Reconfigurable Real-Time Middleware for Distributed Cyber-Physical Systems with Aperiodic Events *

Yuanfang Zhang, Christopher Gill and Chenyang Lu
Department of Computer Science and Engineering
Washington University, St. Louis, MO, USA
{yfzhang, cdgill, lu}@cse.wustl.edu

Abstract

Different distributed cyber-physical systems must handle aperiodic and periodic events with diverse requirements. While existing real-time middleware such as Real-Time CORBA has shown promise as a platform for distributed systems with time constraints, it lacks flexible configuration mechanisms needed to manage end-to-end timing easily for a wide range of different cyber-physical systems with both aperiodic and periodic events. The primary contribution of this work is the design, implementation and performance evaluation of the first configurable component middleware services for admission control and load balancing of aperiodic and periodic event handling in distributed cyber-physical systems. Empirical results demonstrate the need for, and the effectiveness of, our configurable component middleware approach in supporting different applications with aperiodic and periodic events, and providing a flexible software platform for distributed cyber-physical systems with end-to-end timing constraints.

1 Introduction

Many distributed cyber-physical systems (CPS) must handle a mix of aperiodic and periodic events, including aperiodic events with end-to-end deadlines whose assurance is critical to the correct behavior of the system. Computer-integrated manufacturing is representative of many distributed CPS. For example, in an industrial plant monitoring system, an aperiodic alert may be generated when a series of periodic sensor readings meets certain hazard detection criteria. This alert must be processed on multiple processors within an end-to-end deadline, e.g., to put an industrial process into a fail-safe mode. User inputs and other sensor readings may trigger other real-time aperiodic events.

*This work was supported in part by NSF grant CCF-0615341 and NSF CAREER award CNS-0448554.

While traditional real-time middleware such as Real-Time CORBA [16] and Real-Time Java [5] have shown promise as distributed software platforms for systems with time constraints, existing middleware systems lack the flexibility needed to support cyber-physical systems with diverse application semantics and requirements. For example, load balancing is an effective mechanism for handling variable real-time workloads in a distributed cyber-physical system. However, its suitability for cyber-physical systems highly depends on their application semantics. Some digital control algorithms (e.g., proportional-integral-derivative control) for physical systems are stateful and hence not amenable for frequent task re-allocation caused by load balancing, while others (e.g., proportional control) do not have such limitations. Similarly, job skipping (skipping the processing of certain instances of a periodic task) is an admission control strategy for dealing with transient system overload. However, job skipping is not suitable for certain critical control applications in which missing one job may cause catastrophic consequences on the controlled system. In contrast, other applications ranging from video reception to telecommunications may be able to tolerate varying degrees job skipping [12].

Therefore, a key open challenge for distributed cyber-physical systems is to develop a flexible middleware infrastructure that can be easily configured to support the diverse requirements of cyber-physical systems. Specifically, middleware services such as load balancing and admission control must support a variety of strategies. However, providing middleware services with configurable strategies faces several important challenges: (1) services must be able to provide configurable strategies; (2) the specific criteria that distinguish which service strategies are preferable must be identified, and applications must be categorized according to those criteria; and (3) appropriate combinations of services' strategies must be identified for each such application category, according to its characteristic criteria. To address these challenges, and thus to enhance support for diverse cyber-physical systems with aperiodic and periodic events,

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.96

we have designed and implemented a new set of component middleware services including end-to-end event scheduling, admission control, and load balancing.

Research Contributions: In this work, we have (1) developed what is to our knowledge the first set of configurable component middleware services supporting multiple admission control and load balancing strategies for handling aperiodic and periodic events; (2) defined categories of cyber-physical applications according to specific characteristics, and related them to suitable combinations of strategies for our services; and (3) provided a case study that applies different configurable services to a domain with both aperiodic and periodic events, offers empirical evidence of the tradeoffs among service configurations, and demonstrates the effectiveness of our approach in that domain. Our work thus significantly enhances the applicability of real-time middleware as a flexible infrastructure for distributed cyber-physical systems.

2 Background

Task Model: We consider cyber-physical systems comprised of physical systems generating aperiodic and periodic events that must be processed on distributed computing platforms subject to end-to-end deadlines. Henceforth the processing of a sequence of events is referred to as a *task*. A task T_i is composed of a chain of *subtasks* $T_{i,j}(1 \leq j \leq n_i)$ located on different processors. The release of the first subtask $T_{i,1}$ of a task T_i is triggered by a periodic timer event or an aperiodic event generated by the physical system. Upon completion, a subtask $T_{i,j}$ pushes another event which triggers the release of its successor subtask $T_{i,j+1}$. Each release of a subtask is called one *subjob*, and each release of a task is a *job* composed of a chain of subjobs. Every job of a task must be completed within an end-to-end deadline that is its maximum allowable response time. The period of a periodic task is the interarrival time of consecutive subjobs of the first subtask of the periodic task. An aperiodic task does not have a period. The interarrival time between consecutive subjobs of its first subtask may vary widely and, in particular, can be arbitrary small. The worst-case execution time of every subtask, the end-to-end deadline of every task, and the period of every periodic task in the system are known.

Component Middleware: Component middleware platforms are an effective way of achieving customizable reuse of software artifacts. In these platforms, *components* are units of implementation and composition that collaborate with other components via *ports*. The ports isolate the components' contexts from their actual implementations. Component middleware platforms provide execution environments and common services, and support additional tools to configure and deploy the components.

In previous work we developed the first instantiation of a middleware admission control service supporting both aperiodic and periodic events [24] (on TAO, a widely used Real-Time CORBA middleware). However, our previous admission control service only included a fixed set of strategies. As is shown in Section 4, a diverse set of interoperating services and strategies is needed to support cyber-physical systems with different application semantics. Unfortunately, it is difficult to extend implementations that rely directly on distributed object middleware such as our original admission control service. Specifically, in those middleware systems changing the supported strategy requires explicit changes to the service code itself, which can be tedious and error-prone in practice.

The Component-Integrated ACE ORB (CIAO) [1] implements the Light Weight CCM specification [17] and is built atop the TAO [2] real-time CORBA object request broker (ORB). CIAO abstracts real-time policies as installable and configurable units. However, CIAO does not support aperiodic task scheduling, admission control or load balancing. To develop a flexible infrastructure for cyber-physical systems, in this work we develop new admission control and load balancing services, each with a set of alternative strategies on top of CIAO.

Aperiodic Scheduling: Aperiodic tasks have been studied extensively in real-time scheduling theory, including work on aperiodic servers that integrate scheduling of aperiodic and periodic tasks [13]. New schedulability tests based on aperiodic utilization bounds [3] and a new admission control approach [4] also were introduced recently. In our previous work [24], we implemented and evaluated admission control services for two suitable aperiodic scheduling techniques (aperiodic utilization bound [3] and deferrable server [19]) on TAO. Since aperiodic utilization bound (AUB) has comparable performance to deferrable server, and requires less complex scheduling mechanisms in middleware, we focus exclusively on the AUB technique in this paper. Our experiences with AUB reported in this paper show how configurability of other techniques can be integrated within real-time component middleware in a similar way.

With the AUB approach, three kinds of strategies must be made configurable to provide flexible and principled support for diverse cyber-physical systems with aperiodic and periodic tasks: (1) when admissibility is evaluated (to tradeoff the granularity and thus the pessimism of admission guarantees), (2) when the contributions of completed jobs of subtasks can be removed from the schedulability analysis used for admission control (to reduce pessimism), and (3) when tasks can be assigned to different processors (to balance load and improve system performance).

In AUB [3], the set of *current* tasks $S(t)$ at any time t is defined as the set of tasks that have been released but whose

deadlines have not expired. Hence, $S(t) = \{T_i | A_i \leq t < A_i + D_i\}$, where A_i is the release time of the first subtask of task T_i, and D_i is the deadline of task T_i. The synthetic utilization of processor *j* at time t, $U_j(t)$, is defined as the sum of individual subtask utilizations on the processor, accrued over all current tasks. According to AUB analysis, a system achieves its highest schedulable synthetic utilization bound under End-to-end Deadline Monotonic Scheduling (EDMS) algorithm under certain assumptions. Under EDMS, a subtask has a higher priority if it belongs to a task with a shorter end-to-end deadline. Note that AUB does not distinguish aperiodic from periodic tasks. All tasks are scheduled using the same scheduling policy. Under EDMS task T_i will meet its deadline if the following condition holds [3]:

$$\sum_{j=1}^{n_i} \frac{U_{V_{ij}}(1 - U_{V_{ij}}/2)}{1 - U_{V_{ij}}} \leq 1 \qquad (1)$$

where V_{ij} is the j^{th} processor that task T_i visits. A task (or an individual job) can be admitted only when this condition continues to be satisfied for all admitted tasks and this task. Since applications may or may not tolerate job skipping, whether this condition is checked only when a task first arrives or whenever each job arrives should be configurable.

Note that a task remains in the current task set even if it has been completed, as long as its deadline has not expired. To reduce the pessimism of the AUB analysis, a *resetting rule* is introduced in [3]. When a processor becomes idle, the contribution of all completed subtasks to the processor's synthetic utilization can be removed without affecting the correctness of the schedulability condition (1). Since the resetting rule introduces extra overhead, it should be made configurable whether the contribution of only aperiodic subtasks or of both aperiodic and periodic subtasks can be removed early. Under AUB-based schedulability analysis, load balancing also can effectively improve system performance [3]. However some applications require persistent state preservation between jobs of the same task, so it should be made configurable whether a task can be reallocated to a different processor at each release.

3 Middleware Architecture

To support end-to-end aperiodic and periodic tasks in diverse cyber-physical applications, we have developed a new middleware architecture. The key feature of our approach is a *configurable component framework* that can be customized for different sets of aperiodic and periodic tasks. Our framework provides configurable *admission controller* (AC), *idle resetter* (IR) and *load balancer* (LB) components which interact with application components through *task effector* (TE) components. The AC component provides online admission control and schedulability tests for tasks that arrive dynamically at run time. The LB component provides an acceptable task assignment plan to the admission controller if the new arrival task is admittable. The IR component reports all completed subtasks on one processor to the AC component when the processor becomes idle, so the AC component can remove their contributions to the synthetic utilization to reduce the pessimism of the AUB analysis at run-time according to the idle resetting rule.

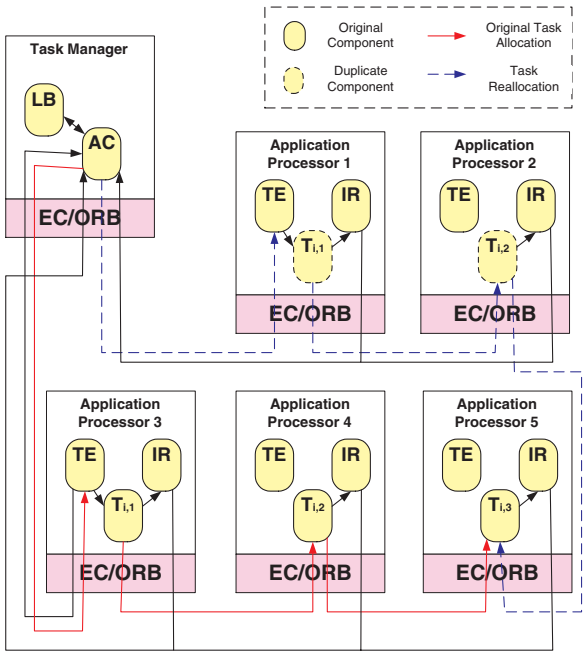

Figure 1. Distributed Middleware Architecture

Figure 1 illustrates our distributed middleware architecture. All processors are connected by TAO's federated event channel [10] which pushes events through local event channels, gateways and remote event channels to the events' consumers sitting on different processors. We deploy one AC component and one LB component on a central task manager processor, and one IR component and one task effector (TE) component on each of multiple application processors. Figure 1 shows an example end-to-end task T_i composed of 3 consecutive subtasks, $T_{i,1}$, $T_{i,2}$ and $T_{i,3}$, executing on separate processors. $T_{i,1}$ and $T_{i,2}$ have duplicates on other application processors. When task T_i arrives at an application processor, the task effector component on that processor pushes a "Task Arrival" event to the AC component and holds the task until it receives an "Accept" command from the AC component. The AC component and LB component decide whether to accept the task, and if so, where to assign its subtasks. The solid line and the dashed line show two possible assignments of subtasks. If the first subtask $T_{i,1}$ is not assigned to the processor where T_i arrived, we call this assignment a *task re-allocation*.

An advantage of this centralized AC/LB architecture is that it does not require synchronization among distributed admission controllers. In contrast, in a distributed architecture the AC components on multiple processors may need to coordinate and synchronize with each other in order to make correct decisions, because admitting an end-to-end task may affect the schedulability of other tasks located on the multiple affected processors. A potential disadvantage of the centralized architecture is that the AC component may become a bottleneck and thus affect scalability. However, the computation time of the schedulability analysis is significantly lower than task execution times in many distributed cyber-physical systems, which alleviates the scalability limitations of a centralized solution [24]. In summary, while our real-time component middleware approach can be extended to use a more distributed architecture, we have adopted a centralized approach with less complexity and overhead, which allows us to focus on achieving system flexibility through component middleware services.

4 Mapping CPS Characteristics to Middleware Strategies

A key contribution of this paper is categorizing a set of characteristics of different cyber-physical systems and mapping them to suitable middleware service strategies. In this section, we present a set of criteria used to categorize cyber-physical systems, and analyze how to map those criteria to different service strategies supported by our middleware.

4.1 CPS Characteristics

Three criteria distinguish how different cyber-physical systems with aperiodic tasks should be supported: Job skipping (criterion **C1**); State persistency (criterion **C2**); and Component replication (criterion **C3**).

Job Skipping means that some jobs of a task are executed while other jobs of the same task may not be admitted. Some applications, such as video streaming, and other loss-tolerant forms of sensing can tolerate job skipping, while in critical control applications, once a task is admitted, all its jobs should be allowed to execute.

State Persistency means that states are required to be preserved between jobs of a same task. For proportional control systems [18], task are stateless and only require current information, so jobs can be re-allocated dynamically. However, for integral control systems [18], tasks require incremental calculation and are not suitable for job re-allocation.

Component Replication depends on an application's throughput requirements. Replication is used here to reduce latency through load distribution, not for fault tolerance purposes. Only those applications with replicated components can support task re-allocation, whereas those that cannot be replicated (e.g. due to constraints on the locality of sensors or actuators) cannot support task re-allocation.

According to these different application criteria, the AC, IR and LB components can be configured to use different strategies. For each component, which strategy is more suitable depends on these criteria and the application's overhead constraints. Table 1 shows how these criteria help to classify CPS applications, which in turn allows selection of corresponding middleware strategies. We now examine the different strategies for each component and the trade-offs among them.

	No	Yes
C1:Job Skipping	AC per Task	AC per Job
C2:State Persistency	LB per Job	LB per Task
C3:Component Replication	No LB	LB

Table 1. Criteria and Middleware Strategies

4.2 Admission Control (AC) Strategies

Admission control offers significant advantages for systems with aperiodic and periodic tasks, by providing online schedulability guarantees to tasks arriving dynamically. Our AC component supports two different strategies: **AC per Task** and **AC per Job**. **AC per Task** performs the admission test only when a task first arrives while **AC per Job** performs the admission test whenever a job of the task arrives. Only applications satisfying criterion C1 are suitable for the second strategy, since it may not admit some jobs. Moreover, the second strategy reduces pessimism at the cost of increasing overhead. The application developer thus needs to consider trade-offs between overhead and pessimism in choosing a proper configuration.

AC per Task: Considering the admission overhead and the fixed inter-arrival times of periodic tasks, one strategy is to perform an admission test only when a periodic task first arrives. Once a periodic task passes the admission test, all its jobs are allowed to be released immediately when they arrive. This strategy improves middleware efficiency at the cost of increasing the pessimism of the admission test. In the AUB analysis [3], the contribution of a job to the synthetic utilization of a processor can be removed when the job's deadline expires (or when the CPU idles if the resetting rule is used and the subjob has been completed). If admission control is performed only at task arrival time, however, the AC component must reserve the synthetic utilization of the task throughout its lifetime. As a result, it cannot reduce the synthetic utilization between the deadline of a job and the arrival of the subsequent job of the same task, which may result in pessimistic admission decisions [3].

AC per Job: If it is possible to skip a job of a periodic task (criterion C1), the alternative strategy to reduce pessimism is to apply the admission test to every job of a *periodic* task.

This strategy is practical for many systems, since the AUB test is highly efficient when used for AC, as is shown in [23] by our overhead measurements.

4.3 Idle Resetting (IR) Strategies

The use of a resetting rule can reduce the pessimism of the AUB schedulability test significantly [3, 24]. There are three ways to configure IR components in our approach. The first of these three strategies avoids the resetting overhead, but is the most pessimistic. The third strategy removes the contribution of completed aperiodic and periodic subjobs more frequently than the other two strategies. Although it has the least pessimism, it introduces the most overhead. The second strategy offers a trade-off between the first and the third strategies.

No IR: The first strategy is to use no resetting at all, so that if the subjobs complete their executions, the contributions of completed subjobs to the processor's synthetic utilization are not removed until the job deadline. This strategy avoids the resetting overhead, but increases the pessimism of schedulability analysis.

IR per Task: The second strategy is that each IR component records completed aperiodic subjobs on one processor. Whenever the processor is idle, a lowest priority thread called an *idle detector* begins to run, and reports the completed *aperiodic* subjobs to the AC component through an "Idle Resetting" event. To avoid reporting repeatedly, the idle detector only reports when there is a newly completed aperiodic subjob whose deadline has not expired.

IR per Job: The third strategy is that each IR component records and reports not only the completed aperiodic subjobs but also the completed subjobs of *periodic* subtasks.

4.4 Load Balancing (LB) Strategies

Under AUB-based AC, load balancing can effectively improve system performance in the face of dynamic task arrivals [3]. We use a heuristic algorithm to assign subtasks to processors at run-time, which always assigns a subtask to the processor with the lowest synthetic utilization among all processors on which the application component corresponding to the task has been replicated (criterion C3). [1] Since migrating a subtask between processors introduces extra overhead, when we accept a new task, we only determine the assignment of that new task and do not change the assignment plan for any other task in the current task set. This service also has three strategies. The first strategy is suitable for applications which cannot satisfy criterion C3. The second strategy is most applicable for applications which satisfy both C3 and C2. The third strategy is most suitable for applications which only satisfy C3, but can not satisfy criterion C2.

No LB: This strategy does not perform load balancing. Each subtask does not have a replica and is assigned to a particular processor.

LB per Task: Each task will only be assigned once, at its first arrival time. This strategy is suitable for applications which must maintain persistent state between any two consecutive jobs of a periodic task.

LB per Job: The third strategy is the most flexible. All jobs from a periodic task are allowed to be assigned to different processors when they arrive.

[1] The focus here is not on the load balancing algorithms themselves. Our configurable middleware may be easily extended to incorporate LB components implementing other load balancing algorithms according to each application's needs.

4.5 Combining AC, IR and LB Strategies

When we use the AC, IR and LB components together, their strategies can be configured in 18 different combinations. However, some combinations of the strategies are invalid. The AC-per-Task/IR-per-Job combination is not reasonable, because per job idle resetting means the synthetic utilizations of all completed subjobs of periodic subtasks are to be removed from the central admission controller, but per task admission control requires that the admission controller reserves the synthetic utilization for all accepted periodic tasks, so an accepted periodic task does not need to go through admission control again before releasing its jobs. These two requirements are thus contradictory, and we can exclude the corresponding configurations as being invalid. Removing this invalid AC/IR combination means removing 3 invalid AC/IR/LB combinations, so there are only 15 reasonable combinations of strategies left. Looking to this degree of complexity to make "right" performant design decision, an application developer needs definitely some kind of cognitive support.

As Figure 2 shows, we divided scheduling of real-time tasks into three axes of configurability: admission control, idle resetting and load balancing. Different configuration options in each of these axes and the impact they may have, as well as conflicting configurations, are delineated thoroughly in this section.

4.6 Implementation

Configurable component middleware standards, such as the CORBA Component Model (CCM) [15], can help to reduce the complexity of developing distributed cyber-physical systems by defining a component-based programming paradigm. They also help by defining a standard configuration framework for packaging and deploying reusable software components. The CIAO [21] is an implementation of the Light Weight CCM specification [17] that is suitable

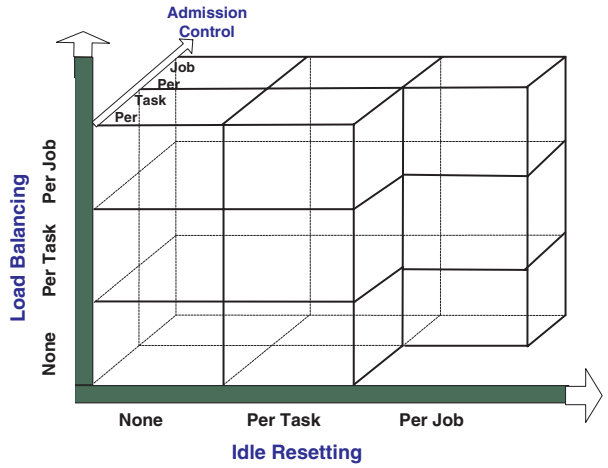

Figure 2. Middleware Services and Strategies

for distributed cyber-physical systems. To support the different strategies described in Section 4, and to allow flexible configuration of suitable combinations of those strategies for a variety of applications, we have implemented admission control, idle resetting and load balancing in CIAO as configurable components. Each component provides a specific service with configurable attributes and clearly defined interfaces for collaboration with other components, and can be instantiated multiple times with the same or different attributes. Component instances can be connected together at run-time through appropriate ports to form a distributed cyber-physical system. We have designed and implemented 6 configurable components to support distributed real-time aperiodic and periodic end-to-end tasks, using ACE/TAO/CIAO version 5.6/1.6/0.6. We also provided a configuration pre-parser and a component configuration interface to allow developers to select and configure each service flexibly, according to each application's specific needs, while disallowing invalid combinations of services' strategies. More details of the implementation can be found in [23].

5 Experimental Evaluations

The experiments were performed on a testbed consisting of six machines connected by a 100Mbps Ethernet switch. Two are Pentium-IV 2.5GHz machines with 1G RAM and 512K cache each, two are Pentium-IV 2.8GHz machines with 1G RAM and 512K cache each, and the other two are Pentium-IV 3.4GHz machines with 2G RAM and 2048K cache each. Each machine runs version 2.4.22 of the KURT-Linux operating system. One Pentium-IV 2.5GHz machine is used as a central task manager where the AC and LB components are deployed. The other five machines are used as application processors.

5.1 Random Workloads

We first randomly generated 10 task sets, each including 4 aperiodic tasks and 5 periodic tasks. The number of subtasks per task is uniformly distributed between 1 and 5. Subtasks are randomly assigned to 5 application processors. Task deadlines are randomly chosen between 250 ms and 10 s. The periods of periodic tasks are equal to their deadlines. The arrival of aperiodic tasks follows a Poisson distribution. The synthetic utilization of every processor is 0.5, if all tasks arrive simultaneously. Each subtask is assigned to a processor, and has a duplicate sitting on a different processor which is randomly picked from the other 4 application processors.

In this experiment, we evaluated all 15 reasonable combinations of strategies on each task set. Each task set ran for 5 minutes for each combination. The performance metric we used in these evaluations is the *accepted utilization ratio*, i.e., the total utilization of jobs actually released divided by the total utilization of all jobs arriving. To be concise, we use capital letters to represent strategies: **N** when a service is **not** enabled in this configuration; **T** when a service is enabled for each **task**; and **J** when a service is enabled for each **job** of a task. In the following figures, a three element tuple denotes each combination of settings for the three configurable services: first for the admission control service, then for the idle resetting service, and last for the load balancing service.

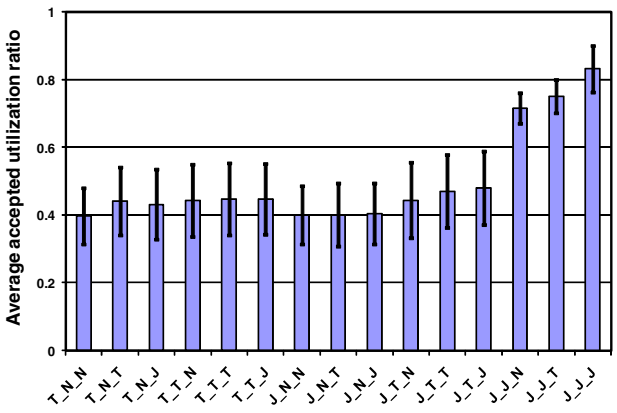

Figure 3. Accepted Utilization Ratio

The bars in Figure 3 show the average results over the 10 task sets. As is shown in Figure 3, enabling either idle resetting or load balancing can increase the utilization of tasks admitted. Moreover, the experiment shows that enabling IR-per-Job (*_J_*) significantly outperforms the configurations which enable IR-per-Task (*_T_*) or not at all (*_N_*). This is because IR-per-Job removes the contribution of all completed periodic subjobs to the synthetic utilizations which greatly helps to admit more jobs. En-

abling all three services per job (J_J_J) performed comparably to the other (J_J_*) configurations (averaging higher though the differences were not significant) and outperformed all other configurations significantly, even though the J_J_J configuration introduces the most overhead. We also notice the difference is small when we only change the configuration of the LB component and keep the configuration of other two services the same. This is because when we randomly generated these 10 task sets, the resulting synthetic utilization of each processor was similar. To show the potential benefit of the LB component, we designed another experiment that is described in the next subsection.

5.2 Imbalanced Workloads

In the second experiment, we use an imbalanced workload. It is representative of a dynamic CPS in which a subset of the system processors may experience heavy load. For example, in an industrial control system, a blockage in a fluid flow valve may cause a sharp increase in the load on the processors immediately connected to it, as aperiodic alert and diagnostic tasks are launched. In this experiment, we divided the 5 application processors into two groups. One group contains 3 processors hosting all tasks. The other group contains 2 processors hosting all duplicates. 10 task sets are randomly generated as in the above experiment, except that all subtasks were randomly assigned to 3 application processors in the first group and the number of subtasks per task is uniformly distributed between 1 and 3. The synthetic utilization for any of these three processors is 0.7. Each subtask has one replica sitting on one processor in the second group.

Each of 10 task sets was run for the 15 different valid combinations, and for each combination we then averaged the utilization acceptance ratio over the 10 results. These 15 combinations can be divided into 5 sets. Each set contains three combinations represented by three adjacent bars in Figure 4. In each set, we kept the admission control and idle resetting strategies the same, but changed the load balancing strategy from none to per task, then to per job. As figure 4 shows, LB-per-Task provides a significant improvement when compared with the results without LB. However, there is not much difference between LB-per-Task vs. LB-per-Job.

From these two experiments, we found that configuring different strategies according to application characteristics can have a significant impact on the performance of a cyber-physical system with aperiodic and periodic events. Our design of the AC, IR and LB services as easily configurable components allows application developers to explore and select valid configurations based on the characteristics and requirements of their applications, and based on the tradeoffs indicated by these empirical results.

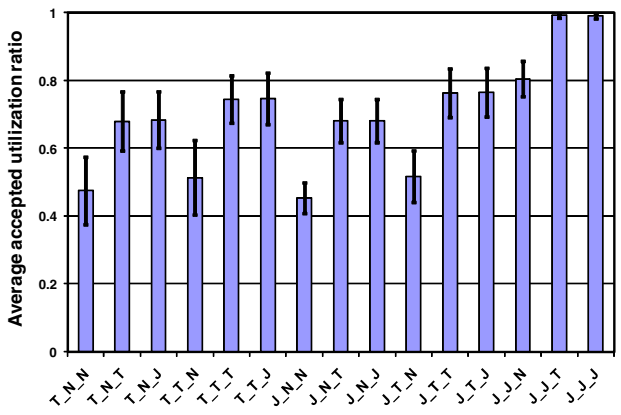

Figure 4. LB Strategy Comparison

6 Related Work

Component Middleware: The architectural patterns used in the CORBA Component Model (CCM) [15] are also used in other popular component middleware technologies, such as J2EE [20]. Among the existing component middleware technologies, CCM is the most suitable for distributed cyber-physical systems since CORBA is the only standards-based COTS middleware that explicitly considers the real-time requirements of distributed cyber-physical systems.

QoS-aware Middleware: Quality Objects (QuO) [25] is an adaptive middleware framework developed by BBN Technologies that allows developers to use aspect-oriented software development techniques to separate the concerns of QoS programming from application logic in distributed cyber-physical systems. A Qosket is a unit of encapsulation and reuse for QuO systemic behaviors. In comparison to CIAO, Qoskets and QuO emphasize dynamic QoS provisioning where CIAO emphasizes static QoS provisioning and integration of various mechanisms and behaviors during different stages of the development lifecycle. The dynamicTAO [11] project applies reflective techniques to reconfigure Object Request Broker (ORB) components at run-time. Similar to dynamicTAO, the Open ORB [9] project also aims at highly configurable and dynamically reconfigurable middleware platforms to support applications with dynamic requirements. Zhang et al. [22] use aspect-oriented techniques to improve the customizability of the middleware core infrastructure at the ORB level.

QoS-aware Component Middleware: Component middleware's container architecture enables meta-programming of QoS attributes in component middleware. de Miguel [7] develops QoS-enabled containers by extending a QoS EJB container interface to support a QoSContext interface that allows the exchange of QoS-related information among component instances.

To take advantage of the QoS-container, a component must implement QoSBean and QoSNegotiation interfaces. However, this requirement increases dependence among component implementations. The QoS Enabled Distributed Objects (Qedo) [8] project is another effort to make QoS support an integral part of CCM. Qedo's extensions to the CCM container interface and Component Implementation Framework (CIF) require component implementations to interact with the container QoS interface and negotiate the level of QoS contract directly. Although this approach is suitable for certain applications where QoS is part of the functional requirements, it tightly couples the QoS provisioning and adaptation behaviors into the component implementation, which may limit the reusability of the component. In comparison, CIAO explicitly avoids this coupling and composes the QoS aspects into applications declaratively. There have been several other efforts to introduce of QoS in conventional component middleware platforms. The FIRST Scheduling Framework (FSF) [14] proposes to compose several applications and to schedule the available resources flexibly while guaranteeing hard real-time requirements. A real-time component type model [6], which integrates QoS facilities into component containers, was also introduced based on the EJB and RMI specifications. None of these approaches provides the configurable services for mixed aperiodic and periodic end-to-end tasks offered by our approach.

7 Conclusions

The work presented in this paper represents a promising step towards configurable middleware services for diverse distributed cyber-physical systems with aperiodic and periodic events. We first analyzed a set of key characteristics of different cyber-physical systems and mapped them to suitable strategies for middleware services. We then designed and implemented configurable middleware components that provide effective on-line admission control and load balancing and can be easily configured and deployed on distributed computing platforms. Empirical results showed that our configurable component middleware is well suited for satisfying different applications with alternative characteristics and requirements.

References

[1] Component-Integrated ACE ORB (CIAO). www.dre.vanderbilt.edu/CIAO/.

[2] The ACE ORB (TAO). www.dre.vanderbilt.edu/TAO/.

[3] T. F. Abdelzaher, G. Thaker, and P. Lardieri. A Feasible Region for Meeting Aperiodic End-to-end Deadlines in Resource Pipelines. In *ICDCS*, 2004.

[4] B. Andersson and C. Ekelin. Exact Admission-Control for Integrated Aperiodic and Periodic Tasks. In *RTAS*, 2005.

[5] G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[6] M. A. de Miguel. Integration of QoS Facilities into Component Container Architectures. In *ISORC*, 2002.

[7] M. A. de Miguel. QoS-Aware Component Frameworks. In *IWQoS*, 2002.

[8] FOKUS. Qedo Project Homepage. `http://qedo.berlios.de/`.

[9] G. S. Blair et al. The Design and Implementation of Open ORB 2. *IEEE Distributed Systems Online*, 2(6), June 2001.

[10] T. H. Harrison, D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-time CORBA Event Service. In *OOPSLA*, 1997.

[11] F. Kon, F. Costa, G. Blair, and R. H. Campbell. The Case for Reflective Middleware. *Commun. ACM*, 45(6):33–38, June 2002.

[12] G. Koren and D. Shasha. Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips. In *RTSS*, 1995.

[13] L. Sha et al. Real Time Scheduling Theory: A Historical Perspective. *The Journal of Real-Time Systems*, 10:101–155, 2004.

[14] M. Aldea et al. FSF: A Real-Time Scheduling Architecture Framework. In *RTAS*, 2006.

[15] Object Management Group. *CORBA Components*, OMG Document formal/2002-06-65 edition, June 2002.

[16] Object Management Group. *Real-Time CORBA Specification*, 1.1 edition, Aug. 2002.

[17] Object Management Group. *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 edition, May 2003.

[18] F. H. Raven. *Automatic Control Engineering*. Mcgraw-Hill, New York, New York, fifth edition, 1994.

[19] J. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[20] M. Volter, A. Schmid, and E.Wolff. *Server Component Patterns – Component Infrastructures illustrated with EJB*. Wiley & Sons, New York, 2002.

[21] N. Wang, C. Gill, D. C. Schmidt, and V. Subramonian. Configuring Real-time Aspects in Component Middleware. In *DOA*, 2004.

[22] C. Zhang and H.-A. Jacobsen. Resolving Feature Convolution in Middleware Systems. In *OOPSLA*, 2004.

[23] Y. Zhang, C. Gill, and C. Lu. Reconfigurable Real-Time Middleware for Distributed Cyber-Physical Systems with Aperiodic Events. Technical Report WUCSE-2008-5, WUSTL, 2008.

[24] Y. Zhang, C. Lu, C. Gill, P. Lardieri, and G. Thaker. Middleware Support for Aperiodic Tasks in Distributed Real-Time Systems. In *RTAS*, 2007.

[25] J. A. Zinky, D. E. Bakken, and R. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1):1–20, 1997.