Term Validation of distributed hard real time applications

Gaëlle Largeteau-Skapin, Dominique Geniet

Term Validation of Distributed Hard Real-time Applications

Gaëlle Largeteau and Dominique Geniet

Laboratoire d'Informatique Scientifique et Industrielle,
Université de Poitiers & ENSMA,
Téléport 2 - 1 avenue Clément Ader
BP 40109 86961 Futuroscope Chasseneuil cédex, France
largeteau@ensma.fr, dgeniet@ensma.fr

Abstract. We show that, when the modeled physical process is closed, finite automata and product operators are sufficient to valid distributed systems on an operational way.

1 Introduction

A real-time system is **reactive** and **concurrent** (all operations associated with a process managing have to run simultaneously). It is a set of elementary **tasks**, each of them coding a reaction to incoming events. This set is composed of **periodic** and **non periodic** tasks (related to alarm signals and user actions). Validity of a real-time system is based on both the correctness of its results and its conformity to timing constraints. There are two classes of real-time systems: **hard** and **soft** systems. If not respecting terms implies irretrievable consequences, the system is **hard**, otherwise the system is **soft**. Here, we deal with hard real-time systems composed of periodic tasks.

Validating a real-time system consists in proving that it will always be able to react in conformity with its timing constraints, whatever the incoming event flow. The term validation is then a decision process related to task scheduling sequences. It usually follows two main approaches: first, *in-line* approach consists in choosing the task to elect for any context switch during the application run. Since computing task system scheduling issue with critical resources is NP-complete [2], this approach is not optimal[1] for almost every task configuration, and it has an exponential complexity. To solve this problem, the *off-line* approach uses formal models to search for existence of at least one scheduling sequence satisfying constraints (through model checking techniques).

The fast technological evolution in recent years (especially in network communications) has resulted in using distributed systems as a base for hard real-time applications more and more frequently. These systems stand founded on real-time protocols integrating timing constraints for messages transmissions.

[1] A scheduling algorithm is optimal if and only if it gives a valid scheduling sequence when there exists one.

Since scheduling distributed systems is NP-difficult, the *in-line* approach is still not optimal. We use the *off-line* approach defined in [3][4] to suggest a method that validates distributed systems on an operational way. Its principle is to integrate communication protocols in the model and to adapt the model to targets with different processor speeds. The physical architecture is composed of a set of sites, that communicate through a network. Each site dispose of many processors, and of a RAM shared between its processors. All the processors follow the local clock of the site. Moreover, each site dispose of a network board, which contains a specialised processor. Real-time system validation involves the foreseeability of behaviors. Since using cache or pipeline induces nondeterminism, they are always disabled in real-time systems: here, we make the assumption, in the framework of validation, that none are used.

Firstly, the model is presented in the framework of centralised real-time systems with fixed execution time tasks. Then, we describe a modelling technique for distributed real-time systems integrating speed differences. We show how product automata can be used to model simultaneity in distributed systems. We assume that there is no task migration. The task placement is not considered.

2 Centralised systems validation

2.1 Task temporal modelling

A real-time software is a set of atomic tasks. We denote $(\tau_i)_{i\in[1,n]}$ such a system. Each task τ_i is specified by: its arrival time r_i, its deadline D_i, its period T_i, and C_i, the CPU execution time of each instance of the task. Parameters r_i, D_i and T_i come from the specifications of the external system, but C_i depends on both the code of the task and the performances of the target processor. We assume that all atomic statements have the same duration of one *time unit*[2]. Hence, C_i is constant in time units. On the opposite, r_i, D_i and T_i do not depend on the CPU frequency: they are not fixed in time unit. In the following, we use this property to build a model language which takes into account both term specifications, and performances of the processor.

Let τ be a task of timing constraints r, D and T. The code ξ of τ is a word over the set P of atomic statements. The execution duration C of τ is extracted from this code. During its execution, τ can be active or suspended whether it owns the processor or not. Then, for each time unit, observing τ state allows to build an activity/inactivity sequence for τ. The set of sequences that respects the specified timing constraints is the τ temporal model. Our aim is to build this set, i.e to build the regular language $L(\tau)$. We consider time as implicit: each task processes one action by time unit. Letter a models the activity state of τ for one time unit, and $\bullet$ models its suspended state for one time unit. We note Σ={a, $\bullet$}. The temporal model associated with ξ is the word $\phi(\xi)$, where ϕ is the concatenation morphism $P \rightarrow \{a\}$. The length of $\phi(\xi)$ is the duration C of τ. The τ temporal model $\mathrm{L}_u(\tau)$ is obtained by given the system inactivity periods

[2] Time unit is the duration of a non-preemptive instruction (assembler).

(using $\bullet$). We use the *Shuffle*[3] operator ш, the generic expression of the model is given in [3] by[4] $L(\tau) = Center(\bullet^r((\bullet^{D-C} ш \phi(\xi))\bullet^{T-D})^*)$. Each word **w** of this language has got the same length T and is called **valid temporal behavior** of τ.

Task τ is running on a processor with particular temporal features : we define a **time unit** as the time interval between two clock ticks, and a **cadence** as the inverse of its duration. In the multi-processor case, all processors of a same site work in a synchronous way. The duration of τ is then equal to $\mid \phi(\xi) \mid \times u_S$ on site S, and $\mid \phi(\xi) \mid \times u_T$ on site T. The τ temporal features (D, T, etc.) are no longer expressed in the language directly as the occurrence number of $\bullet$, but as the occurrence number of $\bullet$ that is necessary to model the inactivity time corresponding to the target processor. We note $L_u(\tau)$ the set of τ valid temporal behaviors on a processor that have **u** for time unit.

Example: Let be τ with (r,D,T,C)=(3ms,8ms,10ms,3t.u.[5]). A τ model can be $L_{1ms}(\tau) = Center(\bullet^3((\bullet^5 ш a^3)\bullet^2)^*)$, or $L_{250\mu s}(\tau) = Center(\bullet^{12}((\bullet^{29} ш a^3)\bullet^8)^*)$.

The rate between **a**'s and $\bullet$'s depend on the target *cadence*. In the following, we show that task τ, defined with temporal characteristics (r,D,T,C), and designed to run on a processor with a cadence **c**(**u**=1/**c**), can always be associated with a regular language $L_u(\tau)$. In order to integrate processor features in the model, values expressed in seconds (r, D, T) have to be converted in time units. Let u be the time unit associated with the processor, x seconds correspond to $\frac{x}{u}$ t.u. on this processor. The values in seconds of r, D and T equal respectively to $\frac{r}{u}$, $\frac{D}{u}$ and $\frac{T}{u}$ t.u.. Usually, timing constraints r, D and T are of the order of 10^{-1}s and the time unit **u** is of the order of 10^{-5}s. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, we can assume that $\frac{r}{u}$, $\frac{D}{u}$ and $\frac{T}{u}$ are integers. By using the same approach than in [3], we get $L_u(\tau)$= Center($\bullet^{\frac{r}{u}}((\bullet^{(\frac{D}{u}-C)} ш \phi(\xi))\ \bullet^{(\frac{T-D}{u})})^*$) as task model. We note $\dot{\tau}$ the set $\{L_u(\tau), u \in \mathbb{Q}^{*+}\}$.

2.2 Validation

We have defined in [3][4] a technique, based on the Arnold-Nivat [1] model, to collect all valid scheduling sequences of a task system. The principle consists first in associating each critical resource R_j (processor, resource, message, etc.) with a virtual task V_{Rj} (modeled by a regular language $L(V_{Rj})$), and then associating the system $(\tau_i)_{i\in I}$ with the homogeneous product Ω of the $L(\tau_i)$ and the $L(V_{Rj})$. Let call S the subset of $\prod_{i\in I}(\Sigma_i)$ of vectors describing valid configurations (respecting mutual exclusion on processors or resources). We prove in [4] that language[6] $Proj_I(Center(L(\tau)_{i\in I} \cap S^*))$ collects the set of valid scheduling sequences, from resources management and timing constraints point of view.

---

[3] *Shuffle*(ш), is defined by the formula: $\forall a \in \Sigma$, a шε = a and $\forall(a,b,w,w') \in \Sigma^2 \times (\Sigma^*)^2$, aw ш bw'= a (w ш bw') $\cup$ b (aw ш w').

[4] The L *center* is the set of L prefixes indefinitely extendable in L, algebraically, Center (L^*) = L*.LeftFactors(L).

[5] t.u. for time unit

[6] $Proj_I(Center(L(\tau)_{i\in I} \cap S^*))$ is noted $\Omega_{i\in I\,S}(L(\tau_i))$ in the following.

Validating a real-time tasks system $(\tau_i)_{i\in I}$ consists in deciding if the configuration $(\tau_i)_{i\in I}$ can be scheduled in conformity with its time constraints. This decision is reached by evaluating the predicate $(center(\Omega_{i\in I\,S}(L(\tau_i))) = \emptyset)$ using an automaton associated with the language. If the language is empty, there exists a valid temporal behavior, then the configuration can be scheduled, otherwise there is no way to schedule the system.

3 Model for distributed systems

A distributed system is defined by a lack of common memory, a use of communication system and by the fact that there is no global state that can be observed [5]. Such a system is characterised by a set of sites, running with different speeds. Each site has a local clock that does not depend on others and that is the reference for every processor of the site. A clock is defined as an increasing sequence depending on time. A model that collects behaviors of a distributed system must be able to express simultaneity of different tasks placed on different sites, with no assumption concerning both a *global* time and correlations between the different speeds of the sites.

The model presented in section 2 is useful to validate real-time systems placed on a single site, possibly multi-processor. The speed of the site is implicitely modeled by the the time unit associated with the labels of the edges of the product automaton which models the software. A distributed system can be viewed (on a model way) as a set of such automata, each automaton being associated with its own time unit. As far as the target architecture is known and static, we know a priori the different speeds of the differents sites. To build an automaton that collects all behaviors of the system, we need two tools. First, a *zoom* technique, to accord the different automata with the same time semantics: we can not give a semantics to a product automaton $A\Omega B$, when A and B do not share the same time semantics. Second, a *start* result, to show that respective starting times of different sites have no incidence on the time validity of the software. The *zoom* technique is presented in section 3.1. The *start* result is obtained as an obvious corollary of properties of words of a regular language center.

3.1 *Zoom languages*

In $\dot{\tau}$ building process (recall that $\dot{\tau}=\{L_u(\tau), u \in \mathbb{Q}^{*+}\}$), we take various CPU speeds into account. We obtain a language class which satisfies the following property: for each word of each language of $\dot{\tau}$, the rate between **a**'s and •'s is a function of the cadence. Consider $L_u \in \dot{\tau}$. We call **granularity** the time associated with each letter duration into L_u words. It is the time semantics of each edge of the automaton. We note ${}_gL_u$ the observation of L_u with the granularity g (i.e. the *zoom rate* $\frac{g}{u}$).

To build the product $L_u \Omega L_v$, L_u and L_v must be observed with the same granularity. Then, we must be able to get $g \in \mathbb{Q}$ such that both ${}_gL_u$ and ${}_gL_v$

exist. Then, we must build the set $\overline{L_u(\tau)}$ of languages that collects behaviors of L_u in different granularities, i.e. the set of ${}_gL_u(\tau)$ associated with task τ running on a site that has **u** as a time unit and observed with a granularity **g**. To build $\overline{L_u(\tau)}$, we use the isomorphism ψ, defined by:
$\psi_{u,g} : P_c \cup \{a, \bullet\} \rightarrow (P_c \cup \{a, \bullet\})^k$ such that $k = u/g$, $\forall x \notin P_c, \psi(x) = x^k; \forall x \in \{P, S\}, \psi(x) = a^{k-1}.x; \forall x \in \{V, R\}, \psi(x) = x.a^{k-1}$.

Then $\overline{L_u(\tau)}$=$\{{}_gL_u(\tau), {}_gL_u(\tau) \subset \Sigma^* / \exists g \in \mathbb{Q}^{+*}, u \in g\mathbb{N}^*, {}_gL_u(\tau) = \psi_{u,g}(L_u(\tau))\}$. Given ${}_{g_1}L_{u_1}$ and ${}_{g_2}L_{u_2}$. We remark that ${}_{g_1}L_{u_1} \Omega {}_{g_2}L_{u_2}$ have a time semantics if and only if g_1=g_2. The model expresses simultaneity through the homogeneous product Ω of languages, our goal is then to find a granularity g which, applied to all sites, gives a temporal semantics to the product. To reach this aim, we extend in a natural way the GCD notion to $\mathbb{Q}$ (GCD operator is noted $\wedge$).

Theorem 1. *Given* $L_{u_1}(\tau_1)$ *and* $L_{u_2}(\tau_2)$. *Then,* $\exists g \in \mathbb{Q}^{+*}$, $g = u_1 \wedge u_2$ *and* ${}_gL_{u_1}(\tau_1) \subset \Sigma^*$, ${}_gL_{u_2}(\tau_2) \subset \Sigma^*$ *such that* ${}_gL_{u_1}(\tau_1) \in \overline{L_{u_1}(\tau_1)}$ *and* ${}_gL_{u_2}(\tau_2) \in \overline{L_{u_2}(\tau_2)}$

Obtained languages are maximal: $\nexists$ g'>g/ ${}_gL_{u_1}(\tau_1) \in \overline{L_{u_1}(\tau_1)}$, ${}_gL_{u_2}(\tau_2) \in \overline{L_{u_2}(\tau_2)}$. Moreover, ψ gives a constructing algorithm for this languages class. This theorem gives a technique to build a set of languages sharing the same granularity. This set allows to use homogeneous product for the composition of systems placed on different sites. This approach stands whatever the site speeds and start times and it can be used in the frame of multi-processor centralised systems that do not have a global clock.

3.2 Communication integration

In the previous part, we have established our model validity in the distributed case. We apply the homogeneous product to languages corresponding to each site. We note $L_{u_n}(S_n)$ the language associated to the site S_n.

Let $(L_{u_i}(S_i))_{i \in J}$ be the set of languages associated with sites. We use Theorem 1 (section 3.1): let be G=$\wedge_{i \in J}(u_i)$, and $({}_GL(S_i))_{i \in J}$ the set of languages such that: $\forall i \in J$, $L(S_i) = \psi_{u_i,G}(L_{u_i}(S_i))$. Languages $({}_GL(S_i))_{i \in J}$ are all built on the same granularity G, we can therefore build ${}_GL = (\Omega({}_GL(S_i))_{i \in J})$. ${}_GL$ gives the system $(\tau_i)_{i \in I}$ model on sites $(S_j)_{j \in J}$.

To temporarily validate this system, we have to integrate communication protocols into the model. Our aim is to warrant that message transmissions stay in temporal terms. Then, it is necessary to have a model for network behaviors. To obtain the model for all the drivers, we first model one of them, and then their simultaneous run using the Arnold-Nivat's product. The driver task is duplicated in order to run on each site of the system, on a dedicated processor (the network board CPU). The model language that collects driver behaviors is called D, it is built on the network. The language D has the granularity g_D of one bit transmission duration. For all j in J, ${}_{g_D}D_{g_D}$ is the driver i associated language. All drivers share the same code and then the same language. Let *Prot* be the synchronisation set expressing protocol communication constraints. Then ${}_{g_D}R = \Omega_{i \in I\, Prot}({}_{g_D}D)$, is the model for the network. This method can be used for any protocol that supports an automaton based model.

We check then the compatibility of message transmission and application timing constraints. To warrant application terms, a message must be transmitted in a limited time: it has a deadline. We use a virtual task Stw (stopwatch) that keeps a record of the elapsed time between a message *Send* and its *receive*. This deadline, associated with a network model, allows to decide the compatibility between transmission and timing constraints. Granularity of Stw is G. Let ${}_G\text{Msg}=\text{Proj}_{Stw}({}_G\text{L}\ \Omega_{Sr}\ {}_G\text{Stw})$, computed with a resource synchronisation (Sr) on the stopwatch [sec.2.2]. To test transmission validity, we must compute the languages ${}_G\text{Msg}$ and ${}_{gD}\text{R}$ homogeneous product. We apply Theorem 1: using $\text{H}=\text{G}\wedge\text{g}$, we get ${}_H\text{Msg}$ and ${}_H\text{R}$.

Language ${}_H\text{L}=\text{Center}({}_H\text{Msg}\Omega_{Sr\,H}\text{R})$ collects the set of valid messages scheduling (respecting timing constraints) on the network. The validity test is the same as in centralised system validation: If ${}_H\text{L}=\emptyset$ then there is no valid behavior, otherwise, there exists at least one.

4 Conclusion

Languages $\text{L}_u(\tau)$ are useful to validate hard real time distributed systems, if they are based on protocols that can be modeled by regular languages. The centralised model was extended by considering processors speed and by defining a *zoom* operation on languages. This last tool, associated with a generalisation of GCD to $\mathbb{Q}$, is useful to model with finite automata distributed systems with no addition of restrictive hypothesis (the only one is the closure of the modeled system: this is not a restriction when considering real-time systems!).

The result is a schedulability decision for the application on a distributed architecture. One of the central corollaries of this approach is the cyclicity of scheduling sequences in distributed multi-processor environnement: this result is an immediate corrolary (star lemma) of the fact that valid scheduling set is a regular language.

This work is ongoing. Our present studies concern both the integration of task migration and the integration of a small level of non determinism by considering alarm events.

References

1. A.Arnold: Finite transition systems. Prentice Hall. (1994).
2. S.K.Baruah, L.E.Rosier, R.R.Howell: Algorithms and Complexity Concernig the Preemptive Scheduling of Periodic Real-Time Tasks on one Processor. RTS. (1990).
3. D.Geniet: Validation d'applications temps réel à contraintes strictes à l'aide de languages rationnels. RTS. (2000).
4. D.Geniet, G.Largeteau: Validation d'applications temps réel strictes à durées variables à l'aide de languages rationnels. MSR. Toulouse. (2001).
5. M.Raynal: Synchronisation et état global dans les systèmes répartis. Eyrolles.(1992).