A Practical Comparison Between the TAO Real-Time Event Service and the Maestro/Ensemble Group Communication System*

Carlo Marchetti[1], Paolo Papa[2], Stefano Cimmino[1], Leonardo Querzoni[1], Roberto Baldoni[1], and Emanuela Barbi[2]

[1] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198, Roma, Italy
{marchet,cimmino,querzoni,baldoni}@dis.uniroma1.it

[2] AMS (Alenia Marconi Systems) S.p.A.
Radar & Technology division
Via Tiburtina km 12,400, Roma, Italy
{ppapa,ebarbi}@amsjv.it

Abstract. In this paper we present the results of a practical experience on the evaluation of two message-passing middleware platforms for developing distributed applications, i.e. the ACE/TAO Real Time Event Channel (RTEC) and the Maestro/Ensemble group communication toolkit (M/E). In particular, we compare their functionalities and their performances in a simple yet meaningful deployment configuration. The functional comparison points out the different characteristics of the two systems. In particular, M/E simplifies the coding of applications with strong requirements in terms of group membership tracking and ordered message delivery guarantees, while RTEC provides users with unreliable message delivery between loosely coupled processes. The performance comparison shows that, under stressing conditions, M/E sacrifices throughput stability for enforcing reliable and ordered message delivery, while RTEC offers a more stable throughput of unordered messages sacrificing message delivery reliability under heavy load. In normal operating conditions, the two systems perform almost similarly.

1 Introduction

The growing interest in open platforms for building mission critical systems currently motivates increasing R&D efforts in the area of middleware platforms [Sch02], as well as on the assessment and evaluation of previous works [CKV01,MCGS03,BCMT02]. In these areas, both group communication and event notification services are nowadays widely recognized as good candidates for implementing systems with stringent non-functional requirements, e.g. fault-tolerance, timeliness, throughput, scalability. For this reason, we are currently evaluating and assessing functionalities and performance of these platforms, in order to implement a Flight Data Processor (FDP) that is one of the

* This work is partially supported by the european project EUPubli.com funded by the European Community and by the italian project MAIS funded by the Italian Ministry of Research

R. Meersman, Z. Tari (Eds.): CoopIS/DOA/ODBASE 2004, LNCS 3291, pp. 1558–1570, 2004.

core components of an air traffic control (ATC) system. At a high abstraction level, FDP has the function of relating radar traces to the flight information stored in a database, and to provide flight controllers with information about the current and expected status of flights. Concerning non-functional requirements, the FDP implementation is expected to be highly available, and to respect stringent timeliness and throughput requirements.

In this context, group communication systems have several success stories showing efficiency and effectiveness of some well-known features [CDD96,Bir99], i.e. the *group membership service*, and the support of several distinct *multicast primitives* with various reliability and message ordering guarantees (e.g. reliable, causal, and atomic) [CKV01], easing the achievement of fault tolerance and high availability.

After being successfully applied in other domains, e.g. stock tracking [Bet00], event notification services [EFGK03,RK98] are gaining popularity in ATC environments [Obj00,LBB99,LMB00]. Indeed, allowing the coding of applications that react to incoming events, and favoring scalability by sharply decoupling senders from receivers, event-based systems seem well suited for distributed systems with no central control, and to implement ATC applications that must monitor and react to changes, e.g. in the environment or in process status.

In this paper we report the results of our experiences on the practical evaluation of two distinguished representatives of group communication systems (Maestro/Ensemble group communication toolkit (M/E) [Vay98,VB98,Hay98]) and event-based systems (ACE/TAO Real Time Event Channel (RTEC) [HLS97,SLM98,Obj03]). We compare these systems under both the functionalities they provide[1] and the achievable performance in a simple yet meaningful configuration. Concerning functionalities, M/E confirms success stories on group communication systems by simplifying the development of fault-tolerant services based on the process group approach and virtual synchrony [GS97,Bir99], but it also suffers from the tight coupling between senders and receivers imposed by the process group approach and from the absence of advanced filtering mechanisms provided by event-based systems. In contrast, RTEC confirms the advantages of event-based systems, i.e. strong decoupling of senders and receivers, sophisticated event filtering and correlation mechanisms, and the possibility of associating timeouts and priorities to events, but it suffers from the lack of mechanisms for reliable and ordered message delivery.

Concerning performance, we present experimental results on average message latency and its standard deviation evaluated under several distinct load conditions in a small-scale system. Results show that both systems under normal operating conditions offer quite reliable message delivery with predictable latency. In contrast, under stressing load conditions, RTEC can loose messages, while M/E introduces significant and highly unpredictable delays in order to continue guaranteeing reliable message delivery. We conclude elaborating on these results, outlining some desired functionalities not supported by current event-based systems, which could be implemented using group communications or their internal mechanisms.

[1] Let us note that a former relevant debate started about ten years ago, i.e. the so-called CATOCS controversy [CS93,Bir94], resulted in pointing out the main differences between the large classes of group communication and event systems. This paper focuses instead on two specific systems, which results in a more granular - even if less general - analysis.

The remainder of this paper is organized as follows: Sections 2.1 and 2.2 summarize the main features of RTEC and M/E, respectively; Section 3 illustrates the functional comparison; Section 4 deals with the performance comparison; finally, Section 5 presents our concluding remarks.

2 Overview of RTEC and M/E

In this section we outline the main features of the two analyzed platforms. Interested readers can find further details in [HLS97,SLM98,Obj03] about RTEC, and in [Vay98, VB98,Hay98] about M/E.

2.1 The TAO Real-Time Event Channel (RTEC)

The ACE ORB (TAO) is a freely available, open-source implementation of CORBA. The CORBA Event Service [Obj01] component was defined to offer developers a basic event service, other than the simple CORBA RPC-like invocation mechanism. The service's programming model consists of (i) *suppliers*, i.e. senders of events, (ii) *consumers*, i.e. receivers of events and (iii) *event channels* (EC), that allow consumers to receive events sent on the channel by suppliers. Consumers and suppliers must register with event channels in order to receive/send events. Event channels are standard CORBA objects and they can be found in a way similar to any CORBA object, so that a global registration service is not required. The registration of a new consumer or supplier is transparent to existing ones and thus it doesn't affect the overall computation. RTEC extends the CORBA Event Service to satisfy the quality of service (QoS) needs of real-time applications in many domains, such as avionics, telecommunications and process control. The main advanced features provided by RTEC are (i) support for centralized event filtering and correlation, enabling consumers to identify a desired subset of events by specifying logical OR and AND event dependencies; (ii) efficient and predictable event dispatching, obtained through application-specified number and priority of threads responsible for dispatching events; (iii) scalability achieved through the efficient use of network and computational resources, e.g. using multicast protocols and building collections of channels that share filtering information (multiple EC can join into a *federation*, and the resulting federated EC acts as one logical EC). Concerning communications, RTEC channels can use IIOP, UDP, and IP multicast. In this work we consider suppliers and consumers running over a UDP-based Event Channel Federation, as suggested in [Obj03] for implementing scalable and efficient systems (especially if compared with IIOP-based configurations requiring at least a TCP connection between each pair of channels). Let us note that in UDP and IP-multicast based configurations, RTEC is admitted to lose messages due to the unreliability of the underlying network protocol. In this case, consumers are not notified of lost messages, unless implementing application-level acknowledgement mechanism.

2.2 Maestro/Ensemble (M/E)

M/E is a middleware platform enabling distributed system developers to use group communication abstractions in an object-oriented manner. The core of the system is Ensemble

[Hay98], which is a flexible and efficient group toolkit, offering typical group communication services, e.g. a group membership service and several multicast primitives with different semantics.

The basic notion underlying group communications is the *process group*, i.e. a set of processes, called *members*, which communicate and coordinate to provide a service. Groups are *dynamic*, i.e. processes are allowed to join and voluntarily leave a group using appropriate primitives. Furthermore, faulty processes are excluded from groups after crashing. The *group membership service* provides each process of a group with a consistent *view* v_i composed by the identifiers of all non-crashed processes currently belonging to the group. Upon a membership change, processes agree on a new view through a *view change protocol*. At the end of this protocol, group members are provided with a view v_{i+1} that (i) is delivered to all the members of v_{i+1} through a view change event, and (ii) contains the identifier of all the members that deliver v_{i+1}. Concerning processes joining a group, M/E provides a built-in state transfer mechanisms which allows automatic alignment of new members with the computation of older ones when needed. However, since the join or leave of a member leads to run the membership change protocol, these kind of events slow down the computation of all group members.

Communications among processes of a group is provided with different reliability and ordering guarantees, including fifo, causal and total order (or atomic) multicast. In this paper we focus on the total order multicast primitive, which intuitively ensures that exchanged messages are delivered in the same order by participating processes.

Ensemble provides fine-grained control over its functionalities, which can be selected simply layering several *micro-protocols*, i.e. well-defined stackable components, each implementing a simple and specific function. The group membership and total order multicast services for instance can be enabled by including the corresponding micro-protocols into the stack. The Maestro toolkit lays on top of Ensemble services to provide developers with an advanced object-oriented framework comprising an open and extensible hierarchy of classes deemed useful to build complex distributed applications.

3 Functional Comparison

Even if different in nature and originally designed at different times for different classes of systems, RTEC and M/E can be compared by abstracting out their peculiarities. From a distributed systems point of view, both RTEC and M/E can be seen as message passing systems mainly differing in the following features: (i) message addressing scheme, (ii) communication reliability, (iii) message filtering, (iv) ordering, (v) real-time guarantees, and (vi) fault-tolerance support.

Addressing scheme. One of the main advantages of event-based systems is their message addressing scheme, enabling processes to send information over the wire without caring about the recipients that receive messages according to their preferences. Furthermore, recipients of messages are typically not aware of senders. Therefore, senders and receivers in event-based systems are *loosely coupled*. RTEC inherits this advantage from the CORBA Event Service, in which suppliers put events on channels and consumers simply bind to channels in order to receive anonymous events. An event channel is an intervening object that allows multiple suppliers to communicate with multiple consumers

asynchronously. In contrast, in M/E processes are *tightly coupled*, as each process has to become a member of the group before initiating the communication with other members. In particular, a process joining a group obtains the full list of active members by delivering the view change event, and explicitly specifies that set of recipients of the messages it sends. Processes can send messages to a single member of the group (in case of a point-to-point communication) or to the entire group (in case of a broadcast communication). Finally, receivers are aware of senders, as each sent message is labelled with the sender's identifier. Tightly coupling favors the possibility to introduce properties into the system, e.g. ordered communications, but reduces to some extent scalability. In fact, the size of the view grows linearly with the number of participating processes. Furthermore, a process joining or leaving the group forces the group to run the membership algorithm, which slows down the overall computation (leading the system to block in the worst case [CHTCB96]). In contrast, having loosely coupled processes favors scalability, as (i) processes do not have to store information about other participants, and (ii) the join and leave of a process are transparent to other participants.

Communication reliability. RTEC, as well as the CORBA Event Service, provides no guarantee of reliable message delivery[2]. This is confirmed by the experiments described in Section 4 that use UDP as underlying transport protocol. A valuable alternative could be the use of IIOP based on TCP and thus providing reliable message delivery, at the price of significantly reducing the service scalability. In contrast, even if based on UDP, M/E is able to provide reliable message delivery. Reliability is achieved by including into the Ensemble's stack a specific set of micro-protocols, which implement well-suited retransmission strategies to ensure reliable message delivery even in the presence of process crashes. Retransmission are usually requested through an efficient NAK-based mechanism. It is worth noting that M/E is able to provide a very strong reliability property, guaranteeing that all members of a view v that neither crash nor leave the group deliver the same set of messages in v. This property, called *atomicity or agreement*, obviously comes at the cost of a performance overhead, as shown in Section 4. It is also important to note that this property would not be achieved by RTEC even using IIOP as underlying transport protocol.

Message filtering. Standard COS Event Channels can be chained together to create an event filtering graph, but traversing such a graph usually introduces an unacceptable overhead for RT applications. RTEC extends the CORBA Event Service specification providing a centralized feature of event filtering with a correlation mechanism that allows consumers to specify logical OR and AND event dependencies in order to obtain flexible filtering. Events can be filtered based on the event type and source id, and can be correlated in a group so that the delivery of some events can be delayed until all events in the group are available. On the contrary, M/E does not provide any type of filtering: once a process has joined a group, it will receive all messages sent by any other group member. Message filtering can be added either implementing a specific application level filtering mechanism, or designing a specific micro-protocol to be included into Ensemble's stack.

[2] The CORBA *Notification* Service augments the Event Service taking into account reliable message delivery [RTD01,Obj02].

Note that in such cases, messages would be filtered out by destinations. In contrast, RTEC is able to filter events at the channel level, thus reducing traffic load and relieving consumers from implementing filtering mechanisms.

Message ordering. RTEC provides no means to implement ordered message delivery. Note that even if some ordering mechanism could be implemented at the application level, e.g. exploiting filters, the lack of reliable message delivery could result in distinct consumers receiving distinct sets of ordered events. In contrast, M/E implements several multicast primitives with different ordering guarantees, i.e. FIFO, causal and total order multicast. In particular, total order multicast [BCM04,DSU03] is a communication abstraction deemed useful for several application scenarios, e.g. active software replication [Sch93]. M/E achieves this primitive by enhancing the reliability properties of the multicast service with an ordering property, provided by a specific micro-protocol. In particular, M/E's total order multicast guarantees that correct processes (i.e. processes that do not crash) deliver the same ordered sequence of messages. In contrast, faulty processes are allowed to exhibit a wider set of behaviors, e.g. to deliver messages in different orders. Total order multicast incurs a performance overhead which is due to the mechanisms enforcing reliable message delivery and to the necessary synchronization among members to achieve total order.

Real time guarantees. RTEC extends the standard CORBA Event Service interfaces by allowing consumers and suppliers to specify their execution requirements and characteristics using QoS parameters such as worst-case execution time, rate, etc. These parameters are then used by the channel's dispatching mechanism to integrate with the system-wide real-time scheduling policy to determine event dispatch ordering and preemption strategies. Moreover RTEC allows consumers to specify event dependency timeouts. These timeouts are used by the service to propagate temporal events in coordination with system scheduling policies. On the contrary, M/E is not concerned with real time guarantees. Members of a group have no guarantees about message transmission and/or processing delays, and they are not able to specify real-time related requirements. In fact, the experiments of Section 4 show that M/E sacrifices throughput stability and message latency to guarantee the reliable message delivery. Despite the extensibility of M/E, it is not possible to provide real time guarantees simply by adding a few micro-protocols. Indeed, this would require adopting specific design principles and mechanisms which would drive to a complete re-design of M/E into a new system.

Fault-tolerance. RTEC is not concerned with fault-tolerance. TAO partially supports the ORB level requirements to achieve Fault Tolerance for CORBA Objects (including the Event Service). The implementation of the full set of requirements mandated by the FT-CORBA specification is under development. As a consequence, failures in RTEC are not automatically handled by the infrastructure. Concerning M/E, it was primarily conceived to support the development of dependable applications. As such, it offers a complete set of mechanisms and protocols to nicely handle failures. Several micro-protocols are used to achieve fault tolerance. First, a micro-protocol provides suspicions of members' failures. The group membership protocol, exploiting these suspicions, is then able to provide each member with an updated and consistent view of the group.

Furthermore, M/E is able to provide the well-known *virtual synchrony* property, which guarantees that any two processes belonging to view v_i and installing view v_{i+1} deliver the same set of messages in v_i. This is an important property as it greatly simplifies the design and development of distributed fault-tolerant applications. Moreover M/E provides a built-in state transfer mechanisms which allows automatic alignment of new members joining the group.

Table 1. Functional comparison summary

Feature	RTEC	M/E
Addressing scheme	loosely coupled	tightly coupled
Communication reliability	unreliable	reliable
Message filtering	complex filters	no filtering
Message ordering	unordered	total order
Real-time guarantees	supported	not supported
Fault-tolerance	not supported	supported

Table 1 summarizes the previous discussion. The following section presents performance analysis that integrates the comparison.

4 Performance Comparison

As aforementioned, the FDP system has stringent timeliness and high-availability constraints, and the number of hosts on which FDP will run mainly depends on the latter requirement, i.e. scalability is not a main concern. In contrast, the FDP is expected to run over a network whose load is not a-priori known[3]. Therefore, to compare how the two platforms perform, we analyzed how they behave with respect to message latency in a small-scale setting, but under a wide range of load conditions.

4.1 Experimental Setting

Testbed platform. The testbed platform consists of two 1,2Ghz Pentium IV hosts equipped with 1Gb of RAM, interconnected via a private Fast Ethernet (100 Mbps) hub and running Linux RedHat 7.2 as operative system. On both hosts, we installed and configured TAO 1.3 and Maestro/Ensemble 1.40. No other program was running on the hosts during the experiments.

Testbed applications. We coded two simple test applications (one for each system) that run 1000 times the following basic experiment (see Figure 1): every D msec a process p_a on host A sends a timestamped message m with a payload of S bytes to a process p_b on host B (step 1). Upon receiving m from the network (step 2), p_b immediately

[3] The network load is mainly due to the number of radars, of flights, and of flight control workstations, which could vary from a few units to hundreds.

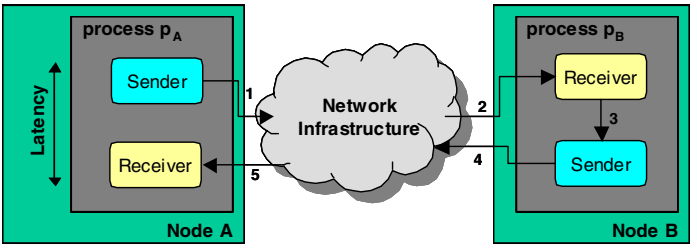


Fig. 1. Test-bed scenario

sends m back to p_a (step 4). Upon receiving m (step 5), p_a evaluates the time T elapsed from the first sending of m and stores $T/2$ for further analysis[4], as a likely estimation of message latency. In the RTEC test application, each process embeds both a consumer and a supplier bound to a single event channel, while in the M/E application both processes join the same group and use the Ensemble's total order multicast primitive (based on a fixed sequencer ordering protocol [DSU03,BCM04]) to exchange messages. Each application finally returns data of the overall experiment, i.e. the estimated average message latency (aL) and its standard deviation ($stdL$), as well as the 1000 estimated values of message latencies. For both platforms, we let vary D (message sending rate expressed in milliseconds) and S (message size expressed in bytes) in the sets reported in Table 2.

Table 2. Ranges of experimental parameters

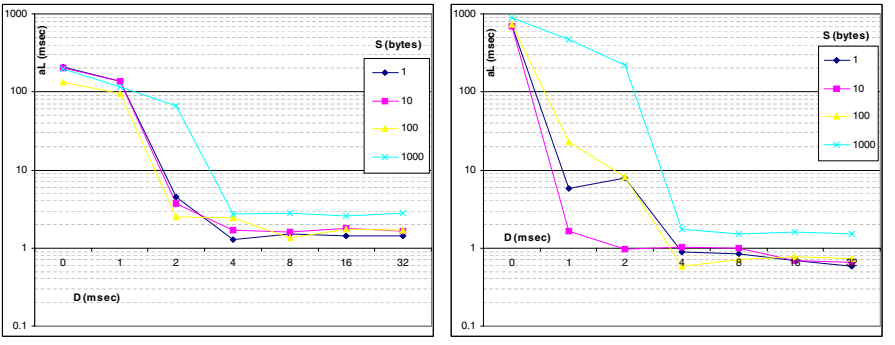
Parameter	Values
$D(msec)$	$\{0, 1, 2, 4, 8, 16, 32\}$
$S(bytes)$	$\{0, 10, 100, 1000\}$

For each pair of values of D and S, each application was run 10 times. Each value plotted in the following figures is thus evaluated on the basis of a batch of 10.000 basic experiments.

4.2 Experimental Results

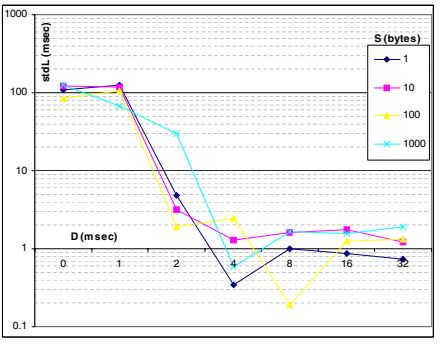
Figure 2 plots the estimated average latency aL and standard deviation $stdL$ as functions of S and D for both RTEC and M/E. It is first possible to observe that not surprisingly both systems suffer from heavy load conditions determined by low values of D and high values of S. However, the message sending rate D has higher impact on performance than

[4] To take measurements, we used the high-resolution timer available on Pentium machines (with resolution $\frac{1}{1396}\mu sec$).

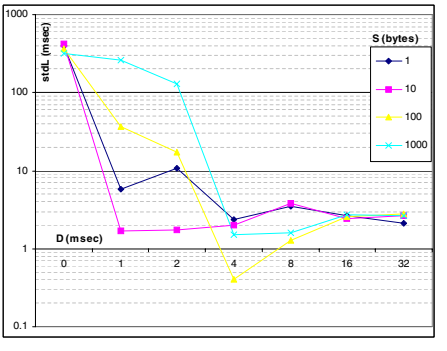


(a) RTEC average latency

(b) M/E average latency



(c) RTEC latency standard deviation



(d) M/E latency standard deviation

Fig. 2. Average latency (aL) and standard deviation ($stdL$) of RTEC and M/E as functions of S and D

message size S, at least for values of S and D ranging in those reported in Table 2. Indeed, when the message sending rate exceeds 4ms both systems reach a quite stable throughput state in which both aL and $stdL$ fall under 3ms. In contrast, if $D \in \{0, 1, 2\}$ms, aL and $stdL$ grow exponentially reaching values of tens or even hundreds of milliseconds, depending on S. In order to fairly compare the platforms, it is important to complement information on latency under stressing conditions with the rate of message loss exhibited by RTEC. Indeed, even if M/E shows higher aL and $stdL$ values when D falls within 4ms, it also continues to deliver the whole set of sent messages according to its reliable delivery guarantees. In contrast, RTEC suffers from message losses. Table 3 shows the percentage $e\%$ of events correctly received at step 5 of Figure 1 for each event sent at step 1 (see Section 4.1), as a function of S and D. Note that (i) reductions of $e\%$ can be

due to the loss of some event sent at step 1 *as well as* of some event sent at step 4, and (ii) no loss was observed for each value of S when D exceeded 4ms.

Table 3. Percentage of events correctly received ($e\%$) in RTEC experiments as a function of S and D

D	S=0	S=10	S=100	S=1000
0	89.23%	88.9%	59.6%	15.77%
1	91.81%	90.77%	70.32%	37.79%
2	100%	100%	99.25%	82.47%
4	100%	100%	99.99%	100%

Admitting message loss allows RTEC to outperform M/E in heavy load conditions. Indeed, under these conditions, the average latency of messages delivered by RTEC is lower than the latency of M/E, that stores and keeps retransmitting messages at the sender in order to ensure reliability. In other words, M/E sacrifices latency for implementing reliable message delivery.

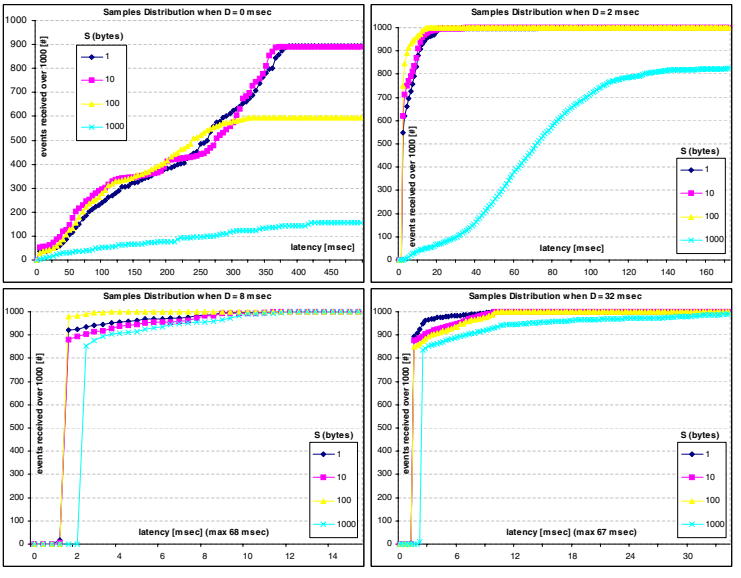
This behaviour appears clear in figures 3(a) and 3(b) where the detailed distributions of message latencies as a function of D and S for both RTEC and M/E are shown. Note that we omit to report distributions for $D = 4ms$ as they are very similar to the obtained setting $D = 8ms$.

Each distribution plots on the Y axis the number of messages having an estimated latency falling in the range plotted on the X axis. Note that the ranges of X and Y axis vary from distribution to distribution in order to show differences.

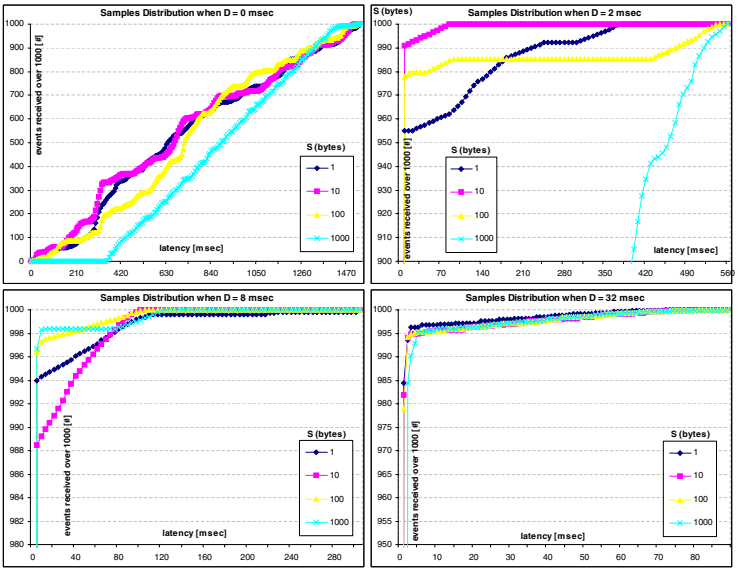
Figure 3(a) shows how RTEC looses messages under heavy load conditions in order to maintain good performance, while M/E always delivers all messages at the cost of an higher latency.

5 Concluding Remarks

In this paper we reported the results of a practical experience on the comparison of two systems deemed suitable for developing a Flight Data Processor, i.e. the TAO Real-time Event Channel (RTEC) and the Maestro/Ensemble toolkit (M/E). We remark that a former scientific debate, namely the CATOCS controversy [CS93,Bir94], emphasized benefits and limitations of event-based and group communication systems at the beginning of research on these platforms. This work reinforces these results by shedding some light on the specific differences and similarities, even in performance terms, of two representative systems belonging to these classes. Let us also note that the results of the comparison can be regarded as a starting point for future research activities. In particular, from Table 1, it follows that it could be interesting to investigate (i) how to extend a group communication system, e.g. M/E, with advanced message filtering and correlation mechanisms, as well as with weaker addressing schemes, and (ii) how to extend an event-based system, e.g. RTEC, with powerful communication semantics as



(a) RTEC distributions



(b) M/E distributions

Fig. 3. Latency distributions of RTEC and M/E. Please note that latency values on x axis vary among graphs

reliable and totally ordered message delivery without significantly impacting on performance. Furthermore, it could be interesting to analyze how fault-tolerance and real-time support could be integrated into a system. Investigating these issues could take to the design of a platform collecting all the benefits achievable among the analyzed systems.

References

[BCM04] R. Baldoni, S. Cimmino, and C. Marchetti. Total order communications over asynchronous distributed systems: Specifications and implementations. Technical Report 06/04, Università di Roma "La sapienza", January 2004.

[BCMT02] R. Baldoni, S. Cimmino, C. Marchetti, and A. Termini. Performance Analisys of Java Group Toolkits: a Case Study. In *Proc. of the International Workshop on scientiFic engIneering of Distributed Java applIcations (FIDJI'2002)*, pages 49–60, Luxembourg, November 2002.

[Bet00] Katherine Betz. A scalable stock web service. In *Proceedings of the 2000 International Conference on Parallel Processing, Workshop on Scalable Web Services*, pages 145–150, Toronto, Canada, 2000. IEEE Computer Society.

[Bir94] K. P. Birman. A response to Cheriton and Skeen's criticism of causal and totally ordered communication. *SIGOPS Oper. Syst. Rev.*, 28(1):11–21, 1994.

[Bir99] K. P. Birman. A Review of Experiences with Reliable Multicast. *Software – Practice and Experience*, 29(9):741–774, 1999.

[CDD96] F. Cristian, B. Dancey, and J. Dehn. Fault-tolerance in air traffic control systems. *ACM Trans. Comput. Syst.*, 14(3):265–286, 1996.

[CHTCB96] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the Impossibility of Group Membership. In *Proc. of the 15th ACM Symposium of Principles of Distributed Computing*, 1996.

[CKV01] G. V. Chokler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[CS93] D. R. Cheriton and D. Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 44–57. ACM Press, 1993.

[DSU03] X. Dèfago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. Technical Report IC/2003/56, École Polytechnique Fédérale de Lausanne, Switzerland, September 2003.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[GS97] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer - Special Issue on Fault Tolerance*, 30:68–74, April 1997.

[Hay98] M. Hayden. The Ensemble system - PhD theses. Technical Report Technical Report TR98-1662, Dept. of Computer Science, Cornell University, Ithaca (NY), 1998.

[HLS97] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt. The design and performance of a real-time CORBA event service. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 184–200. ACM Press, 1997.

[LBB99] C. Liebig, B. Boesling, and A. Buchmann. A notification service for next-generation it systems in air traffic control. In *GI-Workshop: Multicast-Protokolle und Anwendungen*, Braunschweig, Germany, May 1999.

[LMB00] C. Liebig, M. Malva, and A. Buchmann. X^2TS: Unbundling active object systems. In J. Sventek and G. Coulson, editors, *Middleware 2000, IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, volume 1795 of *LNCS*. Springer-Verlag, April 2000.

[MCGS03] S. Mena, X. Cuvellier, C. Grégoire, and A. Schiper. Appia *vs.* Cactus: Comparing protocol composition frameworks. In *22nd Symposium on Reliable Distributed Systems. Florence, Italy*, October 2003.

[Obj00] Object Management Group (OMG). Air Traffic Control, version 1.0. OMG Domain Specification - document formal/00-05-01, 2000.

[Obj01] Object Management Group (OMG). Event Service Specification, version 1.1. OMG Adopted Specification - document formal/01-03-01, 2001.

[Obj02] Object Management Group (OMG). Notification Service Specification, version 1.0.1. OMG Adopted Specification - document formal/02-08-04, 2002.

[Obj03] Object Computing, Inc. *TAO Developer's Guide - Building a standard in performance - version 1.2a*, volume 2. 2003.

[RK98] Adam Rifkin and Rohit Khare. The evolution of internet-scale event notification services: Past, present, and future. Draft to be published - http://www.ics.uci.edu/ rohit/wacc, 1998.

[RTD01] S. Ramani, K. S. Trivedi, and B. Dasarathy. Reliable Messaging Using the CORBA Notification Service. In *Proc. of the Third International Symposium on Distributed Objects and Applications (DOA'01)*, pages 229–238, September 2001.

[Sch93] Fred B. Schneider. Replication Management Using the State Machine Approach. In S. Mullender, editor, *Distributed Systems*. ACM Press - Addison Wesley, 1993.

[Sch02] D. C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(16):43–48, June 2002.

[SLM98] D. C. Schmidt, D. L. Levine, and S. Mungee. The design and performance of real-time object request brokers. *Computer Communications*, 21:294–324, April 1998.

[Vay98] A. Vaysburd. Building reliable interoperable distributed objects with the Maestro tools - PhD theses. Technical Report Technical Report TR98-1678, Dept. of Computer Science, Cornell University, Ithaca (NY), 1998.

[VB98] A. Vaysburd and K. Birman. The Maestro Approach to Building Reliable Interoperable Distributed Applications with Multiple Execution Styles. *Theory and Practice of Object Systems*, 4(2), 1998.