Journal of Universal Computer Science, vol. 15, no. 11 (2009), 2142-2165
submitted: 18/1/09, accepted: 29/5/09, appeared: 1/6/09 © J.UCS

An Approach for Estimating Execution Time Probability Distributions of Component-based Real-Time Systems

Ricardo Perrone
(Federal University of Bahia, Distributed Systems Lab. (LaSiD)/Computer Science Dep., Salvador-BA, Brazil
perrone@ufba.br)

Raimundo Macêdo
(Federal University of Bahia, Distributed Systems Lab. (LaSiD)/Computer Science Dep., Salvador-BA, Brazil
macedo@ufba.br)

George Lima
(Federal University of Bahia, Distributed Systems Lab. (LaSiD)/Computer Science Dep., Salvador-BA, Brazil
gmlima@ufba.br)

Verônica Lima
(Federal University of Bahia, Statistics Dep., Salvador-BA, Brazil
cadena@ufba.br)

Abstract: In recent years, many component-based real-time systems have been proposed as a solution to modular and easily maintainable distributed real-time systems. This paper proposes a methodology for estimating probability distributions of execution times in the context of such systems, where no access to component internal code is assumed. In order to evaluate the proposed methodology, experiments were conducted with components, and related compositions, implemented over CIAO and ARCOS. CIAO is a known real-time component-based middleware and ARCOS is a software framework devoted to the construction of real-time control and supervision applications, also developed over CIAO. The collected experimental data show that the proposed approach is indeed a good approximation for component execution time probability distributions.

Key Words: Real-Time Systems, Response Time Estimation, Distributed Middleware, COTS

Category: D.2, D.2.2, D.2.6

1 Introduction

The requirements of modern real-time distributed systems, such as more flexibility, interoperability, and cost savings, have motivated both the use of software-intensive solutions and the exploitation of hardware and software COTS (Commercial Off-The-Shelf). In this context, distributed software components appear as a promising technology, since their modular and uncoupled approach used for implementing and combining components leads to reusable and easily maintainable systems. Therefore, the use of real-time component-based middleware for the construction of real-time distributed systems has deserved a great deal of attention from the research community in the last years [Wang et al., 2004a].

Whereas it is widely acknowledged that applications can benefit from the adoption of such modular solutions, where COTS can be integrated into a system provided that their implementations comply rigorously with the specified interfaces, the lack of knowledge of the component implementation (black-box approach) makes it difficult to verify the timeliness guarantees of the whole system, which may turn out to be a barrier to use such an approach in the real-time arena. As a result, several aspects on the general use of COTS need to be analyzed. Indeed, possible risks and impacts that are imposed to the project due to the adoption of external components must be taken into consideration [Li et al., 2008].

A conventional way of verifying the timeliness of a safety-critical real-time system is by schedulability analysis [Liu, 2000], which determines whether some system task may miss its deadline in worst-case scenarios. To do so, one must first calculate the worst case execution time (WCET) of each task in isolation and then combine such WCETs in formulas that capture the worst-case execution scenarios for the analyzed system. In order to precisely calculate the WCET of a given task, one must take into account each instruction execution time of the task worst case execution path. Variations of instruction execution times due to mechanisms such as memory caches and pipelines should be also considered.

Unfortunately, when COTS are used it is not always possible to apply such an approach. Indeed, only the component interface may be available and traditional time analysis requires access to the component code. Hence, such COTS-based real-time systems have limited applicability on safety-critical applications where missing deadlines can cause great losses. On the other hand, there are other real-time scenarios, such as multimedia, telecommunication, and some industrial applications, where missing a deadline causes only a quality of service degradation, but it is still tolerable given that the probability of such misses is below a certain limit. Therefore, finding alternative ways of estimating response times of such COTS-based real-time systems is an important challenge to be faced by the research community.

This paper tackles this challenge by proposing and validating a methodology for estimating probability distributions of execution times in the context of a component-based distributed real-time system. These estimations can help designers to verify the timeliness of component-based systems by applying alternative models of timeliness analysis [Kim et al., 2005], [Manolache et al., 2001]. The ultimate motivation of the proposed approach is to apply it to estimate response times of services developed in the context of ARCOS, a component-based framework implemented atop CIAO and devoted to the construction of industrial control and supervision distributed systems [Andrade and Macêdo, 2007].

In order to evaluate the proposed methodology, experiments have been conducted with components implemented over CIAO and ARCOS, and the related probability distributions estimated. The analysis of performance data shows that the proposed approach is indeed a good approximation for estimating component execution time probability distributions.

This is an extended version of our paper originally published in SBCARS2008 (2^{nd} Brazilian Symposium on Components, Architectures and Software Reuse), which introduced a methodology for estimating component execution time probability distributions. For the journal version, the paper was entirely revised. The original approach was extended to deal with component compositions and a simulated car cruise control application, developed on top of ARCOS, was applied as a new case-study to evaluate the proposed method.

The remainder of this paper is structured as follows. In section 2 related work is discussed. Section 3 summarizes the component-model used for implementing the proposed approach. In section 4 the assumed system model and the new approach for estimating the component execution time probability distributions are presented. A case study is also used in this section to illustrate and validate the efficacy of the presented approach. In section 5 the presented approach is extended to deal with component compositions. Finally, in section 6 conclusions are drawn and future work is pointed out.

2 Related Work

Most work in the field of time analysis of real-time systems is on estimating WCET with the best possible accuracy. Together, these studies show different approaches to addressing the factors of influence (input data, execution path, cache memory and pipeline) that determine the occurrence of the worst case system behavior. Usually, they aim at reducing the pessimism and increase the accuracy of WCET calculations. However, given the extreme difficulty in determining the exact value, the maximum they can achieve is an estimate of WCET that is sufficiently secure for many real-time applications.

The techniques related to this topic of study can be broadly divided into the following categories: static, probabilistic, and hybrid. The publication list on static analysis is considerably large where the focus is on analyzing the execution paths of the application code to derive the values of WCET for a given hardware platform. Recent work applies such an approach to component-based real-time systems [Estévez-Ayres et al., 2005], [Ballabriga et al., 2007], [Fredriksson, 2006], [Moss and Muller, 2004]. Usually static approaches tend to be complex and produce over-pessimistic estimations and/or need the knowledge to the source code of application. This has motivated the development of probabilistic and hybrid estimation techniques, which are more related to our work.

A pioneering work on probabilistic WCET estimation is based on sampling task execution times by measurements and then their WCET are estimated using extreme value statistics [Edgar and Burns, 2001]. The authors assume that the sampled data follow a Gumbel distribution and are independent and identically distributed. Although this is a black-box approach, the assumption on data independence may limit the applicability of this approach for practical systems. The work by [Bernat et al., 2002] is hybrid since it combines both the static and the probabilistic approaches. They determine the execution time profiles of code blocks by measurements. Execution time profiles are actually a table that contains the execution time observed and its frequency. Then, they obtain a WCET value (with an associated probability) by convolving and combining these profiles. Certain data and execution paths dependencies are taken into consideration, although the fine-grained instrumentation of application code is necessary. A scheme to store execution time profiles of component-based systems is also presented by [Nolte et al., 2003]. They argue that the components of the system should keep monitoring themselves. Based on this scheme, a hybrid WCET estimation technique is proposed [Möller et al., 2005].

Instead of estimating a value for WCET, this paper focus on deriving the execution time probability distributions in component-based systems. Such distributions have been used in approaches to schedulability analysis [Kim et al., 2005], [Manolache et al., 2001]. Similar to probabilistic and hybrid approaches, the proposed estimation method is based on measurements. However, it is not required any prior knowledge about the internal code of the application for carrying out the measurements. The instrumentation of the code was used only for the purpose of validation of the final results. We assume, for example, that there may be components developed by third-part development teams, so the knowledge of application code may not be available. This makes static or hybrid estimation approaches clearly unsuitable for this context. Moreover, unlike some probabilistic approaches [Edgar and Burns, 2001], we do not assume any particular distribution and may consider complex systems, where data independence may not be ensured. The proposed method is therefore appropriate to COTS oriented models - compiled version. In this paper we first present how to estimate a probability distribution to a simple component, and after that we extend the proposed method to apply it on a composition of components developed with the ARCOS framework.

3 Architecture of Components

3.1 System Model and Assumptions

The work developed considers a system of components that communicate through the connection of standardized interfaces, known as ports. Through their ports, a component specifies the features that will be made available externally as services to other components. Thus, each component can provide or consume services through connections. A service may be built from a number of components, and the execution paths that can be activated will depend on the related component connections.

Components are implemented by processes and communicate by exchanging messages through communication channels. Processes and channels are assumed to be timely: process steps and message delivery are carried out within bounded times. However, such bounds or deadlines may be missed from time to time. Hence, it is assumed to be a soft real-time system.

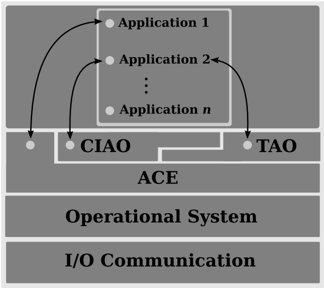
In order to evaluate the methodology proposed in this paper, we have implemented case studies on top of the middleware CIAO (Component Integrated ACE ORB) [Wang et al., 2004a] and the framework ARCOS, recently developed (*Architecture for Control and Supervision*) [Andrade and Macêdo, 2005], [Andrade and Macêdo, 2007]. In the next two sections we briefly present CIAO and ARCOS and the corresponding terminology that will be used throughout the text. However, the proposed method is generic enough to be applied to other types of component-based middleware.

3.2 The CIAO Real-time Middleware

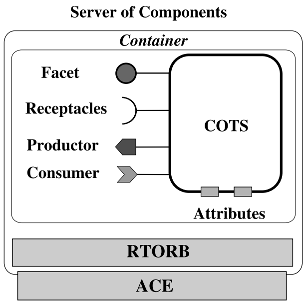
The model of CIAO is compliant with a ligthweight version of Corba Component Model - CCM [OMG, 2006], a specification that describes a standard architecture for CORBA components. CIAO implements a server of real-time components, called container, which supports non-functional services such as security, data persistence, and component life-cycle operations, and inter-component communication facilities. Such services are configurable through deployment descriptors by using XML files (*eXtensible Markup Language*).

CIAO was developed on top of TAO (The ACE ORB) [Schmidt et al., 1998], a middleware that provides an ORB (*Object Request Broker*) coded in the C++ language, and is compliant with the Corba Real-time specification. The internal structure of TAO is based on ACE (*Adaptative Communication Environment*), an object oriented framework that applies design patterns for communication with distribution and concurrency features [Schmidt and Huston, 2003]. The services provided by TAO (e.g. scheduling and real-time event) were designed using the patterns available in ACE, aiming at optimizing the efficiency and predictability of the real-time ORB [Schmidt and Cleeland, 2001].

Therefore, CIAO supports the development of predictable, portable and interoperable applications. Figure 1(a) presents a general view of CIAO and Figure 1(b) illustrates the real-time container. According to the terminology adopted by CIAO, synchronous communication ports are called facets and receptacles. For asynchronous communication, event channels are called producers and consumers ports. For more details on CIAO the reader can refer to other references [Wang et al., 2004b].



(a) Architecture



(b) Server of components

Figure 1: Real-time middleware based on components

3.3 ARCOS

ARCOS (*Architecture for Control and Supervision*) is a component-based framework devoted to the construction of distributed industrial applications to perform tasks of acquisition, control and supervision. ARCOS is developed atop CIAO and implements DAIS (*Data Acquisition from Industrial Systems*), a standard CORBA specification created by OMG (*Object Management Group*) for the acquisition of industrial data. Such specification supports the simultaneously transfer of large amounts of data to multiple clients, usually called consumer applications. Because ARCOS is based on CIAO, it is also compliant with the CCM specification.

In order to allow for a standard representation of common operations of industrial control and supervision applications, some extensions to DAIS were introduced in ARCOS. Such extensions make it possible, for instance, to specialize the system for the acquisition in specific devices, considering different techniques of control (e.g. a PID controller - *Proportional-Integral-Derivative*).

Internally, ARCOS uses TAO's real-time event service as an integrating mechanism for communication between ARCOS' components. The DAIS clients can use the *DAISFacade* class available on ARCOS to create new data acquisition groups interested in specific services. Thus, ARCOS provides an appropriate environment for the development of new distributed applications of control and supervision, where most core ARCOS' components are re-used for each new application deployed. Figure 2 illustrates the main software layers upon which ARCOS was designed.

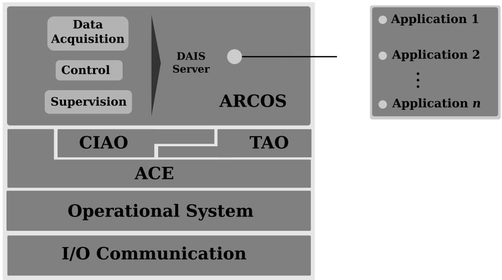


Figure 2: ARCOS *framework*

4 Estimating Component Execution Time Probability Distributions

The proposed method is based on measurements, where a *monitor* component is responsible for measuring the response time for executing a service provided by another component. It also measures the response time to call a null-code service, which is called round-trip time. Thus, response time (R) and round-trip time (RT) are defined as the variables of interest. It is important to emphasize that no knowledge about the application code is being used. Application code is seen as black-box entities. The model of measurement is illustrated in Figure 3. The monitor follows rules defined by designers so that it is able to measure different states of the component execution properly, considering different inputs over a range of values of interest.

The main goal of the proposed approach is to estimate component execution time probability distributions, where the execution time of a component service can be seen as the variable $C = R - RT$. To do so, the monitor component is designed to measure several values of R and RT. After describing the proposed method in the next section, its use is illustrated with a case study.

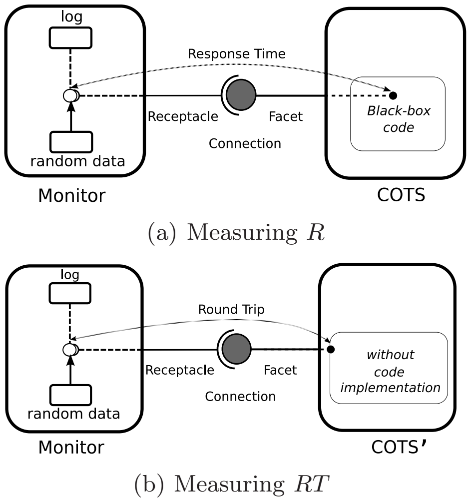


Figure 3: Model of measurement applied

4.1 Estimation Method

Let S_r and S_{rt} denote two collections of measured values that contain response times and round-trip times obtained by the monitor, respectively. The elements of a collection are not necessarily distinct. The set of distinct values in a collection S is given by $D(S)$. The number of times a value v is observed in a collection S is denoted by $\eta(v, S)$. Also, define the function $f(v, S)$ that gives the relative frequency of value v in a collection S. For example, if the monitor measured the same value $r \in S_r$ three times in 10 measurements, $f(r, S_r) = 0.3$. More precisely,

$$f(v, S) = \frac{\eta(v, S)}{\sum_{\forall u \in D(S)} \eta(u, S)} \tag{1}$$

It is possible to derive the probability distribution of C by computing the joint probability distribution

$$P(C = c) = \sum P(R = r, RT = rt) \qquad \forall r \in S_r, rt \in S_{rt} : c = r - rt \tag{2}$$

Nevertheless, not all combinations of values in $r \in S_r$ and $rt \in S_{rt}$ are possible since it is known that $P(C \leq 0) = 0$. More generally, assuming that there must exist a lower bound $c_{min} > 0$ on the execution time, it is necessary to consider those combinations that satisfy $r - rt \geq c_{min}$. It is important to point out that non-valid values of $r - rt$ may appear because the measurements in S_r and S_{rt} are carried out independently. Thus, let us define a collection of possible values for C as

$$S_c = \{r - rt | r - rt \geq c_{min}, r \in S_r, rt \in S_{rt}\} \tag{3}$$

Assuming $c_{min} \approx 0$ is simple but non-realistic. In order to provide better estimations when deriving the probability distribution of C, it is convenient to use better estimations for c_{min}. Since larger values of execution times are of more interest, the following procedure can be used to estimate c_{min}:

1. Find the minimum value $rt_u \in S_{rt}$ such that the probability $P(RT \leq rt_u) \geq p$, where p, $0 < p < 1$, is a threshold on the desired probability given by the user and $P(RT \leq rt_u)$ can be computed as

$$P(RT \leq rt_u) = \sum_{\forall rt \in D(S_{rt}):rt \leq rt_u} f(rt, S_{rt}) \tag{4}$$

2. Find the minimum value $r_{min} \in S_r$ such that $r_{min} - rt_u > 0$.

3. Define $c_{min} = r_{min} - rt_u$.

Once c_{min} is found, collection S_c and function (1) produce the desired distribution of C. In order to illustrate the proposed estimation approach, let $S_r = \{1, 2, 3, 6, 6, 7\}$ and $S_{rt} = \{1, 2, 3, 3, 3, 4\}$. Also, consider the desired threshold $p = 0.8$. In this case, $rt_u = 3$ since $P(RT \leq 2) = 1/3$ and $P(RT \leq 3) = 5/6 \approx 0.83$. Thus, $r_{min} = 6$ and so the values of C cannot be less than $c_{min} = 3$. Therefore, $S_c = \{3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 6\}$.

It is important to emphasize that the above definition of rt_u is used to derive the lower bound c_{min}. Discarding values of C below this bound is useful since it improves the quality of the estimation and does not compromise the estimation procedure. Indeed, the discarded values are too low and can be only observed for too high and very unlikely round-trip time values. Since it is of more interest to better estimate higher values of C, this discarding strategy is also a safe approach. Preliminary tests indicated that to get a good estimate for c_{min}, the value of p should be considered as close as possible to one. Lower values of p may result in discarding too many measurements. Good values of p depends on the application characteristics, though. The next section shows a case study that illustrates the effectiveness of the proposed estimation method.

4.2 Case Study: Applying the Methodology to a Component

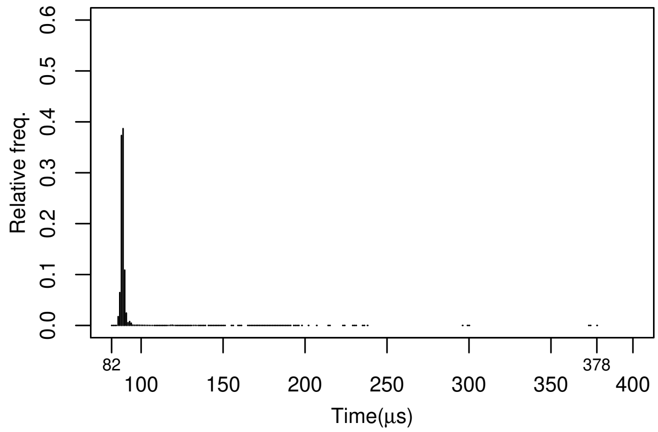
The proposed method was applied in an experiment executed on CIAO, a middleware described in section 3.2.

We implemented both the monitored and the monitor components, hosting them in the same container. The monitored component provides a service that simulates a kind of data analyzer, which keeps sampled data records and provides predictions on the analyzed data behavior. Since the semantics of this component service is not relevant here, it will not be further described. To obtain round-trip times, one third component was hosted latter in the same container with no internal code implementation (*null-service*). In order to minimize the interference in the measured times of the monitored component, its execution priority was set to the maximum value.

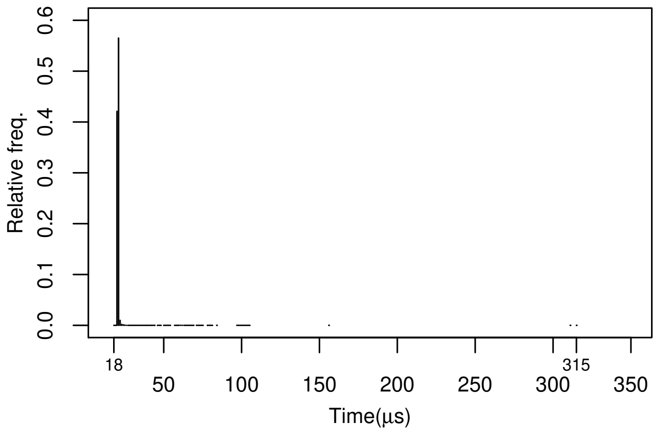
The default setting of CIAO was used as a base during the installation procedure, due to the fact that it already has features configured to guarantee optimization and efficiency of the platform, providing an environment suitable for many real-time applications. However, given the purpose of measuring in the experiment, some ORB parameters have been specifically configured with XML file descriptors: FIFO scheduling policy (*first-in-first-out*), direct mapping of CORBA priorities to the native priorities, priority policy defined by server and use of static tasks for the execution of the components installed in the container. These parameters mean that the interference in measurements will be limited, because these settings reduce the overhead. The number of concurrent tasks and their priorities were fixed during the experiment. Enough resources on the server components (e.g. prior allocation of memory) were also guaranteed.

CIAO 0.6.0 was compiled with GCC 4.2.1 and installed on Linux operating system with kernel with 2.6.20-16 and running on Intel Core Duo to 1.83GHz, 1GB of RAM and 2MB of cache memory. The operating system was configured with the parameters `maxcpus` = 1 in order to consider only one processor core, and `runlevel` = 1, restricting the operating system to its basic services. The definitions of services available and consumed by the components of the experiment were made through the language IDL (*Interface Definition Language*), which allows for the specification and subsequent generation of the structure of components in the CIAO architecture. The procedure for the installation of components followed a deployment plan defined for the components used in the experiment. This plan was also based on XML descriptors.

To carry out the necessary measurements, the monitor performed n calls through the connections with the analyzer and the *null-service*. For each completed call, the times measured were recorded. Both services use an identical port (named facet in the CIAO terminology) and the calling parameters were randomly generated by the monitor based on a range of values of interest. The measurements performed by the monitor were not correlated whatsoever. In our experiments, first the values of variable R were measured and then those of RT. The calling frequencies used in those measurements were set to 25Hz, 50Hz, 75Hz and 100Hz so that there were $n/4$ measurements for each calling frequency. In order to measure these values, we used the class *High Resolution Timer* provided by the framework ACE [Schmidt and Huston, 2003], on top of which CIAO was implemented. Thus, time was measured in microseconds (μs). The number n of measurements should be defined by the user and must be large enough so that the collected values of R and RT are representative. In our experiments, $n = 2 \times 10^6$.



(a) Response Time



(b) Round-Trip

Figure 4: Probability distributions of R and RT.

Figure 4 plots the probability distributions derived from the collected values in S_r and S_{rt}. As can be seen, these distributions exhibit similar behaviors although the dispersion of S_r is higher. This is due to the fact that they are subject to higher interference when compared to S_{rt}. Given that we did not use a real-time operating system, the observed dispersion is not so high and can be considered as expected [Liu, 2000], [Regnier et al., 2008].

Given the collections S_r and S_{rt}, the proposed estimation method was carried out. Table 1 shows some descriptive statistics for the obtained results. In order to evaluate the proposed method, we also measured the execution time of the

Table 1: Descriptive Statistics.

Summary Statistics	Measured Data R	RT	C'	Estimated Data C
Mode	89	21	66	68
Median	89	21	66	68
Mean	88.8	20.62	65.87	68.20
Std. Dev.	2.317528	1.143541	1.699584	2.381033
Max.	378	315	352	360
Min.	82	18	61	60
Range	296	297	291	300

code of the monitored service - *analyzer*. Thus, it was necessary to instrument the monitored service internal code. The measured execution times are represented by variable C' in the table. Also, it can be noted that there is a higher standard deviation for R when compared with RT. This is caused by the already mentioned higher dispersion of S_r as compared to S_{rt}.

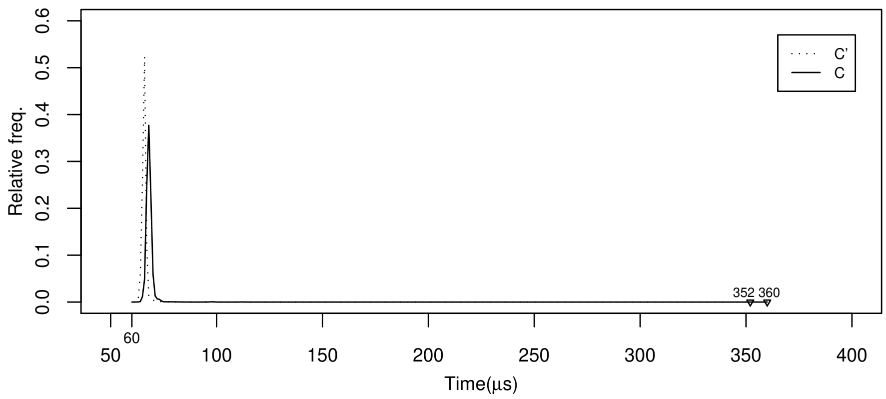


Figure 5: Probability distribution of C and C'

For the estimative of C_{min} was adopted $p = 99.5\%$, and the lower value of rt_u that answered this requirement was $22\mu s$ with probability 99.59%. From that value the distributions of C were calculated. Figures 5 and 6 illustrate better the effectiveness of the proposed approach. As can be seen, the probability distributions of C and C' and their respective mass functions are very similar. We also carried out the Kolmogorov-Smirnov test [Conover, 1971], finding that

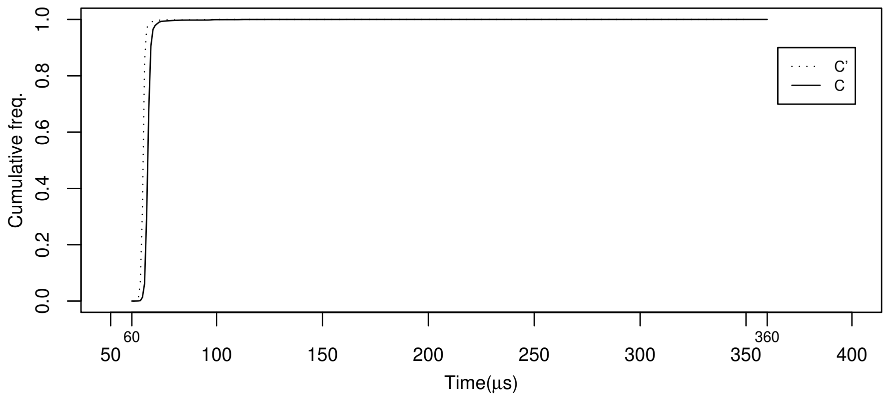


Figure 6: Cumulative distribution of variables C and C'.

both distributions can be considered the same with level of significance of 5%.

As can be seen in Figure 5, the distribution of C exhibits a right-shifted curve when compared with C' distribution. It is due to the influence caused by the joint probability computation when the Equation (2) was applied - i.e. $P(r = 89, rt = 21) = 21.86\%$. However, this right-shift does not compromise the results since the estimated probabilities in C distribution are related to time values that are higher than time values related to the curve of probabilities in C' distribution. So the approach is safe for the adoption of the estimated value. The same occurs with the cumulative distribution of C in Figure 6, that exhibits a right-shift (e.g. $P(C \leq 100) = 99.93\%$ and $P(C' \leq 100) = 99.99\%$).

Finally, it is should be noticed that the estimated probability distribution provides a simple way of estimating probabilistic WCET, for instance, $360\mu s$ with probabilistic guarantee $P(C < 360) = 99.999\%$. Likewise, the probability distribution produced can be used in probabilistic models like in [Kim et al., 2005] or in [Manolache et al., 2001] where the ratio of missed deadlines are computed per task, and the analysis uses a pseudo-continuous distribution based on probabilistic density curves.

5 Estimating Component Compositions Execution Time Probability Distributions

We have extended the previous estimation method to cope with probability distributions of component compositions. We observe that instead of estimating such compositions by convolving the probability distributions related to the execution time of isolated components, we take a simpler approach. Indeed, using such convolutions yields greater complexity when considering COTS - compiled version. This is because there may be dependencies between components. Observe also that if a measured component activates another component, this latter component has to be immediately measured after its activation and such an evaluation should also consider the actual state of the computer architecture. In general, the time analysis must consider the state of execution environment related to all successive activations, usually called *execution history.* As will be seen, our approach is more attractive and appropriate to COTS since component dependence is not directly considered. Instead, the whole composition is treated as a black-box.

We believe that the proposed black-box method ensures greater portability compared with other methods that tend to be complex and dependent on the hardware architecture used. However, we need a strategy that makes the adoption of the statistical method in the context of component compositions feasible. First, some definitions are introduced.

– Consider a chain of j components, where the first and last components in the chain is represented by $cots_1$ and $cots_j$, respectively. Also, consider that when the *monitor* calls a service of $cots_1$, these j components are activated in sequence (see Figure 7(a)). Hence, the time measured by the *monitor* accounts for the total response time that includes the activation chain, defined as R_{total}. In this sense, all components activated by the *monitor* can be seen as a unique entity. More precisely,

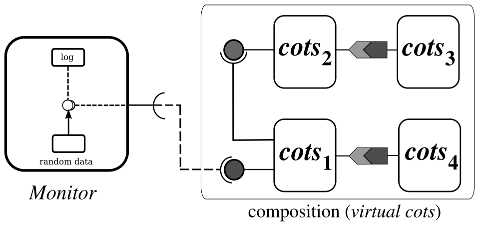
$$R_{total} = R_{cots_1} + R_{cots_2} + R_{cots_3} + \cdots + R_{cots_j} \tag{5}$$

– Similarly, to evaluate the *round-trip* of a composition, the whole composition should be treated as a black-box. The composition should be created and installed in the *container* so that a chain of null services can be measured. Thus,

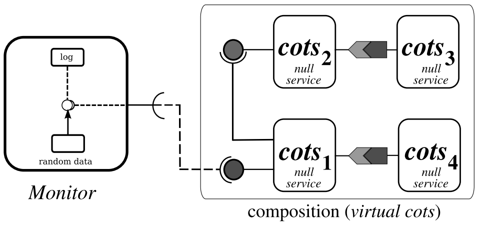
$$RT_{total} = RT_{cots_1} + RT_{cots_2} + RT_{cots_3} + \cdots + RT_{cots_j} \tag{6}$$

– We will use the variables R and RT as before. However, their values represent now the response and *round-trip* times for compositions, respectively.

Given the new definitions for R and RT, the equation for C is the same as presented in Section 4, so that the values of S_c can be obtained. $S_{c_{total}}$ now denotes the value of S_c for a composition. The execution time probability distribution of $S_{c_{total}}$ refers to a measured composition (C_{total}), following the rules of joint probability defined by Equation (2). Given that a composition is executed for a given system configuration, it is necessary that such configuration be the same for both the measurements of the R_{total} and RT_{total} variables to avoid any erros in the measured values. It should be observed that distinct probability distribution curves of C_{total} can be obtained from distinct system configurations and by



(a) Measurement of R_{total}



(b) Measurement of RT_{total}

Figure 7: Model of measurement for a composition of components.

comparing these curves a designer can evaluate the impact of the adoption of components from different suppliers.

5.1 Case Study: Applying the Methodology to a Component Composition

In order to evaluate the proposed method we applied it to a component composition developed on top of ARCOS: a simulated car cruise control application [Pont, 2001].

As the car cruise control is simulated, the values for the acceleration, distance, and speed are obtained from a mathematical model given by the Equations (7), (8), and (9), respectively.

$$Accel = \frac{Trothle \times ENGINE\_POWER - (FRIC \times Old\_speed)}{MASS} \tag{7}$$

$$Dist = Old\_speed + Accel \times \left(\frac{1}{SAMPLE\_RATE}\right) \tag{8}$$

$$Speed = \sqrt{Old\_speed^2 + (2 \times Accel \times Dist)} \tag{9}$$

This car cruise control application is made of two modules: a data acquisition module and a control module.

The data acquisition module includes the following components: a *DAISSimulatedCarProvider* component that represents the car being monitored, providing the speed data and implementing the *IDAISProviderBaseFacet* and *ISimulatedCarProviderFacet* interfaces; a *DAISDAGroupManager* component that manages all operations for a data acquisition group; a *DAISDAGroupClock* component that is responsible for generating events according to the frequencies previously defined for the execution of acquisition and control tasks; a *DAISServer* component that is responsible for creating connection points and data access sessions.

The control module is made of the following components: the *PIDController* component that represents a PID controler and implements the *IControllerBaseFacet* and *IPIDControlerFacet* interfaces; the *ControlManager* component responsible for managing the control cycle; the *ControlManagerDAISCallback* component that includes the *callback* object required by the DAIS implementation for the periodic transmission of data.

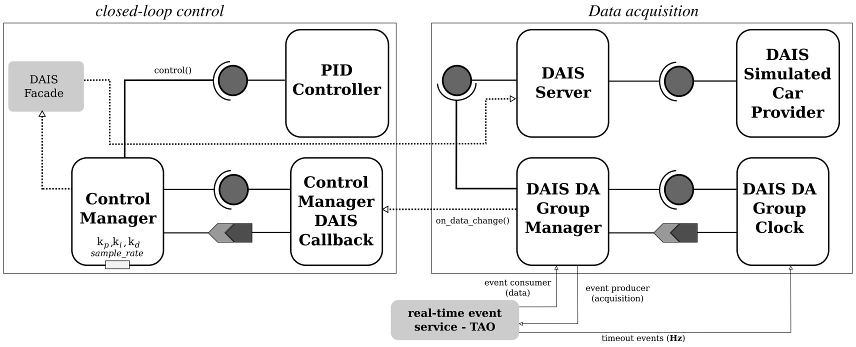


Figure 8: Speed control system for a vehicle.

Figure 8 gives a general view of the components and their inter-connections. Basically, the application works as follows. Periodically, the *DAISDAGroupClock* component generates a data acquisition event that is consumed by the *DAISDAGroupManager* component. The *DAISDAGroupManager* component in turn obtains the data from the *DAISSimulatedCarProvider* component and passes them on to the *ControlManager* component through the *ControlManagerDAISCallback* component - *on_data_change*() method. At this point the control loop begins managed by the *ControlManager* component that passes on the acquired data - method *control*(). The *PIDController* component performs the speed control task. Its goal is to maintain the vehicle speed close to the set-point, as illustrated in Figure 9. The controller tunning is done by the attributes *setpoint*, k_p, k_i e k_d, and the control cycle frequency is adjusted by the attribute *sample_rate*.

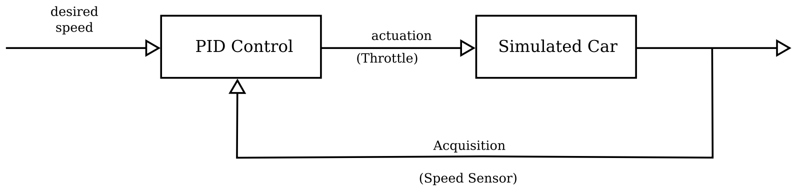


Figure 9: Feedback Control Loop

5.2 Experiment Configuration and Assessment

The speed control application, described in the former section, was originally developed in ARCOS for an older version of CIAO. Such a version did not provide some real-time facilities such as the real-time container. Hence, we decided to adapt the application code to the CIAO version 0.6.5 by slightly modifying the XML descriptor file used in the component composition. It was included in this file specific markers to indicate which resources would be reserved to which component instance. These markers were defined according to the resource utilization policies pre-defined for each application component. Hence, as described in section 4.2, some ORB parameters were specifically configured: FIFO scheduling policy; direct mapping from CORBA priorities to native ones; priority policies defined by the server; and static tasks for the execution of components installed in the *container*.

The *DAISDAGroupClock* component works as a periodic event generator. The event periods are defined during the DAIS client group creation procedure. Since this component has one of the functions carried out by the `monitor` component (recall Figure 3), an adaptation was carried out in the code of *DAISDAGroupClock* enabling it to measure and register event occurrences generated by the TAO real-time event service. This adaptation made it unnecessary to connect a *monitor* component to the composition.

The application was configured considering three set-point values for the controller, namely 30, 60 and 90 km/h. The values of $k_p = 0.05$, $k_i = 0$ and $k_d = 0$ were used for computing the accelerator actuation behavior. It is important to emphasize that several different combinations of speed values, control frequency, and k_p, k_i and k_d are possible.

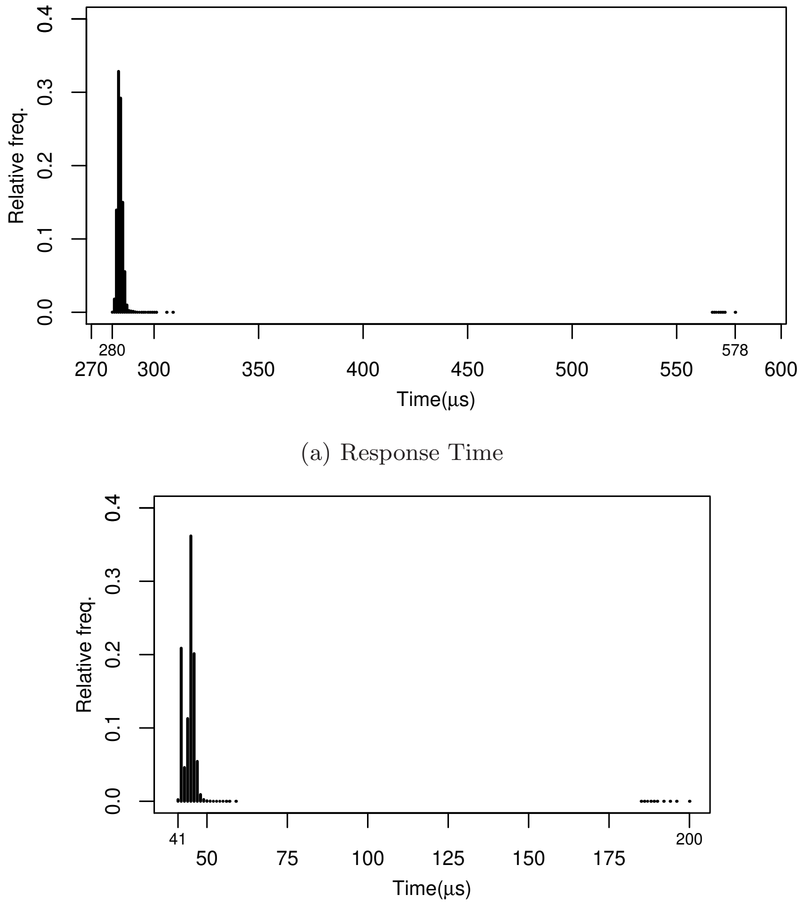
The set of evaluation experiments was carried out making use of Linux kernel version 2.6.24-19. In order to obtain better temporal predictability, the 2.6.24.19-rt preemptive patch was applied. This operating system environment gives reasonably good real-time behavior [Regnier et al., 2008] since it handles aspects related to interrupt and activation latencies, data sharing (e.g. *rt_mutex*) etc. CIAO 0.6.5 and ARCOS were compiled by GCC 4.2.3 and these middleware platforms were installed on a 1.67GHz Intel(R) Core(TM) 2 Duo, with 2GB

RAM and 2MB cache. The operating system was configured with `maxcpus` $= 1$ so that only one core was used. Also, only basic operating system services were installed by option `runlevel` $= 1$. The definitions of the published and consumed services and the component category used were specified through IDL. The container component installation procedure was carried out making use of DAnCE. The priority of each component was set to the maximum allowed by the FIFO Linux scheduling policy.

The measurement procedure was initialized for each timeout event consumed by *DAISDAGroupClock*. An execution path was activated to each acquisition-control cycle. The measurements related to each cycle were carrried out and registered in a log file. Initially, the values of R_{total} were collected. This corresponds to the configuration illustrated in Figure 8. Then, similar measurements were carried out but with the null-service configuration. The collected values correspond to the RT_{total} variable, which represents the total *round-trip* time for the whole activated execution path. In both measurement phases each component was executed by a dedicated task.

The measured values of R_{total} and RT_{total} were not related since they were observed independently. Also, these measurements were carried out taking four frequency values into consideration: 25Hz, 50Hz, 75Hz e 100Hz. These frequencies were specified in the group configuration *ControlAquisition*, created in the *DAISDAGroupManager* component. We carried out 120000 measurements for three set-point values, that is, 30000 measurements were carried out for each considered frequency value. Again, we used the *High Resolution Timer* class provided by the *framework* ACE so that microsecond precision can be obtained (μs). The interference generated by this measurement mechanism was estimated to be 35 nanoseconds (ns). Clearly, this is a small value. Also, considering that this interference is approximately constant, when computing $C_{total} = R_{total} - RT_{total}$, its effects on the estimation is canceled. Thus, we considered that the interference due to measurements can be neglected when computing C_{total}. However, the same may not be true for measuring C'_{total}, the actual application execution time, used for validating the proposed methodology. Indeed, $C'_{total} = \sum_{i=1}^{n} C'_{cots_i}$, where C'_{cots_i} represents the i-th component in the execution path to be measured. Hence, the interference due to measurements depends on n and may not be neglected since it is cumulative. This problem was dealt with by subtracting the accumulated interference value from the measurements. For the performed experiments, this accumulated interference was estimated to be approximately $385ns$.

Let $S_{r_{total}}$ and $S_{rt_{total}}$ be collections containing the measured values of R_{total} and RT_{total}, respectively. Figure 10 shows the respective frequency distributions derived from these values. Although these distributions preserve some similarities, it can be seen that $S_{r_{total}}$ exhibits higher dispersion. This is due to the fact



(b) Round-Trip

Figure 10: Probability distribution of R_{total} and RT_{total}.

that the values of $S_{r_{total}}$ belong to components whose internal code present a much higher number of execution states. Recall that RT_{total} are implemented with null-service code. Also, the actual application components are subject to more interference from the execution infrastructure since their execution time are higher. Finally, some dispersion is expected since the operating system does not provide hard real-time guarantees.

Once the collections $S_{r_{total}}$ and $S_{rt_{total}}$ were obtained by measurements, the proposed methodology was carried out as described in section 5 so that $S_{c_{total}}$ could be estimated. As previously mentioned, the actual execution times were measured so as to determine C'_{total}. This measurement was made by instrument-

Table 2: Descriptive Statistics.

	Measured Data			Estimated Data
Summary Statistics	R_{total}	RT_{total}	C'_{total}	C_{total}
Mode	283	45	233	238
Median	284	46	233	239
Mean	283.7	44.53	233.3	239.1766
Std. Dev.	3.131618	2.163726	2.869205	3.515307
Max.	578	200	528	537
Min.	280	41	228	230
Range	298	159	300	307

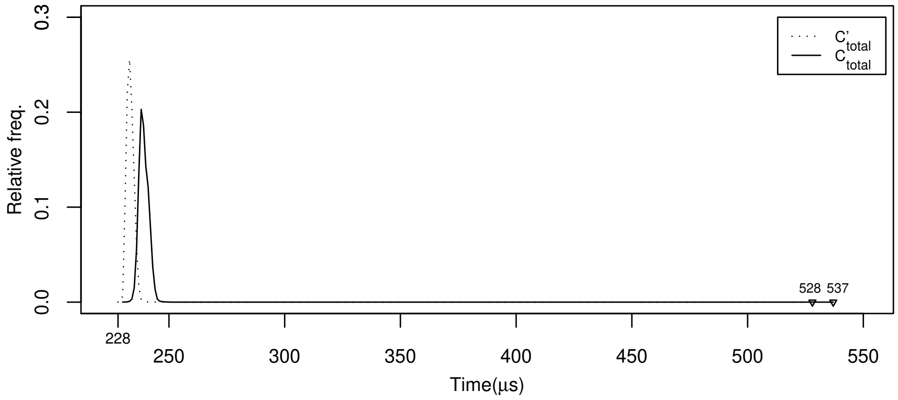


Figure 11: Probability distribution of C_{total} and C'_{total}

ing the corresponding component code. The descriptive analysis of the obtained data can be seen in Table 2. As can be noted, the standard deviation for C_{total} is the highest one. This reflects the higher dispersion mentioned before. This effect causes a shift to the right in all values computed for C_{total} when compared to the respective values of C'_{total}.

The value of c_{min} was estimated taking $p = 99.90\%$ and the lowest value of rt_u corresponding to this value of p was found to be $50\mu s$ with probability 99.93%. Then the distribution of C_{total} was determined and plotted in Figures 11 and 12. For the sake of comparison, these graphs also show the distribution of C'_{total}. Again, its possible to observe similarities between C'_{total} and C_{total}. Similarly to what was done for a single component, we carried out the Kolmogorov-Smirnov test for the data obtained in this set of measumenets [Conover, 1971]. Again, the test showed that both C'_{total} and C_{total} can be considered as following

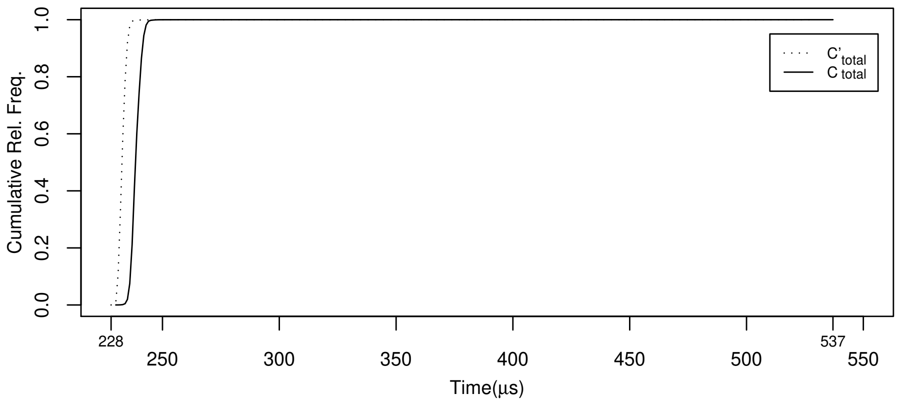


Figure 12: Cumulative Distribution of C_{total} and C'_{total}

the same distribution with significance level of 5%.

In Figure 11 we can see a peak representing the relative frequency of mode equal to 0.2557 while the corresponding estimated frequency for the mode was 0.2029. The mode time values for both variables C'_{total} and C_{total} presented a small difference equal to $5\mu s$. This is because the already mentioned higher dispersion observed in the data, which in turn is taken into consideration when computing the joint probability distribution. More precisely, Equation (2) - e.g. $P(r = 283, rt = 45) = 11.88\%$. It is important to emphasize that this is an interesting characteristic of joint probability computation: The combination of higher probable data values of R and RT tends to give good and safe approximations with respect to the actual values. As a result, we can observe that the relative frequency of C_{total} is right-shiftted when compared to C'_{total}. This makes the proposed approach safe since the given estimation is not lower than the corresponding actual value. Figure 12 shows the cumulative distributions for C'_{total} and C_{total}. As can be noticed, the same right-shifft effect can be observed. This information may be useful for the designers since they can compute a upper bound on the execution time with probabilistic gurantees so that the requirements for the considered applications can be taken into consideration. For example, $P\left(C_{total} \leq 531\right) = 99.999\%$ and $P\left(C'_{total} \leq 526\right) = 99.999\%$.

6 Conclusions and Future Work

This paper presented and evaluated a new methodology for estimating execution time probability distributions of COTS-based real-time systems. Nowadays, such probabilistic approaches are being considered very relevant in the design of modern real-time systems, where the traditional methodologies can not be applied or face difficulties, due to the lack of access to the internal component code. The benefits generated by this work allow the designer to evaluate the time properties of black-box COTS-based systems at early development stages. As a result, different COTS can be analyzed into different configuration of compositions to compare and evaluate the time properties.

Though the proposed methodology is indented to black-box components, it can also be used in some white-box scenarios (where internal knowledge of software components is available), if the application of conventional methodologies - such as static analysis - is not cost-effective. Also, the statistical method we proposed can be applied to others component technologies as long as such technologies include usual real-time facilities as those available in CIAO. However, some improvements need to be done to face specific issues in component compositions. For instance, conditional component activation along execution paths triggered by a service need to be considered. This is a challenge when we are dealing with black-box COTS and techniques based on measurements. To evaluate the proposed methodology, two case studies were fully implemented over the component-based middleware CIAO and the framework ARCOS. In both case studies, component execution time probability distributions were estimated from interactions via the component interfaces and from instrumented code inside the components, bypassing the component interfaces. The comparison of these distributions considering the summary statistics showed that the proposed methodology produces results very close to the real values. This indicates that the statistical method applied can be considered a good approach for estimating execution time probability distributions in such context.

Future work also includes complementing the proposed methodology with mechanisms to derive probabilistic worst-case execution times of components and related probabilistic scheduling feasibility tests.

Acknowledgments

The authors would like to thank the Brazilian funding agencies CAPES (PROCAD/AST), CNPq and FAPESB for their financial support.

References

[Andrade and Macêdo, 2007] Andrade, S. and Macêdo, R. (2007). Engineering components for flexible and interoperable real-time distributed supervision and control systems. In *12th IEEE Conference on Emerging Technologies and Factory Automation*, Patras - Greece.

[Andrade and Macêdo, 2005] Andrade, S. S. and Macêdo, R. (2005). A component-based real-time architecture for distributed supervision and control applications. In *Proc. of the 10TH International Conference on Emergencing Techonologies and Factory Automation*, volume 1, pages 15–22, Catania, Italy. IEEE Computer Society Press.

[Ballabriga et al., 2007] Ballabriga, C., Cassé, H., and Sainrat, P. (2007). Wcet computation on software components by partial static analysis. Junior Researcher Workshop on Real-Time Computing. IRIT - Université de Toulouse, France.

[Bernat et al., 2002] Bernat, G., Colin, A., and Petters, S. M. (2002). Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium*, pages 279– 288, Washington. IEEE Computer Society.

[Conover, 1971] Conover, W. J. (1971). *Practical nonparametric statistics*. John Wiley & Sons, New York.

[Edgar and Burns, 2001] Edgar, S. and Burns, A. (2001). Statistical analysis of wcet for scheduling. In *22nd IEEE Real-Time Systems Symposium*, page 215, Washington, DC, USA. IEEE Computer Society.

[Estévez-Ayres et al., 2005] Estévez-Ayres, I., García-Valls, M., and Basanta-Val, P. (2005). Enabling wcet-based composition of service-based real-time applications. *SIGBED Review*, 2:25–29.

[Fredriksson, 2006] Fredriksson, J. (2006). Increasing accuracy of property predictions for embedded real-time components. In *18th Euromicro Conference on Real-Time Systems*.

[Kim et al., 2005] Kim, K., Diaz, J. L., Lopez, J. M., Bello, L. L., Lee, C.-G., and Min, S. L. (2005). An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. Comput.*, 54(11):1460–1466.

[Li et al., 2008] Li, J., Conradi, R., Slyngstad, O. P. N., Torchiano, M., Morisio, M., and Bunse, C. (2008). A state-of-the-practice survey of risk management in development with off-the-shelf software components. *IEEE Transactions on Software Engineering*, 34(2):271–286.

[Liu, 2000] Liu, J. W. S. (2000). *Real-Time Systems*. Prentice Hall, 1 edition.

[Manolache et al., 2001] Manolache, S., Eles, P., and Peng, Z. (2001). Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *13th Euromicro Conference on Real-Time Systems*, page 19, Washington, DC, USA. IEEE Computer Society.

[Möller et al., 2005] Möller, A., Peake, I., Nolin, M., Fredriksson, J., and Schmidt, H. (2005). Component-based context-dependent hybrid property prediction. In *Workshop on Dependable Software Intensive Embedded systems*.

[Moss and Muller, 2004] Moss, A. and Muller, H. (2004). Model generation for temporal properties of reactive components. In *1st International Workshop on Software Analysis and Development for Pervasive Systems*, pages 12–19.

[Nolte et al., 2003] Nolte, T., Möller, A., and Nolin, M. (2003). Using components to facilitate stochastic schedulability analysis. In *24th IEEE Real-Time System Symposium - WiP Session*, Cancun, Mexico.

[OMG, 2006] OMG (2006). *CORBA Component Model Specification*.

[Pont, 2001] Pont, M. J. (2001). *Patterns for Time-Triggered Embedded Systems*. Addison-Wesley Professional, 1st edition. ISBN: 0-201-33138-1.

[Regnier et al., 2008] Regnier, P., Lima, G., and Barreto, L. (2008). "Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems". *Operating Systems Review (to apear)*, pages 1–12.

[Schmidt and Cleeland, 2001] Schmidt, D. C. and Cleeland, C. (2001). Applying a pattern language to develop extensible orb middleware. pages 393–438.

[Schmidt and Huston, 2003] Schmidt, D. C. and Huston, S. D. (2003). *C++ Network Programming: Systematic Reuse with ACE and Frameworks*. Addison-Wesley Longman.

[Schmidt et al., 1998] Schmidt, D. C., Levine, D. L., and Mungee, S. (1998). The design of the tao real-time object request broker. *Computer Communications*, 21(4):294–324.

[Wang et al., 2004a] Wang, N., Gill, C., Subramonian, V., and Schmidt, D. C. (2004a). Configuring real-time aspects in component middleware. pages 1520–1537.

[Wang et al., 2004b] Wang, N., Gill, C. D., Schmidt, D. C., Gokhale, A., Natarajan, B., Loyall, J. P., Schantz, R. E., and Rodrigues, C. (2004b). *Middleware for communications*, chapter QoS-enabled Middleware, pages 131–159. John Wiley & Sons.