REAL-TIME DATA DISTRIBUTION

Angela Frolov
University of Rhode Island, nglfrolov@yahoo.com

REAL-TIME DATA DISTRIBUTION

BY

ANGELA FROLOV

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

APPLIED MATHEMATICAL SCIENCE

UNIVERSITY OF RHODE ISLAND

2014

DOCTOR OF PHILOSOPHY DISSERTATION

OF

ANGELA FROLOV

APPROVED:

Thesis Committee:

Major Professor	Lisa DiPippo
	Victor Fay-Wolfe
	Qing Yang
	James Baglama
	Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2014

ABSTRACT

Many Cyber-Physical Systems (CPS), distributed embedded real-time (DRE) applications like military command and control, time critical planning collaboration, and wireless embedded sensor networks, require shared data among various components of the system to be available within stringent deadlines for processing and for making critical decisions on time. In order for these decisions to be correct in accordance with the current situation, the data received and processed must be valid. These applications need a data distribution mechanism that can deliver valid data in a specified time. The goal of this work was to develop such a mechanism. We approached it in the following way. First, since a better understanding of the problems involved in real-time data distribution leads to a better solution, we, by grouping characteristics of different systems that require real-time data distribution, defined the data distribution problem space taxonomy. Then, we targeted specific subspaces (static and dynamic systems) in the real-time data distribution problem space and worked on our solutions for them. The solutions we provided include a theoretical base, data models and algorithms for computation of distribution deadlines to ensure data validity in both static and dynamic environment, and the actual data delivery mechanism Timely Data Distribution Service (TDDS).

ACKNOWLEDGMENTS

First, I wish to thank my two academic advisors, who influenced this work the most. My deepest thank you goes to my major professor Dr. Lisa DiPippo for her tireless guidance and support from the very beginning of this endeavor through the very end, and to Dr. Victor Fay-Wolfe, for all his valuable contributions, advices and comments throughout my research.

Second, I would like to express my gratitude to my former fellow student, presently Dr. Kevin Bryan for his useful suggestions during numerous research meetings in the early stages, and for his help and technical support on the final stage of this project, which made finishing this work possible.

I also wish to thank all former members of RT DOC research group who have influenced this work in so many different ways, and made my study and research here a joyful time. A special thank you goes to Mr. Jie Mao, who worked along with me on his part of this project's implementation.

I would like to thank all faculty and staff of the Computer Science Department for making it a wonderful place to study and research, with special thanks to Ms. Lorraine Berube, always amiable, helping and willing to help.

And the last but not the least, I want to express my everlasting gratitude to all my family for their love, support, and understanding.

Angela Frolov

University of Rhode Island

March 2014

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Many Cyber–Physical Systems (CPS), distributed embedded real-time (DRE) applications like military command and control, time critical planning collaboration, and wireless embedded sensor networks, require shared data among various components of the system. Further, these systems might require that the data be available within stringent deadlines for processing and for making critical decisions on time. In order for these decisions to be in accordance with the current situation and correct, the data received and processed must be valid, or *temporally consistent*, that is, data must be no older than a specified age. There is a need for a mechanism that will distribute valid data in a specified time.

One simple solution to achieve this would be to provide client-server or point-to-point communication to deliver the data within the real-time system. However, this type of communication may become extremely complex and inflexible if there are multiple components requiring the same data at different rates. A more efficient and flexible solution would be decoupled, in which the providers of data do not directly communicate with data consumers. This allows the data providers to produce data at a rate consistent with data production, and allows the consumers to receive data at a rate consistent with application needs.

The challenge of this solution is to provide a mechanism that will synthesize the provisions of the provider with the needs of consumers, so that data arrives at each consumer in time and is temporally valid. The situation becomes even more challenging when the distributed system that requires data sharing is dynamic in its nature, that is, data producers and data consumers may come into and leave the system. In this case the solution mechanism must have the ability to adjust the system based on new requirements.

Before proceeding any further we would like to provide some basic definitions.

Real-Time Data Distribution is the transfer of data from one source to one or more destinations within a deterministic timeframe, regardless of the method and the timescale.

Data temporal consistency is defined by a mean of a certain permissible interval of time, regardless of a time scale within which the data is considered to be valid.

1.2 Research Goals

The goals of this work are to provide solutions for specific subspaces in the real-time data distribution problem space (we target static and dynamic systems). These solutions should include algorithms for computation of distribution deadline to ensure data consistency, and the actual data delivery mechanism (Timely Data Distribution Service).

1.3 Our Approach

Since a better understanding of the problems involved in real-time data distribution leads to better solutions, we started our work with the attempt to define the real-time data distribution problem space. By grouping characteristics of the different systems that require real-time data distribution, we defined the taxonomy of a data distribution problem space. Then we worked on a solution to the data distribution problem in static real-time systems. This solution includes an algorithm that determines data distribution scheduling parameters, an implementation that uses a real-time event service to deliver the data, and a real-time scheduling service to ensure that data is delivered on time. We worked next on a solution to data distribution problems in dynamic real-time systems. This includes an algorithm for calculation of scheduling parameters and transition-implementation that supports proper data delivery from data providers to data recipients.

1.4 Dissertation Outline

The remainder of this thesis is organized as follows. Chapter 2 gives a background on techniques and tools involved in the project. It also provides a summary of current work related to the area of data distribution. Chapter 3 presents the Real-Time Data Distribution (RTDD) problem space, highlights the solution space provided by this work, and describes the (RTDD) model, algorithms and theorems. Chapter 4 discusses Static RTDD, including system design, implementation and evaluation. Chapter 5 deals with Dynamic RTDD, its design, implementation and evaluation. Chapter 6 concludes this thesis with summary of contributions, comparisons with related work, limitations of our work, and possible future directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

This section describes our architecture and some components within the architecture that were used to build our system. It also presents a summary of related work.

Since our system architecture is build upon TAO ORB, an open source middleware based on OMG RT CORBA standard, and we use several CORBA services: RT Event Service, Naming Service and Scheduling service, we start with providing background on these components.

2.1 CORBA

The Common Object Request Broker Architecture (CORBA), developed by The Object Management Group (OMG) is a standard of object-oriented middleware for distributed systems [1]. The goal of this middleware is to facilitate seamless client/server interactions in a distributed system.

CORBA is designed to allow a programmer to construct object-oriented programs without regard to traditional object boundaries such as address spaces or location of the object in a distributed system. This means, that a client program should be able to invoke a method on a server object whether the object is in the client's address space or located on a remote node in a distributed system. The CORBA standard defines a framework to allow communication among applications in a distributed system regardless of platform or programming language differences.

Figure 1 presents the highest level of CORBA specification, which is referred to as the object management architecture and consists of four major components:

- Object Request Broker (ORB) is the middleware that routes requests among all other architectural components. This is the foundation for building applications from distributed objects in homo-and heterogeneous environments.
- CORBA Services provide some basic system level services such as Naming, Persistence, Event Notification, etc.

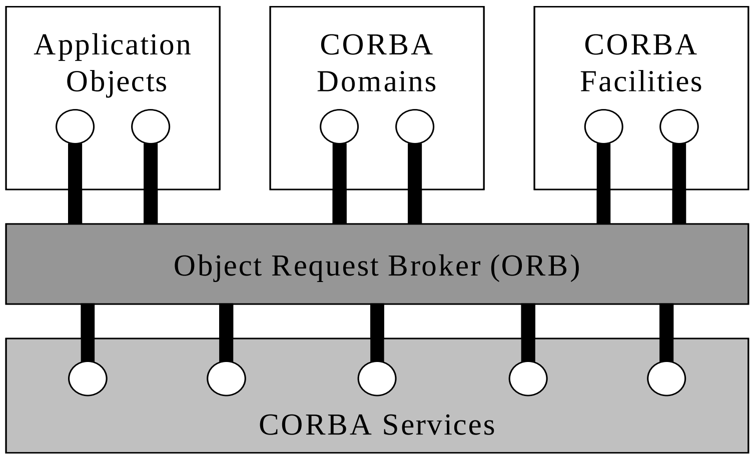


Figure 1. CORBA Architecture

- CORBA Facilities consist of a set of higher–level functions to cover a wide range of generically applicable facilities in areas such as information management and user interface.

- CORBA Domains consists of objects specific to particular application domains. They include financial services, healthcare, manufacturing, telecommunications and business objects.
- Application Objects are the objects (clients and services) created by system implementers to provide tailored business capabilities.

The CORBA specification also includes the Interface Definition Language (IDL), which is the key component to integration of application objects. By providing the standard object interfaces among all applications and data within the CORBA environment, IDL makes communication between application objects independent of their physical locations, platform type, networking protocol, and programming languages.

CORBA's theoretical background is based on three major concepts: an object-oriented model, open distributed computer environments, and component integration and reuse. The latter is achieved through CORBA's uniform access to services, uniform discovery of resources and object names, uniform error handling methods and uniform security policies.

CORBA is one of the major technologies in the field of distributed object management (DOM), in which components grow and specifications are adopted according to emerging needs of the applications involved. To address the needs of broad real-time applications, OMG Real-Time Special Interest Group (RT SIG) defined the standards for the Real-Time CORBA (RT CORBA) [2]. To provide the special capabilities to special applications without restricting non real-time development RT CORBA is positioned as a separate extension to CORBA 2.2 and constitutes an optional, additional compliance point.

2.1.1 Real-Time CORBA

The goal of RT CORBA [2] is to provide a standard for CORBA ORB to deal with expression and enforcement of real-time constraints on executions to support end-to end predictability in a system. RT CORBA consists of the following four major components:

1) The scheduling mechanism in the operating system (OS), which is used to schedule end-to end application activities (to provide a means for programming such activity the term *distributable thread* is used). The RT CORBA specification focuses on OS's that allow applications to specify scheduling priorities and policies. For example, an OS that implements the IEEE POSIX 1003.1-1996 Real-Time Extension has the necessary features to support end-to-end predictability;

2) The real-time ORB provides standard interfaces for allowing RT applications to specify their resource requirements to the ORB and based on that manages end-system and communication resources. It also preserves efficient scalable and predictable end-to-end behavior of high-level services and application components. For example, a global scheduling service which can be used for scheduling and managing of distributed resources;

3) The communication transport, which includes policies and mechanisms to support resource guarantees;

4) The application(s).

To achieve end-to-end predictability, RT CORBA defines standard interfaces and Quality of Service (QoS) policies to allow applications to configure and control all kinds of resources in the system. So for example, the processor resources can be controlled via thread pools, priority mechanisms, intra-process mutexes, and global scheduling service. The communication resources can be controlled via protocol properties and explicit bindings, and the memory resources can be controlled via buffering requests in queues and bounding the size of a thread pool.

Since strict control over scheduling and using of resources is essential for many RT systems, RT CORBA enables client and server applications to determine at which priority a CORBA invocation will be processed, allows servers to predefine the pools of threads and bounds the priority of ORB threads.

While all the above describes the RT CORBA Based Architecture, which ‘covers’ a wide range of fixed priority systems (static systems), the Dynamic Scheduling specification (RTC1.2) generalizes it to meet the requirements of a much greater segment of the real-time computing field. The three major generalizations are: any scheduling discipline may be employed; the scheduling parameter elements associated with the chosen discipline may be changed at any time during execution; and the schedulable entity is a *distributable thread* that may span node boundaries carrying its scheduling context among instances on these nodes.

2.1.2 The ACE ORB (TAO)

The ACE ORB (TAO) is a high quality, freely available, open-source OMG standard-based CORBA middleware platform that was developed by the Distributed

Object Computing DOC group at Washington University in St.Louis [3] to provide an effective instrument for a wide community of researchers and developers. Our RTDOC research group has chosen TAO as the underlying RT CORBA middleware platform.

2.2 Dynamic Scheduling Service

While RT CORBA 1.2 provides a flexible means for expressing and propagating scheduling information across node boundaries in a distributed system, all of its scheduling decisions are assumed to be local. Each endsystem local scheduler uses the same propagated scheduling information to make local scheduling decisions, and they do not have a global view of the overall system. The Real-Time Distributed Scheduling Service (RT DSS) [4] research project in URI RT DOC group attempted to overcome this issue by providing globally sound end-to-end scheduling and overload management using the local enforcement capabilities of the local endsystem.

The RT DSS architecture is presented in Figure 2. It consists of six independent and coordinated components:

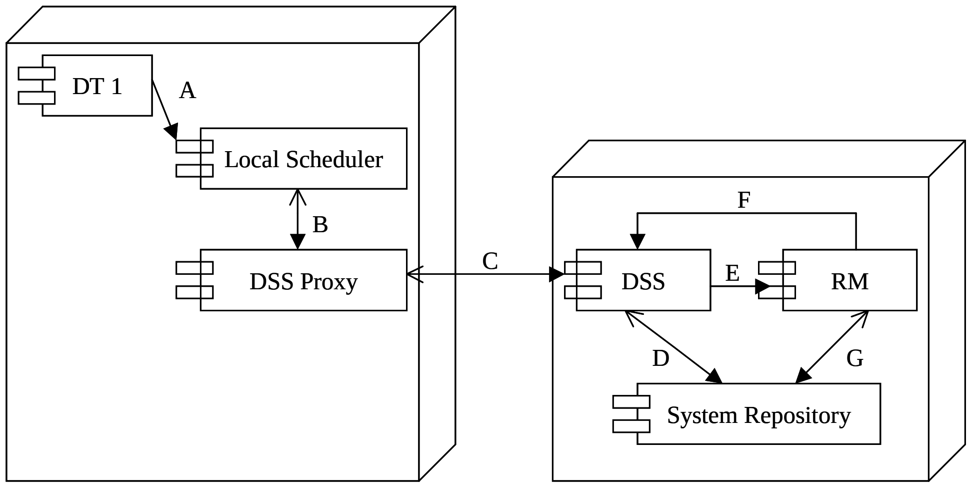


Figure 2. RT DSS System Architecture

Distributable Thread (DT), Local Scheduler, DSS Proxy, DSS, Resource Manager (RM), and System Repository.

A Distributable Thread (DT) is a schedulable entity. When it is spawned by the application, it carries its specified scheduling parameters including the end-to-end deadline. The Local Scheduler is an extension to that defined in RT CORBA 1.2 for managing the local portion of a DT. In this architecture, it interacts with both the DT and the DSS Proxy to obtain and use global information. The DSS Proxy is a running daemon that works as a proxy to the DSS and is always located on the same node as the Local Scheduler. The DSS is a centralized scheduling service with the following responsibilities: online schedulability analysis of an end-to-end task, computation of globally sound scheduling parameters, and triggering of overload management if necessary. If the system becomes unschedulable, the Resource Manager (RM) applies an overload management solution— QoS adjustment, for example. The System Repository stores the information shared between the DSS and the RM.

The implementation of the DSS is supposed to utilize four out of the seven scheduling points defined in RT CORBA 1.2. They are the Begin Scheduling Segment (BSS), at which a DT sends its scheduling parameters to the DSS; the Update Scheduling Segment (USS), at which the DT requires a change to its parameters; the End Scheduling Segment (ESS), at which message is sent to the DSS stating that the DT is no longer in the system; and receive_request, at which a subtask on a new node captures an incoming request of its predecessor.

2.3 Event Service and RT Event Service

A standard CORBA request results in the synchronous execution of an operation by an object, during which data defined by the operation is communicated between client and server. Therefore for the request to be successful, both the client and the server must be available, however there are some scenarios where more decoupled communication between objects is required.

To address this type of communication, OMG issued a specification for CORBA Object Service (COS) Event Service [5]. The Event Service decouples communication between objects by providing for them two roles: the supplier and the consumer. Event data is communicated between supplier and consumer by a standard CORBA call.

The specification describes two approaches to initiate communication between supplier and consumer. They are the push model and the pull model (see Figure 3). In the push model, the supplier is an initiator of communication; it pushes data to the event channel and then the event channel pushes data to consumer. In the pull model, the consumer initiates the connection, it requests data from the event channel, and the event channel in turn pulls data from the supplier. At the heart of Event Service is the Event Channel which plays the role of intermediary between the objects producing data or being changed (suppliers) and the objects interested in data or in knowing about changes (consumers).

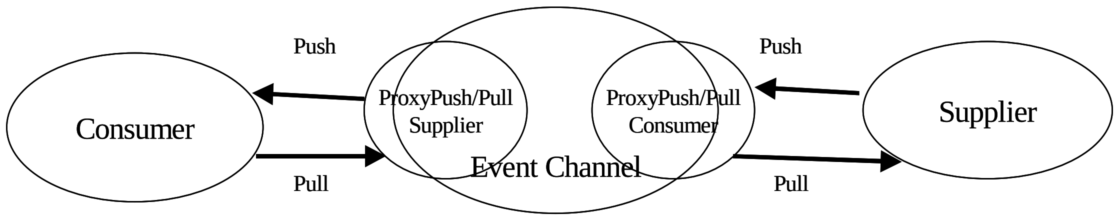


Figure 3 - Event Channel Communication Models

The Event Channel appears to suppliers as a proxy consumer and appears to consumers as a proxy supplier. It is the Event Channel that frees suppliers and consumers from limitations of standard synchronous CORBA calls, and provides flexible communication among multiple suppliers and consumers.

While the CORBA Event Service provides a flexible model for asynchronous communication between objects, its specification lacks important features required by various real-time applications. The work done by Harrison et.al.[6] describes the design and performance of a RT Event Service that was developed as a part of the TAO project at Washington University [3]. This extension is based on enhancements to the push model of CORBA Event Service and supports real-time event scheduling and dispatching, periodic rate based event processing and efficient event filtering and correlation. Figure 4 presents TAO’s Real-Time Event Service (RT ES) architecture and collaborations within it.

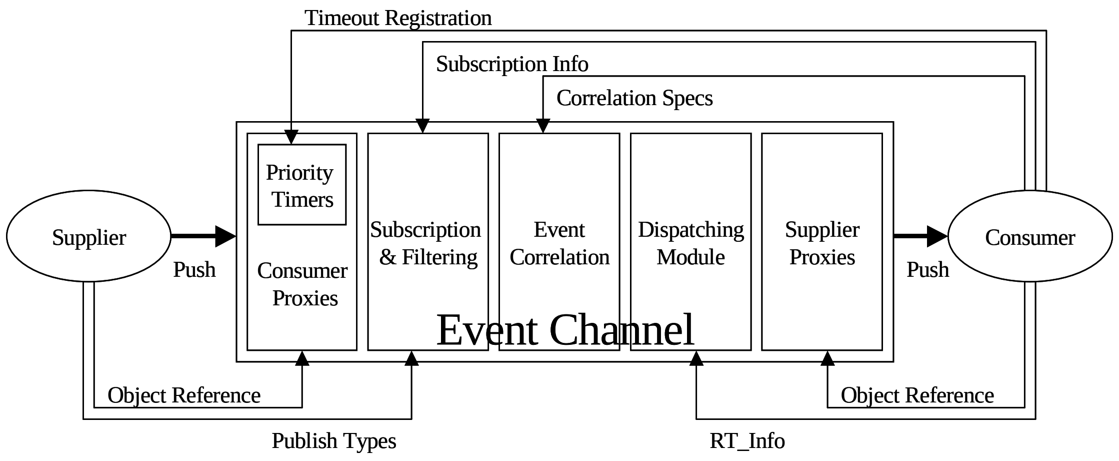


Figure 4. Collaborations in the RT Event Service Architecture

While in this architecture, the Event Channel plays the same role as it does in CORBA Event Service, it consists of several processing modules, each of which encapsulates an independent task of the channel. TAO RT ES's Consumer and Supplier Proxy interfaces extend the standard COS ProxyPushConsumer and ProxyPushSupplier so that suppliers can specify the types of events they provide, and consumers can register with Event Channel their execution dependencies. The Subscription and Filtering module allows consumers to subscribe for particular subset of events, then the channel uses this subscription to filter supplier events to forward them only to interested consumers (In COS Events Service, all events from suppliers are delivered to all consumers). The RT Event Channel provides three types of filtering: Supplier-based filtering that looks for consumers that register for and receive events only from a particular supplier. Type-based filtering that looks for consumers that register for and receive events only of a particular type, and Combined supplier/type-based filtering. The Event Correlation module allows consumers to specify what kind of events are to occur before the Event Channel can proceed. The Dispatching Module determines when events should be delivered to consumers and pushes them accordingly. The architecture of RT ES allows the service to be configured in many ways, since its modules can be added, removed, or modified without changes to other modules. So, for example, for our purposes we configure the ES by removing the Dispatching and Correlation modules, because we use a different mechanism for enforcing real-time event deliveries and we do not assume to have complex inter-event correlation dependencies.

2.4 Naming Service

A *name binding* is a name-to-object association. It is always defined in a *naming context*, which is an object containing a set of name bindings where each name is unique. Different names can be bound to the object in the same or different context at the same time. To *resolve a name* is to find the object associated with the name in a given context. To *bind a name* is to create a name binding in a given context.

Naming Service is the CORBA Object Service (COS) [7] that provides a mechanism through which the ORB clients locate the objects they intend to use.

2.5 Summary of Related Work

Real-time data distribution has become an important area of research. One of the first areas that contributed to the subject is data dissemination in a network. In Karakaya and Ulusov's work[8], for example, the problem of scheduling the broadcast of the data is considered. It provides an approximate version of the Longest Wait First heuristic that reduces overhead. Similar work by Xuan et. al [9] describes a Broadcast on Demand technique that schedules the broadcast using the earliest deadline first, periodic or hybrid algorithms. The work presented by Bestavros [10] describes a speculative data dissemination service that uses geographic and temporal locality of reference to determine which data should be disseminated. These techniques take into account the deadline timing constraints of clients, but do not consider data temporal consistency.

A large amount of real-time data dissemination in wireless sensor networks research is done at the University of Virginia (UVa) [11,12,13,14,15]. While this work addresses the deadline of requests, and the temporal validity is considered in the sense that data is reported before it expires— by corresponding confidence values, the work does not provide assurance that the data is still temporally valid when it arrives to the requestor.

Another application area that has provided various research efforts towards data distribution is embedded sensor networks [16,17,18,19,20]. While all of the work here provides valuable insights into solving the problem of data distribution in sensor networks, none considers real-time characteristics of the data or the applications. That is, neither deadlines on data delivery nor temporal consistency of data is supported.

Quite extensive research for the data consistency problem can be found in the area of real-time databases. The first of such algorithms was the Half-Half (HH) algorithm [21], which suggested that to maintain temporal consistency of data objects, the periods and deadlines of updating transactions should be less or equal to half of the data object validity interval (OV). Then, work by Xiong and Ramamritham [22] presented the More-Less (ML) approach in which periods of updates are assigned to be more than half of the data validity interval and deadlines to be less than a half of the interval with deadline monotonic (DM) scheduling. That allowed maximizing the periods of transactions and hence maximizing the CPU utilization. Then more algorithms were presented based on the ML approach; Further work by Xiong et. al [23,24] considers earliest deadline first based ML (ML_{EDF}) and Deferrable Scheduling (DS-FP). Xiong and Ramamritham later extended their previous work on ML to distributed systems introducing transmission delays of updating jobs [25]. Further, to address variability in transmission delays, recent work by Wang et. al [26] introduces extensions to ML called Jitter-Based More-Less (JB-ML) and Statistical Jitter-Based More-Less (SJB-ML). As with the classical ML approach, all this extra information is used to figure out the deadlines (D) of updates, and then assign the periods (P) according to $D + P \leq OV$, where $D \leq ½ OV$ and $P \geq ½ OV$. All this work assures that data is temporally consistent at the sink, or initial data base storage. Our work extends this assurance to the end point receivers.

All ongoing interest and research in various areas of data dissemination lead the OMG to standardization of data distribution in middleware through a Data Distribution Service (DDS) [27]. This specification describes two levels of interface: Data Centric Publish Subscribe (DCPS) is responsible for efficient delivery of the proper information to the proper recipients, and Data Local Reconstruction Layer (DLRL) is responsible for local reconstruction of data from updates and allows an application to access the data as 'if it were' local. One of the major functionalities of the DCPS along with the topics definition and creation of publishers and subscribers, is attaching various quality of service (QoS) policies to all of the objects it creates. The policy that is responsible for periodic updates is the Deadline QoS policy. The deadline on the publishing site is the contract the application must meet, it means that the publisher is required to send at least one update within the period, the deadline on the subscriber side is a minimum requirement for the remote publisher supplying the data. To "match" a DataWriter and a DataReader, the DDS checks the compatibility of settings (offered deadline ≤ requested deadline). If they don't match (communication will not occur), both sides are informed (via the listeners or condition mechanisms) of incompatibilities. If matching occurs, the DDS monitors the fulfillment of the service agreement and informs the application of any violations by means of the proper listener or condition. Another policy related to our work is a Lifespan QoS. The purpose of this policy is to “avoid” delivering stale data to the application. When a set of data goes beyond its lifespan, it is deleted from all caches. Based on that, there theoretically can be an interval in a periodic data update when an expired data set is already gone, and a new update is not yet complete, so the application trying to read data during this interval might get no data at all.

There are presently several implementations of DDS, both commercial and open source. Two major commercial products are RTI Data Distribution Service from Real-Time Innovations, Inc. [28] and Open Splice DDS from PrismTech [29] that was built upon SPLICE architecture [30], the product of a strategic alliance of THALES [31] and PrismTech. Open Splice DDS is the most complete realization of OMG standard, it fully implements both DCPS and LDRL levels. Other commercially available products are CoreDX DDS from Twin Oaks Computing Inc. [32], InterCOM DDS from Norwegian Kongsberg Gallium Corp. [33], and MilSOFT DDS from Turkish company MilSOFT [34].

OpenDDS is an open-source CORBA-based implementation of OMG DDS by Object Computing Inc. (OCI) [35,36]. It implements all profiles (including optional) of the DCPS layer and none of the DLRL functionality.

Since all these implementations are based on the above specification, none of them can guarantee that applications will always access data that is temporally consistent and that all the specified deadlines will be met.

Another relatively new and fast growing field applicable to data distribution is Cyber-Physical Systems (CPS) [37,38,39]. These systems are integrations of physical processes with computational devices that monitor and control them. By this definition, the CPS can be viewed as similar to embedded sensors networks. However, if the latter are “closed” boxes not exposing their computing capabilities to the outside, the CPS comes from networking such boxes together. Applications of CPS include next era avionic systems, defense systems, high confidence medical systems and devices, assisted living, traffic control and safety, advanced automotive systems, process control, energy conservation, environmental control, critical infrastructure control, etc. Many of these systems require effective and reliable data dissemination from sensors in the physical word to all collaborative entities. Work by Kang et. al [40] discusses the approach to data dissemination in the systems with data continuity (e.g temperature sensors). The authors present a publish/subscribe middleware architecture called Real-time Data Distribution Service (RDDS), with semantic-aware communication, using predictive sensor models. In their approach, both a publisher and its corresponding subscribers maintain the same model for each sensor data stream. A new sensor observation is transmitted from the publisher to the subscribers, and the respective sensor models at both sides are synchronized only when the prediction accuracy of the models becomes lower than the required bound. This architecture implements a broker by which the parties can discover each other, but then communication between publishers and subscribers is performed through multicast. In our work, the sensor data can be discrete (e.g presence of the object in an environment).

As we described above, some of the presented work considers receiver's deadlines, but not considers data validity, some of the work considers deadlines and validity, but at sinks or initial data storages, and not at the end point requestors. The goal of our work is, by taking into account end point requestors' parameters, guarantee them, the delivery of valid data within the specified deadlines.

---

1. OMG. Common Object Request Broker Architecture – Version 3.3, OMG Inc., November 2012, (formal/2012-11-12, formal/2012-11-14, formal/2012-11-16), <http://www.omg.org/spec/CORBA/3.3>. Accessed 19 March 2014.

2. OMG RT CORBA - Version1.2, OMG Inc., January 2005, (formal/2005-01-04), <http://www.omg.org/spec/RT/1.2>. Accessed 19 March 2014.

3. Schmidt, D.C., "Real-time CORBA with TAO (The ACE ORB)," Washington University at St. Louis, 12 November 2013,< http://www.cs.wustl.edu/~schmidt/TAO.html>. Accessed 19 March 2014.

4. Zhang, J., DiPippo, L., Fay-Wolfe, V., Bryan, K., Murphy, M., "A Real-Time Distributed Scheduling Service for Middleware Systems," in Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005), February 2005, Sedona, AZ.

5. OMG Event Service –Specification – Version 1.2, OMG Inc., October 2004, (formal/2004-10-02), <http://www.omg.org/spec/EVNT/1.2>. Accessed 19 March 2014.

6. Harrison, T.H., Levine, D.L., Schmidt, D.C., "The Design and Performance of a Real-time CORBA Event Service," in Proceedings of the Object-Oriented Programming Systems, Languages & Applications Conference (OOPSLA'97), October 1997, Atlanta, GA.

7. OMG Naming Service – Specification – Version 1.3, OMG Inc., October 2004, (formal/2004-10-03), <http://www.omg.org/spec/NAM/1.3>. Accessed 19 March 2014.

8. Karakaya, M., Ulusov, O., "Evaluation of a Broadcast Scheduling Algorithm", Lecture Notes in Computer Science, Springer-Verlag, 2151, 2001.

9. Xuan, P., Sen, S., Gonzales O., Fernandez, J., Ramamritham, K., "Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments," in Proceedings of the 3rd IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'97), June 1997, Montreal, Canada.

10. Bestavros, A., "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems," in Proceedings of the 1996 International Conference on Data Engineering, New Orleans, LA.

11. Lu, B.C., Blum, B.M., Abdelzaher, T., Stankovic, J.A., He, T., "RAP: A Real-Time Communication Architecture for Large Scale Wireless Networks," in Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'02), September 2002, San Jose, CA.

12. Abdelzaher, T., Stankovic, J., Son, S., Blum, B., He, T., Wood, A., Lu, C., “A Communication Architecture and Programming Abstractions for Real-Time Embedded Sensor Networks,” in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

13. Kim, S., Son, S., Stankovic, J., Li, S., Choi, Y., “SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks,” in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

14. Bhattacharya, S., Kim H., Prabh, S., Abdelzaher, T., “Energy Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks,” in Proceedings of the First International Conference on Mobile Systems, Application and Services, May 2003, San Francisco, CA.

15. Li, S., Son, S., Stankovic, J., “Event Detection Service Middleware in Distributed Sensor Networks,” in Proceedings of the International Workshop on Information Processing in sensor Networks (IPSN’03), April 2003, Palo Alto, CA.

16. Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., Estrin, D., “Data-Centric Storage in Sensornets”, in Proceedings of the First ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-I), October 2002, Princeton, NJ.

17. Ye, F., Luo, H., Cheng, J., Lu, S., Zhang, L., “A Two –Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks,” in Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MOBICOM’02), September 2002, Atlanta, GA.

18. Yao, Y. and Gehrke, J., “Query Processing for Sensor Networks,” in Proceedings of the 2003 Conference on Innovative Data System Research (CIDR2003), January 2003, Asilomar, CA.

19. Bonnet, P., Gehrke, J., Seshadri, P., “Towards Sensor Database Systems,” in Proceedings of the Sensor Information Conference on Mobile Data Management, January 2001, Hong Kong, China.

20. Heinzelman, W., Chandrakasan, A., Balakrishan, H., “Energy Efficient Communication Protocol for Wireless Microsensor Networks," in Proceedings of the Hawaii International Conference on System Sciences(HICSS'00), January 2000.

21. Ho, S., Kuo, T., Mok, A., "Similarity-based load adjustment for real-time data-intensive applications." in Proceedings of the IEEE real-time system symposium (RTSS'97), December 1997, San Francisco, CA, pp:144-153.

22. Xiong, M. and Ramamritham, K., “Deriving deadlines and periods for real-time update transactions,” in Proceedings of the 20th IEEE Real-time Systems Symposium, December 1999, Phoenix, AZ.

23. Xiong, M., Han, S., Lam, K.Y., Chen, D., “Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results,” IEEE Transactions on Computers, 57(7), July 2008.

24. Xiong, M., Wang, Q., Ramamritham, K., “On earliest deadline first scheduling for temporal consistency maintenance,” Real-Time Systems, 40, 2008, pp:208–237.

25. Xiong, M. and Ramamritham, K., “Deriving Deadlines and Periods for Real-Time Update Transactions”, IEEE Transactions on Computers, 53(5), 2004.

26. Wang, J., Han, S., Lam, K-Y., Mok, A., “Maintaining Data Temporal Consistency in Distributed Real-Time Systems," Real-Time Systems, 48, 2012, pp:387–429.

27. OMG – Data Distribution Service for Real Time Applications Specification, Version 1.2, OMG Inc., January 2007, (formal/07-01-01), <http://www.omg.org/spec/DDS/1.2>. Accessed 19 March 2014.

28. "RTI Connext DDS Professional," Real-Time Innovations, 2014, <http://www.rti.com/products/data_distribution/index.html>. Accessed 19 March 2014.

29. "OpenSPLICE|DDS", PrismTech, 2014, <http://www.prismtechnologies.com>. Accessed 19 March 2014.

30. van ‘t Hag, J.H., "Data-centric to the Max, the SPLICE Architecture Experience," in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

31. "Thales," Thales Group, <http://www.thalesgroup.com>. Accessed 19 March 2014.

32. "CoreDX DDS Data Distribution Service Middleware," Twin Oaks Computing Inc., <http://www.twinoakscomputing.com/coredx.php>. Accessed 19 March 2014.

33. "InterCOM DDS," Kongsberg Gallium Corp., 2013, <http://www.kongsberg.com/en/kds/kongsberggallium/products/intercom%20dds/>. Accessed 19 March 2014.

34. "Technology Center," MilSOFT, <http://www.milsoft.com.tr>. Accessed 19 March 2014.

35. "The ACE ORB (TAO)," Object Computing, Inc., <http://www.ociweb.com/products/tao>. Accessed 19 March 2014.

36. "OpenDDS," Object Computing Inc., 2013, <http://opendds.sourceforge.net/>. Accessed 19 Mar 2014.

37. Lee, E., "Cyber Physical Systems: Design Challenges." Technical Report No. UCB/EECS-2008-8,University of California, Berkeley, 23 January 2008.

38. Baheti, R. and Gill, H., "Cyber-Physical Systems," The Impact of Control Technology, T. Samad and A.M. Annaswamy (eds.), IEEE Control Systems Society, 2011, available at www.ieeecss.org.

39. Rujkumar, R., Lee, I., Sha, L., Stankovic, J., "Cyber-Physical Systems: The Next Computing Revolution," in Proceedings of the 47th Design Automation Conference, June 2010, Anaheim, CA, pp:731-736.

40. Kang, W., Kapitanova, K., Son, S., “RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems”, IEEE Transactions on Industrial Informatics, 8(2), 2012.

CHAPTER 3

REAL-TIME DATA DISTRIBUTION: MODEL AND THEORY

In this section we present our description of the problem space involved in real time data distribution and existing approaches to data distribution including the solution space they cover. We also present the solution space provided by our work and describe our real time data distribution model and the algorithms we use along with the theorems that verify correctness of our calculations.

3.1 RTDD Problem Space

In systems that require real-time data distribution there are some common characteristics, such as data must be at the right place at the right time and it must be temporally consistent. There are also other specific characteristics that vary from one problem to another. Here we identify these system specific characteristics and group them into three types:

1) System characteristics;
2) Real-time characteristics; and
3) Data characteristics.

These categories are further broken down into specific characteristics, each of which can take on one or more values [1]. Figure 5 illustrates this concept in RTDD Problem Space taxonomy. This section describes each of the characteristics of a RTDD problem, and discusses the values that it may take.

1) System Characteristics

The first layer in the RTDD problem space taxonomy represents system characteristics. These are the overall characteristics of the system that define the general problem.

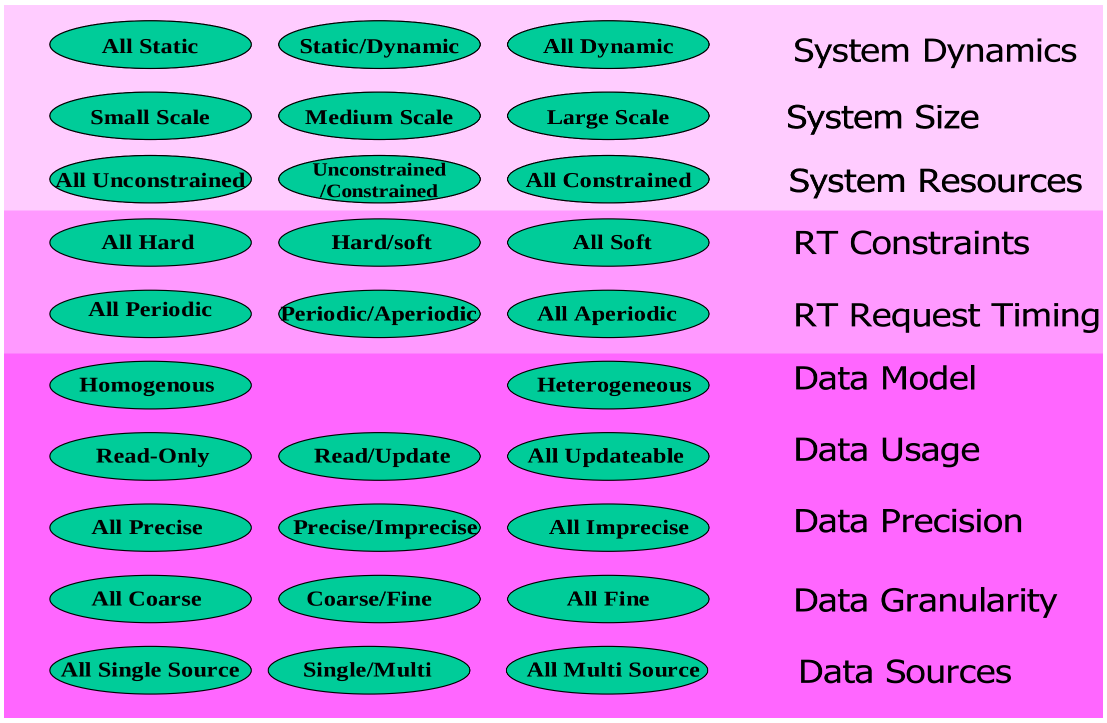


Figure 5. RTDD Problem Space

System Dynamics. Some systems that require real-time data distribution are *static*, that is, the system requirements are fully known in advance and do not change. Therefore, the needs for data distribution can be specified and analyzed prior to system execution to ensure that data that is needed at any particular time and location is delivered on time. For example, an industrial automated system may be static if all of its parts are known at the design stage and do not change during the system's lifetime.

A *dynamic* system is one in which the system specification cannot be predicted before execution time. Requests for data can be made at any time during execution, and the system must be able to either estimate the data needs, or react to dynamic requests in order to meet the timing requirements. An example of this type of system is an electronic stock trading system, in which a client's request for a particular stock price can come at any time during the system's execution.

There are also some systems with a combination of static and dynamic elements. That is, there may be some requirements that remain the same throughout the execution of the system, while others change, or are unpredictable. For instance, in an air traffic control system the requirements for how often to provide wind-speed information may remain the same, while the requirement to receive aircraft information may change based on environmental conditions.

System Size. The size of a system can vary from a single node to thousands of nodes. The size can also affect how much data is being stored, how many suppliers of data there are, and how many consumers there are in the system. An example of a small system that requires RTDD is a patient monitoring system in a hospital. Data about the vital conditions of a patient can be sent to several doctors or other hospital systems. A much larger system might involve thousands of cell phone users requesting stock prices or sports scores from a bank of servers that have the information.

System Resources. The resources of a RTDD system may have various constraints on their operation. For example, a system of small, battery-operated wireless sensors that collect and distribute data about certain environmental conditions has power constraints on each of the nodes, as well as communication constraints based on the strength of the wireless radios. Other systems, such as an embedded network of wired computers aboard a submarine, have fewer physical constraints on the system.

2) Real-Time Characteristics

The next layer in the taxonomy of Figure 5 represents real-time characteristics that involve the timing of the system (periodic vs. aperiodic), as well as the consequences of missing a specified constraint (hard vs. soft).

RT Constraints. Real-time constraints define the system behavior in case of missing specified deadlines. In a *hard* real-time system, if a deadline is missed, the system fails. For example, in an industrial automated system, if data is not delivered on time, the system cannot proceed, leading to further failures down the line. Data itself can have hard deadlines as well. In a submarine contact tracking system, the tracks have to be updated from the sensors within a specified time or they will be considered old, or temporally inconsistent.

A system has *soft* real-time constraints if missing the deadlines causes a degradation of value to the system, but not a failure. For example, a high-availability telecom system may specify that it will deliver data on time a certain percentage of the time. In a soft real-time system, some temporal inconsistency in the data may be tolerated as long as it is corrected in a timely manner.

There are systems with a combination of soft and hard real-time constraints. For instance in a submarine the contact tracking will have hard deadlines, while showing video to the crew will have soft deadlines. The crew could tolerate some frozen video frames while the tracking system is following a potential enemy ship.

RT Request Timing. Requests for data in a real-time distributed system can be made *periodically* or *sporadically* (aperiodically). When a periodic request is made, the data is expected to be delivered at the specified frequency, or else the delivery deadline is considered to be missed. Periodic requests usually occur once, requesting delivery of the data regularly for many periods. The requests can be halted, or the period can change, but while a request is intact, the data should be delivered every period. An example of a system that may require periodic data delivery is a submarine contact tracking system. In order to ensure that the system is representing the real-world contact sufficiently, the system requires that the new real-time data be updated frequently enough to represent a smooth transition from one contact data point to the next.

Sporadic requests for real-time data distribution occur when a client requires data on a one-time basis, or based on events rather than time periods. For example, in the stock trading system described above, a client may specify that they require a stock price whenever its value changes by 5% or more.

3) Data Characteristics

The last layer in the taxonomy represents characteristics that involve the kind of data being shared in a real-time system, and how it is used within the system.

Data Model. The data model used within a real-time data distribution system can be *homogeneous*, where each participant is expected to use the same data model, or *heterogeneous*, where such an expectation is not required. A homogeneous data model makes the sharing of data across the distributed system simpler because no conversion is necessary. However, it may be too restrictive in a large-scale system to expect that various applications that share data will use the same data model. A heterogeneous data model is more flexible, since various applications that are developed at different times, with different requirements can share data without restricting the way in which their own data is stored. However, this type of system may require conversions from one data model to another, or the use of an agreed-upon intermediary representation. For example, in a system that provides data sharing among a coalition of forces from various nations, it is unreasonable to expect the data to be stored in a homogeneous model. For such a system the various data models are stored in their own formats, and a data transfer language, like XML, is used to interpret the data that is shared among the various components.

Data usage. Many real-time data distribution systems only require, that data be disseminated to various clients within timing constraints, but do not expect the data to be updated and written back to the source. These types of systems, which we call *read-only*, do not necessarily require any concurrency control among the distributed clients because they treat the data as their own copies. As long as each client receives data that is temporally consistent, and the data is received within specified timing constraints, the distribution of the data is successful. For example, in an electronic stock trading system, the stock prices are distributed to requesting clients, but the clients do not update them.

However, there are applications in which distributed consumers of the data also update the data and *write it back* to the source, or to other copies of the data. For example, in a submarine contact tracking system, the track data, synthesized from sensor information, may be distributed to various locations so that it can be used, and viewed by other applications and human users. Some of these applications may receive data from other sources that would allow it to make refinements to the track data. In this case, the track data may need to be updated, not only at the source, but possibly also at any other copies of the data. This kind of data usage is much more complicated than read-only data usage because more than one application may wish to update the original data, and therefore concurrency control among these updates is required. If copies of the data also have to be updated, then the system is even more complex. The fact that all of the data must be kept both logically and temporally consistent with each other adds to the complexity of the problem.

Data Precision. Some real-time systems require that the data that they receive be absolutely *precise*, consistent with the real-world entities that are being modeled. In such systems, the concurrency control mechanism that maintains the integrity of the data will not allow multiple updates, even if the locking that might be required will cause deadlines to be missed. Further, the data must be temporally consistent at all times – never becoming older than a specified age. For instance, a command and control system that is closely tracking a target will want to be sure that the data it receives is precise.

On the other hand, some applications allow for the possibility of some *imprecision* in the value as well as the age of the data in order to allow for more flexibility in meeting other constraints. For example, a client of an electronic stock trading system may be willing to receive data that is slightly old, or slightly imprecise, if it means paying a lower fee. As long as the amount of imprecision is bounded, the client can analyze the data with the imprecision in mind.

Data Granularity. The amount or granularity of data that is distributed to clients can vary from entire tree structures, to single atomic elements. In the case of an object-oriented system, entire objects can be distributed to various locations for use by clients. In fact, groupings or hierarchies of objects can be distributed all together; these are *coarse-grained* distributions of data. On the other hand, a *finer grain* of data can be distributed such as individual attribute values, or return values of object methods. The granularity of the data being distributed depends largely on the applications that are using the data, as well as how the data is being used. For example, in a system in which the distributed data is being updated and written back, it might make sense to employ the smallest granularity possible so that large portions of data are not locked due to concurrency control.

On the other hand, when groups of objects are closely related, it may make sense to distribute them together as a group. This way, the values of the related data are more likely to be relatively temporally consistent with each other, and therefore more valuable to the requesting client.

Data Source. In many real-time systems, real-time data comes from sensors that provide the most recent version of the data. In many cases the sensor transaction is the *single source* of update for the data. However, it is also possible for the data to be updated by *multiple sources*. For example, in a target detection system, various sensors may be used to update the data depending upon which is the closest, or most reliable. In this case, it may be possible that both sensors try to update the data simultaneously, requiring concurrency control to ensure the integrity of the data.

All the characteristics described above form the definition of a problem space for real-time data distribution.

3.2 Existing Approaches to RTDD and Solution Space Addressed by Our Work

In this section we discuss different mechanisms of RTDD and show the areas within the problem space that they address. Then, we describe the subset of the problem space that our work addresses, along with the solution provided by our work.

3.2.1. Types of RTDD

Client-Server. The Client-Server, an example of point-to-point communication model, can be considered a pioneer method of data distribution. The Client-Server model is a central idea in network computing. Many business applications existing today use this model. In this model, a server waits for requests from clients, who access data via queries. In some of these applications, clients can read and update information on the server.

The client-server approach to RTDD is very broad. Therefore, the area within the RTDD problem space that can be addressed depends greatly on the application that is being served. A client-server model can address both static and dynamic systems. Most applications that use this approach are dynamic, but in a system in which all requests for data are known a priori, a client-server approach can also work. The client-server model can work in a system of any size. However, in order to provide real-time support for data distribution, a larger size can become unwieldy. Further, if there are a lot of requests for the same data, it becomes difficult for a single server to respond in a timely fashion. Thus, multiple servers might be necessary, which makes the system more complex.

In the client-server model, clients can access data both to read it and to update it. The typical client-server model does not specify any allowance for imprecise data. However, a specialized implementation can build imprecision into a particular application. The granularity of the data depends upon the service provided by a server. Typically, in a client-server model, there is a single source for any data that is available. If more than one server provides the data, it usually originates at the same source.

Broadcast and Multicast. The Broadcast and Multicast are examples of point-to-multipoint communication model. With the broadcast, data or signal is transmitted to anyone and everyone in a particular service area or network. For instance, in the wireless network of portable devices (cell phones, PDA, palmtops etc.) information such as electronic newspapers, weather and traffic information, stock and sports tickers, and entertainment delivery is broadcast to all devices in the network. The difference between broadcast and multicast is that in a multicast communication model, data is transmitted to a select list of recipients and not to everyone in the network. The target systems for broadcast or multicast RTDD are dynamic. Thus, the real-time constraints that a broadcast or multicast system has are usually soft. In order for the supplier to efficiently serve all requestors by broadcasting or multicasting data, the data model must be homogeneous.

This is a read-only approach. Broadcast data can be precise or imprecise depending upon the requirements of the receivers. As long as the receiver is aware of the level of imprecision, it can be factored into how the data is used. Broadcast data can be at any level of granularity. However, due to the widespread use of the network in a broadcast, smaller, more fine-grained data may be more efficient to send. Typically, in a broadcast model, there is a single source for any data that is available.

Streaming. Streaming is a technology in which data is transferred from a server to a client and is processed in a steady and continuous stream, without the need to be stored in a client's space. Typical applications that use streaming for RTDD are video, and continuous backup copying to a storage medium.

Systems that use streaming for RTDD are usually dynamic— clients connect and disconnect at any time. The size of the system can be quite large. In an HDTV application, thousands of users view the stream from a source. Since clients do not need to store data, they can operate with some limited resources. Streaming systems typically have soft real-time constraints, such as minimum frame rate on a video stream.

Data transfer can be periodic or sporadic. In a video streaming application, the frames are transferred periodically so that they can be displayed on the receiving node with a constant frame rate. For an application in which data is streamed for continuous backup, the rate of the stream is not as important, and can be more sporadic. The data model of a streaming application is typically homogeneous. This way, the sender can stream data, such as video frames, and the receiver knows how to process it.

Similar to broadcast, streaming RTDD is a read-only approach. For best quality, streaming data should be precise.

The granularity of the data in a stream depends upon the application. The receiving node has to process the data upon receipt, so it would make sense to use the smallest granularity possible. Typically, in a streaming model, there is a single data source.

Real-Time Data Bases. A real-time database (RTDB) is considered as an extension to a traditional database. It has all traditional database features, but also is able to express and maintain timing constraints, such as deadlines, earliest and latest start time on transactions and timing constraints, such as temporal consistency on data itself. A RTDB consists of RT objects representing real world entities and updated by sensor transactions. To be coherent with the state of the environment, the RT object must be refreshed by a transaction before it becomes invalid, that is within its temporal validity interval, whose length is usually application-dependent. There are many applications that require real-time data, and with advances in networking they are not necessarily located on the same node as the RTDB and therefore require the real-time data to be distributed to them.

A RTDB can handle both static and dynamic systems. A central database can serve small- to medium-scale systems. For larger scale systems, a distributed database is usually used. Computational resources are usually constrained by the timing constraints imposed by the applications that use a RTDB and resource constraints exist in a RTDB that involves mobile, wireless nodes.

Transactions in a RTDB can be hard or soft, and can be periodic or sporadic. The data model is typically homogeneous. Although, in larger systems that combine various RTDBs into a single virtual RTDB, it may be possible to have a heterogeneous data model. In this case, middleware is typically used to synthesize the various models. Most RTDB applications expect precise data.

TAO's Real-Time Event Service is an implementation of point-to-multipoint communication model. Since we gave a thorough description of RT Event Service in the background section (2.3), here we only highlight the solution space provided by this approach in the RTDD problem space.

TAO's RT Event Service can handle static and dynamic systems of various sizes. The computational resources in the system are bound by the timing constraints imposed by the application. The service can provide support for both hard and soft real-time applications. Publish-subscribe nature of the RT Event Service allows processing of both periodic and a periodic types of requests.

The data model for TAO's RT Event Service is homogeneous, since the consumers use the same data model as the suppliers. Only the suppliers can change their data and the consumers are just readers, therefore, the data usage is read-only. Since the service allows the suppliers to register for the ‘whole’ event, and not a part of it, only coarse granularity is supported. On the other hand, if we consider an event as a single piece of information it can be considered fine. Then, if a subscriber wants to impose some event dependencies and get a combination of several events, that can be considered coarse. The RT Event Service allows supplier/type based filtering, therefore it can address multiple sources of data.

OMG Data Distribution Service is an implementation of point-to-multipoint communication model. Since we described the service and explained the way it differs from our work in section 2.2.5, here we only will provide a description of the area in the problem space addressed by the service.

The DDS can be used for both static and dynamic types of systems of various sizes, and it can address soft real-time systems. However, it does not enforce any constraints. For this, an underlying real-time scheduling mechanism must be used. Both periodic and a periodic requests can be specified. The data model assumed by the DDS is homogeneous. However, implementations of DLRL can provide transition among application data formats to the DDS data model, making the service suitable for heterogeneous applications. Since there is a de-coupling between publishers writing to the data and subscribers accessing data, the data usage can be defined as read-only.

Both precise data and imprecise data (by means of TIME_BASED_FITERQoS and HISTORY policies) can be used by DDS. Various levels of granularity can also be supported. By using MultiTopic Class, a subscriber can select and combine data from multiple topics into a single resulting type.

The OWNERSHIPQoS policy allows multiple DataWriters to update the same instance of data-object. There are two settings for this policy: SHARED indicates that the service does not enforce unique ownership, so multiple writers can update the same data instance simultaneously and subscribers can access modifications from all DataWriters; EXCLUSIVE indicates that each data instance can be updated by one DataWriter that "owns" this instance, though the owner of data can be changed. Thus the service provides both multiple and single data source solutions.

3.2.2 Solution Space Provided by Our Work.

In our work we consider two types of application: static and dynamic. For the static model we address the following specific problems in the data distribution problem space.

- System Characteristics:
    - Small- to medium-scale systems consisting of tens to hundreds of nodes;
    - Static applications and infrastructure. All system requirements are known a priori and are invariant;
    - Unconstrained resources. We assume high-powered CPUs and high-speed network with high bandwidth.
- Real-Time Characteristics:
    - Hard.
    - Periodic request timing.

- Data Characteristic:
    - Temporally constrained data;
    - Homogeneous data model;
    - Asynchronous data production;
    - Precise data;
    - Fine or course grained data;
    - Single source for each data item.

Our dynamic model covers the following area in the problem space.

- System Characteristic:
    - Small, medium, or large scale;
    - Dynamic infrastructure;
    - Unconstrained resources.
- Real-Time Characteristics:
    - Soft real-time;
    - Periodic request timing.
- Data characteristics are the same as for the static system.

3.3 RTDD Model

This subsection describes a real-time data distribution model – the basis of our work.

Figure 6 displays our Real Time Data Distribution Model. The model consists of five main elements.

$$\begin{aligned}
&DataObject = <OID, Value, TS, OV> \\
&DataSource = <SID, Node, OID, SP> \\
&DataReader = <RID, Node, OID, SP> \\
&Dist = <DID, OID, SID, <RID, SP>> \\
&SP = <P, D, R, E>
\end{aligned}$$

Figure 6. Real-Time Data Distribution Model

The ***DataObject*** represents the data that is being distributed. ***OID*** is a unique identifier of the data object within the system. ***Value*** is the value of the data object. This can be a simple atomic value, or a structured value depending upon the granularity of the data. ***TS*** is the time (timestamp) at which the object was last updated. ***OV*** is the object validity, a time interval within which the data object is considered to be valid after its update. When the ***OV*** expires, the data is considered temporally invalid. The ***DataSource*** is the entity that produces the data that is to be distributed. ***SID*** is a unique identifier for the data source. The ***DataReader*** is the entity that requests that data be sent to it. ***RID*** is a unique identifier for a data reader. ***Node*** is the computing element on which the source/reader executes. ***SP*** is a set of scheduling parameters. ***P*** is the period of the task. Recall that our solution addresses the problem space of periodic data distribution. ***D*** is a deadline within the period. ***R*** is the release time after which the task may start to execute. ***E*** is the worst-case execution time of the task. Note that the data source and the data reader may have different scheduling parameters. ***Dist*** is a distribution of data from a ***DataSource*** to a ***DataReader***. A distribution has its own unique identifier ***DID.*** It also has its own scheduling parameters that will be determined by the proposed data distribution algorithms. The algorithms consider the scheduling parameters of the ***DataSource***, the scheduling parameters of the ***DataReader***, and the data object validity interval to determine the scheduling parameters of the distribution.

3.4 RTDD Algorithm

In this section we describe the algorithms we use to compute distribution parameters for the static and dynamic models, and provide a theoretical background that ensures the correct work of the algorithms in an actual implementation.

3.4.1 JIT Static Data Distribution (JITS)

The algorithm we are using to ensure that all data readers receive the temporally valid data in time is a modification of the Just-In-Time Real Time Replication algorithm [2] and is called Just in Time Data Distribution Algorithm (JITDD). This algorithm, based on data source and data readers' real-time characteristics, and data validity time, computes appropriate deadlines for data distributions.

For a static system, the algorithm works as follows:

Let d be the deadline that is computed for a distribution *Dist* from source S to a set of m data readers $R_1,\ldots,R_m$ for a request of data object *OID*. The period of S (and therefore of *Dist*) is p. Let N be the least common multiple of the periods of all data readers of *OID* and the period of the source.

We call N the *superperiod* of the distribution because it represents a complete cycle of all readers for the data. We define OV_i to be the point in time in the i^{th} period of the distribution that the object (from the most recent update) becomes temporally invalid. An invalid interval is an interval of time during which the object does not have a valid value associated with it, that is, the object is temporarily inconsistent.

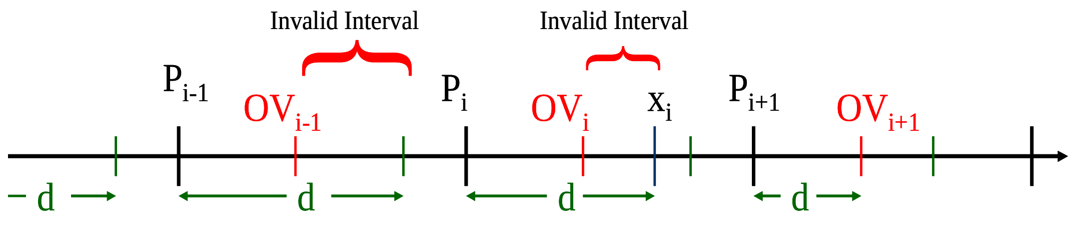


Figure 7. Deadline Computation

Figure 7 depicts an invalid interval. OV_i is the time within period P_i that the data that was updated during period P_{i-1} becomes invalid. The d in the figure represents the deadline of the distribution within its period. The invalid interval is the time between OV_i and this deadline because after the deadline, a new value of the data will have been delivered.

In the algorithm, when computing the deadline of the distribution, initially we set it to be equal to its period ($d=p$). The key to computing the deadline of the distribution is to determine if any of the data readers will be executing in the invalid interval. If so, it is possible that it could use invalid data. For each reader, there is a window, called the *data access window*, within its period when it could access the data. The data access window falls between the release time of the reader and its deadline. There are three cases to consider when calculating the deadline of the distribution:

1) If no reader's data access window overlaps with the invalid interval, the deadline is unchanged because no reader will be using invalid data.

2) If some reader's data access window begins at time x_i, after OV_i, i.e. $OV_i < x_i < P_{i+1}$ and ends before the next invalid interval, then the deadline is changed to $min(d, x_i - P_i)$. That is, the distribution must complete, before this reader's data access window begins.

3) If any reader's data access window has started before, at or after OV_i and continues to execute in the same/next invalid interval, then the deadline is changed to $OV_i - P_i$. This deadline assignment ensures that there will be no invalid interval within the period at all, and thus the reader will use valid data.

Note that if the deadline is changed to $OV_i - P_i$ at any point, the computation of deadline is complete, as we have reached the minimum possible deadline. Otherwise we consider these three cases for each of the *n* periods in the superperiod.

It can be noted that a simple way to compute this deadline would be to always use $OV_i - P_i$. This would provide the required temporal validity, but it could be an overly pessimistic choice, and might cause the system to be nonschedulable. Because in our current implementation this algorithm is computed off-line, the extra work that is required to compute the more flexible deadline is acceptable.

While this algorithm works for a static model, since all the computation can be done off-line, the overhead, which will be imposed by the *superperiod* computation in case of significant amount of data readers, makes it impractical to use this algorithm for on-line computation. Therefore, to suit the needs of dynamic application, we changed the algorithm so that it delivers the same quality of result with significantly less computation overhead.

3.4.2. JIT Dynamic Data Distribution

First, let us observe that the least common multiple (LCM) of two numbers *a* and *b* can be obtained by finding the prime factorization of each

$a = p_1^{a1} \cdot p_2^{a2} \cdots p_n^{an}$

$b = p_1^{b1} \cdot p_2^{b2} \cdots p_n^{bn}$,

where p_is are all prime factors of *a* and *b*, and if p_i does not occur in one factorization then correspondent exponent is taken as 0, then

$\text{LCM}(a,b) = \prod_{i=1,..n} p_i^{\max(ai,\ bi)}$

Also $\text{LCM}(a,b,c) = \text{LCM}(\text{LCM}(a,b),c)$

$= \text{LCM}(\prod_{i=1,..n} p_i^{\max(ai,\ bi)}, c)$

$= \prod_{i=1,..n} p_i^{\max(ai,\ bi,ci)}$

then $\text{LCM}(a,b,c) / \text{LCM}(a,b) = \prod_{i=1,..n} p_i^{\max(ai,\ bi,ci)} / \prod_{i=1,..n} p_i^{\max(ai,\ bi)}$

$= \prod_{i=1,..n} p_i^{\max(ai,\ bi,ci)-\max(ai,bi)}$

and $\text{LCM}(a,b,c) / \text{LCM}(a,c) = \prod_{i=1,..n} p_i^{\max(ai,\ bi,ci)} / \prod_{i=1,..n} p_i^{\max(ai,\ ci)}$

$= \prod_{i=1,..n} p_i^{\max(ai,\ bi,ci)-\max(ai,\ ci)}$

etc.

Based on that, we can observe that *subsuperperiods (SubN)* that are LCMs computed based on the source period and each data reader period may "repeat" in *superperiod (N)*. Therefore we can take *SubN* instead of *N* with the rest of algorithm remaining the same. Minimum deadline, computed for each data reader in *SubN* will be the same throughout *N*.

Therefore, in a dynamic case, when a new reader comes into system, we do not need to re-compute the *superperiod* for all corresponding readers in the system. Instead we compute *subsuperperiod* for the new reader and data source, perform our algorithm and check existing deadline against computed. If existing deadline is less than computed, nothing changes. If it is bigger, then we change it to the computed value, because now this is the minimum deadline that satisfies all readers.

3.5 Theorems

This section presents a theoretical background assuring the correct work of our algorithms.

Lemma 1:

For a set of readers, to preserve the data consistency the Distribution period must be equal to the Source period.

Proof:

Without loss of generality we can assume that the given set of readers is such that the readers may access data during or over each of the invalid intervals. Therefore to preserve the consistency of data, new data must be distributed before or during each of the invalid intervals. This cannot be achieved without the *Distribution* period being equal to the *Source* period. Assume that is not true and that the *Distribution* period can be longer or shorter than the *Source* period. Then, in the first case, depending on the source deadline the Distribution can disseminate data that becomes invalid at OV_1 or OV_2 (see Figure 8), with nothing that can be done to prevent readers from reading old data. The same may happen with the period of distribution being less than data source period (see Figure 9). In both cases we cannot guarantee that we can manage each of the invalid intervals, and hence we cannot guarantee consistency of data. Therefore to preserve the data consistency, the *Distribution* period must be equal to the source period.

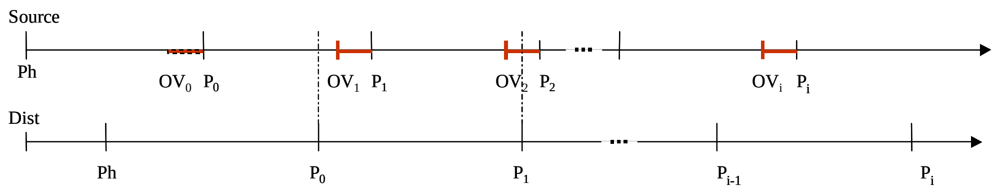


Figure 8. Lemma 1 ($P_{Dist} > P_{Source}$)

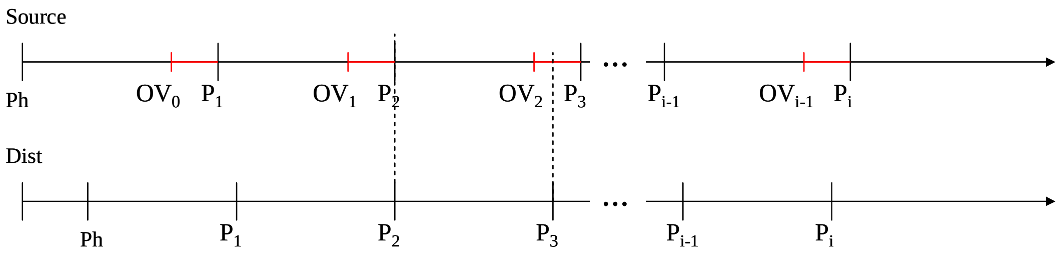


Figure 9. Lemma 1 ($P_{Dist} < P_{Source}$)

Lemma 2:

For a Distribution to distribute fresh data and hence to preserve data consistency, it must start at or after SU_d (bounded by OV-P-ET).

Proof:

To prove the lemma, consider the i_{th} *Distribution* period (see Figure 10). To preserve data consistency in this period, the data must be updated before some computed deadline with the data that is not going to expire during this period. To distribute the data that is not going to expire in the current *Distribution* period, *Distribution* cannot start before the supposed finish of the current (i^{th}) data source update. If it does, it might distribute an old data (e.g. the same unit) expiring at OV, so that readers will access invalid data even though *Distribution* finished before the specified time. Thus to preserve data consistency, the i^{th} *Distribution* must start at or after the current sensor update deadline. Consequently, the first *Distribution* must start at or after the sensor update deadline (SU_d) in its first period.

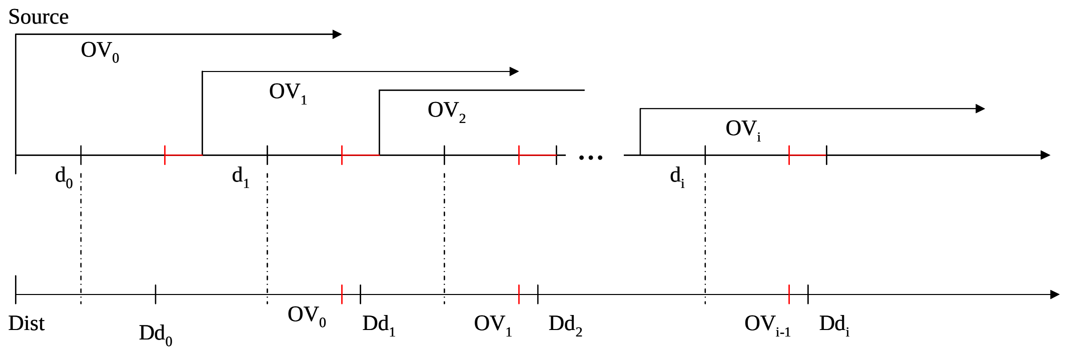


Figure 10. Lemma 2

Though it does not make much sense to delay *Distribution,* since it will reduce the time assigned for its execution, we need to note that the *Distribution* start has to be bounded by OV-P-ET, otherwise *Distribution* will not be able to finish before its deadline (in the case when the computed deadline is equal to OV) because it will leave less time than is necessary for *Distribution* to execute, and as a result data consistency will not be preserved.

Theorem One:

For a set of readers, if Dist period is equal to the period of Data Source, Dist deadline is computed according to the JITDD algorithm and Dist phase is at or after SU_d (bounded by OV-P-ET), (where SU_d is the sensor update deadline, OV is data validity time, P is period of Source, ET is execution time of Dist), then the readers will always read valid data.

Proof:

Now having lemmas 1 and 2, and assuming that the JITDD algorithm works with the specified *Distribution* period and phase, we will show that the deadline computed by the JITDD algorithm guarantees that readers will read valid data.

Recall from the JITDD algorithm that there are three possible cases considered for deadline computation. To prove that no reader reads invalid data, let us re-examine these cases.

Case 1) No readers read in the invalid interval. Conclusion is clear.

Case 2) Some readers start at time X_i such that $OV<X_i<d_i$ and finish before the next invalid interval. The JITDD algorithm changes *Distribution* deadline d to X_i, reducing the size of the invalid interval and making the *Distribution* update an old data set with the fresh one before any reader reads it. Thus no reader reads the data within the invalid interval.

Case 3) Some readers read the data through the invalid interval, that is start before, at or after OV and finishes at some point in the current/next invalid interval. In this case the JITDD algorithm computes the deadline to be equal to OV, and by doing that removes the invalid interval. Therefore no readers can possibly read data within it.

So, we proved that having the distribution's period equal to the data source's period, the distribution's phase at or after $SU_{d,}$ and having the deadline computed by the JITDD algorithm, will guarantee the set of readers always receive temporally consistent data.□

Definition:

The optimal deadline is a deadline that cannot be made any longer.

Theorem Two:

The JITDD algorithm assigns the optimal deadline for ensuring the temporal consistency of data.

Proof:

Theorem 1 proves that with the deadline assigned according to the JITDD algorithm, the data read by all requests is always temporally consistent. To prove that the assigned deadline (d) is optimal, let us assume that there exist another greater data distribution deadline (d1) assigned by some other algorithm, which still preserves data consistency. The JITDD algorithm computes the data distribution deadline and consequently redefines the invalid interval to [OV, d] based on the knowledge that no request reads data during this invalid interval, but there are requests that may start to read data right after d. Now, with another deadline d_1 we have the invalid interval defined as [OV, d_1] and consequently we have an interval [d, d_1] during which a request may read an invalid data set. That is, the data consistency is not preserved, and our assumption about the existence of another greater deadline is wrong. This implies that the JITDD algorithm's deadline assignment is optimal.□

This concludes our theoretical background on modeling and algorithms. In the next two chapters we will present our approach to implementation of data distribution mechanisms for both static and dynamic systems.

1. Uvarova, A. and Fay Wolfe, V., "Towards a Definition of the Real-Time Data Distribution Problem Space," in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

2. Peddi, P., "A Replication Strategy for Distributed Real-Time Object-Oriented Databases," TR01-282, University of Rhode Island, May 2001

CHAPTER 4

STATIC RTDD

This chapter presents system design, implementation and evaluation of static real time data distribution.

4.1 System Design and Implementation

Since in a static system, all system characteristics are known a priori and system analysis can be done ahead of time, the implementation of data distribution is divided into two parts: an off-line analysis and on-line event-based data delivery.

4.1.1 Off-line Analysis

Figure 11 depicts the process that is followed in the off-line analysis of our implementation. It begins with the specification of the system, in the format of our model described in Section 3.3. An ASCII file containing descriptions of all of the data sources, readers, data and nodes is created and stored. The C++ implementation of the JITDD algorithm reads in the system specification and computes the scheduling parameters for each of the data distributions required. The output of the JITDD algorithm is another ASCII file containing the system specification augmented with the computed distribution scheduling parameters.

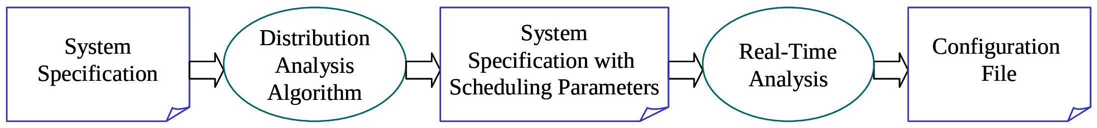

System Specification
Distribution Analysis Algorithm
System Specification with Scheduling Parameters
Real-Time Analysis
Configuration File

Figure 11. Off-line Analysis Process

The augmented system specification is fed into a real-time analysis tool to determine if the system is schedulable. While we were doing this work, the only available choice was the RapidRMA tool by TriPacific Corporation [1]. The use of RapidRMA involved manually translating the specification into the visual model required by the tool. We had to transform all components of our system model, that are the sources, readers, and distributions into a system of tasks, resources, and task dependencies that are required by the RapidRMA. RapidRMA performs a schedulability analysis on the specified model using Deadline Monotonic, end-to-end analysis [2]. If the system is found to be non-schedulable, the system specification must be reworked, perhaps adding more nodes or more powerful nodes to the system. Once the system is deemed schedulable, RapidRMA produces a configuration file that provides scheduling priorities for each of the tasks in the system. This configuration file is used in the on-line implementation described next. At present time the OpenSTARS tool [3] developed by URI RTDOC group is available for the analysis purpose. This tool eliminates the manual translation work, because it gets all necessary information directly from the system specification file.

4.1.2 On-line Implementation

The runtime component of our implementation executes the model specified in the off-line component described above. The implementation was programmed in C++ and ran on Linux Kernel 2.4.21, with TAO v1.3.5 CORBA software [4] to provide real-time middleware support. The implementation also used two of TAO's common object services: the Real-Time Event Service (RTES) [5], and the Real-Time Static Scheduling Service (RTSSS) [6]. The RTES was used as a mechanism for distributing data asynchronously, and the RTSSS provided priority-based scheduling to ensure that deadlines are met. Figure 12 illustrates our implementation using these two services.

Event-based Data Distribution. TAO's RTES provides asynchronous, decoupled communication between sources and readers of data. The RTES uses a supplier/consumer model to deliver events. The supplier sends data from a specific source to the RTES, and the consumer receives data from the RTES. In our implementation, we create a supplier to distribute data that is produced at each source, and we create a consumer to receive data for each reader.

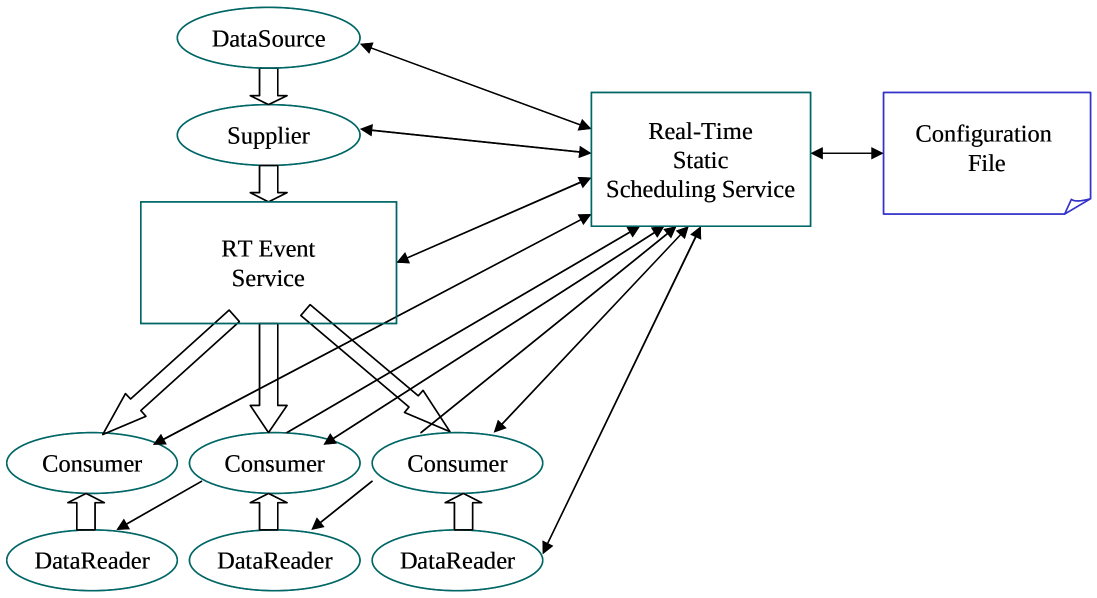


Figure 12. On-line Implementation.

The RTES can be configured in various ways, including a complex configuration with a priority-based thread dispatcher, and a simple, single-threaded configuration that maps one Real-Time Event Channel (RTEC) to each supplier [5]. Because our implementation performs all of the scheduling analysis off-line, we have chosen the simple configuration of the RTES.

The RTES provides an interface for a supplier to register events (data) that it will supply. It also provides an interface for a consumer to register for events that it would like to receive. The RTES matches these requests with the supplied events, and sends the event data to consumers when they are supplied by the suppliers. Consumers in their turn make the data available for the readers to use. Based on our formal model of Section 3.3, a data ***Distribution*** is represented by the delivery of event data from the supplier to each consumer.

Scheduling Real-Time Data Distribution. In previous work, URI RT DOC group developed the Real Time Static Scheduling Service (RTSSS) that is in the TAO code base [3]. It is implemented as a set of library code that is compiled into the programs that use it. The library code creates a mapping of task to priority, using the information in the configuration file produced by the scheduling tool (RapidRMA, in this work). When the system starts up, each of the executing entities (sources, suppliers, consumers, readers, RTES) begins by requesting a priority from the RTSSS. The RTSSS looks the priorities up in the task/priority mapping table, and sets the priorities accordingly. Each of these tasks then executes at its specified priority.

4.2 System Evaluation

In order to demonstrate the effectiveness of our implementation, we developed several test scenarios to make sure that our claim of ensuring temporal validity and deadline of the distribution holds in our implementation. The main metrics we used are *temporal consistency of delivered data*, and *deadline of data delivery*.

The first two of the test scenarios we used examine the system under “normal” conditions and under workload-constrained conditions. In the third set of tests, we developed a system model based on the real Navy weapon alignment application. Below we describe the various test cases, how they were modeled and implemented, and the results of the tests that we performed.

Test Scenarios. We tested tree scenarios, each of which is described here. In each scenario, we used two nodes, with executing entities distributed across these nodes. Recall that in each case, the system is modeled and analyzed up front, so we have chosen systems that are schedulable, but in some cases, may be close to being non-schedulable. Figure 13 illustrates the system layout for the first two test scenarios.

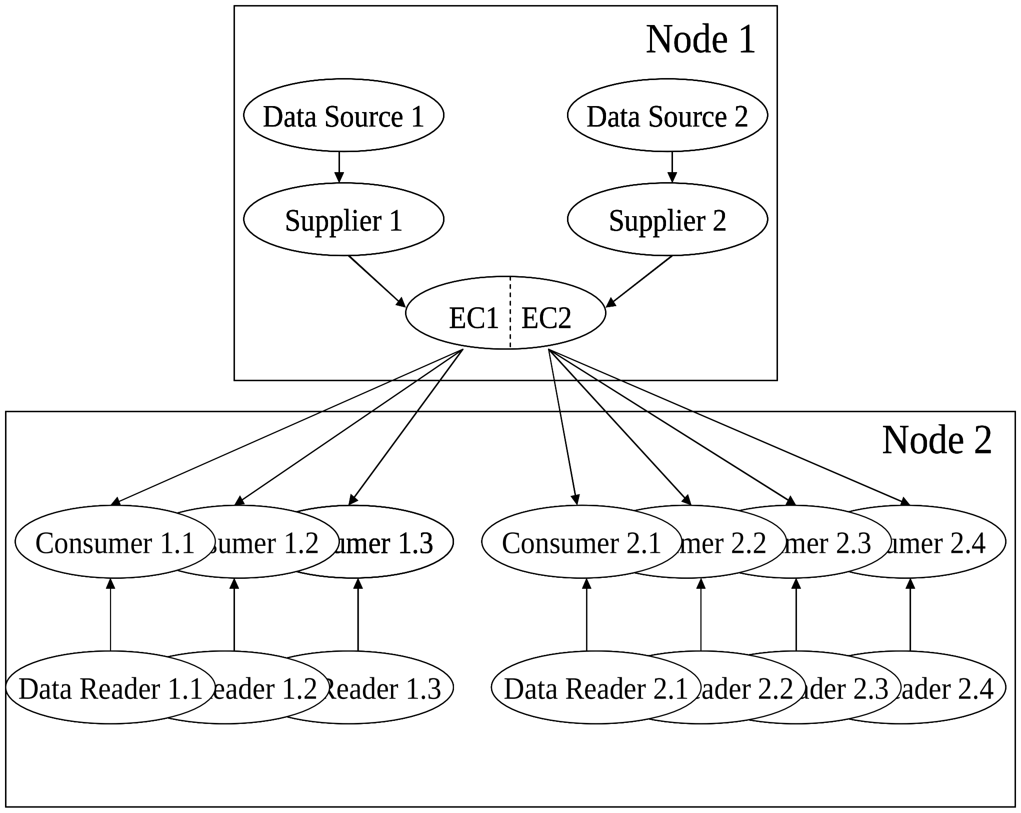


Figure 13. Test Scenario Set Up

Below we describe the specific parameters for these scenarios.

<u>Scenario 1 – Normal Conditions:</u> Figure 14 depicts the layout of entities in the system on the two available nodes. On node 1, there are two data sources, two suppliers and the event channel. In the implementation, there is an instance of the event channel for each supplier. Node 2 has the consumers and the readers that will use the data. Table 1 gives the specific parameters for each of these entities. The table has two rows for the event channel (EC1 and EC2). Each of these represents the distribution from one of the data sources to the set of readers that have requested the data. Additionally, we specified a network delay of 150 μsec for each transmission between node 1 and node 2. The object validity for Data Source 1 is 150,000 μsec, and for Data Source 2 is 140,000 μsec. Note that in Table 1, the deadline listed for each consumer represents the computed deadline for the distribution for the associated data source. These consumer deadlines were computed using the JITDD algorithm, synthesizing the deadlines for each reader that requested data from the data source. The entire system model was analyzed in RapidRMA, and found to be schedulable.

<u>Scenario 2 – Workload Constrained:</u> This scenario is almost identical to Scenario 1, except that extra workload was inserted onto Node 2. This workload increased the utilization on that node from 16.53% to 72.15%.. Again, the model was analyzed using RapidRMA, and while the extra workload on Node 2 caused the system to be more constrained, it was still schedulable. We chose to perform this test to show that under tight workload conditions, when the system is found to be schedulable, our implementation meets all deadlines and temporal consistency constraints.

Name	**Period, µsec**	**Release, µsec**	**Deadline, µsec**	**Exec time, µsec**
DataSource1	100 000	0	10000	1500
DataSource2	80000	0	10000	2000
Data Reader 1.1	100000	80000	30000	1500
Data Reader 1.2	200000	180000	40000	1500
Data Reader 1.3	300000	280000	50000	1500
Data Reader 2.1	100000	80000	40000	2000
Data Reader 2.2	120000	130000	50000	2000
Data Reader 2.3	180000	130000	100000	2000
Data Reader 2.4	200000	160000	80000	2000
Supplier1	100000	10000	70000	1000
Supplier2	80000	10000	60000	1000
EC1	100000	10000	70000	400
EC2	80000	10000	60000	400
Consumer1.*	100000	10000	70000	1000
Consumer2.**	80000	10000	60000	1000
* All consumers of DataSource1 (** and of DataSource2) have the same parameters				

Table 1. Test Scenario Parameters

Scenario 3 – Navy Weapons Alignment Application: In order to demonstrate how our algorithm and implementation work with a real application we have developed a simulation of the Navy weapon alignment system (see Figure 14). This Figure is the property of the Raytheon Company [6].

In this system, a set of navigation subsystems produces navigation data. This data must be distributed along a chain of components so that it can eventually be used by the weapon subsystems to align the weapons according to the latest location of the ship. The data is not only distributed along the chain, but it is also processed along the way. For example, the *Nav Data Interchanges* component receives the raw data and processes it so that the *Process Nav Data* component can use it.

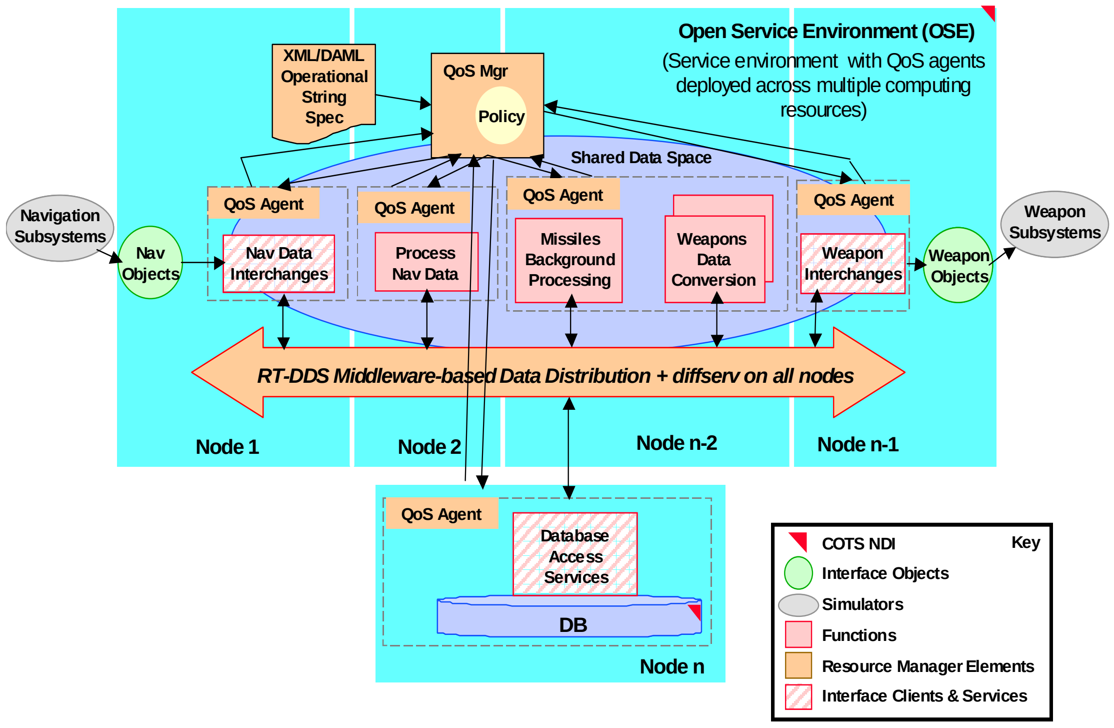


Figure 14 - Navy Weapons Alignment Application

In this application, it is critical that data be delivered within specified deadlines so that the alignment operations can take place in time to get weapons prepared for deployment. Further, the data that is received by the weapons subsystems must be temporally valid. Otherwise, the weapons may end up being aligned according to old navigation data.

This application is static in the sense that all of the components have well-known and stable parameters, such as execution time, period and deadline. Also, the number of components in the system remains the same. That is, it is known a priori how many, and which weapons subsystems will require the navigation data, and when.

Presently, this type of application uses point-to-point communication to send the data along the chain. This is very inflexible since whenever new components are inserted, new direct communications must be added. For example, if more than one weapon subsystem requires the navigation data (i.e. missiles and torpedoes), there would be the need to set point-to-point communication from the *Process Nav Data* component to each of the *Missiles Background Processing* components. Using a decoupled data distribution mechanism we describe in this work, allows for more flexibility in terms of where the data is sent. The data distribution mechanism would allow components to specify the data that they can provide, and the data that they require, and the delivery of the data would be handled by the data distribution. All these make this system a very good real life set up to demonstrate applicability of our algorithm and implementation.

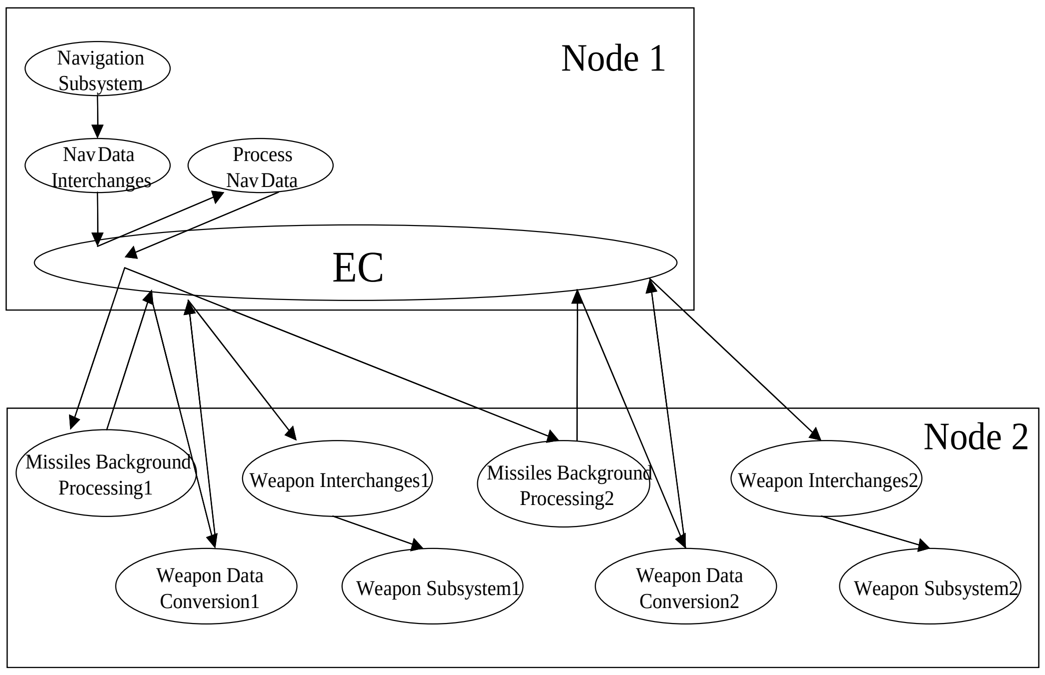


Figure 15 - Navy Weapons Alignment Application Simulation

Figure 15 illustrates how we have simulated the system. Again, we use two nodes, with the shared navigational components and the event channel on Node 1 and the specific weapons components on Node 2. In this implementation, we have implemented two different weapons systems, each with its own final deadline. Table 2 shows the parameters that we used to simulate this application. The object validity of the navigation data being distributed is 800,000 μsec. The values in the table are representative of the numbers for the real application.

Name	**Period, μsec**	**Release, μsec**	**Deadline, μsec**	**Exec time, μsec**
NavigationSubsystem	500,000	0	300,000	100,000
NavDataInterchanges	500,000	300,000	350,000	5,000
EC1	500,000	300,000	350,000	400
ProcessNavData	500,000	300,000	350,000	5,000
EC2	500,000	300,000	350,000	400
WeaponBackground Processing1	500,000	300,000	350,000	5,000
EC3_1	500,000	300,000	350,000	400
WeaponData Conversion1	500,000	300,000	350,000	5,000
EC4_1	500,000	300,000	350,000	400
WeaponInterchanges1	500,000	300,000	350,000	5,000
MissilesBackground Processing2	500,000	300,000	450,000	5,000
EC3_2	500,000	300,000	450,000	400
WeaponData Conversion2	500,000	300,000	450,000	5,000
EC4_2	500,000	300,000	450,000	400
WeaponInterchanges2	500,000	300,000	450,000	1,000
WeaponSubsystems1	500,000	650,000	150,000	10,000
WeaponSubsystems2	1,000,000	750,000	300,000	10,000

Table 2. Navy Weapons Alignment Application Simulation Parameters

The use of the JITDD algorithm for this model was slightly different from its use in the more generic models described above. In this application, the data is sent through the *Navigation Subsystem*, the *Nav Data Interchanges*, and the *Process Nav Data* components in a single path. However, because there are two weapon systems that require the processed navigational data at the end of the chain of components, the path splits. Thus, each weapon system will have a deadline by which it must receive the data, and the delivery of data through the path must meet that deadline. For example, the deadline for *Weapon Subsystem 1* is 150,000 μsec, and the deadline for *Weapon Subsystem 2* is 300,000 μsec. The JITDD algorithm was applied to determine the deadline for the delivery of this data to each weapon subsystem. However, because the original data flows from the same source, there must be a single deadline placed on the receipt of the data at the *Process Nav Data* component, the point where the path splits. This deadline was computed by taking the shorter of the two computed deadlines for the Weapon Subsystems.

Test Results. Here we describe the results of the test scenarios specified above. Again the main metrics of each of these scenarios are deadlines, and data temporal validity. The offline analysis has indicated that each of the scenarios is schedulable, and Theorems One and Two specify that all data that is used is temporally consistent. These test results are meant to demonstrate that the implementation does indeed meet the expected theoretical results. For each of the first three test scenarios, we ran the system over 100 periods of the data source and collected deadline and temporal consistency data. We ran each test 10 times and graphed the maximum completion time/data age values over these 10 tests. The results are displayed in the graphs of Figures 17-24, and explained in detail below.

Scenario 1 – Normal Conditions: Figures 16-17 show the results of these tests. Figure 17 displays the deadline results, one box for each of the data sources. The horizontal line in each graph indicates the deadline for the distribution of the particular data source. The other points in the scatter graph represent the completion times of the data distributions over the 100 periods. As the figure indicates, except for a few statistical anomalies in the first few periods, all of the data distributions complete before the specified deadline, as the theoretical results had predicted. In the first few periods, there may have been some set up execution that caused the tasks to complete after the deadline.

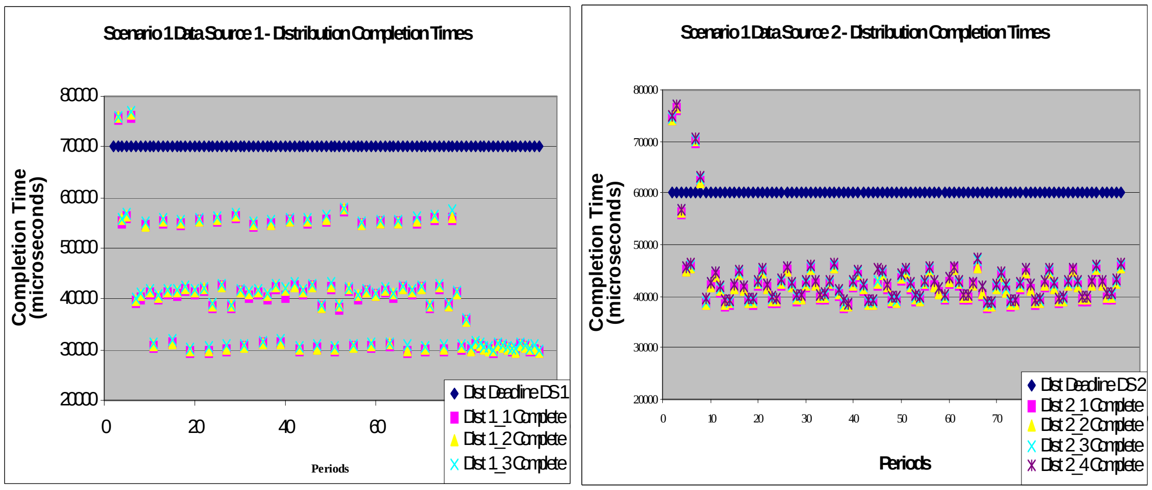


Figure 16. Scenario 1. Distribution Completion Time vs. Deadline

Figure 18 shows the temporal consistency results for scenario 1, one graph for each data source. The horizontal line in each graph represents the object validity of the data object being distributed. The other points in the scatter chart represent the ages of the data objects at the time they were read by the targets. It is clear to see that all of the targets, in each of the periods run, read temporally consistent data.

<u>Scenario 2 – Workload Constrained:</u> Figures 18 and 19 show the results of the Scenario 2 tests.

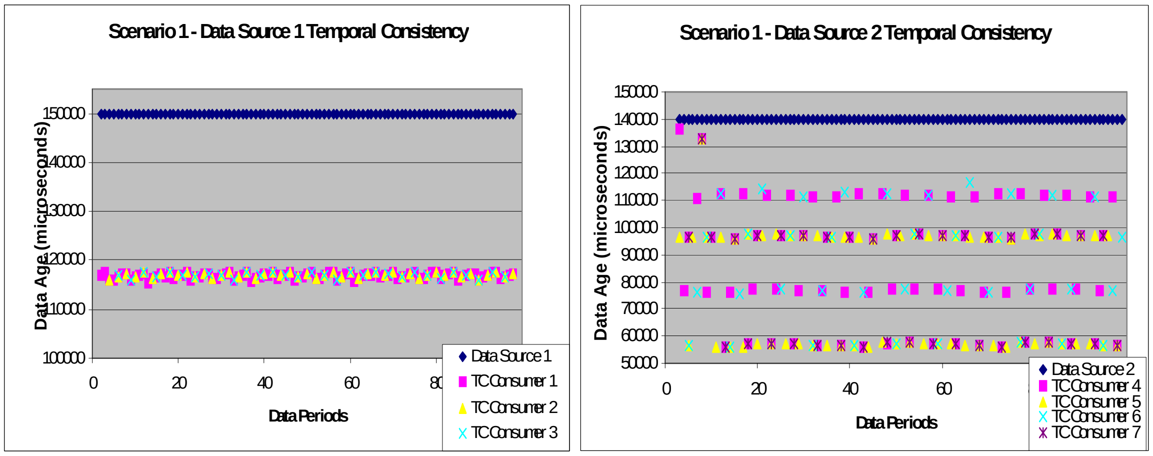


Figure 17. Scenario 1. Temporal Consistency of Data Sources

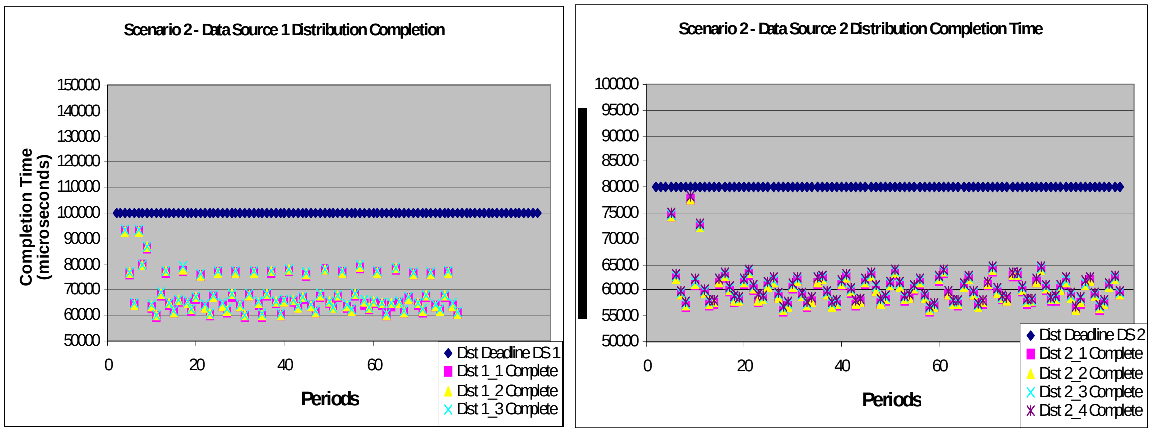


Figure 18. Scenario 2. Distribution Completion Time vs. Deadline

Again, we see that in Figure 18, the deadlines of the distributions are met for each of the periods over which the system was run. Figure 19 indicates that, aside from one statistical anomaly, the data temporal consistency was maintained for the data objects, for each period.

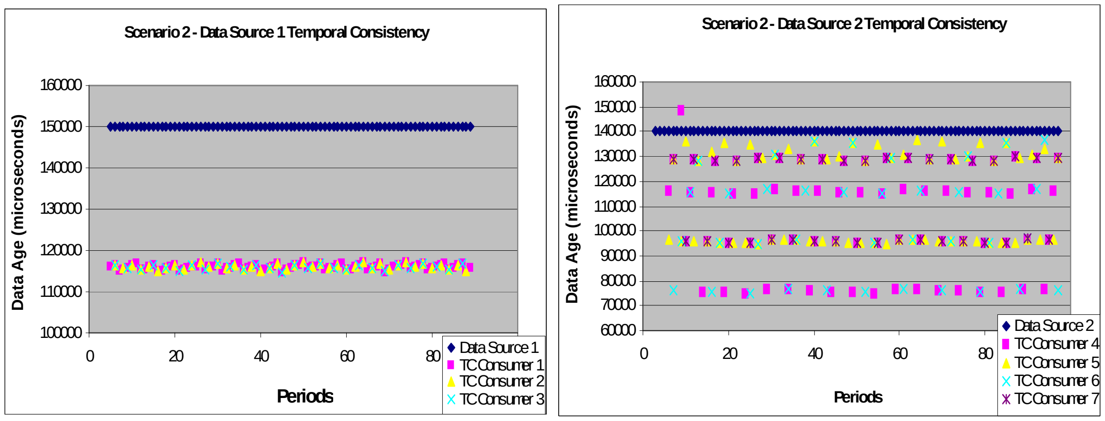


Figure 19. Scenario 2. Temporal Consistency of Data Sources

<u>Scenario 3 – Navy Weapon Alignment Application:</u> For scenario 3, we have run the system over 100 periods of the *Navy Subsystem* component, 10 times. We graphed the maximum values for the completion times of the two *Weapon Subsystems*, and for the object validity of the data arriving at the two *Weapon Subsystems* components.

Figures 20 and 21 show the results these tests. From the figures we can see that our computed deadlines are met each time, and the temporal validity of the data is preserved as well.

The work described in this section was published in [6].

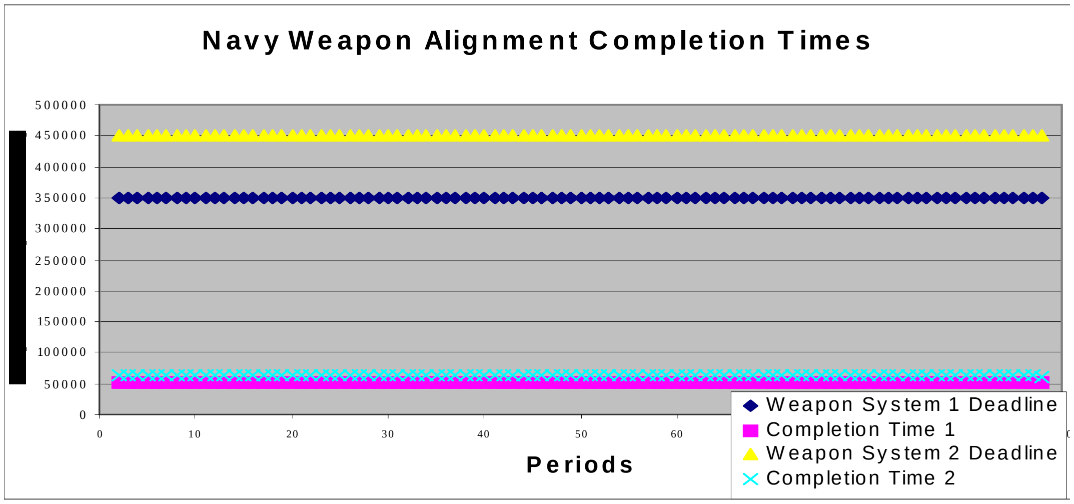


Figure 20 - Navy Weapons Alignment Simulation.
Distribution Completion Time vs. Deadline

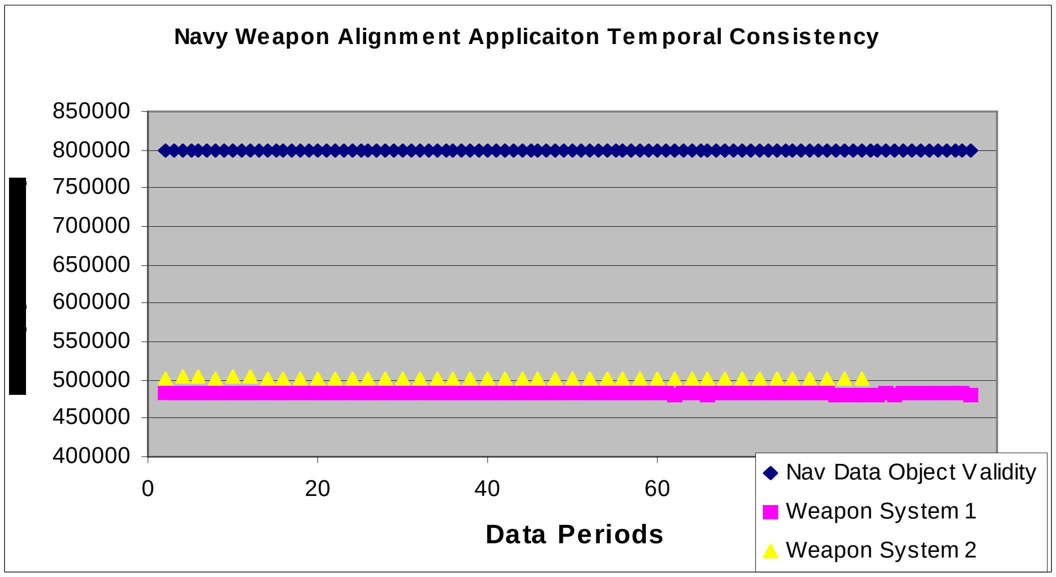


Figure 21 - Navy Weapons Alignment Simulation.
Temporal Consistency of Data Source

1. "Rapid RMA," TriPacific Software, Inc., www.tripac.com, accessed March, 19th, 2014.

2. Liu, J.W.S., Real-Time Systems, Prentice-Hall, June 2000.

3. Murphy, M., Bryan, K., "CORBA 1.0 Compliant Static Scheduling Service for Periodic Tasks Technical Documentation," URI Technical Report TR04-297, University of Rhode Island, January 2004.

4. Schmidt, D.C., "Real-time CORBA with TAO (The ACE ORB)," Washington University at St. Louis, 12 November 2013, http://www.cs.wustl.edu/~schmidt/TAO.html, accessed 19 March 2014.

5. Harrison, T.H., Levine, D.L., Schmidt, D.C., "The Design and Performance of a Real-time CORBA Event Service," in Proceedings of the Object-Oriented Programming Systems, Languages & Applications Conference (OOPSLA'97), October 1997, Atlanta, GA.

6. Uvarov, A., DiPippo, L., Fay Wolfe, V., Bryan, K., Gadrow, P., Henry, T., Murphy M., Work, P.R., DiPalma, L.P., "Static Real-Time Data Distribution," in Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), May 2004, Toronto, Canada

7. The Raytheon Company, www.raytheon.com, accessed March, 19th, 2014

CHAPTER 5

DYNAMIC RTDD

This chapter presents our work on real-time dynamic data distribution that includes description of system design, implementation, and evaluation.

5.1 System Design

In a dynamic system where data sources and data readers may come and leave at any time, all computation and analysis has to be performed on-line. This type of system imposes different requirements on system performance and as a result on its architecture. Our proposed *Timely Data Distribution Service* (TDDS) system architecture for dynamic systems is presented in Figure 22.

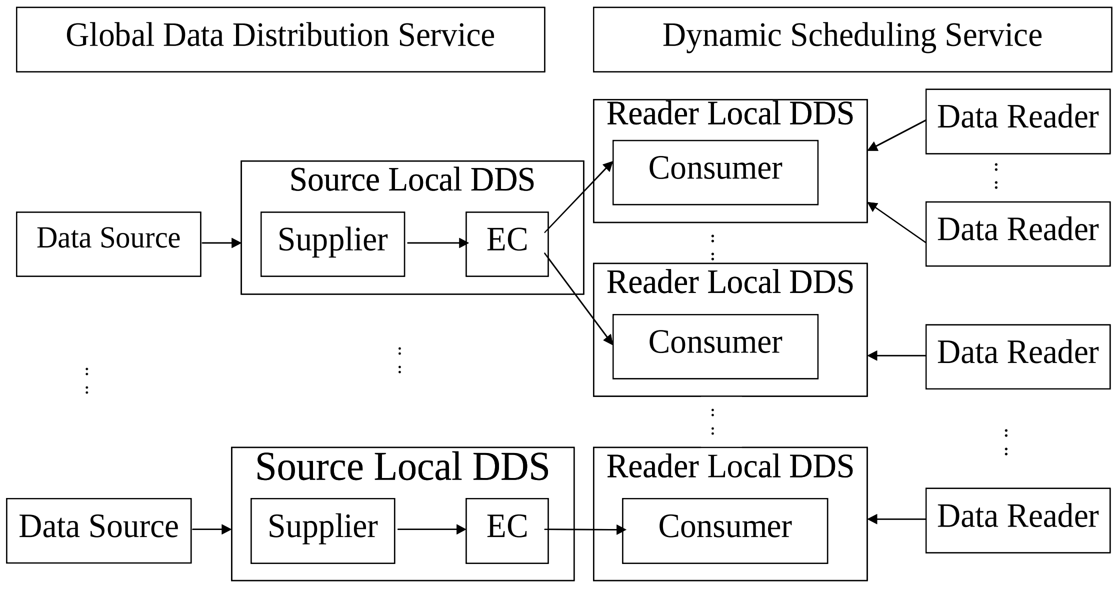


Figure 22. TDDS System Architecture

As the figure shows, the main components of the system are as follows:

DataSource and *DataReader* are the applications analogs to those in a static system.

Source and *Reader Local Data Distribution Service*s (*Source/ReaderLocalDDS*) are the local objects residing at the same nodes as the data producing and data consuming applications and serve as the entrance points of the *Data Distribution Service*. These local DDSs are responsible for *DataSource* and *DataReader* registrations, analysis of data distribution parameters and interactions with other parts of the system such as *GlobalDataDistributionService* and *DynamicSchedulingService* to achieve system goals and actual data distribution.

The Real-Time Event Service (RT ES) is an internal to the DDS data distribution mechanism, responsible for actual data distribution.

The *Global DDS* is used by the *ReaderLocalDDS*s to find the *SourceLocalDDS* associated with the data requested by *DataReader* application. During *DataSource* registration, *SourceLocalDDS* registers itself with *Global DDS* with association to data provided by *DataSource*. Then, this information is used by *ReaderLocalDDS* to locate the appropriate *SourceLocalDDS*.

The *Dynamic Scheduling Service* (DSS) is responsible for system schedulability analysis and priority assignments for all tasks in the systems.

Figures 23 and 24 present components collaboration in our real-time data distribution framework. This collaboration can be split into two phases: the Set-Up phase and the Run-Time phase. *Data Source* Set-Up includes the following steps (the numbers in the steps described below correspond to the numbered events in figures 23 and 24):

1 - *Data Source* connects to the *Scheduling Service* to schedule its own activity on the node.

2 - If schedulable, *Data Source* registers to the *Source Local DDS*.

3, 4, 5 - *Source Local DDS* creates an Event, Supplier and Event Channel.

6 - *Local DDS* requests *DSS* to schedule an event.

7 - If the event is schedulable, *Source Local DDS* registers with the *Global DDS*.

At this point, the data source part is all set and is ready to distribute data.

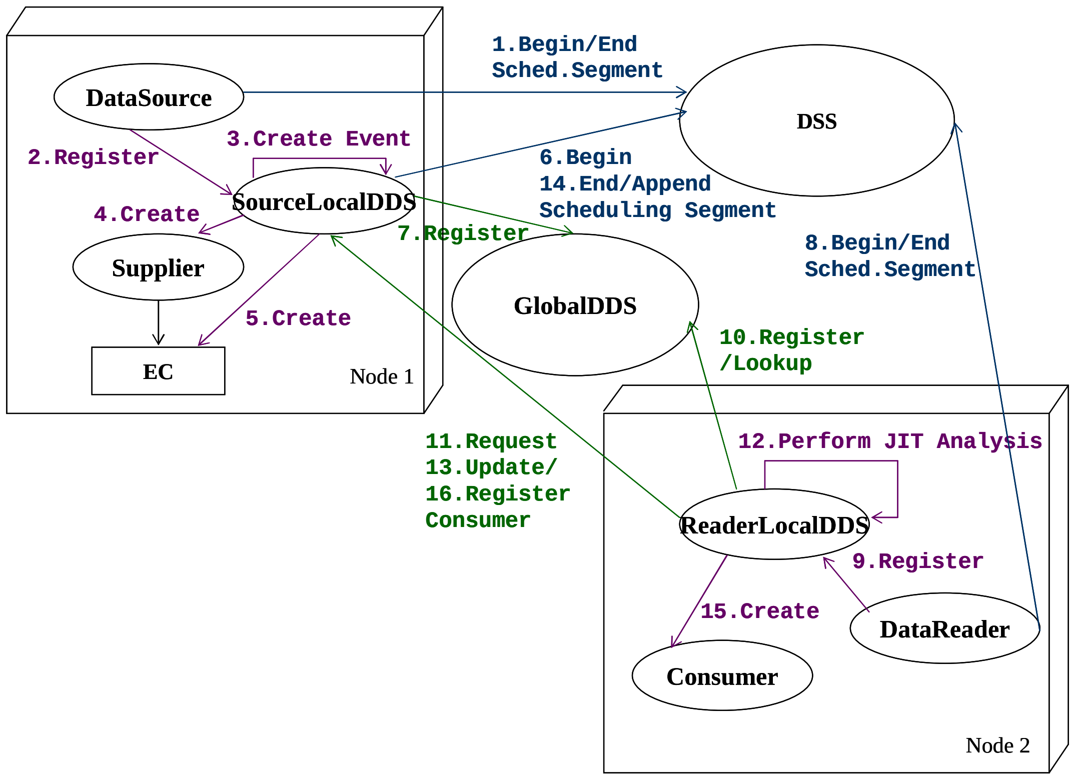


Figure 23. Components Collaboration in TDDS Framework (Set-Up Phase)

The Set-Up phase for *Data Reader* includes the following steps.

8 - *Data Reader* request *DSS* to schedule its own activity.

9 - If schedulable, *Data Reader* registers with the *Reader Local DDS*.

10 - *Reader Local DDS* registers new *Data Reader* to the *Global DDS*. If there is no local consumer for the requested data, the *Reader Local DDS* looks up the *Global DDS* for an available *Source Local DDS*.

11. *Reader Local DDS* creates consumer.

12. *Reader Local DDS* requests Supplier information from the *Source Local DDS*.

13. *Reader Local DDS* performs Just-In-Time analysis for a new *Data Reader*.

14. *Reader Local DDS* updates Supplier information for the *Source Local DDS*.

15. *Source Local DDS* requests *DSS* to schedule new distribution and registers new Consumer.

16. If the new distribution is schedulable, the *Source Local DDS* registers new Consumer with the Event Channel.

This is the end of the Set-Up phase

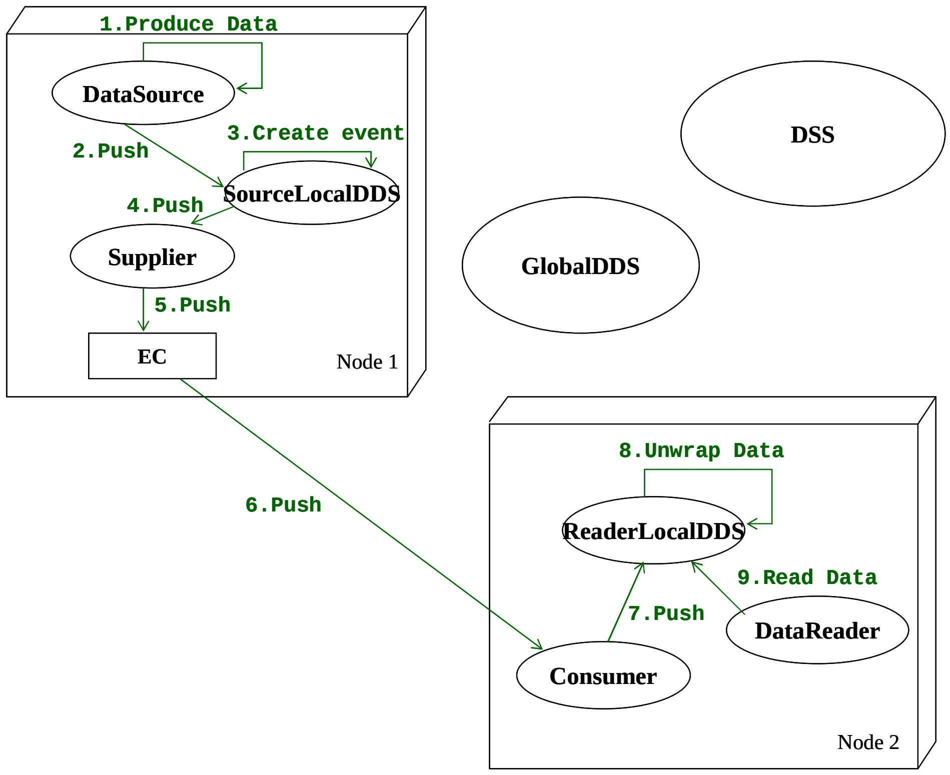


Figure 24. Component collaboration. Run-time Phase.

The Run-Time phase depicted in Figure 24 performs the actual data distribution in the following order.

1,2 - *Data Source* produces data and writes it to the *Source Local DDS*.

3,4 - The *Source Local DDS* wraps the data into event and pushes it to the *Supplier*.

5 - *Supplier* pushes data into the *Event Channel*.

6 - *Event Channel* pushes it to all of its *Consumers*.

7 - Each *Consumer* then pushes data to its *Local Reader DDS*.

8 - The *Reader Local DDS*s un-wrap the data and makes it available for their *Data Readers*.

9 - *Data Readers* access the data according to their own needs.

5.2 System Implementation

The whole system is developed upon the Real-Time ORB in TAO [1]. The Real Time Data Distribution Service framework, excluding scheduling and Just-in-Time analysis interfaces was implemented as part of a Master's Thesis project [2], the system analysis and design, though, were part of this work. The major components of the system and their collaboration are described below.

5.2.1. Components and Use Cases Implementation.

The following two subsections describe all the system's components and their actions during set-up and run-time phases. In comparison to a static system, in a dynamic system, this differentiation is, of course, arbitrary, since components enter and leave the system during run-time. We use these two phases just for separation of

Data Readers and Data Sources registration and connection from the actual data distribution.

5.2.1.1 Set-Up Phase

In the set-up phase, new incoming *Data Sources* and *Data Readers* are introduced, scheduled, and based on schedulability result, registered to the system. The components and their collaborations are as follows.

Global DDS, the keeper of a system-wide repository for event entities, is implemented as a wrapper around CORBA Naming Service. For the purpose of reduction of network communication, it is designed as distributed agents between *Local DDS* and *Naming Service*, residing on each network node.

Source Local DDS is implemented as a multi-threaded server, with *Supplier* and *Event Channel* on each of the threads. To decrease a run time overhead instead of being created when a new *Data Source* is registering to the system, *Suppliers* and *Event Channels* for each type of event are created ahead of time and are kept running.

Reader Local DDS uses the same thread model as *Source Local DDS*. It stores and updates data each time the *Consumer* pushes a new event.

Data Source. After registering to *Reader Local DDS* during the set-up phase, the *Data Source* periodically wraps application data into an internal data structure set and pushes data to its *Local DDS*.

Data Reader performs two tasks. It registers with its *Local DDS* and then periodically reads the data from it.

Some of the interfaces for the above components were developed based on the following four cases of usage: Data Source Registration and Unregistration, and Data Reader Registration and Unregistration.

Case of Data Source Registration (See Figure 25). Upon coming into the system, *Data Source* registers to *Real Time Dynamic Scheduling Service*, then to *Reader Local DDS*. After that, *Reader Local DDS* creates an end-to-end task, representing the producing end of data distribution and schedules it with *RTDSS*. If scheduled, the source *Local DDS* registers a new event with *Global DDS* and requests a list of *Reader Local DDS*s waiting for this event, to inform them of the event's availability.

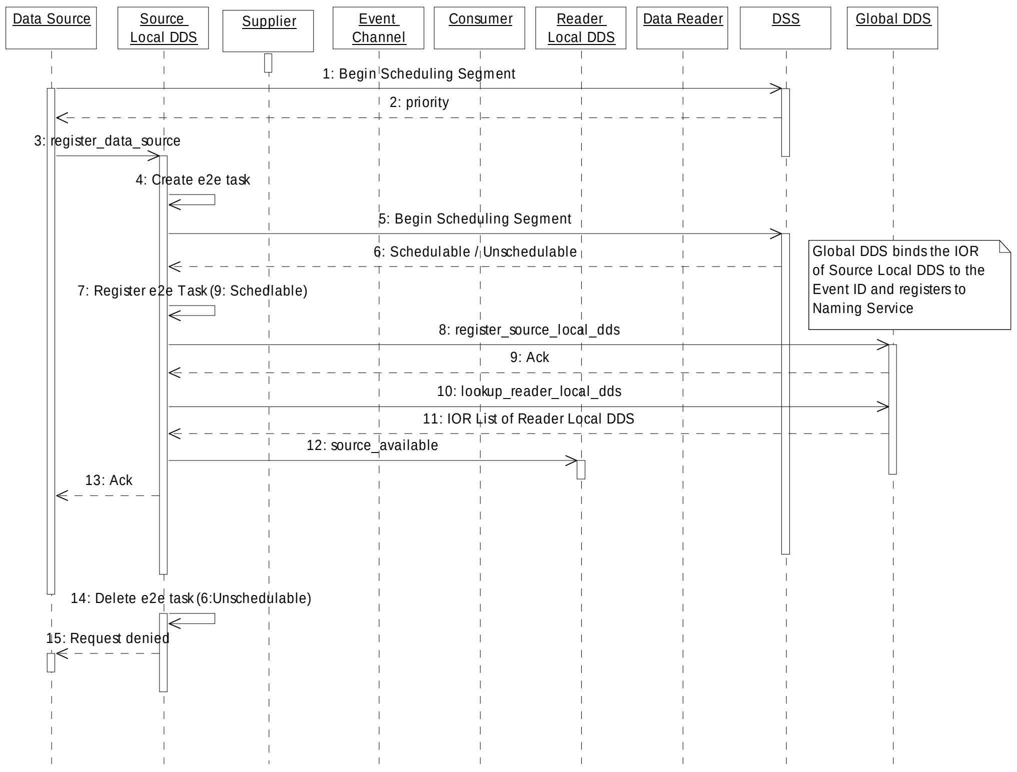

Data Source
Source Local DDS
Supplier
Event Channel
Consumer
Reader Local DDS
Data Reader
DSS
Global DDS
1: Begin Scheduling Segment
2: priority
3: register_data_source
4: Create e2e task
5: Begin Scheduling Segment
6: Schedulable / Unschedulable
Global DDS binds the IOR of Source Local DDS to the Event ID and registers to Naming Service
7: Register e2e Task (9: Schedlable)
8: register_source_local_dds
9: Ack
10: lookup_reader_local_dds
11: IOR List of Reader Local DDS
12: source_available
13: Ack
14: Delete e2e task (6:Unschedulable)
15: Request denied

Figure 25. Data Source Registration Use Case

Case of Data Source Unregistration (See Figure 26). When a *Data Source* deactivates, it unregisters itself from the *Source Local DDS* and *RT DSS*. After that, the *Source Local DDS* associated with the *Data Source* will unregister the end-to-end task (distribution) from the *RTDSS* and unregister itself from the *Global DDS*. Then it will request the list of *Reader Local DDSs* receiving this data, to inform them of the source unavailability. Once that is executed, all involved *Reader Local DDSs* will deactivate their corresponding consumers.

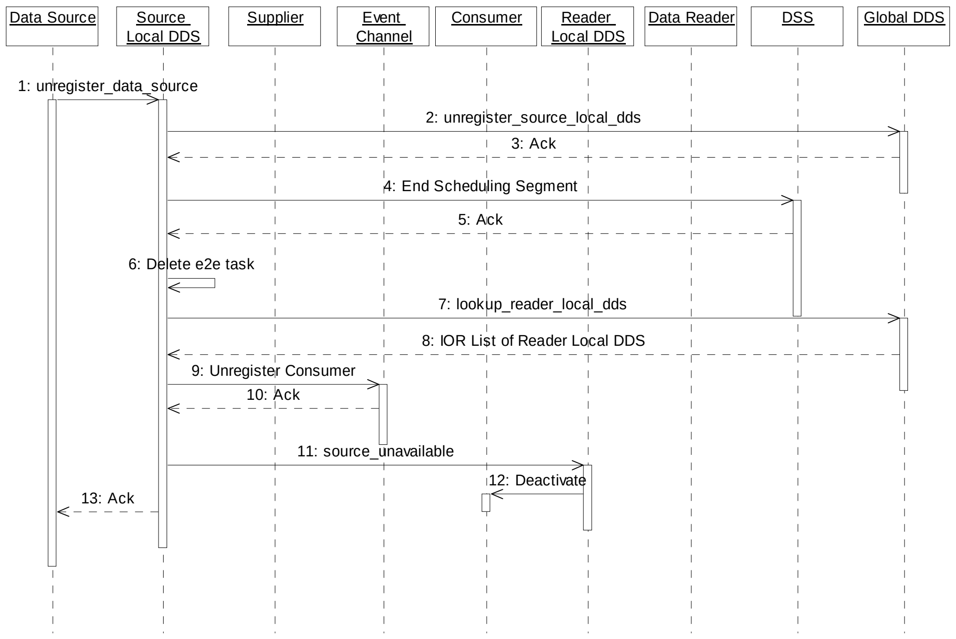
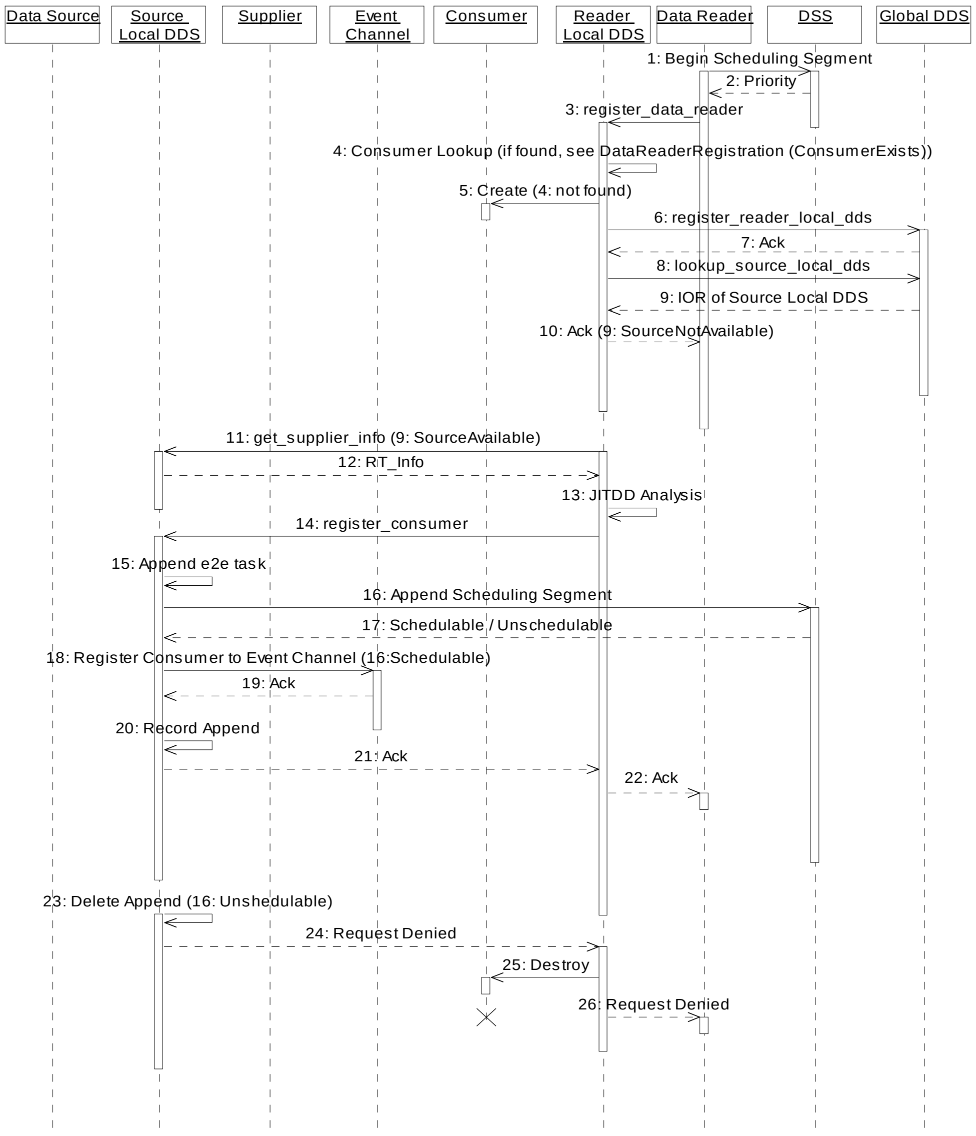


Figure 26. Data Source Unregistration Use Case.

Cases of Data Reader Registration (See Figures 27 and 28). There are two scenarios in this case. In the first, general case, when a new *Data Reader* comes into the system, it registers with *RT DSS* and then with *Reader Local DDS*.



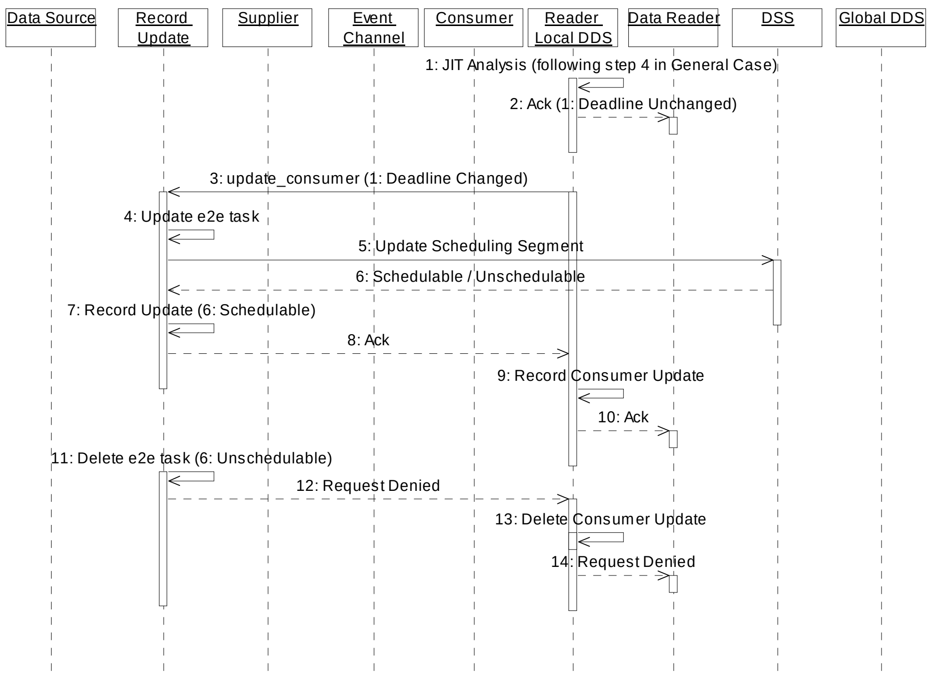
Data Source
Source Local DDS
Supplier
Event Channel
Consumer
Reader Local DDS
Data Reader
DSS
Global DDS
1: Begin Scheduling Segment
2: Priority
3: register_data_reader
4: Consumer Lookup (if found, see DataReaderRegistration (ConsumerExists))
5: Create (4: not found)
6: register_reader_local_dds
7: Ack
8: lookup_source_local_dds
9: IOR of Source Local DDS
10: Ack (9: SourceNotAvailable)
11: get_supplier_info (9: SourceAvailable)
12: RT_Info
13: JITDD Analysis
14: register_consumer
15: Append e2e task
16: Append Scheduling Segment
17: Schedulable / Unschedulable
18: Register Consumer to Event Channel (16:Schedulable)
19: Ack
20: Record Append
21: Ack
22: Ack
23: Delete Append (16: Unshedulable)
24: Request Denied
25: Destroy
26: Request Denied

Figure 27. Data Reader Registration Use Case

Then, the *Reader Local DDS* creates a consumer and looks up the *Global DDS* for an available *Source Local DDS*. If there is no *Data Source*, the *Reader Local DDS* returns the notification.

If the *Data Source* is available, the *Reader Local DDS* calls upon the *Source Local DDS* to get the *Data Source* information to perform Just-In-Time analysis. After that, the *Reader Local DDS* registers the new consumer to the *Source Local DDS*.

The *Source Local DDS* in its turn adds the new *Consumer* to the corresponding End-to-End task and calls upon *RT DSS* to schedule it. If schedulable, the *Consumer* is registered to the *Event Channel* and everything is ready for the data transfer. Otherwise the *Source Local DDS* denies the request for *Consumer* registration and returns Request Deny back to the *Reader Local DDS*, which in turns destroys the *Consumer*.



Data Source
Record Update
Supplier
Event Channel
Consumer
Reader Local DDS
Data Reader
DSS
Global DDS
1: JIT Analysis (following step 4 in General Case)
2: Ack (1: Deadline Unchanged)
3: update_consumer (1: Deadline Changed)
4: Update e2e task
5: Update Scheduling Segment
6: Schedulable / Unschedulable
7: Record Update (6: Schedulable)
8: Ack
9: Record Consumer Update
10: Ack
11: Delete e2e task (6: Unschedulable)
12: Request Denied
13: Delete Consumer Update
14: Request Denied

Figure 28. Data Reader Registration (Consumer Exists) Use Case

The second scenario is applied when *Consumer Look Up* reveals its existence at step 4 of the general case. Here, *Just-In-Time* analysis is performed to compute a new deadline for *Consumer*. Then if the computed deadline is bigger than the existing *Consumer's* deadline, the *Consumer* will continue to perform on existing conditions, and the/a new *Reader* will get valid data. If the new deadline is less than the existing, the *Reader Local DDS* calls upon the *Source Local DDS* to modify deadline parameters on the corresponding *End-to-End task* and schedule it with *RT DSS*. If schedulable, the *Source Local DDS* records the update, otherwise the update is deleted and request is denied.

Case of Data Reader Unregistration (See Figure 29). When a *Data Reader* leaves the system, it unregisters itself with the *Reader Local DDS*. The *Reader Local DDS* calls the *Just-in-Time* block to check if the *Consumer* deadline will change when the *Data Reader* leaves. Based on the result, we observe three possible scenarios.

In the first scenario, the deadline is unchanged (*Reader's* deadline is longer than *Consumer's*). Nothing needs to be done. (Figure 29, Step 3)

In the second scenario, when the leaving *Reader's* deadline was the shortest, the *Reader Local DDS* call the *Just-in-Time* block to compute a new deadline for the *Consumer*. Then it calls the *Source Local DDS* to update the *Consumer's* information. The *Source Local DDS* updates the *End-to-End* task and calls *RT DSS* to adjust the system. (Figure 29, Step 5)

In the third scenario, we consider the case when the leaving *Data Reader* is the last requestor of data from the *Consumer*. In this case, the *Reader Local DDS* unregisters the *Consumer* from the *Source Local DDS*. The *Source Local DDS* updates the *End-*

to-End task, calls *RT DSS*, and unregisters the *Consumer* from the *Event Channel*. (Figure 29, Step 13)

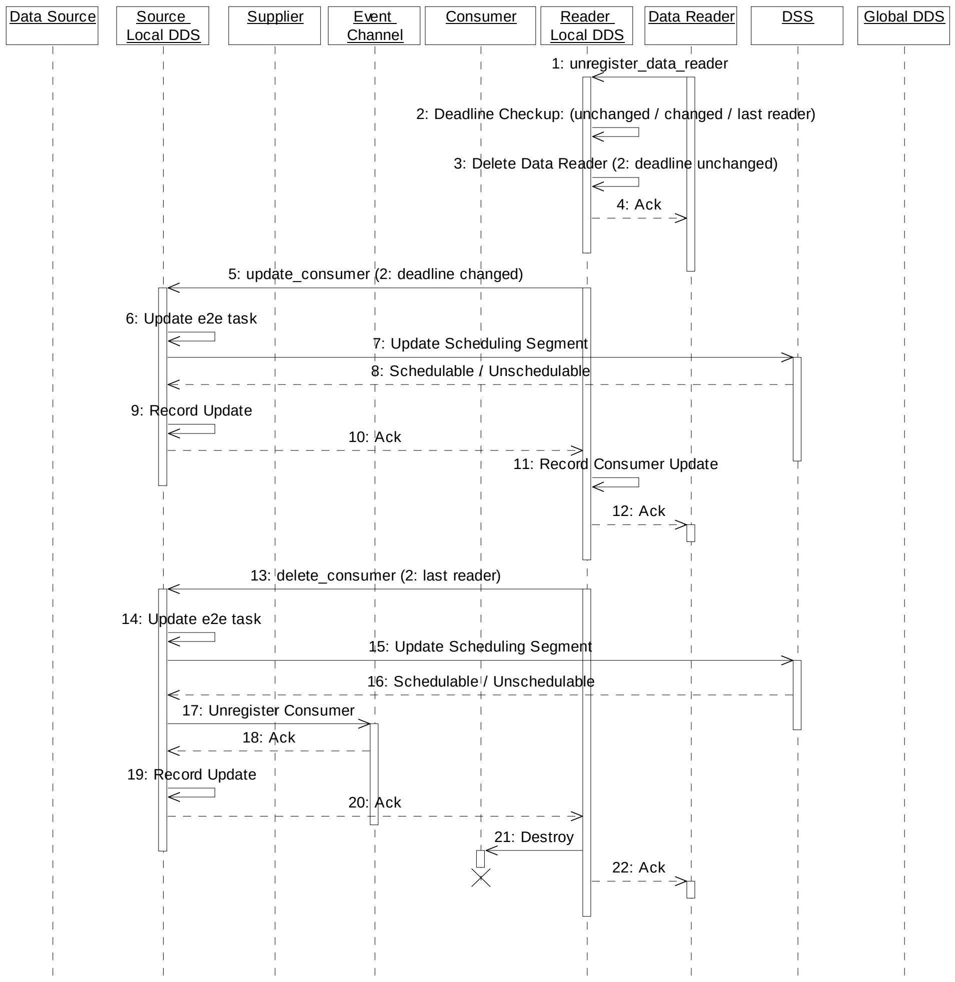

Data Source
Source Local DDS
Supplier
Event Channel
Consumer
Reader Local DDS
Data Reader
DSS
Global DDS
1: unregister_data_reader
2: Deadline Checkup: (unchanged / changed / last reader)
3: Delete Data Reader (2: deadline unchanged)
4: Ack
5: update_consumer (2: deadline changed)
6: Update e2e task
7: Update Scheduling Segment
8: Schedulable / Unschedulable
9: Record Update
10: Ack
11: Record Consumer Update
12: Ack
13: delete_consumer (2: last reader)
14: Update e2e task
15: Update Scheduling Segment
16: Schedulable / Unschedulable
17: Unregister Consumer
18: Ack
19: Record Update
20: Ack
21: Destroy
22: Ack

Figure 29. Data Reader Unregistration Use Case.

5.2.1.2 Run-Time Phase

After completion of registration, Data Sources are ready for periodic data updates, and Data Readers are ready for their periodic data consuming. The case of Data Distribution, the one we associate with the run-time phase, is presented in Figure 30.

Case of Data Distribution. The *Data Source* produces data, wraps it into *Internal Data Structure*, and pushes it to the *Source Local DDS*. The *Source Local DDS* wraps the data into *Event* and pushes it to *Supplier*. The *Supplier* pushes it to *Event Channel* and *Event Channel* to all its *Consumers*. *Consumers* push data to their respective *Reader Local DDSs*. Each *Reader Local DDS* unwraps the data from the event and stores it internally, making it accessible to their *Data Readers*. The *Readers* then check the data's time stamp and validity to determine its freshness. If a *Reader* keeps reading the same old data, it is a sign that there is no *Data Source* providing the data. The reader application then may choose to continue to read with the same interval, to increase the reading interval, or to stop reading.

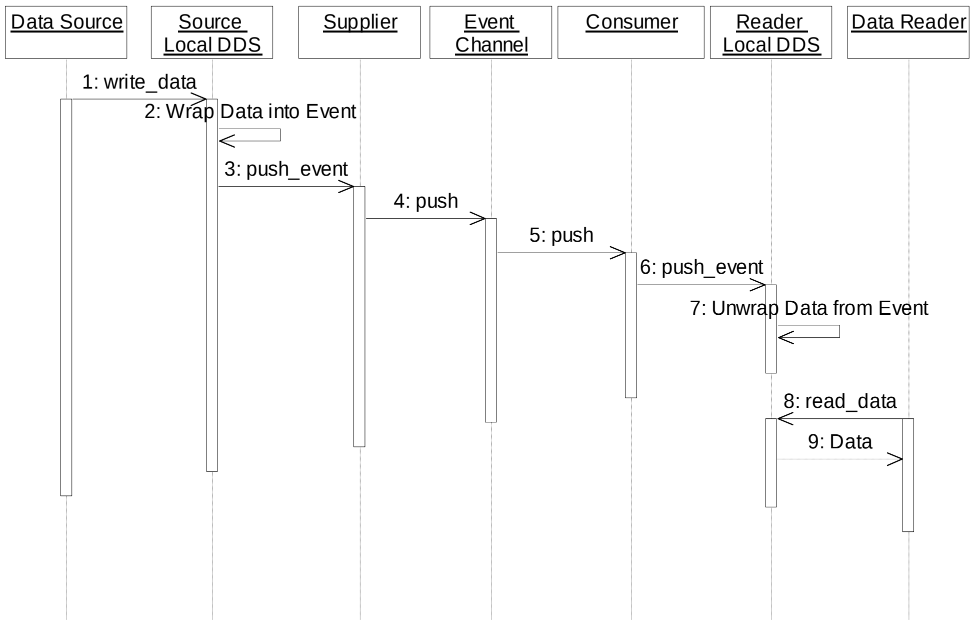


Figure 30. Data Distribution Use Case

5.2.2 Major Data Structures.

There are four major data structures in our implementation.

To provide real-time constraints, the *Data Source* wraps the data into an internal data structure called *Data_Set_t* (Figure 31). During each update, it also stamps the data with the time it was updated. This time stamp along with the data validity time is used by the *Data Reader* to ascertain whether data is still valid at the time of reading.

```
Struct Data_Set_t
{
   EventID_t eid;
   Data_t data;
   TimeType_t validity;
   TimeType_t lastupdate;
};
```

Figure 31. Internal Data Structure

The real-time information structure Rt_Info_t (refer to Figure 32) is used to provide real-time constraints of all major components in the system (Data Sources, Data Readers, Source/Reader Local DDSs, Consumers, Suppliers, Event Channels) for use in JIT computation and in building end-to-end distribution tasks.

```
Struct Rt_Info_t
{
  /// A user define name of the entity
  string name;

  ///The entity’s IOR. Can be null if not a servant.
  IOR_t ior;

  /// The network ID of the computer the entity resides.
  NodeID_t nid;

  ///The event ID that the entity is associated with.
  EventID_t eid;

  TimeType_t period;
  TimeType_t release;
  TimeType_t deadline;
  TimeType_t exec_time;
  TimeType_t validity;
};
```

Figure 32. Real-Time Information Structure.

The *Subtask* structure (refer to Figure 33) is used to keep all real-time info of tasks involved in end-to-end data distribution. This information is used by *RT DSS* to compute all intermediate deadlines and to assign priority to the tasks in the system. This structure is defined as a recursive structure to accommodate the non-linear nature of the data distribution task. (We could also argue that a non-linear task is a more general approach to the end-to-end task presentation, while a linear task is just a basic variation). Along with common real-time parameters, the structure also includes resource usage information (acquisition and deacquisition time) and parameters specific to *RT DSS*.

```
struct Subtask_t
  {
    /// A user defined name of the entity.
    string name;

    /// The network ID of the computer the entity resides
    NodeID_t nid;

    TimeType_t period;
    TimeType_t phase;
    TimeType_t deadline;
    TimeType_t exec_time;

    /// Resources used by the task
    ResourceUsageSet_t resources;

    ///Tasks successors
    sequence<Subtask_t> subtasks;
   };
```

Here `ResourceUsageSet_t` is the list of `ResourceUsage_t` structures, where

```
struct ResourceUsage_t
  {
        string name;
        TimeType_t acqTime;
        TimeType_t deacqTime;
  };
```

Figure 33. Subtask Structure.

The *End2EndTask* structure (refer to Figure 34) is used for definition of actual data distribution, that starts at the *Source Local DDS* and ends at the *Reader Local DDS*. It stores real-time information of all the subtasks involved in the chain, and end-to-end parameters of the task itself. The *RT DSS* uses this information to compute intermediate deadlines of involved subtasks, to perform schedulability analysis, and to assign priority for task's execution.

```
struct End2EndTask_t
{
   /// A user defined name of the entity.
   string name;

   /// The event ID that the entity is associated with.
   EventID_t eid;

   TimeType_t period;
   TimeType_t release;
   TimeType_t deadline;
   Iportance_t importance;
   TimeType_t exec_time;

   /// Set of subtasks
   sequence <Subtask_t> subtasks;
};
```

Figure 34. End-to-End Task Structure

5.2.3 Intermediate Deadlines Computation.

This section presents a description of our suggested approach for intermediate deadline computation in a non-linear distribution *End-to-End* task. Even though this considered to be the part of RT DSS project, we sought it would be beneficial to give our insights on the subject.

For an *End-to-End* task to complete before its deadline, all involved subtasks must complete before this deadline. Since, a task successor starts only after its task predecessor completes, the intermediate deadlines for all subtasks need to be assigned one after another within the end-to-end deadline. The original algorithm in *RT DSS* is accommodated to compute intermediate deadlines in a linear end-to-end task.

There were two approaches discussed for intermediate deadline computation. In the first approach (head-to-tail approach), the deadline assignment starts from the beginning of the *End-to-End* task. The deadline for the first subtask is defined by addition of the first subtask's execution time to the *End-to-End* task's release time, the deadline for the second subtask is subsequently defined by addition of the second subtask execution time to the first subtask deadline, and so on and so forth. In the second approach (tail-to-head), computation starts from the end of the *End-to-End* task. The last subtask deadline is assigned as the *End-to-End* task's deadline. The next to last subtask's deadline is defined by subtraction of last subtask's execution time from its deadline, and so on and so forth.

To illustrate these approaches let us considered the following example. Let *End-to-End* task *E2E* have period (*P*) and deadline (*D*) equal to 10, and its release (*R*) be at the beginning of its period. Let this task consist of 3 subtasks (*ST1*, *ST2*, *ST3*) with execution times (*ST1E, ST2E, ST3E*) equal to 2, 3, and 2, respectively. Then with the first approach, we assign intermediate deadlines *ST1D, ST2D, ST3D*, as follows:

$$\text{ST1D} = \text{R} + \text{ST1E} = 0+2 = 2$$

$$\text{ST2D} = \text{ST1D} + \text{ST2E} = 2 + 3 = 5 \text{ and}$$

$$\text{ST3D} = \text{D} = 10$$

With the second approach:

$$\text{ST3D} = \text{D} = 10$$

ST2D = ST3D – ST3E = 10 – 2 = 8

ST1D = ST2D – ST2E = 8 – 3 = 5

As we can see from Figure 35, all free (slack) time is allocated to the last subtask in the chain in the first approach, and to the first subtask in the second approach.

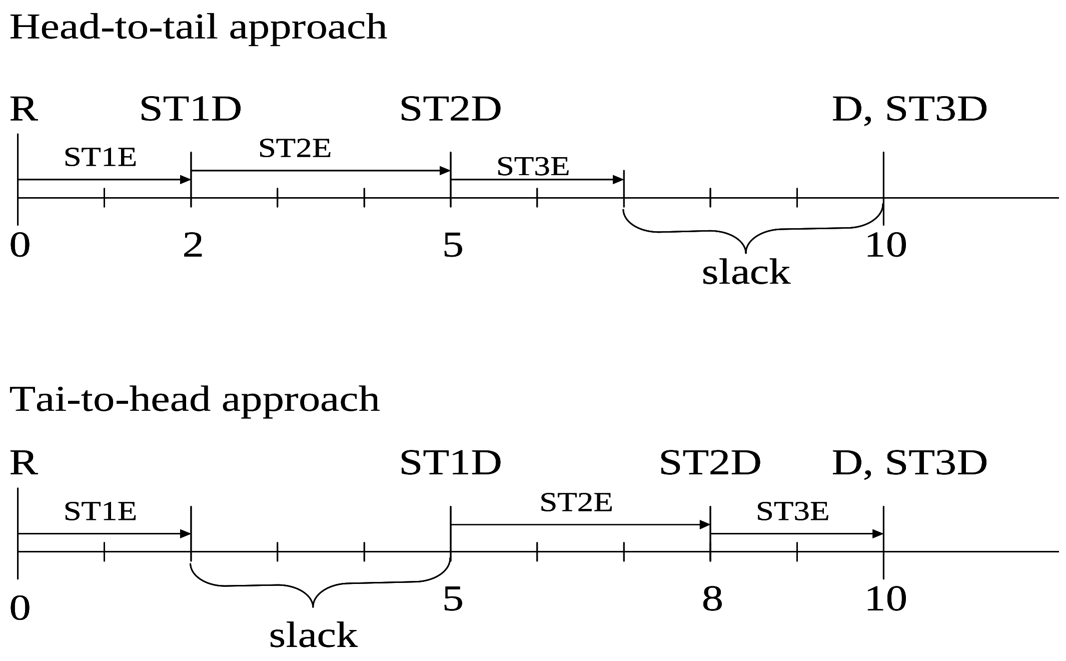


Figure 35. Intermediate Deadlines Assignment in RT DSS

Now let us consider the case of a non-linear distribution task *E2E* (refer to Figure 36), with the same period, deadline and release time as in the previous example. Let the subtasks be *ST1*, *ST2*, *ST3* and *ST4* where subtask *ST2* is a point of spawning. That is, at the end of execution of *ST2,* subtasks *ST3* and *ST4* start to execute simultaneously. Let their execution times be ST1E =2, ST2E =3, ST3E = 2 and ST4E = 3.

With the first approach, subtasks intermediate deadlines will be:

ST1D = R + ST1E = 0+2 = 2

ST2D = ST1D + ST2E = 2 + 3 = 5 and

ST3D = ST4D = D = 10

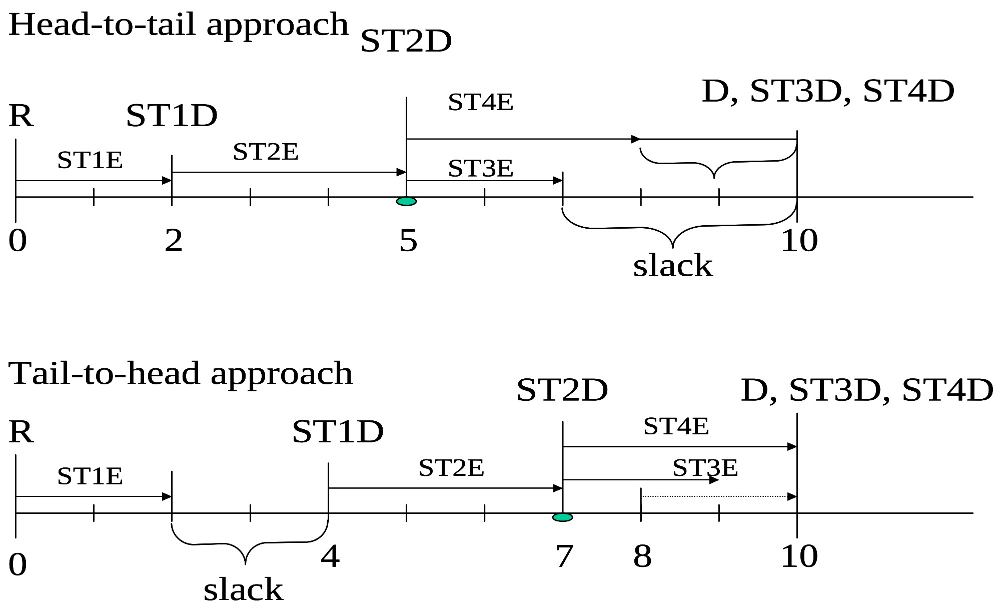


Figure 36. Intermediate Deadlines Assignment for Distribution Task

For the second approach, the algorithm needs to be modified a little. We start from the end of one branch, let us say *ST3*. Then:

ST3D = D = 10

ST2D = ST3D – ST3E = 10 – 2 = 8

Here, we need to take into account another branch:

ST4D = D = 10

ST2D = ST4D – ST4E = 10 – 3 = 7

For both subtasks *ST3* and *ST4* to complete before their deadlines, the deadline of their predecessor subtask *ST2* needs to be assigned as the shortest of these two. That is:

ST2D = 7, and now,

ST1D = ST2D – ST2E = 7 – 3 = 4

Here again the slack time is accumulated either at the last subtasks or at the first.

To spread this slack time more evenly, and hence to relax constraints along the chain, we propose to allocate tasks' deadlines in proportions to their execution times (proportional assignment). For that we need to compute *E2E* task execution time, again taking into account its non-linear nature. So for the branch constructed with subtask *ST3*, we have *E2E* execution time:

E = ST1E + ST2E + ST3E = 2 + 3 + 2 = 7

For the branch constructed with subtask *ST4*, have we have *E2E* execution time:

E = ST1E + ST2E + ST4E = 2 + 3 + 3 = 8

E2E execution time is assigned as the longest of these two. Therefore, for 8 execution time units we have 10 allocation time units, that is, for each execution unit we can assign 1.25 allocation units. With this, obviously, either Head-to-Tail or Tail-to-Head approach will lead to the same intermediate deadlines (refer to figure 37).

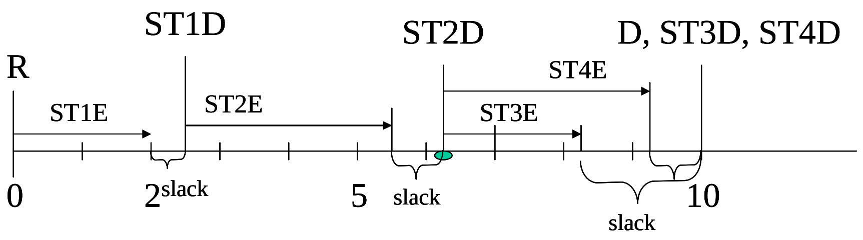
Head-to-Tail computation:

ST1D = R + ST1E * 1.25 = 0+2 * 1.25 = 2.5

ST2D = ST1D + ST2E * 1.25 = 2.5 + 3 *1.25 = 6.25 and

ST3D = ST4D = D = 10

Head-to-tail approach



Tail-to-head approach

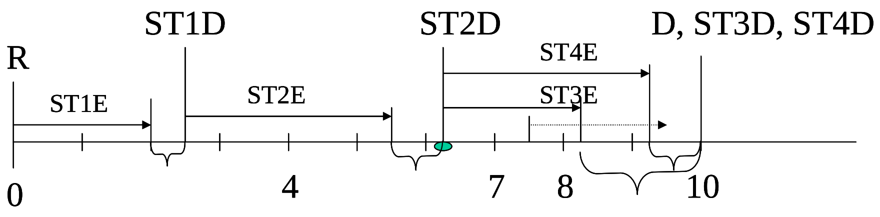


Figure 37. Proportional Intermediate Deadline Assignment

Tail-to-Head computation:

ST3D = D = 10

ST2D = ST3D – ST3E *1.25 = 10 – 2.5 = 7.5

Here again, we need to take into account another branch:

ST4D = D = 10

ST2D = ST4D – ST4E * 1.25 = 10 – 3.75 = 6.25

So, ST2D = 6.25 and,

ST1D = ST2D – ST2E * 1.25 = 6.25 – 3.75 = 2.5

The choice of computational approach in this case should be based on other parameters, such as the effectiveness of the implementation. In our substitute for RTDDS (see below) for intermediate deadlines computation we implemented Tail-to-Head approach.

5.3 System Evaluation

This section describes the empirical studies used/conducted to justify our approach of Dynamic Real-Time Data Distribution Service.

5.3.1 Experimental Platform

Middleware consists of *TAO Real Time ORB* and *TAO Real Time Event Channel*.

The experimental applications use *TAO Real-Time ORB* and *TAO's Real-Time Event Channel* to communicate both between components requiring event-mediated interactions on the same end system and components, distributed across different end systems. The software architecture also was supposed to include the *RTDSS* framework. The implementation of this framework was separate from our project and due to reasons beyond our control is not complete. Since the process of schedulability lays outside of our project's scope, and by knowing that with the low CPU utilization (<= 69%) our set of task is going to be schedulable (classic Rate Monotonic Scheduling), we simply use "dummy" function calls, whenever we need interactions with the *RTDSS*.

Our experimental application is running on a desktop computer, equipped with Gentoo Linux i686 2.6.39-r3, installed with ACE 6.0.3 and TAO 2.0.3.

The computer is running a global *Naming Service*. For a single node simulation it is running a *Global DDS* agent, a *Source Local DDS* server, and a *Reader Local DDS* server. For a multiple node simulation we add additional *Global DDS* agents, *Source Local DDS* and *Reader Local DDS* servers. Multiple data-centric applications providing or receiving different types of data are also running on the computer.

5.3.2 Experimental Design

To describe our experiments we are using Goals-Questions-Metrics-Experiments (GQME) terminology [3].

The **Goa**l was to evaluate TDDS middleware in terms of end-to-end delivery of information with timing constraints and its support for dynamic changes in real-time configurations.

The following **Questions** and subsequently **Metrics** were defined:

1) How much overhead is there for TDDS middleware to perform real-time end-to-end data distribution? This question was addressed by measuring:

- Average time to establish a distribution chain.
- End-to-end latency to deliver data.
- Memory consumption to establish a distribution chain.

2) How well does TDDS middleware respond to dynamic configuration changes? Here as well, average time to establish/destroy a distribution chain was measured.

3) How well does TDDS middleware perform in terms of preserving data temporal consistency? This was measured by the time remaining until data expiration at the time of data access.

4) How well does TDDS middleware decentralize? This was determined by quantifying existence of single point of failure, and possible recovery methods if any.

5) How transparent is TDDS middleware from the application? This was measured by how much a user must know about the system to join.

6) How well does TDDS middleware scale? This was measured, by the effect on system performance of increasing the number of data applications (event types, data providers and receivers).

Since, a lot of tests for questions 1, 4, 5, and 6 were performed after initial framework development and described in a previously published thesis [2] we did not repeat them in this work. Instead we concentrated on the effect of including JIT block to the system (questions 2 and 3). That is, on distribution deadlines, on temporal consistency of delivered data, and the overhead added to the system by JIT computation associated with maintaining data consistency. In the tests we measured the time that was involved in establishing and destroying a distribution chain. We compared the time it took to establish the chain with deadline computation in JIT block, and without it, assuming the worst case scenario and the minimum deadline (OV – P). We also measured the time interval between distribution deadlines and actual time of data delivery, and the interval between data expiration and the time it was accessed by a *Reader*. These parameters were computed as follows:

Time *tsr*0 is recorded when starting up *Data Source*. Time *tsr*1 is recorded when the *Data Source* finishes registration to the *Source Local DDS*. Time *trr*0 is recorded when starting up *Data Reader*. Time *trr*1 is recorded when the *Data Reader* finishes registration to the *Reader Local DDS*. Time *trs* = (*tsr*1 – *tsr*0) and *trr*= (*trr*1 – *trr*0) are the times to establish a source and a reder. For deadline assurance we record the time *ttd* when a data is delivered by a *Consumer* to a *Reader Local DDS*. Then we check it against the *Consumer* deadline *rt_info.deadline*. If the deadline is met, the *rt_info.deadline* - *ttd* ≥ 0. When a *Data Reader* reads the data from its *Reader Local DDS*, the time value associated with it (*ttr)* is used to calculate the data validity. For a data to be valid at the time of access, the *data.validity* - *ttr* ≥ 0.

To destroy the chain tim *tsu*0 is recorded when starting *Data Source* unregistration. Time *tsu*1 is recorded when the *Data Source* finishes unregistration from the *Source Local DDS*. Time *tru*0 is recorded when starting *Data Reader* unregistration. Time *tru*1 is recorded when the *Data Reader* finishes unregistration from the *Reader Local DDS*. Time *tsu* = (*tsu*1 – *tsu*0) and *tru*= (*tru*1 – *tru*0) is the time elapsed to destroy a source and a reader.

We performed the following set of test suits:

Test Suite 1: Baseline. Single Node / Single Data Source / Single Data Reader. Experiments 1-10 (with JIT). Experiments 11-20 (without JIT)

Test Suite 2: Single Node / Single Data Source / Multiple Data Readers. Number of readers increased to 5. Experiments 21-30 (with JIT). Experiments 31-40 (without JIT)

Test Suite 3: Single Node / Multiple Data Sources / Multiple Data Readers. Number of data sources is increased to 5. Experiments 41-42 (with JIT). Experiments 43-44 (without JIT)

Test Suite 4: Multiple Nodes / Multiple Data Sources / Multiple Data Readers. Data readers run on both nodes. Experiment 45 (with JIT). Experiment 46 (without JIT).

For these experiments, we generated 10 random sets of parameters for Data Sources, with values for periods and data validity ranging from 100ms to 2000ms. Then accordingly, we generated 5 sets of Data Reader parameters for each of the Data Sources. During the tests' runs the Data Sources and Data Readers come and leave the system randomly.

5.3.3 Results

In this section we present the results of our tests.

Test Suite 1: For the base line, we repeated experiments for each of ten generated Data Sources with one respective Data Reader from the pool for each Data Source. Then, for the registration/unregistration time analysis for each party, we used the means of the results from these ten experiments. For the Deadline and Validity charts, we used all data as-is. We received the following results (refer to Figure 38): the average registration time of incoming Data Sources in both cases (with JIT, and without) is within 17 ms: the average Unregistration is within 8 ms. Since the Data Sources are not affected by JIT computation, there is no difference in the performance. Registration of incoming Data Readers in both cases is completed within 25 ms. It takes just 3.7% more time to register a Data Reader with the use of JIT computation, than without it. The un-registration for both cases finishes within 8 ms, with 5.8% overhead for the Data Reader with JIT. Figures 39, 40 show baseline performance in terms of accurate data delivery and its validity. Dots on the chart to the left represent the differences between a deadline and actual delivery time, and the dots on the chart to the right represent the difference between data validity time and the time the data was accessed. We can see that all differences are positive. That is, in every instance the distribution is finished before its deadline, and every time the data was accessed, it was valid (the shape of the graph represents Data Readers reading patterns). We can also see that with JIT computation, the distribution deadlines are more relaxed, that is some of them are longer. Longer deadlines mean a better chance of system being schedulable.

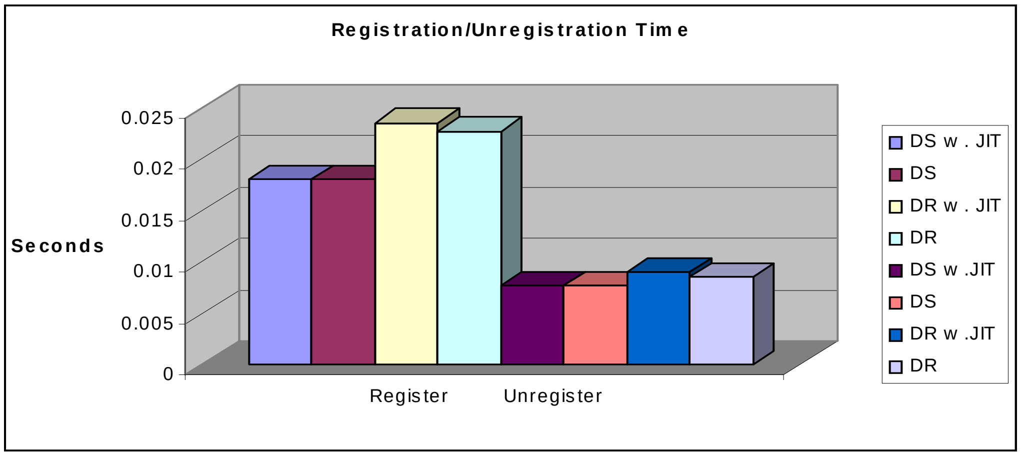
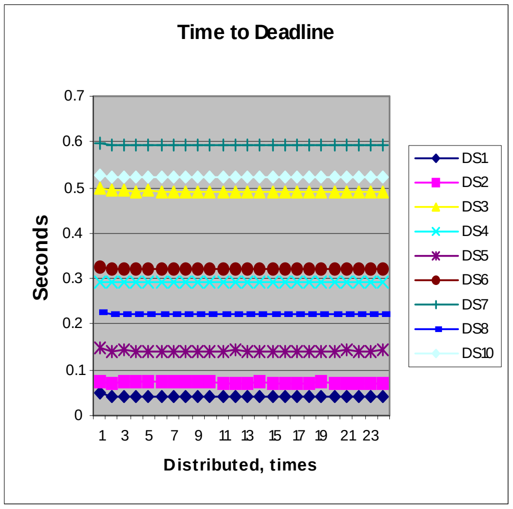


Figure 38. Baseline (Registration/Unregistration)



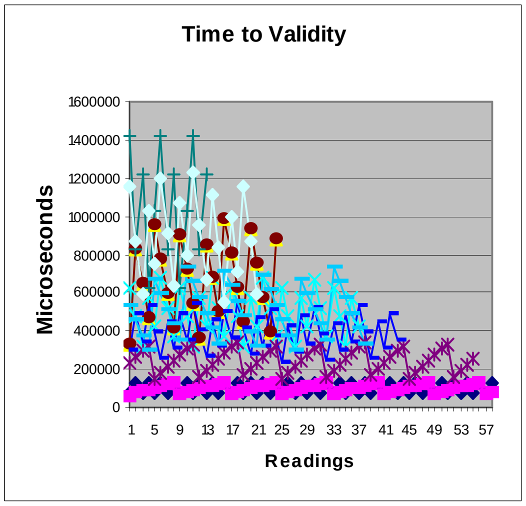
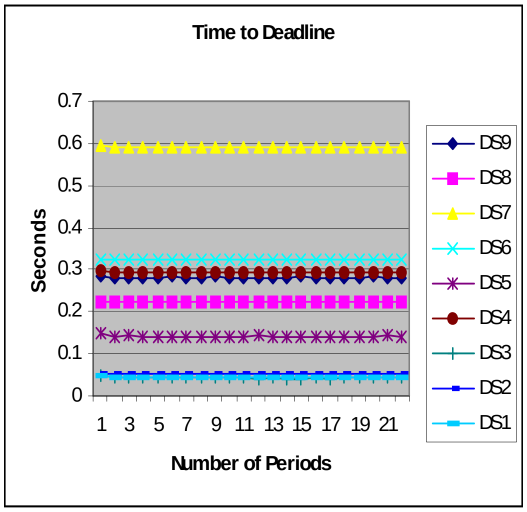


Figure 39. Baseline with JIT.



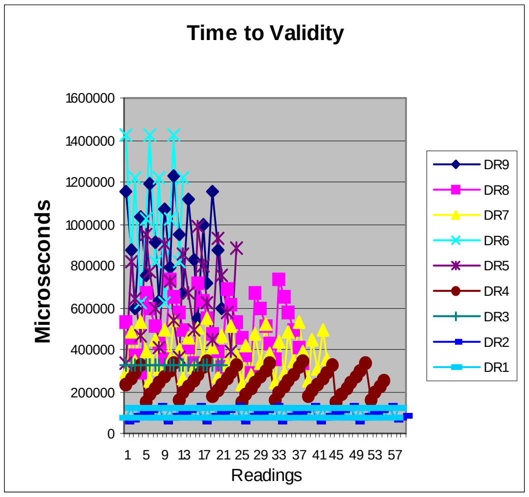


Figure 40. Baseline without JIT.

Test Suite 2: Within the second set of experiments, we ran each of ten Data Sources, but now with all five Readers for each. For the registration/unregistration time we again used the means of the respective results, and for the Deadline/Validity charts, we used all data as-is. We observed that registration/unregistration time for the incoming Data Sources, registration time for the first incoming Reader, and unregistration for the last Reader to leave the system are similar to our baseline time (refer to Figure 41). The average registration for the Data Source is below 20 ms, for the first Data Readers is below 25 ms. Average registration time of incoming Readers two through five, and then unregistration time of Readers one through four (they leave the system in first–in-first –out order) is below 5 ms. This is due to the fact that at the time these Readers enter and leave the system, all entities are running and all distribution chains are set up. An overhead imposed by JIT computation in this set of experiments was no more than 35 % across all readers (13% on average).

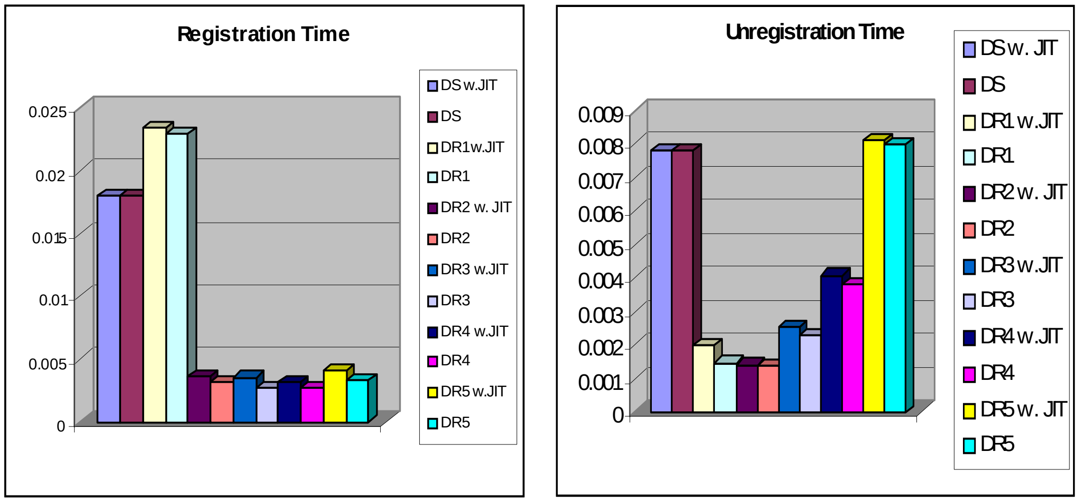


Figure 41. Single Node. Single Data Source. Multiple Data Readers.

From Figures 42 and 43, it can be observed that with JIT computation, deadlines are changing in the process of new readers entering the system, and again they are more relaxed. All deadlines are met, and all the Readers access valid data all the time.

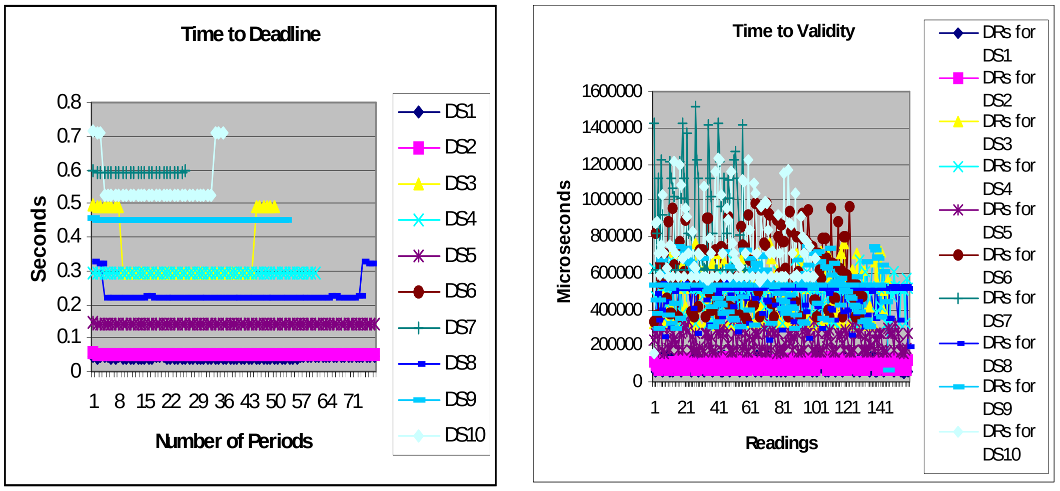


Figure 42. Single Node. Single Data Source. Multiple Data Readers. With JIT

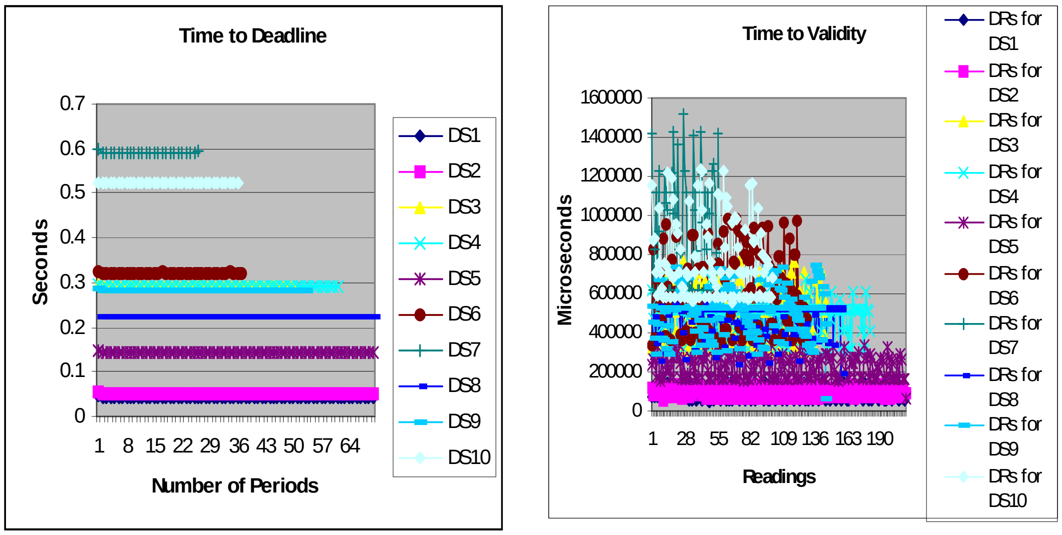


Figure 43. Single Node. Single Data Source. Multiple Data Readers. Without JIT

Test Suite 3: For the single node multiple sources experiments we ran twice five data sources with five readers each. We averaged registration/unregistration times for all ten incoming Data Sources and for all fifty incoming Readers in the order of their registration. Figure 44 presents our results.

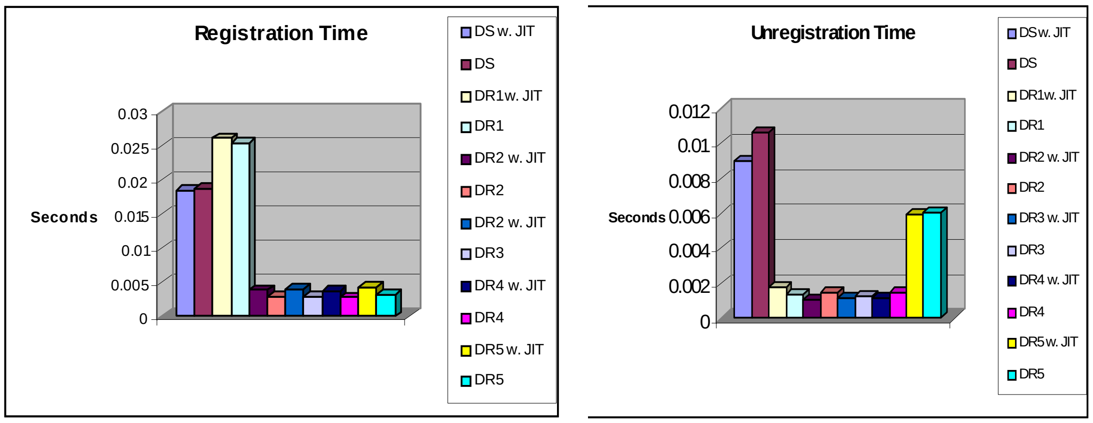


Figure 44. Single Node. Multiple Data Sources. Multiple Data Readers.

These results go along with registration/unregistration time we have already observed, with average overhead imposed by JIT in this set up being about 30%.

Figures 45 and 46 present our observations for distribution deadlines and data validity checks for tests with JIT computation and without it. On the figures we combined results from both experiments in each set up.

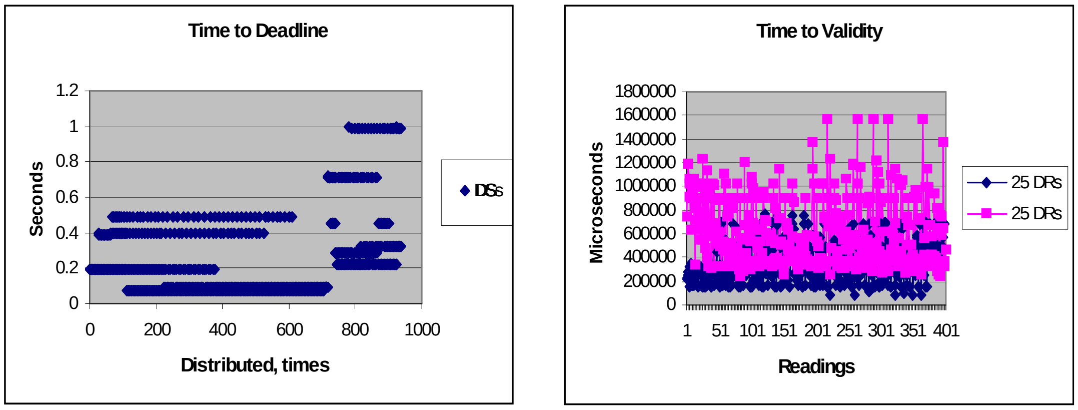


Figure 45. Single Node. Multiple Data Sources. Multiple Readers. With JIT.

On the charts to the left the lines represent time to deadline for each of ten data readers (five from each experiment). On the charts to right data points of one color represent times to validity at readings for each of twenty five Readers related to five Data Sources in one experiment. We can observe that results here are also similar to the above. All measurements are positive, meaning that distributions complete before their deadlines, which in case of JIT computation are longer for some of them, and all the readers always accessed valid data.

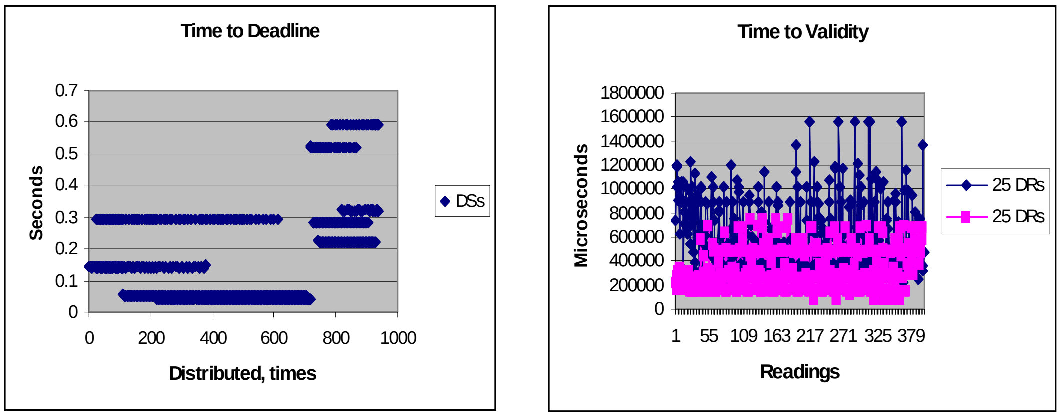
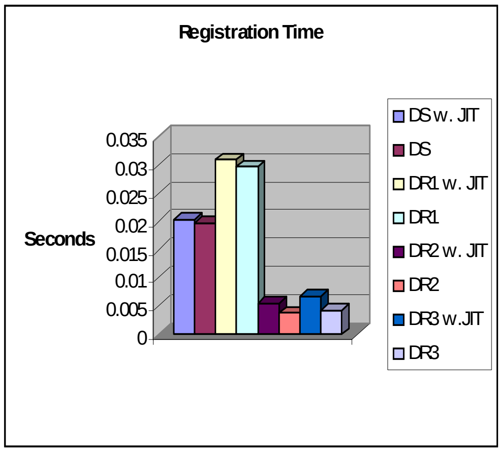


Figure 46. Single Node. Multiple Data Sources. Multiple Readers. Without JIT.

Test Suite 4: For the final experiment we ran five sources on each of nodes one and two with fifty data readers (five for each source) divided between the nodes. With this set up we had either two or three readers for each source on the node. We recorded all registration/unregistration time results and then averaged them to build our charts. For deadlines and validity we used recordings from all ten Data Sources and all fifty Data Readers. Figure 47 shows a slight increase in registration/unregistration times compared to all the previous tests. Here the registration for incoming Data Sources is averaged within 20 ms. For the first incoming Data Readers it is at 30 ms, and for the second and third Readers it is below 7 ms. Unregistration for Data Sources is complete within 14 ms and for the readers it is done within 10 ms. Since with our set up some of the second Data Readers are the last to leave for their Data Sources on a node, we see an increase in their unregistration time, compared to the first Data Readers to leave. The average overhead due to JIT computation runs at about 24% here.



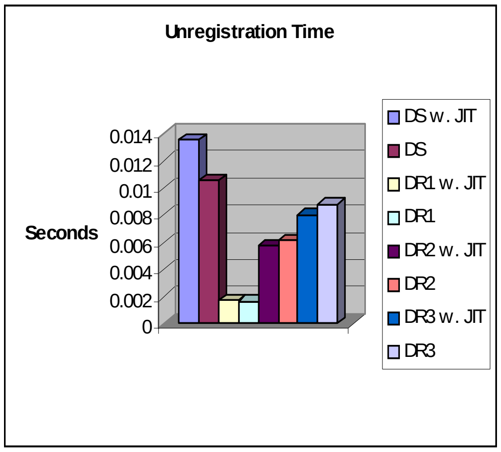


Figure 47. Multiple Nodes. Multiple Data Sources. Multiple Data Readers.

Figures 48 and 49 present our results for the delivering times and the validity of data. The results here are similar to the ones we have already observed in the previous tests. With all the deadlines, either computed with JIT or the worst case, met, the accessed data is always valid.

The results of our experiments show that the JIT computation relaxes system deadlines, the overhead associated with it falls in a reasonable range (averaging less than 35%), across all the tests. And, that all Data Readers always get valid data if it was delivered before specified deadline.

Combining our results with the results published by Mr. Jie Mao [2], along with system design and implementation, we can summarize characteristics of our TDDS middleware.

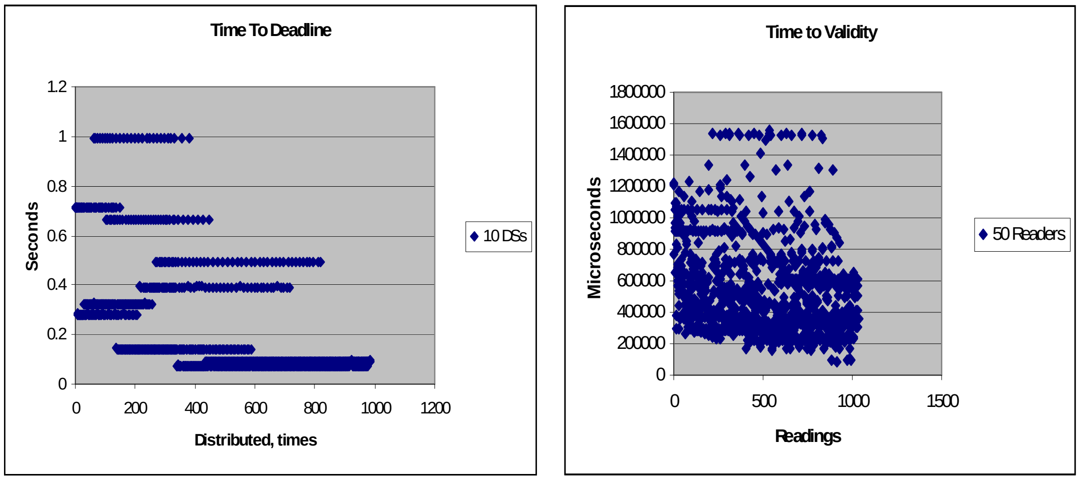


Figure 48. Multiple Nodes. Multiple Data Sources. Multiple Data Readers. With JIT.

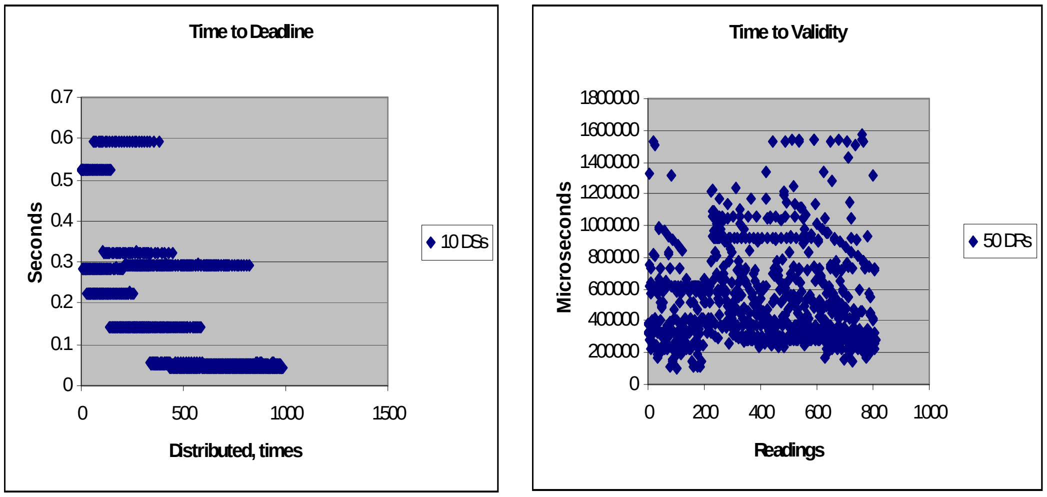


Figure 49. Multiple Nodes. Multiple Data Sources. Multiple Data Readers. Without JIT

The TDDS ensures timely and inerrant data delivery from a proper data provider to a proper data recipient according to their requirements, with the guarantee of data temporal consistency.

The TDDS is completely decentralized, with Reader Local DDSs, Source Local DDSs and Global DDSs, as distributed agents, running on each node.

The TDDS is highly transparent. The service achieves this by hiding all the details of deadline computations, scheduling and actual data distribution from the end user. The end user just employs either Source or Reader Local DDS on their side, provides their real-time parameters and an event type of their interest. After that the middleware processes all the necessary steps to set up data distribution.

The TDDS scales well. Addition of new distribution chains has no effect on existing ones.

---

1. Schmidt, D.C., "Real-time CORBA with TAO (The ACE ORB)," Washington University at St. Louis, 12 November 2013, http://www.cs.wustl.edu/~schmidt/TAO.html, accessed 19 March 2014.

2. Mao, J., "Implementation of a Dynamic Real-Time Data Distribution Service for Middleware Systems," MS Thesis, Computer Science Department, University of Rhode Island, May 2005

3. Basili, V.R., "Software Modeling and Measurement: The Goal/Question/Metric Paradigm", *Goal_Question_Metric.pdf* Retrieved from http://drum.lib.umd.edu on March 19 th, 2014

CHAPTER 6

CONCLUSION

6.1 Contributions

This thesis has focused on real-time data distribution. This subject covers quite a wide area, since there are many real-time distributed systems with various parameters and goals that require different types of data to be dispersed. Since a better understanding of the problem leads to a better solution, we, by combining together various characteristics of the systems, real-time characteristics and data characteristics defined the Real-Time Data Distribution Problem Space Taxonomy. The Taxonomy provides researchers and developers with a more standardized way of looking at the problems being addressed and solutions that might fit them. This part of the work was published in [1].

Further, we defined two specific subspaces within the problem space to address in this work. They are static and dynamic application, with the following main characteristics: hard real-time with periodic timing constrains and consistent data for the static system; and soft real-time with periodic timing constrains and consistent data for the dynamic system. We started with the static solution. We defined parameters of Distribution, and proved their necessity for ensuring the correctness of timely data transfer. We developed Just-In-Time Static (JITS) algorithm for computation of the Distribution deadline. This algorithm combines Data Sources and Data Reader parameters, which ensures data temporal consistency whenever it is accessed by the Readers. We implemented and tested the system with real-life parameters of military command and control application. The results of the tests show that our claim holds. With the data delivered by the computed JITS deadline (which with the static system is always the case, since all the requirements are known and scheduled a priori), it is temporally consistent whenever it accessed by the reading applications. This part of the work was published in [2].

After finishing our work on the static solution, we moved on to the dynamic. For which we first reworked our static JITS algorithm and changed it into a dynamic JIT version that delivers the same result with a lesser computation overhead. This change removed some extra computation and made the algorithm more suitable for the dynamic environment, where all computation is performed on-line. Then, we designed and implemented the Timely Data Distribution Service middleware that, by incorporation of JIT computation in its mechanism, allows to adjust Distribution deadlines according to incoming Data Reader’s requests in a dynamic fashion. The Distribution deadlines computed with JIT can be longer than the ones set by the worst case assumption; that is the absolute data object validity less the data distribution period (OV – P). Longer deadlines, in their turn make the system more flexible in terms of schedulability, with more tasks being accepted. Our tests show that the overhead associated with JIT computation averages at 30%. The results also show that when a system is schedulable and Distribution deadlines are met, the Data Readers that access data according to their own timing constraints always read temporally valid data. Summarizing all the results, we can characterize our Timely Data Distribution

Service as a completely decentralized, highly transparent and scalable data transferring mechanism, with the data validity guaranty.

6.2 Comparison with Related Work

There are several areas applicable to RT Data Distribution. One of the first and very extensive researched is the area of data consistency in real-time databases.

Starting from the HH algorithm[3], that sets data update deadlines and periods to be half of the absolute object validity (OV), to the More-Less approach [4], where the periods are longer than half of the OV, and the deadline are shorter, which by using DM scheduling maximizes CPU utilization, compare to HH. Then the further work in [5,6] considers earliest deadline first based ML (ML_{EDF}) and Deferrable Scheduling (DS -FP), the work in [7] extends ML to distributed systems introducing transmission delays of updating tasks. Later, to address variability in transmission delays, work in [8] introduces extensions to ML called Jitter-Based More-Less (JB -ML) and Statistical Jitter-Based More-Less (SJB-ML). In all this extensions, all extra parameters are used to determine the deadline of a data update(D_{upd}), and then assign the period (P_{upd}) according to $D_{upd} + P_{upd} \leq OV$, where $D_{upd} \leq ½\ OV \leq P_{upd}$. All this work guarantees that data is temporally consistent at the sink, or initial database, where it comes from various physical devices, sensors, cameras, etc. It can’t provide the assurance that data is still valid when it is distributed to the end point users. Our work can be seen as extension to this. To assure the data freshness at the end point of distribution the worst case deadline should be computed as $D = D_{upd} + D_{worst}$, where D_{worst} is equal to the worst case execution time for a distribution to be able to complete within the system. Then, the period of update and respectively of distribution can be computed as above $P + D \leq OV$. This will assure that even with worst case temporally valid data can be physically distributed. Having our distribution period, we start from here and use our computation to relax the worst case deadlines and make the system more flexible.

We guarantee the freshness of data whenever it is accessed by the client, and may leave it inconsistent at some other times that is $D_{dist} + P$ can be more than OV. This allows us to extend some of the distributions deadlines and increase the chances of system schedulability.

Another area applicable to data distribution, that in recent years has become an established technology for a wide application areas, such as monitoring, tracking, event detection, to name a few, is the Wireless Sensor Networks (WSN). A large amount of real-time data dissemination in wireless sensor networks research was done at the University of Virginia (UVa) [9,10,11,12,13]. While authors addressed deadlines of requests, and the temporal validity was considered in the sense that data was reported before it expired— by corresponding confidence values, this, work however did not provide assurance that the data is still temporally valid when it arrived to the requestor. In their recent work [14] authors presented a data abstraction layer for collaborative 2-tier sensor network applications. The layer implements a model-driven predictive replication mechanism, the goal of which is to maintain an overall data consistency, by disseminating sensor updates to the parties only when data, predicted by an established model, is outside of specified data accuracy threshold. Decreasing the amount of dissemination, leads to decreasing CPU utilization, but for this approach to work data must be continuous. In our work we do not place restriction on data, and decrease CPU utilization by extending distribution deadlines.

To address the needs of various types of applications requiring data dissemination the OMG issued a specification for Data Distribution Service (DDS) [15]. Two QoS policies supported by DDSs DCPS interface and related to our work are the DeadlineQoS and a LifespanQoS. Where the DeadlineQoS specifies a period during which the data must be distributed, and the LifespanQoS enables middleware to delete expired data. Based on these policies, there is no way to define and enforce a deadline within the period, which can lead to the situation when the previous data is stale and deleted from the data space, but a new sample is not delivered. Therefore we believe that DDS can not guarantee the temporal consistency of data.   Our work can ensure that the reading applications get valid data whenever they access it.

The work in [16] presents an extension to OMG DDS, called RDDS. RDDS tries to achieve overall system data consistency by the mean of semantic-aware communication, using predictive sensor models on publisher and subscriber sides in the systems with data continuity. The approach here is very similar to the one described in [14], except that it is built upon DCPS instead of embedded databases. In our work we place no restriction on data, and use original sensor updates.

6.3 Limitations and Future Work

We recognize that there are some limitations to the work presented here. Some of them are highlighted below and can be considered for a future work.

(1) The TDDS framework was supposed to work with the RTDSS framework to enforce real-time scheduling. The RTDSS framework was not completed by the reasons beyond our control. Therefore it would be beneficial to finish this project, and to evaluate the system as a whole to ensure its overall functionality and performance.

(2) Currently, we only allow one system-wide Data Source for each type of Event. It is challenging but interesting to investigate a data distribution service allowing multiple Data Sources providing the same type of data into the system, and delivering data from a certain Data Source to certain Data Readers according to some pre-set policy, or reconnecting a Data Reader to another Source if its original Data Source leaves the system.

(3) It also would be interesting to accommodate our JIT algorithm to different DataSource – Data Reader patterns. For example, if the DataSourse produces data much faster than the DataReaders need it, the distribution period could be set to n*P, n={1,2,3...}. That could reduce the amount of distributions in the system, and decrease the workload and amount of communication.

---

1. Uvarov Frolov, A., Cingiser Dipippo, L., Fay-Wolfe, V., “Real Time Data Distribution,”*Handbook of Real-Time and Embedded Systems*, Lee, I., Leung, J. Y-T., Son, S. H., Boca Raton: Chapman & Hall, 2008.

2. Uvarov, A., DiPippo, L., Fay Wolfe, V., Bryan,K., Gadrow, P., Henry, T., Murphy, M., Work, P.R., DiPalma, L.P., Static Real-Time Data Distribution, *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, 2004: 502-509.

3. Ho, S., Kuo, T., Mok, A., "Similarity-based load adjustment for real-time data-intensive applications." in Proceedings of the IEEE real-time system symposium (RTSS'97), December 1997, San Francisco, CA, pp:144-153.

4. Xiong, M. and Ramamritham, K., “Deriving deadlines and periods for real-time update transactions,” in Proceedings of the 20th IEEE Real-time Systems Symposium, December 1999, Phoenix, AZ.

5. Xiong, M., Han, S., Lam, K.Y., Chen, D., "Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results," IEEE Transactions on Computers, 57(7), July 2008.

6. Xiong, M., Wang, Q., Ramamritham, K., "On earliest deadline first scheduling for temporal consistency maintenance," Real-Time Systems, 40, 2008, pp:208–237.

7 Xiong, M. and Ramamritham, K., "Deriving Deadlines and Periods for Real-Time Update Transactions", IEEE Transactions on Computers, 53(5), 2004.

8. Wang, J., Han, S., Lam, K-Y., Mok, A., "Maintaining Data Temporal Consistency in Distributed Real-Time Systems," Real-Time Systems, 48, 2012, pp:387–429.

9. Lu, B.C., Blum, B.M., Abdelzaher, T., Stankovic, J.A., He, T., "RAP: A Real-Time Communication Architecture for Large Scale Wireless Networks," in Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'02), September 2002, San Jose, CA.

10. Abdelzaher, T., Stankovic, J., Son, S., Blum, B., He, T., Wood, A., Lu, C., "A Communication Architecture and Programming Abstractions for Real-Time Embedded Sensor Networks," in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

11. Kim, S., Son, S., Stankovic, J., Li, S., Choi, Y., "SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks," in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems, May 2003, Providence, RI.

12. Bhattacharya, S., Kim H., Prabh, S., Abdelzaher, T., "Energy Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks," in Proceedings of the First International Conference on Mobile Systems, Application and Services, May 2003, San Francisco, CA.

13. Li, S., Son, S., Stankovic, J., "Event Detection Service Middleware in Distributed Sensor Networks," in Proceedings of the International Workshop on Information Processing in sensor Networks (IPSN'03), April 2003, Palo Alto, CA.

14. Kang, W., Son, S.H., Stankovic, J.A., "Quality-aware data abstraction layer for collaborative 2-tier sensor network applications", *Real-Time Systems* (2012) 48:463–498 DOI 10.1007/s11241-012-9154-0

15. OMG – Data Distribution Service for Real Time Applications Specification, Version 1.2, OMG Inc., January 2007, (formal/07-01-01), <http://www.omg.org/spec/DDS/1.2>. Accessed 19 March 2014.

16. Kang,W., Kapitanova, K., Son, S., "RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems", *IEEE Transactions on Industrial Informatics*, Vol. 8, NO. 2, May 2012

BIBLIOGRAPHY

Abdelzaher, T., Stankovic, J., Son, S., Blum, B., He, T., Wood, A., Lu, C., "A Communication Architecture and Programming Abstractions for Real-Time Embedded Sensor Networks," *in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems*, May 2003, Providence, RI.

Aksoy, D., Altinel, M., Bose, R., Cetintemel, U., Franklin, M., Wang, J., Zdonik, S., "Research in Data Broadcast and Dissemination," *International Conference on Advanced Multimedia Content Processing (AMCP)*, November 1998, Osaka, Japan, pp:196-211.

Baheti, R. and Gill, H., "Cyber-Physical Systems," The Impact of Control Technology, T. Samad and A.M. Annaswamy (eds.), *IEEE Control Systems Society*, 2011, available at www.ieeecss.org.

Basili, V.R., "Software Modeling and Measurement: The Goal/Question/Metric Paradigm", *Goal_Question_Metric.pdf* Retrieved from http://drum.lib.umd.edu on March 19 th, 2014

Bestavros, A., "AIDA-based real-time fault-tolerant broadcast disks," *in Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS'96)*, June 1996, Boston, MA, pp:49-58.

Bestavros, A., "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems," *in*

Proceedings of the 1996 International Conference on Data Engineering, New Orleans, LA.

Bhattacharya, S., Kim H., Prabh, S., Abdelzaher, T., “Energy Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks,” *in Proceedings of the First International Conference on Mobile Systems, Application and Services*, May 2003, San Francisco, CA.

Bonnet, P., Gehrke, J., Seshadri, P., “Towards Sensor Database Systems,” *in Proceedings of the Sensor Information Conference on Mobile Data Management*, January 2001, Hong Kong, China.

Gore, P., Pyarali, I., Gill, C.D., Schmidt, D.C., "The Design and Performance of a Real-time Notification Service," *in Proceedings of the 10th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS’04)*, May 2004, Toronto, Canada.

Harrison, T.H., Levine, D.L., Schmidt, D.C., "The Design and Performance of a Real-time CORBA Event Service," *in Proceedings of the Object-Oriented Programming Systems, Languages & Applications Conference (OOPSLA’97),* October 1997, Atlanta, GA.

Harrison, T.H., Levine, D.L., Schmidt, D.C., "The Design and Performance of a Real-time CORBA Event Service," *in Proceedings of the Object-Oriented Programming Systems, Languages & Applications Conference (OOPSLA’97),* October 1997, Atlanta, GA.

Heinzelman, W., Chandrakasan, A., Balakrishan, H., “Energy Efficient Communication Protocol for Wireless Microsensor Networks," *in Proceedings* *of the Hawaii International Conference on System Sciences(HICSS'00),* January 2000.

Ho, S., Kuo, T., Mok, A., "Similarity-based load adjustment for real-time data-intensive applications." *in Proceedings of the IEEE real-time system symposium (RTSS'97),* December 1997, San Francisco, CA, pp:144-153.

Kang, W., Kapitanova, K., Son, S., "RDDS: A Real-Time Data Distribution Service for Cyber-Physical Systems", *IEEE Transactions on Industrial Iinformatics,* 8(2), 2012.

Kang, W., Son, S.H., Stankovic, J.A., "Quality-aware data abstraction layer for collaborative 2-tier sensor network applications", *Real-Time Systems* (2012) 48:463–498 DOI 10.1007/s11241-012-9154-0

Karakaya, M., Ulusov, O., "Evaluation of a Broadcast Scheduling Algorithm", *Lecture Notes in Computer Science, Springer-Verlag, 2151, 2001.*

Kim, S., Son, S., Stankovic, J., Li, S., Choi, Y., "SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks," *in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems,* May 2003, Providence, RI.

"InterCOM DDS," Kongsberg Gallium Corp., 2013, *http://www.kongsberg.com/en/kds/kongsberggallium/products/intercom%20dds/*, accessed 19 March 2014.

Lam,W. and Garcia-Molina, H., "Slicing Broadcast Disks," *Stanford University Technical Report*, 2003.

Lee, E., "Cyber Physical Systems: Design Challenges." *Technical Report No. UCB/EECS-2008-8,University of California, Berkeley,* January 23, 2008.

Li, S., Son, S., Stankovic, J., "Event Detection Service Middleware in Distributed Sensor Networks," *in Proceedings of the International Workshop on Information Processing in sensor Networks (IPSN'03),* April 2003, Palo Alto, CA.

Liu, J.W.S., *Real-Time Systems, Prentice-Hall,* June 2000.

Lu, B.C., Blum, B.M., Abdelzaher, T., Stankovic, J.A., He, T., "RAP: A Real-Time Communication Architecture for Large Scale Wireless Networks," *in Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Application Symposium (RTAS'02),* September 2002, San Jose, CA.

Ma, C. and Bacon, J., "COBEA: A CORBA Based Event Architecture," *in Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS' 98),* April 1998, Santa Fe, NM.

Mao, J., "Implementation of a Dynamic Real-Time Data Distribution Service for Middleware Systems," *MS Thesis, Computer Science Department, University of Rhode Island,* May 2005.

"Technology Center," MilSOFT, http://www.milsoft.com.tr, accessed 19 March 2014.

Mungee, S., Surendran, M., Krishnamurthy, Y., Schmidt, D.C., "The Design and Performance of CORBA Audio/Video Streamin Service," *Design and Management of Multimedia Information Systems: Opportunities and Challenges, Syed, M. (ed), Idea Group Publishing.* Hershey, USA, 2001.

Murphy, M., Bryan, K., "Corba 1.0 Compliant Static Scheduling Service for Periodic Tasks Technical Documentation", *URI Technical Report TR04-297, January 2004*.

Object Mgmt. Group, Real-time Notification: Request For Proposals, *OMG Doc. orbos/00-06-10*, June, 2000.

OMG. Common Object Request Broker Architecture – Version 3.3, OMG Inc., November 2012, *(formal/2012-11-12, formal/2012-11-14, formal/2012-11-16)*, http://www.omg.org/spec/CORBA/3.3

OMG CORBA Audio/Visual (A/V) Streams Specification, OMG Inc., 2000 (formal/2000-01-03)

OMG – Data Distribution Service for Real Time Applications Specification, Version 1.2, January 2007, *(formal/07-01-01)*, http://www.omg.org/spec/DDS/1.2

OMG Event Service –Specification – Version 1.2, OMG Inc., October 2004, *(formal/2004-10-02)*, http://www.omg.org/spec/EVNT/1.2

OMG Naming Service – Specification – Version 1.3, October 2004, *(formal/2004-10-03)*, http://www.omg.org/spec/NAM/1.3

OMG Notification Service – Specification – Version 1.1, OMG Inc., 2004 *(formal/04-10-11)*

OMG RT CORBA - Version1.2 OMG Inc., January 2005, *(formal/2005-01-04),* http://www.omg.org/spec/RT/1.2

"OpenDDS," Object Computing Inc., 2013, http://opendds.sourceforge.net/, accessed 19 Mar 2014.

"OpenSPLICE|DDS", PrismTech, 2014, http://www.prismtechnologies.com, accessed 19 March 2014.

Peddi, P., "A Replication Strategy for Distributed Real-Time Object-Oriented Databases," *TR01-282, University of Rhode Island*, May 2001.

"RTI Connext DDS Professional," Real-Time Innovations, 2014, http://www.rti.com/products/data_distribution/index.html, accessed 19 March 2014.

Rujkumar, R., Lee, I., Sha, L., Stankovic, J., "Cyber-Physical Systems: The Next Computing Revolution," *in Proceedings of the 47th Design Automation Conference*, June 2010, Anaheim, CA, pp:731-736.

Shenker, S., Ratnasamy, S., Karp, B., Govindan, R., Estrin, D., “Data-Centric Storage in Sensornets”, *in Proceedings of the First ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-I),* October 2002, Princeton, NJ.

"The ACE ORB (TAO)," Object Computing, Inc., http://www.ociweb.com/products/tao, accessed 19 March 2014.

Schmidt, D.C., "Real-time CORBA with TAO (The ACE ORB)," Vanderbilt University, 12 November 2013, http://www.cs.wustl.edu/~schmidt/TAO.html, accessed 19 March 2014.

"Thales," Thales Group, http://www.thalesgroup.com, accessed 19 March 2014.

"Rapid RMA," TriPacific Software, Inc., www.tripac.com, accessed 19 March 2014.

"CoreDX DDS Data Distribution Service Middleware," Twin Oaks Computing Inc., http://www.twinoakscomputing.com/coredx.php, accessed 19 March 2014.

Uvarov, A., DiPippo, L., Fay Wolfe, V., Bryan, K., Gadrow, P., Henry, T., Murphy M., Work, P.R., DiPalma, L.P., "Static Real-Time Data Distribution," *in Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, May 2004, Toronto, Canada, pp:502-509.

Uvarova, A. and Fay Wolfe, V., "Towards a Definition of the Real-Time Data Distribution Problem Space," *in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems*, May 2003, Providence, RI.

van ‘t Hag, J.H., "Data-centric to the Max, the SPLICE Architecture Experience," *in Proceedings of the First International Workshop on Data Distribution for Real-Time Systems*, May 2003, Providence, RI.

Wang, J., Han, S., Lam, K-Y., Mok, A., “Maintaining Data Temporal Consistency in Distributed Real-Time Systems," *Real-Time Systems*,48, 2012, pp:387–429.

Xiong, M. and Ramamritham, K., “Deriving Deadlines and Periods for Real-Time Update Transactions”, *IEEE Transactions on Computers,* 53(5), 2004.

Xiong, M., Han, S., Lam, K.Y., Chen, D., “Deferrable scheduling for maintaining real-time data freshness: algorithms, analysis, and results,” *IEEE Transactions on Computers*, 57(7), July 2008.

Xiong, M. and Ramamritham, K., “Deriving deadlines and periods for real-time update transactions,” *in Proceedings of the 20th IEEE Real-time Systems Symposium*, December 1999, Phoenix, AZ.

Xiong, M., Wang, Q., Ramamritham, K., “On earliest deadline first scheduling for temporal consistency maintenance,” *Real-Time Systems*, 40, 2008, pp:208–237.

Xuan, P., Sen, S., Gonzales O., Fernandez, J., Ramamritham, K., “Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments,” *in Proceedings of the 3rd IEEE Real-Time and Embedded Technology and Application Symposium (RTAS’97)*, June 1997, Montreal, Canada.

Yao, Y. and Gehrke, J., “Query Processing for Sensor Networks,” *in Proceedings of the 2003 Conference on Innovative Data System Research (CIDR2003)*, January 2003, Asilomar, CA.

Ye, F., Luo, H., Cheng, J., Lu, S., Zhang, L., “A Two –Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks,” *in Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MOBICOM’02)*, September 2002, Atlanta, GA.

Zhang, J., DiPippo, L., Fay-Wolfe, V., Bryan, K., Murphy, M., “A Real-Time Distributed Scheduling Service for Middleware Systems,” *in Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005)*, February 2005, Sedona, AZ.