updates the possible states of the program $<P_1, P_2 \ldots P_Q>$ ($P, Q$ – are amount of tasks interactions and program states correspondingly).

Checking properties of a CPS in the global state space is the main method to guarantee its fault-tolerance. For verification we check if all synchronization conditions remain invariants in the process of program execution.

One of approaches for verification of concurrent software is the logic of Floyd and Hoare [10; 11]. Hoare logic puts pre-conditions at the beginning of a program, and post-conditions at the end of it, which allows to check successful execution of the program.

Let's consider an expansion of proposed model of tasks synchronization by its considering in terms of Hoare triplets [11]. It is defined by the relationship

$$\{P\}C\{Q\}$$  (10)

where $P$ designates pre-condition, $Q$ - post-condition predicates and $C$ is the command (action, in our terminology).

Synchronization pre-condition and action imply post-condition

$$P \text{ re – condition } \wedge \text{Action} \rightarrow \text{Post – condition}$$  (11)

As an example let's consider the scenario of tasks synchronization on a mutex:

**Pre-conditions:**
- A waiting on a mutex task AND
- Mutex is locked.

**Actions:**
- Set state of a task as Active;
- Unlock a mutex;

**Post-conditions:**
- Waiting task runnable again AND
- Mutex is unlocked.

Adding the post-condition in the model of synchronization needed for further development of validation methods, especially for designing fault-tolerant CPS. Note, that traditionally used testing can only identify program incorrectness, but not prove program validness.

6. FUTURE WORK

The model of CPS will be expanded by physical elements, such as sensors and actuators. Consequently, we will add in the model guards, defining the logical conditions of physical interactions in a CPS. Physical properties of a CPS will be linked with geometry of a space. We will also introduce operations on guards, which will allow to make composition of physical and software elements of a CPS.

7. CONCLUSION

This paper discussed a model of a cyber-physical system in the context of a process oriented programming paradigm. Building CPS on a formal base correlates programming techniques with formal methods and gives the possibility of designing fault-tolerant CPS. The main result and advantage of the disused approach is the generalization of different tasks synchronization scenarios into unique and customizable model. This allows the CPS designer to express the synchronization semantics as best matching the intended behavior of a software part of a system.

The presented approach also results in turning a task and a corresponding synchronization object into a component of a CPS. By combining such the components it is possible to develop more complex systems linked together by processes communications.

REFERENCES