Workshop on Quality of Service for Application Servers - WQSAS 2004

Scheduling Approaches for Component-Based Real-Time Distributed Applications

Cássia Yuri Tatibana , Rômulo Silva de Oliveira , Carlos Montez

DAS - Departamento de Automação e Sistemas UFSC - Universidade Federal de Santa Catarina
Campus Universitário, Caixa Postal 476 - CEP 88040-900 - Florianópolis - SC -Brazil

{cytatiba,romulo,montez}@das.ufsc.br

Abstract. *Real-time systems grow in complexity and applicability each day. The development of these systems through component-based approaches has proved to be a promising alternative. This paper aims a qualitative comparison between the proposals found in the literature, placing them in the real-time context, and presenting some gaps to be explored by the research community. We also propose the adoption of dynamic guarantee in real-time component-based systems through the implementation of an acceptance test in component containers.*

1. Introduction

Historically, real-time systems developers have avoided indirections concerning implementation aspects. This approach had been used to avoid the lack of predictability inherent in the use of too many software layers between real-time applications and required physical resources. Although it assures a certain degree of reliability concerning applications timing behavior, the increase in utilization and complexity of these systems raises the necessity to adopt methods, tools and technologies to improve productivity, management facilities and quality.

These issues justify the number of works involving components and real-time systems in the literature [6, 20]. The diversity of real-time applications leads to the development of proposals towards the solution of specific problems. Also, component-based development includes a set of procedures with great potential for investigation focusing real-time systems development. In this sense, a real-time component model may be elaborated based on many different aspects and premises. In this paper we present important proposals about components and real-time and point out gaps that raise research possibilities towards dynamic guarantees in component-based real-time systems. We also propose the adoption of an acceptance test to be implemented by the container in order to provide dynamic guarantee for these systems.

The rest of this paper is organized in the following manner: Section 2 describes real-time systems. Section 3 presents the relevant issues when dealing with real-time systems and components. In the next section, two important propositions in the literature are described. Section 5 describes our proposal, the assumptions and related issues. Finally, our conclusions are presented in section 6.

2. Real-Time Systems

General purpose systems concentrate efforts in the quality of their results. Although fast executions are desirable, the approach is always ”do the work using the necessary time”. Real-time systems have a different approach because the time is limited. They must assure that it will be possible to satisfy the deadlines imposed by the system environment. So, the issue is ”do the work using the available time” [11].

In the literature different proposals about how real-time systems are scheduled are identified [11]. They may be classified in three groups according to the offered guarantee: static guarantee (pre-runtime) [18], dynamic guarantee [12] and best effort [8]. The static guarantee group is the one capable of offering deterministic predictability. It is employed in systems that need to assure at pre-runtime that all tasks will be executed before their respective deadlines. Obviously, this guarantee is based on a set of assumptions, including a determined workload and a fault hypothesis. The static guarantee implies in resource reservation for the worst case and resource subutilization. Another problem related to this approach is the need of a bounded and static workload.

In the best-effort approach there is no pre-runtime guarantee about meeting deadlines. The best-effort scheduling, at most, provides probabilistic predictability based on an estimated workload. The immediate consequence is the possibility of system overload. This situation is characterized when it is not possible to execute all tasks before their respective deadlines.

Finally, some proposals provide ”dynamic guarantee” by determining at runtime which deadlines will be satisfied. This approach group executes an acceptance test each

time a new task arrives in the system. The test is based on analysis that considers the worst case hypothesis only for the tasks to be guaranteed and all tasks already guaranteed. If the test indicates that it is not possible to meet the deadline of the new task, together with all previously accepted tasks, the new task is rejected. Therefore, this mechanism implements dynamic guarantee of newly arrived tasks while preserving the tasks previously guaranteed as schedulable.

3. Real-Time Component-Based Systems

In this section, some of the main aspects related to real-time component-based system development will be discussed.

Components and Containers

A real-time component model may be structured in two ways. The component may be complete, containing application logic and the necessary infrastructure for its correct functioning, including real-time aspects. This choice assures few indirections concerning the component implementation but requires specific tools as it will require the component code for configuration.

The other way to structure the component is by separating the functional part in the component, and the non-functional part in the container. While the component contains only business logic, the container implements all the functions needed to manage this component, including the mechanisms for real-time behavior. The separation of functional and non-functional aspects of an application between component and container is used in EJB (Enterprise Java Beans) [2] and CCM (CORBA Component Model) [3]. This implementation structure contributes to component reuse and facilitates its configuration process. All the real-time aspects to be configured are confined in the container.

Execution Environment

The execution environment of a proposed model may consider two hypothesis: open system and closed system.

In an open system clients are outside the system considered in the work. This system does not know the client, it only acknowledges clients requests at the moment they arrive at the system. Open systems work with unknown computational workload and usually adopt best effort policies.

In a closed system clients are part of the system. All the elements involved in the system execution are predicted and the communication may be planned with increased predictability. In these systems, modules or mechanisms can be deployed as a client part in order to achieve the configuration of end-to-end communication (between components and between client and server). Since the computational workload may be known a priori, it is possible to provide end-to-end predictability to the application.

The choice of a closed system implies the possibility of obtaining increased predictability from the generated application. The whole system can be designed, configured, deployed and integrated (establishing connections between server components and between clients and server) together. During configuration and implementation steps it is possible to choose an optimum allocation of components in containers (considering more than one component in each container) and of containers in the distributed system hosts.

It is important to notice that the end-to-end predictability concerns different issues according to the context environment. In open systems end-to-end predictability refers to timing behavior inside the server: from the moment the request is acknowledged by the server to the produced response. For closed systems by the other hand, end-to-end predictability means the joined behavior of clients and server: it includes the client timing behavior and all the server components timing behavior as well.

Communication Latency, Clock Synchronization and Worst Case Execution Time

Once the real-time system components may be hosted by different computers, the problem of predictability in a distributed environment must be considered. The time between messages sending and receiving must be known or estimated.

Clock synchronization may be applied to offer a common time reference to all components involved. This reference allow the computation of a message travel between components in different hosts, which improves timing behavior management in real-time systems.

The worst case execution time - WCET, is a component timing property dependent on all the underlying software and hardware layers. As component connections influences component execution and cannot be predicted, dealing with the WCET is a complex job.

4. Main Proposals in the Literature

Many real-time component proposals can be found in the literature. Each of them presents different objectives, approaches and mechanisms to insert time constraints in the component-based system lifecycle.

The study of the many approaches exploring aspects and lifecycle steps of real-time component-based devel-

opment brings about gaps to be fulfilled. The real-time component-based development is still a relatively young research area. Many of the works envolving the theme are concentrated on the early stages of application development. The execution model or how all the mechanisms provided by the component structure will be used at runtime are not presented yet. As some of these approaches are very dependent on tools, the development of a whole system according to them are labored and error prone.

Some interesting researches developed real-time components architectures aiming determinism, however most of them target embedded applications [16, 1, 5]. The works presented in [19, 15] focus a component framework and configuration approach intended for distributed environment, but do not explore the component model execution behavior or communication aspects.

Most of the approaches addressing embedded systems offer some sort of timing behavior guarantee since they are built for specific real-time target platforms. By the other hand, the approaches concerning distributed environment have to cope with different problems.

In the distributed real-time context and considering the previous discussion, two proposals are examined: Cadena [6] and CIAO/QuO [17](Figure 1). Many others can be found in the literature [10, 9, 4, 7], however, they are in accordance with the scenario illustrated in this section.

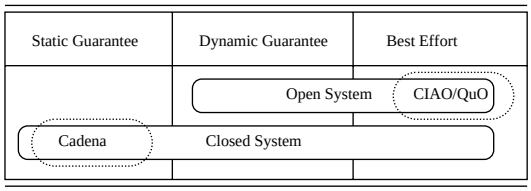

Figure 1: System Types and Scheduling Approaches Combination

Different from open systems, in closed systems any of the scheduling approaches may be implemented: static guarantee, dynamic guarantee or best effort. As they have pre-runtime workload information, they can assure the satisfaction of the tasks timing constraints. Cadena is an example of a tool used to model, build and verify systems with these characteristics.

Cadena is a development environment to model CCM systems [3]. It is employed in the construction of Boeing applications. This environment is composed by a set of tools to analyze timing behavior and works as a layer on top of OpenCCM [13]. Cadena adds forms to the IDLs of the applications being developed. Through this forms, time constraints are defined, dependence analysis between components are made and also scheduling information are provided. The application under development together with Cadena forms and correct functioning references (set by the developer) is translated into a formal analysis model.

As Cadena is developed by Boeing many assumptions are made, for example, all Cadena components are event-triggered. The communication is based on events produced according to frequencies specified by Cadena forms and associated to component ports. The Cadena analyzer checks if the time constraints imposed are feasible and through interactions with the developer, adjusts the timing constraints and the components allocation in the distributed environment.

The CIAO/QuO approach aims at the development of real-time component-based open systems. Although it allows the development of client components, it cannot predict or limit the number of clients connected to the server, neither prevent server overloads. CIAO/QuO is an attempt to provide services to applications implementing the best effort policy: CIAO (Component Integrated ACE ORB) [17] provides static QoS and QuO (Quality Objects) [21] provides dynamic QoS. This combination CIAO/QuO results in an environment capable of building components with QoS properties defined pre-runtime and adaptability during the application execution.

CIAO proposes the decoupling of reusable, multi-purpose, off-the-shelf, resource management aspects of the middleware from aspects that need customization and tailoring to the specific application preferences. It allows developers to select real-time policies and QoS aspects to be applied to the server and the client side. It includes capabilities to configure CPU policies, communication policies and distributed middleware end-to-end.

QuO allows the specification of QoS constraints, system monitoring and adaptability according to changes in system current state. All QuO facilities are enclosed in Qoskets, behavior units for reuse that can be deployed in CIAO components.

The CIAO/QuO approach separates application communication into functional paths and QoS systemic paths. Functional paths are flows of application specific for information between client and server. QoS systemic paths are responsible for determining how well the functional interactions behave between client and server with respect to QoS properties set by the developer.

The real-time handling provided by this approach focus on one client-server communication. Although it allows one to configure both sides of this communication, it can

at most assure a best effort behavior concerning the satisfaction of constraints imposed by the client.

It is important to notice the way real-time constraints are dealt with in both approaches. Cadena, despite of being a moddeling tool, provides a full application view. It envelopes server and clients in its development environment providing the application timing behavior. CIAO/QuO by the other hand, works in the communication aspect only. Although it is capable to configure client and server, it is not able to assure the complete application behavior.

As Cadena, CIAO and QuO approaches are developed upon existent technologies, what makes possible to explore runtime aspects towards real-time behavior in distributed environment during execution. Yet, these approaches are based on a general purpose component model. There is a separation between the approaches investigating the early steps of real-time component-based development and the ones addressing timing behavior at runtime. It is expected that the progress of researches will converge the many approaches towards complete real-time component-based development proposals.

5. Acceptance Test

In this section we propose a scheduling approach based on acceptance test for component-based real-time systems. It is assumed an open system, that is, clients are known only when their requests arrive at a the server side component. We chose to use containers in order achieve greater benefits from the reuse property of component based development.

Considering component-based development, after components design and implementation, they must be deployed and then executed. Connections between the server components may be established during system deployment (static bind) . These connections are predicted during system assembly, or even before, in earlier stages. Therefore, at pre-runtime the set of components that provide one service may have informations such as WCET, dependence or precedence relations between each other. Tools to verify dependence relationship among components are already available in Cadena, for instance.

However, only at runtime connections (dynamic bind) between clients and server takes place and clients may request the services provided by the server. For real-time purposes, these requests must also inform the server about the timing constraints this service must obey. This is how server components will base their execution in an attempt to satify all the clients requests and timing constraints.

Before the client become capable of requesting services, the connection between this client and the server component must explicity take place. At the bind time, the server has not yet accepted the client, so the client can provide all the timing information the server needs to judge its capacity to perform this client requests. At this moment, the server is able to consider the client timing constraints, its own timing properties (WCET, components methods precedence relation and dependence) and the server current state (resources availability and current workload). Therefore, the dynamic guarantee is made possible through the application of an acceptance test at bind time.

Real-time dynamic guarantee approach is based on an acceptance test to verify the schedulability of the set composed by a newly arrived task and the tasks previously accepted by the system. The acceptance tests are based on worst case hipothesis analysis considering timing parameters. This approach was developed for critical systems that operate in non deterministic environment.

An acceptance test [14] could be implemented by the server components containers, requiring no aditional functionality from the component. The acceptance test would be a new container responsability as all the others concerning real-time aspects. The server could accept or not the new client considering the guarantee of older (already accepted) clients constraints.

The parameters used by an acceptance test would be captured at two moments of the application lifecycle. Let's consider components $C1$, $C2$ and $C3$ from a given server and a service S provided by the sequence of methods $C1.m2$, $C2.m5$ and $C3.m1$. At the application server deployment time, its components are connected to each other. So, during the static bind of components $C1$, $C2$ and $C3$ information, as WCET about the methods they implement could be provided. Through the components ports connections, dependence relationships would be established. By the end of deployment step, all the server provided services would be mapped to component methods sequences of execution. So, from the arrival of a service request, the possible components and methods to be executed could be figured. These components would be the ones negotianting the acceptance of a newly arrived client request in the server.

At runtime, candidate new clients should submit their bind request together with their timing constraints to the server components acceptance test. Timing constraints examples would be: a deadline for each service request and a minimum time interval between service requests. If the S is the service required, components $C1$, $C2$ and $C3$ would be appointed to apply the test. Based on the client request timing constraints and their own workload upon methods $C1.m2$, $C2.m5$ and $C3.m1$, these components

should reach a result and allow or discard the client bind request. Once accepted, the server would guarantee to accepted clients timing constraints accomplisment.

Concerning the issues described in section 3, our proposal adopts containers and considers an open system. Also, in order to make possible an acceptance test implementation as proposed, communication latence and WCET informations, as well as clock sinchronization will be necessary.

The proposed approach addresses the gap emphasized in Figure 1, dynamic guarantee is not provided in any of the runtime proposals described. Different from Cadena, dynamic guarantee is intended to provide flexibility to the system, new clients may be accepted at runtime. Although this proposal does not address clients configuration, as happens in CIAO/QuO, it would provide better application timing behavior since it can assure at least the timing constraints of accepted clients instead of providing best effort only, for all clients.

6. Conclusion

Component-based development presents advantages desirable in real-time systems. However, conventional component models are incapable of providing essential features of real-time systems. Despite of the many component models that started to be developed, there is still the lack of models that focus real-time components runtime. Most of the proposals are concentrated in pre-runtime aspects: component architecture and initial building and configuration steps. The runtime behavior has not been well explored.

One research opportunity is the real-time component-based development including dynamic guarantee. The dynamic guarantees can be provided by applying an acceptance test at bind time, at the moment a client bind request is received by the server.

The container may be used to manage the acceptance test at runtime. This capability assigned to the container does not interfere with the common used component technology structure: the component contains only the code related to the application logic. The container manages all other aspects related to assure component functioning according to its configuration, including timing aspects.

The containers managing components which work together to provide one service must be able to decide (as a whole) if they are capable of providing the service according to time constraints imposed by the client request. The decision made by the set of containers considers (beyond the request constrains) the component WCET and the component and host workload. Each container must be capable of applying the acceptance test to the request and then to allow or not the new client connection.

The acceptance test of new clients depends on the component behavior required by the client and the server capacity. The negotiation among server components is necessary to guarantee service provision. The behavior specification demanded by a client bind request may be set through QoS contracts that would include timing parameters such as deadline and minimum time interval between service requests. This scheduling approach provides dynamic guarantee and it complements the other two approaches (offline guarantee and best effort) already proposed in the literature.

Dynamic guarantee in real-time component-based systems is a promising approach towards predictability in these systems. The other two real-time scheduling approaches imposes strict limitations to the system. It is our objective to continue exploring this approach towards real-time component-based systems that provides timing guarantee in applications with some flexibility.

References

[1] Pecos project. http://www.pecos-project.org/.

[2] Sun microsystems, java 2 platform enterprise edition. http://java.sun.com/j2ee/.

[3] Corba component model. http://www.omg.org/technology/documents/formal/components.htm, June 2001.

[4] CHEN, D., MOK, A., AND NIXON, M. Real-time support in com. In *in Proceedings of the 32nd Hawaii International Conference on System Sciences* (Maui, Hawaii, USA, January 1999), Emerging Technologies. IEEE Computer Society,. Emerging Technologies.

[5] FREDRIKSSON, J., AKERHOLM, M., SANDERSTROM, K., AND DOBRIN, R. Attaining flexible real-time systems by bringing together component technologies and real-time systems theory. In *29th Euromicro Conference* (Turkey, September 2003).

[6] HATCLIFF, J., DENG, W., DWYER, M. B., JUNG, G., AND RANGANATH, V. Cadena: An integrated development, analysis, and verification environment for component-based systems. In *ICSE 2003* (Oregon, 2003).

[7] KO, R., AND MUTKA, M. W. A component-based approach for adaptive soft real-time within heterogeneous environments. In *the special issue of Parallel and Distributed Computing Practices* (September 2003). vol 5, no 1.

[8] LOCKE, C. D. *Best-effort decision-making for real-time scheduling*. PhD thesis, 1986.

[9] LUDERS, F. Adopting a software component model in real-time systems development. IEEE Computer Society Press. 28th Annual NASA/IEEE Software Engineering Workshop.

[10] NOLTE, T., MOLLER, A., AND NOLIN, M. Using components to facilitate stochastic schedulability analysis. 24th IEEE Real-Time Systems Symposium (RTSS'2003), Work in Progress Session.

[11] RAMAMRITHAN, K., AND STANKOVIC, J. A. Scheduling algorithms and operating systems support for real-time systems. In *Proceedings of IEEE* (January 1994), pp. 55 – 67.

[12] RAMAMRITHAN, K., AND STANKOVIC, J. A. Scheduling algorithms and operating systems support for real-time systems. vol. 82, pp. 55–67.

[13] SCHMIDT, D., LEVINE, D. L., AND MUNGEE, S. Goal. the openccm platform. http: //corbaweb.lifl.fr/OpenCCM/, 2002.

[14] S.LIU, J. W. *Real-Time Systems*, 1st ed. Prentice Hall, 2000.

[15] TEŠANOVIĆ, A., NYSTRÖM, D., HANSSON, J., AND NORSTRÖM, C. Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing* (February 2004).

[16] VAN OMMERING, R., VAN DER LINDEN, F., AND KRAMER, J. The koala component model for consumer eletronics software. *IEEE Computer* (March 2000).

[17] WANG, N., SCHMIDT, D. C., AND GOKHALE, A. Total quality of service provisioning in middleware and aplications. In *Elsevier Journal of Microprocessors and Microsystems* (January 2003). vol.26, number 9-10.

[18] XU, J., AND PARNAS, D. L. On Satisfying Timing Constraints in Hard Real-Time Systems. *IEEE Transactions on Software Engineering 19* (January 1997), 70–84. No 1.

[19] YAU, S., AND TAWEPONSOMKIAT, C. Component customization for object-oriented distributed real-time software development. In *ISORC 2000* (2000).

[20] YAU, S., AND TAWEPONSOMKIAT, C. An approach to object-oriented component customization for real-time software development. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC 2002)* (2002).

[21] ZINKY, J. A., BAKKEN, D. E., AND SCHANTZ, R. E. Architectural support for quality of service for corba objects. *Theory and Practice of Object Systems 3*, 1 (1997).