Towards Middleware for Fault-tolerance in Distributed Real-time and Embedded Systems

Jaiganesh Balasubramanian[1], Aniruddha Gokhale[1], Douglas C. Schmidt[1], and Nanbor Wang[2]

[1] Department of Electrical Engineering and Computer Science,
Vanderbilt University, Nashville, TN 37203, USA
[2] Tech-X Corporation, Boulder, CO, USA.

Abstract. Distributed real-time and embedded (DRE) systems often require support for multiple simultaneous quality of service (QoS) properties, such as real-timeliness and fault tolerance, that operate within resource constrained environments. These resource constraints motivate the need for a lightweight middleware infrastructure, while the need for simultaneous QoS properties require the middleware to provide fault tolerance capabilities that respect time-critical needs of DRE systems. Conventional middleware solutions, such as Fault-tolerant CORBA (FT-CORBA) and Continuous Availability API for J2EE, have limited utility for DRE systems because they are heavyweight (e.g., the complexity of their feature-rich fault tolerance capabilities consumes excessive runtime resources), yet incomplete (e.g., they lack mechanisms that enable fault tolerance while maintaining real-time predictability).
This paper provides three contributions to the development and standardization of lightweight real-time and fault-tolerant middleware for DRE systems. First, we discuss the challenges in realizing real-time fault-tolerant solutions for DRE systems using contemporary middleware. Second, we describe recent progress towards standardizing a CORBA lightweight fault-tolerance specification for DRE systems. Third, we present the architecture of FLARe, which is a prototype based on the OMG real-time fault-tolerant CORBA middleware standardization efforts that is lightweight (e.g., leverages only those server- and client-side mechanisms required for real-time systems) and predictable (e.g., provides fault-tolerant mechanisms that respect time-critical performance needs of DRE systems).

1 Introduction

Emerging trends and challenges. Distributed object computing (DOC) middleware, such as CORBA and Real-time CORBA (RT-CORBA), is used to support the development and deployment of many distributed real-time and embedded (DRE) systems, such as shipboard computing systems and intelligence, surveillance, and reconnaissance systems. Such systems often operate in resource-constrained environments and consist of soft real-time applications whose avail-

ability and timeliness requirements must be satisfied simultaneously. For example, target tracking systems should provide timely response for analyzing sensor readings even when hardware and software failures occur.

Prior research on providing quality of service (QoS) using DOC middleware has addressed the timeliness [20] and availability [16, 19] requirements of DRE systems. Moreover, several standards have defined interfaces and provide services and strategies to enhance the timeliness and fault-tolerance capabilities of DRE systems. For example, RT-CORBA [14] and Distributed Real-time Java [7] provide capabilities to ensure predictable end-to-end behavior for remote object method invocations. Similarly, Fault-Tolerant CORBA (FT-CORBA) [13] and Continuous Availability API for J2EE [23] provide services and strategies to enhance the dependability of DRE applications.

Despite promising prior work on providing timeliness and fault-tolerance capabilities for DRE systems, key problems remain unsolved. Existing approaches provide solutions that address only one QoS dimension (*e.g.*, timeliness) at a time. As such, these approaches do not simultaneously satisfy multiple QoS requirements, such as timeliness *and* availability. For example, fault-tolerance solutions are often not designed to honor timeliness while recovering from failures, whereas real-time solutions often do not recover from failures while ensuring predictable end-to-end behavior for remote object method invocations.

Moreover, *ad hoc* solutions—where availability and timeliness capabilities are obtained by simply adopting a combination of one or more solutions (*e.g.*, FT-CORBA and RT-CORBA) described above—are brittle and hard to maintain and upgrade. Likewise, many DRE systems run in dynamic operating environments where workloads and resource availabilities fluctuate, which affect availability and timeliness requirements of applications. DRE systems therefore need middleware that (1) integrates real-time and fault-tolerance by design, rather than in an *ad hoc* manner, (2) is lightweight so that it is suitable for resource-constrained deployments, and (3) is adaptive so that availability and timeliness properties can be tuned dynamically at runtime to maintain soft real-time and fault-tolerant performance.

Solution approach → Lightweight Real-time Fault-tolerant Middleware. To address these unresolved challenges with prior work, this paper describes FLARe (*F*ault-tolerant *L*ightweight *A*daptive *Re*al-time (FLARe)), which is a CORBA-based middleware characterized by the following contributions:

• *Lightweight middleware architecture*, that integrates fault-tolerance and real-time solutions by design, instead of via an *ad hoc* combination of the complete FT-CORBA and RT-CORBA specifications. FLARe supports the provisioning of fault-tolerance functionality based on application time requirements, *e.g.*, to make failure recovery faster and more predictable for critical (as opposed to non-critical) applications.

• *Resource-aware adaptive fault-tolerance*, where the middleware supports flexible fault-tolerant system configurations (rather than inflexible configuration prevalent in conventional FT solutions) whose behavior depends on resource availability and utilization levels. When resource availability fluctuates due to

failures, FLARe allocates the most suitable resources amongst the available resources for critical applications to increase the probability of meeting deadlines after failure recovery.

FLARe's design is based on the Object Management Group (OMG)'s standardization efforts to define a *Lightweight Fault-tolerance for Distributed Real-time Systems* (Lw-FT-RT-CORBA) [15] specification for CORBA-based DRE systems. In addition to summarizing these efforts, this paper focuses on the novel techniques that FLARe uses to provide fast, predictable, and resource-aware failure recovery for DRE systems. FLARe is developed atop the TAO (`www.dre.vanderbilt.edu/TAO`) RT-CORBA object request broker (ORB).

2 Objectives of the Lw-FT-RT-CORBA Effort

The goal of Lw-FT-RT-CORBA is to provide middleware mechanisms that simultaneously support availability and timeliness QoS for DRE systems. This section first describes the system and fault model of DRE systems that Lw-FT-RT-CORBA is intended to support. We then describe the key challenges of simultaneously providing availability and fault-tolerance capabilities for DRE systems and explain why the FT-CORBA [13] standard is inadequate for DRE systems. Finally, we summarize how FLARe achieves the objectives of Lw-FT-RT-CORBA to resolve these challenges effectively.

2.1 System and Fault Model

This paper focuses on request/response-based DRE systems, where clients invoke remote operations on servers and where client timeliness and availability requirements must be satisfied. Many real-time services (*e.g.*, sensor data acquisition and processing) are inherently stateless. For example, in target tracking systems the coordinate calculator that receives images from an image forwarding base station and calculates coordinates of surveillance images can be designed to process each image independently to avoid maintaining state between each invocation. Such systems need to provide real-time performance to clients, even in the presence of failures and load fluctuations. The goal of Lw-FT-RT-CORBA is to support soft real-time and fault-tolerant QoS properties for these applications.

Replication style. ACTIVE and PASSIVE replication are two approaches for building fault-tolerant distributed systems [16]. In ACTIVE replication, client requests are multicast and executed at all replicas to maintain strong consistency and provide fast failure recovery. ACTIVE replication, however, can incur excessive overhead for DRE systems composed of (1) stateless applications, such as the coordinate calculator systems which do not maintain state from prior sampling period's request processing as the processing in the current sampling period is independent from previous sampling periods, and (2) soft real-time applications that can tolerate occasional deadline misses. Prior research [4,18] has shown that PASSIVE replication and its variants are more effective for DRE systems because of its low execution overhead, and hence our focus is on how Lw-FT-RT-CORBA

can effectively support real-time and fault-tolerant requirements of applications using PASSIVE replication.

System and fault model. The clients and servers (*e.g.*, the image forwarding base station and coordinate calculator services in the target tracking example) are implemented as RT-CORBA objects. The processors and the processes hosted by the processors are designed using a *fail-stop* model, where (1) each processor or a process halts in response to a failure rather than produce erroneous results and (2) a processor's or process' halted state can be detected by a failure detector. These types of faults may occur due to aging or acute damage. Considering unpredictable behavior of processes or processors is beyond the scope of this paper. We assume that networks provide bounded communication latencies and do not fail. This assumption is reasonable for many DRE systems, such as avionics mission computing and shipboard computing environments, where nodes are connected by highly redundant high-speed networks.

2.2 Resource-aware Fault Recovery Challenges for Lw-RT-FT CORBA

In the context of the system and fault model described in Section 2.1, the following are key unresolved failure recovery challenges for using PASSIVE replication effectively in CORBA-based DRE systems.

• **Challenge 1: Providing efficient and predictable system/failure management**. As described in Section 1 and Section 2.1, DRE systems operate in dynamic operating environments, where new applications are deployed in response to changing workloads and failures. This dynamic deployment causes (1) increased resource utilization in certain processors and (2) load imbalance amongst the processors in the system. Middleware that is designed to provide failure recovery and management in a timely manner needs mechanisms that can react to changing load conditions and failures. Such dynamic environment changes must be communicated to the fault-tolerant middleware quickly and predictably so that failure management decisions, such as failover target selection, can be adapted and updated at runtime.

• **Challenge 2: Providing adaptive failover target selection**. When a CORBA application fails due to a processor/process failure, the respective client-side ORB receives a CORBA COMM_FAILURE exception [13]. Fault-tolerant ORBs therefore need to mask clients from those exceptions and transparently redirect clients to appropriately chosen backups. After a failover, the CPU utilization of the processors hosting the failover targets increase and the response times of the clients depend on the utilization levels of those processors.

If the failover targets are chosen statically—and without the knowledge of current system resource availability—client failovers could cause system resource overload, where different processor failures cause all the clients to failover to the same processor. A well-known approach for maintaining deadlines of application tasks in a processor is to ensure that its utilization remains below its schedulable utilization bound [10]. If resource overloads occur, however, this could cause increased utilization that exceeds the schedulable bound in those processors,

thereby causing applications to miss deadlines. Failover targets must therefore be chosen based on system's resource availability, as well as replica's resource requirements, so that application timeliness requirements are not compromised.
• **Challenge 3: Providing transparent and predictable failure recovery**. One way to provide appropriate failure recovery is to decide on a failover target after receiving the CORBA COMM_FAILURE exception. This approach increases the time clients need to failover to an appropriate backup, however, and thus affects application deadlines. Failover target information must therefore be available at the client-side fault-tolerant middleware ahead of the failure time, so that the clients can failover to appropriate backups quickly and predictably.

2.3 Limitations of FT-CORBA for DRE Systems

To support PASSIVE replication, the FT-CORBA [13] specification collects CORBA objects into *replication groups*. Replica addresses are grouped by a standard mechanism called an *interoperable object group reference* (IOGR), which comprises a sequence of CORBA *interoperable object references* (IORs), each of which points to a server replica IOR. FT-CORBA clients invoke operations using IOGRs as if they were making invocations using IORs.

If a server object fails in the IOGR model the client-side ORB catches the exception, and cycles through the IORs in the IOGR to handle the request at a different replica. This approach ensures faster client failover and provides clients with a transparent abstraction as though the service was provided by a single server. If no IORs in the IOGR list are valid (*e.g.*, if no replicas are live) an exception is propagated to the client application so it can start a recovery process to find a new set of server object addresses.

Although the IOGR provides a standardized, transparent mechanism for client-side failover if a server replica fails, the overall architecture has the following shortcomings:

• **No seamless integration with RT-CORBA**. Not all RT-CORBA ORBs support the FT-CORBA IOGR feature. Even if it is supported, there are no guidelines on how the FT-CORBA services will work with RT-CORBA features, such as *thread pool with lanes* and *banded connections*. Without these features higher priority applications cannot be provided with fault-tolerance capabilities in a timely manner due to lack of support for for prioritizing failure detections and notifications.

• **Fixed and load-unaware replica selection**. FT-CORBA's mechanism of selecting the next IOR from a sequence provides fast failover. The default FT-CORBA replica selection policy, however, does not consider each server's resource utilization, which may affect client response times after failover. For example, due to dynamic task arrivals and changing system utilization levels, a replica that was a suitable failover target at deployment time may be a poor choice at runtime.

These shortcoming of FT-CORBA for DRE systems described above motivate the need for—and approach taken by—the Lw-FT-RT-CORBA standardization effort.

2.4 Salient Features of Lw-FT-RT-CORBA

To overcome the limitations of FT-CORBA for DRE systems, Lw-FT-RT-CORBA requires the integration of real-time and fault-tolerance capabilities into a DRE system by design. Lw-FT-RT-CORBA combines the following capabilities: (1) *FT-enabled middleware*, which provides fault-tolerance capabilities that does not require any real-time features, *e.g.*, a client-side interceptor can catch failure exceptions irrespective of the priority of the server process that has failed, (2) *FT-enabled real-time middleware*, which provides fault-tolerance capabilities that requires real-time features, *e.g.*, a failure detector needs to differentiate between the reporting of the failure of a higher priority object from that of a lower priority object so that fault recovery can be prioritized, and (3) *middleware-independent fault-tolerance mechanisms*, which support adaptive fault-tolerance, *e.g.*, fault-tolerant decision making, such as failover target selection, can be made using algorithms that are independent of the supported middleware.

The Lw-FT-RT-CORBA approach is different than the FT-CORBA approach, which provisions all fault-tolerance capabilities using FT-enabled middleware. For example, in FT-CORBA fault recovery is provided by (1) a *fault detector*, which is a CORBA component that detects CORBA object failures, (2) a *fault notifier*, which is a CORBA component that notifies CORBA object failures, and (3) a *client-side interceptor*, which is a CORBA component that detects client-side failure exceptions to redirect clients to the next profile in the server IOGRs. As described in Section 2.3, however, these capabilities do not function properly due to the non-adaptive, resource-unaware recovery mechanisms in FT-CORBA. To address these limitations, Lw-FT-RT-CORBA uses a micro-kernel approach that provisions fault-tolerance functionality via the combination of capabilities described above that collaborate to provide real-time fault-tolerance capabilities for DRE systems.

3 The Design of FLARe

This section describes the design of FLARe and shows how it addresses the resource-aware fault recovery challenges described in Section 2.2. FLARe is designed to address the requirements of Lw-FT-RT-CORBA *i.e.*, provide both availability and timeliness capabilities for DRE systems. Figure 1 shows the key components of FLARe's architecture, which includes protocols, mechanisms, and services for supporting fault-tolerance capabilities using PASSIVE replication for DRE systems.

The novel aspects of FLARe's design include the combination of (1) *client-side FT-enabled middleware components*, which transparently provide client redirection and request reinvocation, (2) *server-side FT-enabled real-time middle-*

ware components, which monitor replica and process failures along with system parameters, such as CPU utilization, and help provide resource-aware and priority-aware tunable fault-tolerance, and (3) *infrastructure-specific middleware-independent mechanisms*, which use interfaces for replica registration and specifying application QoS requirements to support fine-grained tuning of fault-tolerance policies to ensure timely performance of DRE applications.

The interactions between the FLARe components combine real-time and fault-tolerance features, and hence provide an open platform for evaluating key issues in real-time fault-tolerance capabilities for DRE systems. Moreover, while describing the interactions between these different components, we also elaborate on the design choices we made and patterns used to implement various entities of FLARe's architecture. FLARe's pattern-based design enhances its flexibility and portability, without impeding the primary objectives of fault tolerance and real-time.

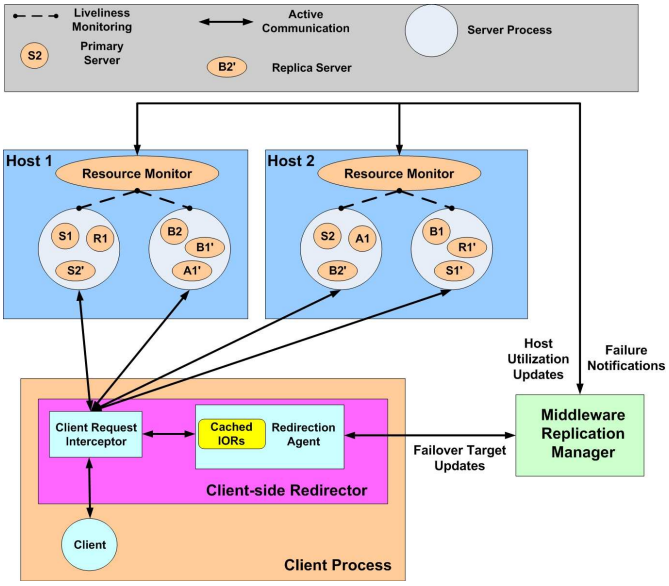

Fig. 1: **FLARe Middleware Architecture**

3.1 Providing Efficient and Predictable System/Failure Management

DRE systems often operate in dynamic operating environments, where processor utillizations fluctuate due to dynamic application deployments and failures. Changes in the system (*e.g.*, increase in CPU load) must therefore be conveyed to the fault-tolerant middleware quickly so appropriate actions can be taken.

Problem. In FT-CORBA, liveness checking is typically accomplished via an *is_alive* message from a *fault detector* component to all the CORBA objects it monitors. However, as described in [15], the failures and recovery occur at

the granularity level of a process and its address space. Liveness checking of individual objects for failure detection can therefore introduce unwanted and substantial overhead that adversely impacts real-time requirements. Moreover, introducing messaging for liveness check introduces additional system overhead.

Moreover, multiple objects and processes could fail simultaneously in DRE systems. Since the objects differ by their priority, failure and recovery management of those objects must also be prioritized. What is needed therefore is a resource monitoring infrastructure that is (1) decentralized, so that processor-specific local monitors can monitor the liveness of processes and their hosted objects, and (2) scalable and predictable, so that the failure as well as utilization reports are communicated with the fault-tolerant middleware according to the priority of the applications monitored.

Solution → Predictable and scalable resource monitors. As shown in Figure 1, FLARe employs a pair of FT-enabled real-time middleware components namely *middleware replication manager* and *resource monitor* to provide a decentralized, and predictable failure and system management for DRE systems. The middleware replication manager is composed of several sub-components, including a (1) *failure manager*, (2) *system manager*, (3) *resource manager*, and (4) *fault-tolerance manager*.

The failure manager receives failure notifications and works with the system manager to start new replicas if the replication degree of replica is below an acceptable threshold. The system manager receives system runtime information (such as CPU utilizations at different processors) and works with the resource manager to tune fault-tolerance decisions (*e.g.*, failover targets) dyhamically. The fault-tolerance manager works with the client-side and server-side middleware to notify the fault-tolerance decisions made by the resource manager.

FLARe runs a *resource monitor* on each processor to track the CPU utilization and liveness of the processes hosted by the processor. On Linux platforms, for example, the resource monitor uses the `/proc/stat` file to estimate the CPU utilization in each sampling period. This file records the number of "jiffies" (a default duration of 10ms in Linux) when the CPU is in user, nice, system, and idle modes. At the end of each sampling period, the resource monitor reads the counters and estimates the CPU utilization as the fraction of time when the CPU is not idle.

To perform liveness checking of processes in a processor, FLARe uses the *Acceptor/Connector* [21] pattern that decouples connection establishment and service initialization in a distributed system from the processing performed once the service is initialized. Since the server process and resource monitor run on the same host, the connection uses local connection mechanisms, such as a POSIX local socket (also known as a UNIX domain socket) or Windows named pipes.

For example, on Linux each application process opens a passive (*i.e.*, Acceptor role) POSIX local socket, and registers the port number with the resource monitor. The resource monitor connects to (*i.e.*, Connector role) and performs a blocking read on the socket. If an application process crashes, the socket and the

opened port will be invalidated. The resource monitor then receives an invalid read error on the socket, which indicates the failure of the process.

Resource monitors generate periodic and event-driven notifications regarding failures and system utilization. FLARe's replication manager (the system and failure manager sub-components) must handle these periodic requests from all hosts it manages. The replication manager must therefore allocate appropriate resources to serve these requests concurrently and these events may be treated at different levels of priorities, depending on the criticality of the process and processor being monitored.

Addressing the challenges outlined above requires an approach that can handle incoming requests concurrently with negligible overhead stemming from context switching and data copying activities. The client-side (resource monitors) defines the priority at which the requests will be executed at the system and failure managers. FLARe therefore uses RT-CORBA's CLIENT_PROPAGATED priority model, which allows clients to dictate the CPU priority using which the server executes the client request.

To allow the system and failure managers to serve the requests arriving at different priorities, FLARe uses the RT-CORBA *thread pool with lanes* feature, which partitions the available number of threads across different priorities, so that each server can simultaneously serve multiple client requests arriving at multiple priorities. The number of threads is configured at deployment time depending on the number of resource monitors deployed in the system. By selecting real-time features, such as *thread pool with lanes*, and integrating them with fault-tolerance features, such as process liveness checking, FLARe provides prioritized failure management for applications using the combination of the FT-enabled real-time middleware components.

3.2 Providing Adaptive Failover Target Selection

For every replica in the system, failover targets should be determined based on updated information about the processor utilizations and failures, so that clients do not failover to replicas that (1) *are overloaded*, which can cause potential deadline misses, and (2) *have already failed*, which can potentially increases failure recovery time.

Problem. Fault-tolerant middleware needs to make per-replica failover target decisions based on algorithms [**?**, 12]. DRE systems, however, often have a wide variety of applications with different characteristics and priorities. Hence, a single decision making algorithm will not suffice for the needs of all applications. What is needed, therefore, is middleware that can support real-time fault-tolerant decision making based on various algorithms specialized for the needs of different applications.

Solution → Adaptive resource manager. As described in Section 3.1, the middleware replication manager has a subcomponent *resource manager* that works with the *system manager* to tune fault-tolerance configurations of the system in response to changing system configurations. FLARe's resource manager makes run-time, resource-aware decisions about the fault-tolerance configura-

tions so that the clients can access the services in a fault-tolerant and timely manner. Example fault-tolerance configurations include per-replica failover targets, per-replica physical mapping onto processors, and per-replica weaker consistency optimizations. Research has been done in each of these decision-making dimension (*e.g.*, failover target selection) and many algorithms have been proposed [1, 3, 12].

To allow the resource manager to make decisions using a wide variety of algorithms, FLARe uses the *Strategy* pattern [5] to factor out similarities among algorithmic alternatives. For each decision-making dimension, the resource manager can be configured at deployment time with an algorithm strategy that is customized for application-specific availability and timeliness requirements.

The capability to plug-in many different decision-making algorithms allows FLARe to cater to the needs of a wide variery of applications. FLARe provides a failover target selection algorithm that determines a list of failover targets ordered by the predicted CPU utilization of the processors if a failover occurs (the processor with the lowest predicted CPU utilization is the first in the list). The algorithm and the subsequent performance within the context of FLARe is described in [?]. Moreover, as described in Section 3.1, the system manager receives information about the processor utilizations in a prioritized manner. Hence, the resource manager can provide predictable fault-tolerance by working on tuning the fault-tolerance configurations of higher priority objects rather than lower priority objects, whenever there are changes in resource availability and utilizations.

3.3 Providing Transparent and Predictable Failure Recovery

Client-side middleware in DRE systems must transparently handle failure exceptions caught as a result of process and processor failures and redirect clients to appropriate failover targets in a predictable and faster manner.

Problem. The per-replica failover target information computed by FLARe's resource manager is used by the client-side middleware to redirect clients after receiving a failure exception. The latency and timeliness properties of applications can be negatively affected, however, by invoking a remote method on the resource manager to obtain the failover target address after receiving a failure. What is needed therefore are mechanisms that can proactively update the failover targets on the client side.

Solution → Client-side redirectors. FLARe provides fast failover with predictable latencies by proactively updating the failover targets on the client side. It therefore employs a *client-side redirector* in each client process to handle failures transparently to each client object. The client-side redirector comprises a *client request interceptor* for each client object and a *redirection agent* in each client process.

Interceptors are software components that can increase the flexibility of client and server applications by modifying their behavior with little or no change to existing application software [21]. FLARe redirection agent uses a CORBA *client request interceptor* [2] at system initialization time to handle CORBA

COMM_FAILURE exceptions that are raised in response to server or service failures. CORBA in turn relies on the underlying network transport protocol's (*e.g.*, TCP) connection timeout mechanisms to detect server failures. Since TAO supports client/server communications using many different protocols, its failure detection mechanism can be configured to use advanced fault-tolerance protocols, such as SCTP [22].

After catching a failure exception the client request interceptor attempts to redirect the clients to the appropriate failover target, rather than propagating that exception to the client application. As mentioned in the solution to challenge 2 above, the resource manager maintains information about the failover targets for each replica. One way to update the client request interceptor with these failover target decisions would be to establish remote communications between the resource manager and the client request interceptor. As discussed in [2], however, portable interceptors are not remote objects and do not have their own thread of control. No external service or object can thus invoke a remote operation on the client request interceptor (which is a CORBA-based portable interceptor) and the client request interceptor cannot periodically invoke a remote operation on an external object or service.

Moreover, such a remote invocation will increase failover or failure recovery latency. If an appropriate failover target information is available at the client request interceptor even before the failure happens, then client redirection will be predictable, fast, and timely, *i.e.*, failover latency will only depend on the time taken for the clients to receive the COMM_FAILURE exception after a server failure. FLARe's redirection agent is a CORBA object that runs in its own thread within the client process to allow FLARe's resource manager to send object failover information to the client request interceptor.

FLARe's redirection agent communicates with FLARe's resource manager so it is updated with the failover information proactively, *i.e.*, before failures occur. Since it is conceivable that multiple clients may invoke the same server, the resource manager uses real-time publish-subscribe communication to scalably and efficiently disseminate the failover targets to all the concerned redirection agents. After catching an exception, the client request interceptor contacts the redirection agent to obtain the failover object address, and redirects the client to that server object. By proactively selecting failover target updates, FLARe can provide timely and predictable failover.

4 Related Work

Our work on FLARe can be compared with related research along the following dimensions:

Real-time fault-tolerant systems. Delta-4/XPA [18] provided real-time fault-tolerant solutions to distributed systems by using the semi-active replication model. MEAD [17] and its proactive recovery strategy for distributed CORBA applications can minimize the recovery time for DRE systems. The Time-triggered Message-triggered Objects (TMO) project [9] considers replication schemes such

as the primary-shadow TMO replication (PSTR) scheme, for which recovery time bounds can be quantitatively established, and real-time fault tolerance guarantees can be provided to applications. DARX [11] provides adaptive fault-tolerance for multi-agent software platforms by dynamically changing replication styles in response to changing resource availabilities and application performance.

FLARe builds upon and extends this prior work by focusing on a combination of server-side, client-side, and infrastructure-specific middleware components. These together address an important challenge of using PASSIVE replication in fault-tolerant real-time systems: maintaining soft real-time performance after failure recovery.

Scheduling algorithms. Fundamental ideas and challenges in combining real-time and fault tolerance are described in [24], where imprecise computations are used to provide degraded QoS to applications operating in the presence of failures. [6] proposes adaptive fault tolerance mechanisms to choose a suitable redundancy strategy for dynamically arriving aperiodic tasks based on system resource availability. The Realize middleware [8] provides dynamic scheduling techniques that observes the execution times, slack, and resource requirements of applications to dynamically schedule tasks that are recovering from failure, and make sure that non-faulty tasks do not get affected by the recovering tasks.

FLARe differs from these approaches in providing fault tolerance capabilities to soft real-time applications. Rather than ensuring hard deadlines are met in the presence of failures, FLARe focuses on minimizing the impact of failure recovery on client response times and system resource utilization, and also provides timely client failover to appropriate failover targets.

5 Concluding Remarks

The FLARe middleware described in this paper provides both timeliness and availability to distributed real-time and embedded (DRE) systems. FLARe focuses on passive replication to meet the needs of resource-constrained environments. FLARe identifies and provisions those fault-tolerance functionalities, which if not designed properly could also affect the timeliness properties of the applications. To design and implement those functionalities, FLARe overcomes limitations of current middleware approaches, by providing a proactive, adaptive, and resource-aware fault-tolerance solution for clients.

The lessons we learned developing and applying FLARe thus far include:

• Common CORBA features, such as portable interceptors, and POSIX features, such as local sockets, can be leveraged to provide fault tolerance capabilities to soft real-time systems without modifying the implementation of standard-compliant RT-CORBA ORBs. Moreover, well-known architectural and design patterns can be carefully chosen to design key components of a fault-tolerant middleware, so that the fault-tolerance functionalities can be provided in an effective and timely manner.

• FLARe currently does not support stateful applications, so its resource manager uses a failover target selection algorithm that selects failover targets

without considering the consistency levels of the replicas. Supporting stateful applications in DRE systems not only requires timely failover, but also supporting different client consistency requirements, such as weak or strong consistency models. This is part of our future work.

• FLARe is designed for environments where the networks provide bounded communication latencies and have no single point of failure. Certain DRE systems, however, run in environments where networks fail, which can cause resource contention in the remaining links. Our future work is therefore focusing on integrating FLARe with network QoS mechanisms, such as DiffServ and Bandwidth Brokers so that critical communications can use network QoS mechanisms to meet critical QoS requirements.

FLARe is open-source software that can be downloaded from `www.dre.vanderbilt.edu/~jai/FLARe`.

References

1. Ismail Assayad, Alain Girault, and Hamoudi Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *DSN '04*, page 347, Florence, Italy, 2004.
2. Taha Bennani, Laurent Blain, Ludovic Courtes, Jean-Charles Fabre, Marc-Olivier Killijian, Eric Marsden, and Francois Taiani. Implementing Simple Replication Protocols using CORBA Portable Interceptors and Java Serialization. In *DSN' 04*, pages 549–554, Florence, Italy, 2004.
3. Zhongtang Cai, Vibhore Kumar, Brian F. Cooper, Greg Eisenhauer, Karsten Schwan, and Robert E. Strom. Utility-Driven Proactive Management of Availability in Enterprise-Scale Information Flows. In *Proceedings of ACM/Usenix/IFIP Middleware*, pages 382–403, 2006.
4. A. M. Déplanche, P. Y. Théaudière, and Y. Trinquet. Implementing a semi-active replication strategy in chorus/classix, a distributed real-time executive. In *SRDS '99: Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, page 90, Washington, DC, USA, 1999. IEEE Computer Society.
5. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
6. O. Gonzalez, H. Shrikumar, J. A. Stankovic, and K. Ramamritham. Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. In *RTSS '97*, page 79, San Francisco, CA, USA, 1997.
7. E. Douglas Jensen. Distributed Real-time Specification for Java. `java.sun.com/aboutJava/communityprocess/jsr/jsr\_050\_drt.html`, 2000.
8. V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser. Dynamic Scheduling of Distributed Method Invocations. In *21st IEEE Real-time Systems Symposium*, Orlando, FL, November 2000. IEEE.
9. K. H. (Kane) Kim and Chittur Subbaraman. The pstr/sns scheme for real-time fault tolerance via active object replication and network surveillance. *IEEE Trans. on Know. and Data Engg.*, 12(2), 2000.
10. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *RTSS '89: Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, Washington, DC, USA, 1989. IEEE Computer Society.

11. Olivier Marin, Marin Bertier, and Pierre Sens. Darx: A framework for the fault-tolerant support of agent software. In *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering*, page 406, Washington, DC, USA, 2003. IEEE Computer Society.
12. Aad P. A. Van Moorsel. The 'qos query service' for improved quality-of-service decision making in corba. In *SRDS '99*, page 274, Lausanne, Switzerland, 1999.
13. Object Management Group. *Fault Tolerant CORBA, Chapter 23, CORBA v3.0.3*, OMG Document formal/04-03-10 edition, March 2004.
14. Object Management Group. *Real-time CORBA Specification v1.2 (static)*, OMG Document formal/05-01-04 edition, November 2005.
15. Object Management Group. *Lightweight Real-Time Fault Tolerant CORBA DRAFT RFP*, OMG Document realtime/06-06-06 edition, June 2006.
16. Pascal Felber and Priya Narasimhan. Experiences, Approaches and Challenges in building Fault-tolerant CORBA Systems. *Computers, IEEE Transactions on*, 54(5):497–511, May 2004.
17. Soila Pertet and Priya Narasimhan. Proactive recovery in distributed corba applications. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 357, Washington, DC, USA, 2004. IEEE Computer Society.
18. David Powell. Distributed fault tolerance: Lessons from delta-4. *IEEE Micro*, 14(1):36–47, 1994.
19. Francisco Prez-Sorrosal, Marta Patino-Martinez, Ricardo Jimenez-Peris, and Jaksa Vuckovic. Highly available long running transactions and activities for j2ee applications. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 2, Washington, DC, USA, 2006. IEEE Computer Society.
20. Binoy Ravindran, Edward Curley, Jonathan S. Anderson, and E. Douglas Jensen. On best-effort real-time assurances for recovering from distributable thread failures in distributed real-time systems. In *ISORC '07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.
21. Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.
22. Randall Stewart and Qiaobing Xie. *Stream Control Transmission Protocol (SCTP) A Reference Guide*. Addison-Wesley, Boston, 2001.
23. Sun Microsystems. *Java Specification Request, JSR 117, J2EE APIs for Continuous Availability*, JSR 117 edition, April 2001.
24. Fuxing Wang, Krithi Ramamritham, and John A. Stankovic. Determining redundancy levels for fault tolerant real-time systems. *IEEE Transactions on Computers*, 44(2):292–301, 1995.