Component-Based Design in Multiprocessor Real-Time Systems

Sanjoy Baruah  
Department of Computer Science  
The University of North Carolina at Chapel Hill, USA  
Email: baruah@cs.unc.edu

Nathan Fisher  
Department of Computer Science  
Wayne State University  
Detroit, MI USA  
Email: fishern@cs.wayne.edu

Abstract—Component-based design for real-time systems has primarily focused upon integrating subsystems upon single processor platforms. In this paper, we propose an abstraction for components with real-time requirements co-executing upon a multiprocessor platform. We show this abstraction may be efficiently computed and integrated into known schedulability tests from multiprocessor platforms. Furthermore, we discuss the advantages and disadvantages of our proposed abstraction and highlight some areas of future research for component-based design for multiprocessor real-time systems.

Index Terms—Real-time systems; compositional analysis; schedulability analysis; multiprocessor platforms; interface specification.

I. INTRODUCTION

As real-time embedded systems become increasingly more complex, the benefits of constructing such systems by assembling simpler components is becoming ever more apparent. Each component is a (sub)system; larger components are obtained by integrating or composing smaller ones. There are many advantages to such a component-based approach towards complex system design, including component reuse, modular synthesis and analysis, and reconfigurability.

Several component-based design and analysis methodologies have been proposed for real-time systems that are to be implemented on uniprocessor platforms. (For example, the server-based approaches studied extensively in the real-time systems research community may be considered in this light. A non-exhaustive list of different uniprocessor component-based approaches includes [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]) Although some research has addressed multiprocessor platforms (the work reported in [13], [14] being a couple of notable examples), far less work has been done in this regard even though, due to cost and related considerations, there has recently been a lot of interest in exploring the possibility of implementing embedded real-time systems on multiprocessor platforms. Hence, there is a strong need for more work to be done on component-based design and analysis techniques for real-time embedded systems that are to be implemented on multiprocessor platforms.

In [15], we discussed preliminary ideas for techniques in multiprocessor component-based design for real-time systems. In this paper, we explore these ideas and provide details on our proposed abstraction for component-based multiprocessor real-time systems. Section II introduces our machine and computation model. Section III describes our proposed abstraction for representing real-time components for multiprocessor systems. Section IV gives (and derives) a multiprocessor schedulability test for components represented by this abstraction. Section V discusses some drawbacks of our approach and ideas for future work.

II. MODEL AND SCOPE

We restrict ourselves in this report to a very simple platform and component model. We assume that a hard-real-time system is being designed for implementation on a platform comprised of multiple shared-memory preemptive processors. In our model, a multiprocessor platform is comprised of \( m \) identical processors each of unit-speed (i.e., a processor executes one unit of execution for each time unit).

We assume that the workload of the real-time system under consideration is comprised of basic units of work known as independent jobs. Each job \( J = (A, E, D) \) is characterized by an arrival time \( A \), a worst-case execution requirement (WCET) \( E \), and a relative deadline \( D \). (A job’s hard or absolute deadline is at time \( A + D \).) We define the density of a job \( J = (A, E, D) \) to be the ratio of its WCET to its relative deadline: \( \text{density}(J) = E/D \). The jobs in the system may be generated by recurrent tasks which are specified according to some task model like the periodic [16] or sporadic [17], [18] model. Throughout the remainder of this paper, we will use the sporadic task model (introduced below) as an example of a recurrent task.

For the sake of concreteness, we will assume that the processors of our multiprocessor platform are scheduled according to the global earliest-deadline-first (EDF) scheduling algorithm. At any given time-instant, Global EDF schedules, from the set of jobs with remaining execution, at most \( m \) jobs with the nearest absolute deadline. We make the simplifying assumption that a job may be preempted at any time and resume execution at a later time (potentially on a different processor) with no execution penalty. Future work on multiprocessor scheduling is required to remove this assumption.

a) Sporadic Task Model: A sporadic task \( \tau_i = (c_i, d_i, p_i) \) is characterized by a worst-case execution requirement \( c_i \), a (relative) deadline \( d_i \), and a minimum inter-arrival...
PECIFYING COMPONENTS

maximum is the sum of the and a deadline's of its constituent components. Each job has a worst-case execution requirement equal to \(e_i\) and a deadline that occurs \(d_i\) time units after its arrival time. A useful metric for a sporadic task \(\tau_i\) is the task utilization \(u_i = \frac{e_i}{p_i}\).

III. SPECIFYING COMPONENTS

As stated in Section II above, we are studying hard-real-time systems of jobs that are to be scheduled on a preemptive multiprocessor platform, using global EDF. Our goal is to synthesize such systems under the guarantee that all job deadlines will always be met; our approach towards achieving this goal is to use a component-based design technique.

b) Components.: We assume a fairly standard notion of components: each component is either a single recurrent task, or is comprised of a finite number of constituent components. We make the independence assumption regarding the constituent components: i.e., we assume that the different constituent components of a component are independent of one another, in the sense that the jobs generated by each constituent component do not synchronize in any manner with jobs of another constituent component. (The independence assumption may result in some increased pessimism in that any analysis of the component must necessarily assume that each individual constituent component separately exhibits worst-case behavior even if such simultaneous worst-case behavior is not possible in practice; however, it appears that such pessimism is unavoidable if analysis is to be kept tractable. See [19] for a discussion concerning similar independence assumptions in the context of recurrent task models.)

With this definition, we can represent our overall system as a tree in which each node represents a component with the root node representing the entire system, the children of a node denoting its constituent components, and each leaf node denoting a component comprised of a single recurrent task. (Note that this tree structure rules out the sharing of constituent components by different components.)

c) Component interface.: The component interface represents an abstraction that encapsulates the resource (in our simple case, processor) requirements of the component. Experience has shown that designing the appropriate interface is critical to the success or failure of a component framework. It is particularly desirable that interfaces be efficiently composable: given the interfaces of the constituents of a component, it should be possible to efficiently determine the interface of a component. With this compositability requirement in mind, we propose that a component be specified by specifying two attributes: its demand bound function (DBF) and its maximum job density (DENSITY):

DBF: For any component \(C\) and any positive real number \(t\), the demand bound function \(DBF(C, t)\) of component \(C\) for interval-length \(t\) denotes the maximum cumulative execution requirement by jobs of component \(C\) that may both arrive in, and have deadlines within, any time interval of length at most \(t\).

DENSITY: For any component \(C\), the maximum job density \(DENSITY(C)\) of \(C\) denotes the maximum density of any job that could possibly be generated by \(C\); e.g., if \(J_C\) is the set of jobs generated by component \(C\), then \(DENSITY(C) = \max_{J_i \in J_C} \{DENSITY(J_i)\}\).

d) Computing the DBF.: For any component \(C\), \(DBF(C, t)\) is a monotonically non-decreasing function of the time parameter \(t\). Closed-form representations are known for the DBF’s of many recurrent task models such as the sporadic task model [18] and the recurrent real-time task model [20]; these representations immediately yield closed-form representations of the DBF’s of components that are comprised of a single recurrent task. For components comprised of multiple constituent components, the independence assumption implies that a component’s DBF is the sum of the DBF’s of its constituent components.

e) Computing the DENSITY.: The DENSITY\((C)\) of a component \(C\) comprised of a single recurrent task is equal to the density of the largest job that can be generated by the task. For a component \(C\) comprised of multiple constituent components, \(DENSITY(C)\) is equal to the maximum of the DENSITY’s of its constituent components.

f) Components comprised of sporadic tasks.: As a concrete example, we will now describe how to determine the DENSITY and DBF parameters for a component comprised of sporadic tasks. Obviously, the density of a sporadic task \(DENSITY(\tau_i)\) is equal to \(e_i/d_i\). Baruah et al. [18] have shown that for sporadic tasks that DBF can be calculated as follows.

\[
DBF(\tau_i, t) = \max\left(0, \left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1 \right) \cdot e_i.
\]

Figure 1 gives a visual depiction of the demand-bound function for a sporadic task \(\tau_i\). Observe from the above definition and Figure 1 that if the DBF is a right continuous function with discontinuities at time points of the form \(t \equiv d_i + a \cdot p_i\) where \(a \in \mathbb{N}\). A component \(C\) comprised of only sporadic tasks has \(DBF(C, t)\) equal to \(\sum_{\tau_i \in C} DBF(\tau_i, t)\).

IV. Schedulability Analysis

In Section III above, we proposed an interface for components, and described how such interfaces may be composed. We now explain why we have proposed the particular interface that we did.

Several sufficient tests have been proposed over the past few years (see, e.g., [21], [22], [23], [24], [25]) for performing global EDF schedulability analysis of sporadic task systems upon preemptive multiprocessor platforms. Comparisons via extensive simulations (see, e.g., [26]) have revealed that these tests are incomparable to one another, in that there are different schedulable task systems that are successfully identified as being so by only one of the tests but by none of the others.

We have chosen to base our schedulability testing on the sufficient schedulability test presented in [25], [27]. This is
Consider any sequence of job arrivals for components of the system such that each component does not violate DBF or DENSITY parameters. Suppose a job $J_i = (A_i, E_i, D_i)$ of component $C$ misses its deadline at time $t_d$ when scheduled by EDF and that $t_d$ is the earliest such deadline miss of any job in the system. Due to $J_i$’s job parameters, $A_i = t_d - D_i$.

For the above sequence of jobs, we remove jobs with absolute deadlines after $t_d$, since they have no effect on the deadline miss at time $t_d$ (i.e., job $J_i$ of component $C$ will still miss a deadline if these jobs are removed). Thus, we have a new sequence of job arrivals where each job has deadline at or prior to $t_d$ and job $J_i$ misses a deadline at $t_d$. We now introduce some notation that will be useful for the remainder of the proof. For any $t \leq t_d$:

- $W(t)$ denotes the cumulative execution requirement of all the jobs in the above sequence, minus the total amount of execution completed by EDF prior to time-instant $t$.
- $\Omega(t)$ denotes $W(t)/(t_d - t)$.

Because $J_i$ missed a deadline, it executed for strictly less than $E_i$ time units over the interval $[A_i, t_d]$ (equivalent to $[A_i, A_i + D_i]$). Thus, over this interval all $m$ processors must have been executing jobs other than $J_i$ for a total duration of at least $D_i - E_i$ time units. Thus, it must be (due to the definition of $W(t)$) that $W(A_i) > (D_i - E_i)m + E_i$. From the definition of $\Omega(t)$, it follows that

$$\Omega(A_i) \geq \frac{(D_i - E_i)m + E_i}{D_i} \geq m - (m - 1) \frac{E_i}{D_i} \geq m - (m - 1) \text{DENSITY}(C)$$

Let $\mu \equiv m - (m - 1) \text{DENSITY}(C)$.

Let $t_0$ denote the smallest value of $t \leq A_i$ such that $\Omega(t) \geq \mu$. Denote $\Delta \equiv t_d - t_0$.

Now consider total execution that contributes to the value of $W(t_0)$. The job execution contributing to this value arise from two different types of jobs:

1) Jobs that arrive at or after time $t_0$. (Note these jobs necessarily have absolute deadlines prior to $t_d$ since we have removed any job with deadline later than $t_d$.)

2) Jobs that arrive prior to time $t_0$ but execute in the interval $[t_0, t_d]$ in the EDF schedule. We refer to these jobs as carry-in jobs.

Below we prove a lemma regarding the amount each carry-in job may execute over the interval $[t_0, t_d]$.

Lemma 1: Each carry-in job of component $C$ has $< \Delta \times \text{DENSITY}(C)$ remaining execution requirement at time $t_0$.

Proof: Consider any carry-in job $J_k = (A_k, E_k, D_k)$ of component $C$. (Observe that $A_k < t_0$ and $J_k$ has not completed its execution by $t_0$, by definition of carry-in job).

Let $\phi = t_0 - A_k$. From the definition of $t_0$, it must be that $\Omega(A_k) < \mu$. That is,

$$W(A_k) < \mu(\Delta + \phi).$$

However, $\Omega(t_0) \geq \mu$, implying that

$$W(t_0) \geq \mu \Delta.$$
From inequalities 4 and 5 and the definition of \( W(\cdot) \), the amount of execution of jobs of \( C \) in the EDF schedule over \([A_k, t_0]\) is less than \( \mu \phi \) (i.e., the execution over this interval is equal to \( W(A_k) - W(t_0) \)). Let \( y \) denote the time over the interval \([A_k, t_0]\) during which \( J_k \) is executing. All \( m \) processors must be executing jobs for the remaining \( \phi - y \) time units in this interval. This implies that the total amount of work done by EDF over \([A_k, t_0]\) is at least \( m(\phi - y) + y \). From the upper and lower bound on the amount of work over the interval \([A_k, t_0]\), we have

\[
\begin{align*}
\text{upper bound:} & \quad m(\phi - y) + y < \mu \phi \\
\text{lower bound:} & \quad m \phi - (m - 1)y < (m - (m - 1)\text{DENSITY}(C))\phi \\
& \quad \text{implies} \quad y > \phi \times \text{DENSITY}(C).
\end{align*}
\]

The above inequality implies that \( J_k \) must have completed, over the interval \([A_k, t_0]\), at least \( \phi \times \text{DENSITY}(C) \) units of execution. Thus, the remaining execution of job \( J_k \) at time \( t_0 \) (which contributes to the execution quantified by \( W(t_0) \)) is

\[
(E_k - y) < (D_k \times \text{DENSITY}(C) - \phi \times \text{DENSITY}(C)) = (D_k - \phi) \times \text{DENSITY}(C).
\]

Since our sequence only consists of jobs with absolute deadline prior to \( t_d \), it must be that \( J_k \)'s absolute deadline is at most \( t_d \) (i.e., \( A_k + D_k \leq t_d \)). Therefore, \((D_k - \phi) \leq \Delta \). By the above inequality, it follows that the remaining execution of job \( J_k \) at time \( t_0 \) is at most \( \Delta \times \text{DENSITY}(C) \). \( \blacksquare \)

In the next lemma, we bound the number of carry-in jobs that may exist for the interval \([t_0, t_d]\). The proof of the lemma is nearly identical to one found in [25]; thus, we provide only a brief proof sketch.

**Lemma 2**: The number of carry-in jobs is bounded from above by \( \lceil \mu \rceil - 1 \).

**Proof Sketch**: By definition of time-instant \( t_0 \), for any time \( t \) prior to \( t_0 \), \( \Omega(t) < \mu \). However, if there exist \( \lceil \mu \rceil \) or more carry-in jobs for the interval \([t_0, t_d]\), then there exists an arbitrarily small \( \epsilon > 0 \) where each of the carry-in jobs is active over the interval \([t_0 - \epsilon, t_0]\); this implies that over the interval \([t_0 - \epsilon, t_0]\) at least \( \lceil \mu \rceil \) jobs of \( C \) are executing (i.e., \( \Omega(t_0 - \epsilon, t_0) \geq \mu \)). However, this directly contradicts the definition of \( t_0 \). Thus, the number of carry-in jobs must at most \( \lceil \mu \rceil - 1 \). \( \blacksquare \)

Lemmas 1 and 2 can be combined to prove the desired schedulability test. Again, we provide only a proof sketch as the full proof is nearly identical to one contained in [25].

**Theorem 1**: Component \( C \) is global-EDF schedulable upon a processing platform comprised of \( m \) unit-capacity processors, if equation 2 is satisfied.

**Proof Sketch**: The proof relies upon three main facts:

1) From Lemma 1, the amount of execution that each carry-in job contributes to the interval \([t_0, t_d]\) (where \( \Delta = t_d - t_0 \)) is strictly less than \( \Delta \times \text{DENSITY}(C) \).

2) From Lemma 2, the number of carry-in jobs is at most \( \lceil \mu \rceil - 1 \).

3) The maximum contribution of jobs that arrive (and have absolute deadline) within the interval \([t_0, t_d]\) is \( \text{DBF}(C, \Delta) \). The expression \( \text{DBF}(C, \Delta) \) may be upper bounded by \( \Delta \times \max_{t>0} \left( \frac{\text{DBF}(C, t)}{t} \right) \).

From the above facts, we may obtain an upper-bound on \( W(t_0) \) in the case of a deadline miss at time \( t_d \):

\[
W(t_0) < \left[ \Delta \times \max_{t>0} \left( \frac{\text{DBF}(C, t)}{t} \right) \right] + \left( \lceil \mu \rceil - 1 \right) \Delta \times \text{DENSITY}(C) \cdot \text{DENSITY}(C) \cdot \Delta.
\]

By dividing both sides of the above inequality and noting that \( \Omega(t_0) \geq \mu \), we obtain

\[
\max_{t>0} \left( \frac{\text{DBF}(C, t)}{t} \right) \leq \mu - (\lceil \mu \rceil - 1) \text{DENSITY}(C).
\]

Thus, for a deadline miss to occur for component \( C \), the above inequality is a necessary condition. By negation of the above necessary condition, a sufficient condition for component \( C \) to be EDF-schedulable is obtained:

\[
\max_{t>0} \left( \frac{\text{DBF}(C, t)}{t} \right) \leq \mu - (\lceil \mu \rceil - 1) \text{DENSITY}(C).
\]

\( \blacksquare \)

\( h) \) Pragmatic considerations: approximating \( \text{DBF} \)'s.: For reasons of computational efficiency, we would probably like to approximate the \( \text{DBF} \)'s. Much work has been done on such approximations in the context of individual tasks — see, e.g., [28], [29], [30], [31], [32]. The essence of these approximations is this — the \( \text{DBF} \) of a task, considered as a function of \( t \), appears as a step function. This step function can be approximated by representing it exactly for the first several steps (equivalently, for small \( t \)), and then bounding it from above by a straight line with slope that is equal to the utilization of the task. Such a notion of approximations to the \( \text{DBF} \) can be generalized for components. As is the case with tasks, there is a trade-off between the accuracy of the approximation and the computational complexity of the schedulability analysis: we can render the test more efficient (in the sense that the “\( \max \)” in Condition 2 needs to be evaluated over fewer values of \( t \)) by over-estimating the \( \text{DBF} \) by a larger amount. We can also trade off computational complexity for efficient usage of processor capacity by choosing to apply such approximations at each level of the component hierarchy (i.e., we could approximate the \( \text{DBF} \) of each component rather than performing the approximation just once, at the root node that represents the entire system).

Again, to be concrete, we will briefly describe how such an approximation would be done for sporadic tasks. Albers and Slomka [29] proposed the following approximation to \( \text{DBF} \) to reduce the number of discontinuities (we will show below how this speeds up the computation of Condition 2).

\[
\text{DBF}(\tau, t, k) \triangleq \begin{cases} 
\text{DBF}(\tau, t), & \text{if } t < d_i + (k - 1)p_i; \\
\max_i (u_i \cdot (t - d_i) + e_i), & \text{otherwise.}
\end{cases}
\]

(7)
Notice the \( \overrightarrow{DBF}(\tau, t, k) \) is that it “tracks” DBF for exactly \( k \) discontinuities (i.e., “steps”). After \( k \) discontinuities, \( DBF(\tau, t, k) \) uses a linear interpolation of the subsequent discontinuous points (with slope equal to \( u_j \)). The steps with the thick lines and the sloped-dotted line in Figure 1 correspond to \( DBF(\tau, t, 3) \). Albers and Slomka show [29] that the above approximation may be integrated into a uniprocessor EDF-schedulability condition. We may extend their approximation to Condition 2 for multiprocessor systems. Consider a component \( C \) comprised of sporadic tasks and a fixed \( k \in \mathbb{N}^+ \). We will use \( DBF(C, t, k) \) to denote \( \sum_{\tau_i \in C} DBF(\tau_i, t, k) \). Albers and Slomka [29] prove that \( DBF(C, t) \leq DBF(C, t, k) \) for all \( t \geq 0 \) and any value of \( k > 0 \). Thus, we may replace Condition 2 with the following inequality:

\[
\max_{t>0} \left( \frac{DBF(C, t, k)}{t} \right) \leq \mu - (\lfloor \mu \rfloor - 1) \times DENSITY(C). \tag{8}
\]

How does using \( \overrightarrow{DBF} \) improve the computational complexity of the schedulability test? For Condition 2, it has been shown [31] (in general) that the number of values of \( t \) for which the ration in the left-hand side must be evaluated may be exponential in the number of tasks in \( C \). However, for Condition 8, Albers and Slomka [29] show that it suffices to evaluate the left-hand side of Condition 8 condition for all \( t \) in the following set:

\[
\overrightarrow{TS}(C, k) \triangleq \bigcup_{\tau_i \in C} \left\{ t \equiv d_i + a \cdot p_i \mid (a \in \mathbb{N}^+) \wedge (a < k) \right\}, \tag{9}
\]

and to check whether \( \sum_{\tau_i \in C} u_i \leq \mu - (\lfloor \mu \rfloor - 1) \times DENSITY(C) \). Thus, the number of values of \( t \) necessary to evaluate Condition 8 is \( \mathcal{O}(k \times |C|) \) which is polynomial in the size of \( C \).

Albers and Slomka also show the following desirable property of \( DBF \): the accuracy of the approximation increases with larger values of \( k \).

**Lemma 3 (from [29]):** Given a fixed integer \( k \in \mathbb{N}^+ \), \( DBF(\tau_i, t) \leq DBF(\tau_i, t, k) \leq \left( \frac{k+1}{k} \right) DBF(\tau_i, t) \) for all \( \tau_i \in \tau \) and \( t \in \mathbb{R}_{\geq 0} \).

The above lemma implies the following: \( \overrightarrow{DBF} \left( C, t, \left\lfloor \frac{1}{\epsilon} \right\rfloor \right) \leq \left( 1 + \epsilon \right) DBF(C, t) \) for all \( t \geq 0 \). Therefore, we may approximate DBF using \( \overrightarrow{DBF} \) to within an arbitrary multiplicative factor \( (1 + \epsilon) \) for \( \epsilon > 0 \), by setting \( k \) to be equal to \( 1/\epsilon \).

1. **A design choice: choosing DENSITY.** As a practical matter, the system designer would probably specify a system-wide bound on DENSITY, and require all “base” components (those comprised of single recurrent tasks) to be compliant with this DENSITY bound. With such a pre-specified value for DENSITY, it becomes easier to compare different components: in choosing between two components a component with DBF greater than the other component’s for most values of \( t \) is more likely to compromise the schedulability of the overall system.

V. SHORTCOMINGS AND FUTURE WORK

While the suggested component abstraction has the significant advantage of being easily integrated into multiprocessor schedulability tests, there are some shortcomings and inadequacies of the approach advocated in this document. We discuss a few of them below.

1. The assumption of independence among constituent components is a cornerstone for the framework proposed in this document. However (as pointed out in Section II above), the independence assumption results in increased pessimism; in addition, it does not consider at all other resources — memory, bandwidth, I/O, etc. — that are essential to the synthesis of actual embedded systems.

2. We are assuming that all components can be trusted; the interface of a component represents its actual behavior. For non-trustworthy components, it is necessary to have strategies in place for run-time policing of the behavior of components, and policies for isolating well-behaved components from the ill-effects of misbehaving ones. Policing for DBF-compliance is a very non-trivial issue, which we have not addressed in this document.

3. Meanwhile, research continues on better global schedulability tests. Although we have based the discussion here on the sufficient EDF-schedulability test of [25], it would probably be premature to commit to any particularly schedulability test at the present time. It is quite likely that significantly superior tests will soon be developed; if such tests turn out to also be amenable to compositional analysis, they may prove to be superior bases upon which to build component-based synthesis and analysis techniques.

Due to these limitations, the framework proposed in this document should be thought of as a collection of integrated ideas that may motivate further research. We hope that these ideas serve as a foundation for future more sophisticated component abstractions for multiprocessor systems that address some of the shortcomings identified above.

VI. CONCLUSIONS

Real-time component-based design has been successfully applied to uniprocessor systems. The real-time component-based design frameworks proposed for uniprocessors have enabled designers to integrate different components with real-time requirements and effectively predict the timing behavior of each individual component. The purpose of our paper is to propose a component-based design framework for multiprocessor systems. For this goal, we propose that each component abstraction be specified by a demand-bound function and a density function. The combination of these two functions quantifies the real-time requirements of a component upon a multiprocessor platform. We show that the proposed component abstraction has the significant advantage of being easily integrated into known schedulability conditions. (We formally prove the integration). Furthermore, approximation techniques may be applied to the component to reduce the computational complexity of the schedulability analysis for
components. We also discuss possible disadvantages of the proposed abstraction which are interesting basis for future work on real-time multiprocessor component-based design.

ACKNOWLEDGMENT

This work was supported in part by a Wayne State University Faculty Research Award, NSF Grant Nos. CNS-0834270, CNS-0834132, and CNS-0615197, ARO Grant No. W911NF-06-1-0425, and funding from IBM, Sun Microsystems, and the Intel Corporation.

REFERENCES