Static Real-Time Data Distribution*

Angela Uvarov, Lisa DiPippo, Victor Fay-Wolfe, Kevin Bryan, Patrick Gadrow, Timothy Henry, Matthew Murphy
University of Rhode Island
Department of Computer Science
Kingston, RI 02881 USA
{uvarovaa, dipippo, wolfe, bryank, gadrowp, henry, murphym}@cs.uri.edu

Paul R. Work, Louis P. DiPalma
Raytheon Company
Portsmouth, RI USA 02871
{paul_r_work, louis_p_dipalma}@raytheon.com

Abstract

This paper describes the design and implementation of a static real-time data distribution mechanism that uses a real-time event service and a real-time scheduling service to ensure the on-time, and temporally valid delivery of data. The mechanism implements an algorithm, called the Just-In-Time Data Distribution algorithm to determine scheduling parameters that will ensure that data that arrives at a requesting target is valid. The paper uses a Navy weapons alignment application to demonstrate the necessity and usefulness of the algorithm and implementation. The paper also presents results of tests that demonstrate that our implementation fulfills the guarantees predicted by the theoretical results.

Keywords: *real-time, data distribution, real-time events, data temporal consistency*

1 Introduction

In many distributed embedded real-time (DRE) applications, like military command and control, time-critical planning collaboration, and wireless embedded sensor networks, data produced in one component of the system needs to be shared with other components of the system. Such applications may have stringent deadlines by which the data must be delivered in order to process it on time to make critical decisions. Further, the data that is distributed must be valid when it arrives at its target. That is, if the data is too old when it is delivered, it could produce invalid results when used in computations. A simple solution would be to provide point-to-point or client-server communication to deliver data within the real-time system. However, this communication can become extremely complex when multiple components require the same data at differing rates. Furthermore the communication infrastructure is inflexible. A decoupled solution where suppliers of data do not communicate directly with consumers of data is more efficient and flexible. Such a solution would allow the suppliers of data to produce the data at a rate that is consistent with the data production, and would allow consumers of the data to receive the data at a rate that consistent with the needs of the application. The challenge in this type of solution is to synthesize the provisions of the suppliers with the needs of the consumers so that data arrives at each consumer on time, and temporally valid.

This paper presents a solution to the data distribution problem in static real-time systems. The solution includes an algorithm that determines scheduling parameters to ensure that data that is delivered will be valid when it is used and an implementation using a real-time event service to deliver the data, and a real-time scheduling service to ensure that data is delivered on time.

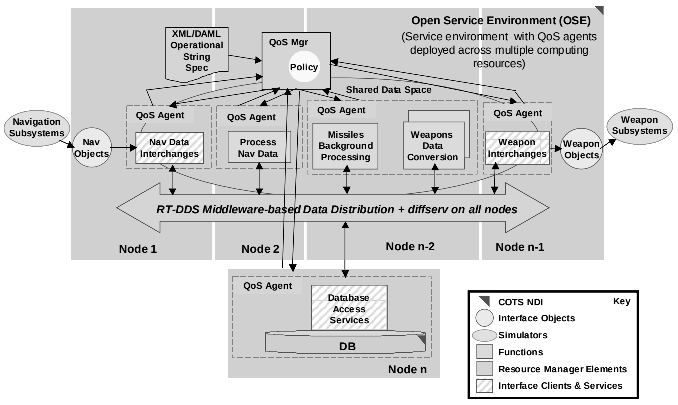


Figure 1 – Navy Weapons Alignment Application

1.1 Navy-Weapons Alignment Application

To illustrate the utility of our static real-time data distribution solution we use a Navy weapons alignment system, depicted in Figure 1. 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

* This work is partially supported by the U.S. Office of Naval Research grant N00014-03-C-0080.

align the weapons according to the latest location of the ship. The data is also processed along the chain. For example, the *Nav Data Interchanges* component receives the raw data and processes it so that the *Process Nav Data* component can use it.

Presently, this type of application uses point-to-point communication to send the data along the chain. This is very inflexible when new components are inserted. For example, if more than one weapon subsystem requires the navigation data (i.e. missiles and torpedoes), there would need to be point-to-point communication from the *Process Nav Data* component to each of the *Background Processing* components. Using a decoupled data distribution mechanism, depicted in Figure 1 as RT-DDS, 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.

The application as depicted in Figure 1 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.

System Characteristics	Data Characteristics
small and medium scale	temporally constrained data
static applications	homogenous data model
static infrastructure	asynchronous data production
unconstrained resources	precise data
hard real-time	fine or course grained data
periodic request timing	single source for each data item

Table 1 - Problem Space for this Work

1.2 Problem Space

To define the problem that the work in this paper addresses, we refer to a *Real-Time Data Distribution Problem Space Taxonomy* [1]. Table 1 shows the specific problem in the space that our work addresses. The Navy weapons alignment application is an example of a problem in this space. For more details on the problem space characteristics, see [1].

2 Related Work

Real-time data distribution has recently emerged as an important area of research. There was a workshop dedicated to the topic (The First Workshop on Data Distribution for Real-Time Systems [3]) in May of 2003. There is also an effort in the Object Management Group (OMG) to standardize data distribution in a middleware service [4]. In [5], the problem of scheduling the broadcast of real-time data is considered. It provides an approximate version of the Longest Wait First heuristic that reduces overhead. Similar work [6] describes a Broadcast on Demand technique that schedules the broadcast using earliest deadline first, periodic or hybrid scheduling algorithms. The work described in [7] is a speculative data dissemination service that uses geographic and temporal locality of reference to determine which data to be disseminated. These techniques take into account the deadline timing constraints of the clients, but do not consider the data temporal consistency.

An application area that has provided various research efforts towards data distribution is embedded sensor networks [8,9,10,11,12]. While all of the work described here provides valuable insights into solving the problem of data distribution in sensor networks, none considers real-time characteristics of the data or of the applications. That is, neither deadlines on data delivery nor temporal consistency of the data is supported.

A large amount of real-time data distribution research has been done at the University of Virginia (UVa) in the context of wireless sensor networks [13,14,15,16,]. This work does address the deadlines of requests. Also, temporal validity is considered in the sense that data values are reported before they expire, but with corresponding confidence values. However, it does not provide assurance that the data is temporally valid when it arrives at the requestor.

There are several commercial products that provide data distribution solutions – most of which are working on becoming compliant with the OMG's Data Distribution Service specification [4]. Real-Time Innovations [18] has a product called NDDS that provides a publish-subscribe architecture for time-critical delivery of data. Thales Naval Nederland [19] has a product called SPLICE [20] that provides a data-centric architecture for mission-critical applications. SPLICE agents act as information brokers to decouple the real-time delivery of data. Both of these products provide valuable real-time features in data distribution, but neither guarantees data temporal consistency or deadlines.

3 Real-Time Data Distribution Model

In this section we describe the model on which our work is based. Figure 2 depicts the elements of the 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. For example, in the Navy weapon alignment application, the data delivered to the weapons subsystem is a complex weapon object. ***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.

DataObject = < OID, Value, TS, OV >
Source = <SID, Node, OID, SP>
Target = <TID, Node, OID, SP>
Dist = <DID, OID, SID, TID, SP>
SP = <P, D, R, E>

Figure 2 - Real-Time Data Distribution Model

The ***Source*** is the entity that produces the data that is to be distributed. ***SID*** is a unique identifier for the source. The ***Target*** is the entity that requests that data be sent to it. ***TID*** is a unique identifier for a target. ***Node*** is the computing element on which the source/target 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 within the period after which the task may start to execute. ***E*** is the worst-case execution time of the task. Note that the source and the target may have different scheduling parameters.

Dist is a distribution of data from a ***Source*** to a ***Target***. A distribution has its own unique identifier ***DID.*** It also has its own scheduling parameters that can be determined by the data distribution algorithm described in Section 4. These algorithms consider the scheduling parameters of the ***Source***, the scheduling parameters of the ***Target***, and the data object validity interval to determine the scheduling parameters of the distribution.

4 Just-in-Time Data Distribution Algorithm

This section describes the *Just In Time Data Distribution (JITDD)* algorithm. The main goal of this algorithm is to examine the specified system, and compute the scheduling parameters for the required data distributions. The algorithm considers the periods of the sources and the periods and deadlines of the targets to determine a deadline for the distribution that will ensure that all targets receive their requested data within the specified deadlines, and that all targets read temporally valid data.

4.1 Assumptions

This JITDD algorithm provides a solution to the problem space described in Section 1. Therefore, the following assumptions are made about the environment in which the algorithm works:

1) The system is static. All nodes, data objects, and scheduling parameters are known a priori.
2) For each object there is one source.
3) Each object has a local node, where it originates.
4) The period of an object’s source is always less than the temporal validity of the object that it supplies.

4.2 Deadline Computation

In order for targets to receive valid data, the scheduling parameters of the data distribution must be such that the distribution will finish delivering the data before the target uses it. Further, there can be more than one target that requires the delivery of the same data object, possibly at different rates, with different deadlines. For example, in the Navy weapon alignment application, each weapon system may require alignment data at different times because different weapons require varying amounts of time to be realigned. Thus, the computation of the scheduling parameters for the distribution should be able to consider the deadlines of all targets that require the data. We have proven in previous work [2] that the period of the distribution must be the same as the period of the source of the data. The release time of the distribution should be at the start of its period. Thus, the crucial computation that must be done is that of the deadline of the distribution within its period.

Let d be the deadline that is computed for a distribution *Dist* from source S to a set of m targets $T_1,\ldots,T_m$ 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 targets of *OID* and the period of the source. Let n be the number of periods that should be considered for the analysis, where n is computed as

$$n = N/p$$

We call N the *superperiod* of the distribution because it represents a complete cycle of all targets 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 3 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.

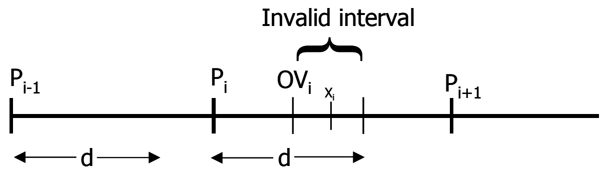


Figure 3 – Deadline Computation

In the algorithm when computing the deadline of the distribution, initially we set it to be equal to its period (*d*=*p*). For each of the *n* periods in the superperiod, the key to computing the deadline of the distribution is to determine if any of the targets will be executing in the invalid interval. If so, it is possible that it could use invalid data. For each target, 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 target and its deadline. There are three cases to consider when calculating the deadline:

1. If no target's data access window overlaps with the invalid interval, the deadline is unchanged because no targets will be using invalid data.
2. If some target's data access window begins at time x_i, after OV_i, i.e. $OV_i < x_i < P_i + d$, then the deadline is changed to x_i-P_i. The target must complete before this data access window begins.
3. If any target's data access window has started before or at OV_i and continues to execute in the invalid interval, then the deadline is changed to OV_i - P_i. This deadline assignment ensures that the target completes its data access before the data becomes invalid, and thus the target 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 target periods in the superperiod. It can also 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.

In [2] we proved that the JITDD algorithm always provides valid data to requesting targets. We also proved that the JITDD algorithm is necessary and sufficient for computing the distribution deadline.

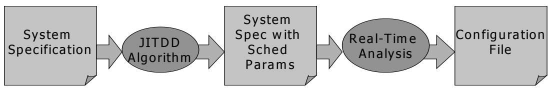


Figure 4 – Off-line Analysis Process

5 Implementation and Evaluation

The implementation of our real-time data distribution is made up of two parts: an off-line analysis and an on-line event-based delivery of data. In this section we describe these two parts of the implementation, as well as a series of tests that we have performed to demonstrate that the implementation correctly represents the theory behind the JITDD algorithm.

5.1 Off-line Analysis

Figure 4 depicts the process that is followed in the off-line analysis of our implementation. The implementation begins with a specification of the system, in the format of our model described in Section 3. An ASCII file containing descriptions of all of the sources, targets, 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.

The augmented system specification is fed into a real-time analysis tool to determine if the system is schedulable. We are currently using the RapidRMA tool by TriPacific Corporation [21]. This requires manually translating the specification into the visual model required by RapidRMA. In ongoing work we are developing tool support to ease this manual translation work (see Section 6). RapidRMA performs a schedulability analysis on the specified model using deadline monotonic, end-to-end analysis [22]. If the system is found to be non-schedulable, then 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.

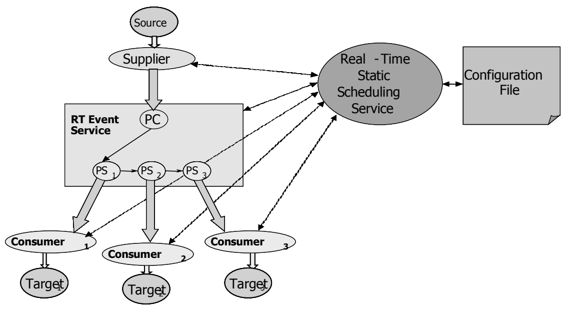


Figure 5 – Runtime Implementation

5.2 On-line Implementation

The runtime component of our implementation executes the model specified in the off-line component described above. The implementation is programmed in C++ and runs on Linux Kernel 2.4.21, with TAO v1.3.5 CORBA software [23] to provide real-time middleware support. The implementation also uses two of TAO's common object services: the Real-Time Event Service (RTES) [24], and the Real-Time Static Scheduling Service

(RTSSS) [25]. The RTES is used as a mechanism for distributing data asynchronously, and the RTSSS provides priority-based scheduling to ensure that deadlines are met. Figure 5 illustrates the implementation using these two services.

Event-based Data Distribution. TAO's RTES provides asynchronous, decoupled communication between sources and targets 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 target.

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. As shown in Figure 5, in our implementation the RTES periodically receives event data from the supplier. The RTES *Proxy Consumer* (PC) receives the event data, and then iteratively passes it along to the *Proxy Suppliers* (PS) for each consumer that has registered to receive this event. The Proxy Suppliers finally deliver the event data to the corresponding consumers, which make the data available for the targets to use. From our formal model of Section 3, a data ***Dist*** is represented by the delivery of event data from the supplier to each consumer.

Scheduling Real-Time Data Distribution. In previous work, we developed the RTSSS that is in the TAO code base [25]. 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 RapidRMA tool. When the system starts up, each of the executing entities (sources, suppliers, consumers, targets, 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. Thus, the deadline computation that is performed in the off-line analysis has provided the deadline for the consumers to deliver to the targets.

5.3 Test Cases

In order to demonstrate the effectiveness of our implementation, we have developed several test scenarios. In each of these scenarios, the main metrics for success are *temporal consistency of delivered data*, and *deadline of data delivery*. That is, we tested to make sure that our claim of ensuring temporal validity and deadline of the distribution holds in our implementation.

The first three of the test scenarios examined the system under "normal" conditions, under network-constrained conditions, and under workload-constrained conditions. In the fourth set of tests, we emulated the Navy weapon alignment application described in Section 1.1. We have developed a system model based on the real application context. 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 have tested four scenarios, each of which is described here. In each scenario, we have 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 6 illustrates the system layout for the first three test scenarios. Below we describe the specific parameters for these scenarios. 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.

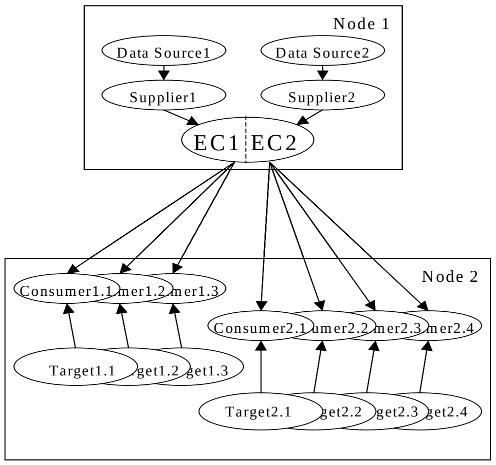


Figure 6 – Test Scenario Set Up

<u>*Scenario 1 – Normal Conditions:*</u> On node 1 in Figure 6, there are two data sources, two suppliers and an event channel for each supplier. Node 2 has the consumers and the targets that will use the data. Table 2 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 targets 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 2, 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 target that requested data from the data source. The entire system model was analyzed in RapidRMA, and found to be schedulable.

Figures 8 and 9 display the deadline results for this scenario, one figure 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 was set up execution that caused the tasks to complete after the deadline.

Name	**Period, µsec**	**Release, µsec**	**Deadline, µsec**	**Exec time, µsec**
DataSource1	100000	0	10000	1500
DataSource2	80000	0	10000	2000
Target1.1	100000	80000	30000	1500
Target1.2	200000	180000	40000	1500
Target1.3	300000	280000	50000	1500
Target2.1	100000	80000	40000	2000
Target2.2	120000	130000	50000	2000
Target2.3	180000	130000	100000	2000
Target2.4	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 2 – Test Scenario Parameters

Figures 10 and 11 show the temporal consistency results for scenario 1. The horizontal line in each graph represents the object validity of the data object being distributed. The other points 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, read temporally consistent data.

Scenario 2 – Workload Constrained: 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%. 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. Test results indicate this to be the case. The graphs for this scenario look very similar to those for scenario 1. Due to space limitations, we do not show these results here.

Scenario 3 – Network Constrained: Scenario 3, is also similar to Scenario 1, except that we assumed that the network was more constrained. That is, we assumed a network delay of 560 µsec. This increased the utilization of the network from 12% to 44.8%. Our intent with this scenario was to show that our implementation still maintains the temporal consistency of both the data and of the execution (deadlines). Test results for this scenario also look very much like Figures 8-11, so we do not show them here.

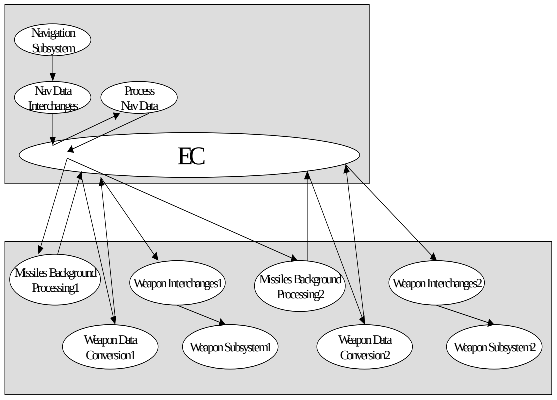


Figure 7 – Navy Weapon Alignment Application Simulation

Scenario 4 – Navy Weapon Alignment Application: We have developed a simulation of the Navy weapon alignment system to demonstrate how our algorithm and implementation work with a real application. Figure 7 illustrates how we have implemented the system. 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 3 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.

The JITDD algorithm was applied to determine the deadline for the delivery of the navigational data to each weapon subsystem. 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. This deadline was computed by taking the shorter of the two computed deadlines for the Weapon Subsystems.

For scenario 4, we have run the system over 100 periods of the *Nav 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 of these tests. The figures indicate that the deadlines are met each time, and the temporal validity of the data is preserved as well.

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 3 – Navy Weapon Alignment Application Simulation Parameters

6 Conclusions and Future Work

This paper has presented a model for real-time data distribution, and an algorithm that determines distribution deadlines such that all data that arrives at specified targets is temporally valid. The paper also describes an implementation that provides the data distribution and enforces the temporal constraints specified by the algorithm. The results of a set of tests indicates that our implementation upholds the theoretical expectations that data be temporally consistent when received, and that deadlines will be made.

As described in Section 1.2, the work in this paper solves a relatively simple problem in the overall real-time data distribution problem space. This work will serve as a foundation for future work in more complex areas in the problem space. For example, in order to support a more dynamic application, our implementation could be extended to provide scenario-based dynamics. This would be a relatively simple change to the current model and implementation. It would require that various scenarios are known and modeled a priori. Then, when the application changes from one known scenario to another, for example, adding another weapons system to the weapon alignment application, the system would be rescheduled based on the previously analyzed model. The system would still be guaranteed to be schedulable, and the data consistency would be guaranteed as well.

A more dynamic application in which data requests or data sources enter with parameters that are not known a priori, presents a more complex problem. In this case, our implementation would require that the computation of distribution deadline be computed online. We would most likely have to rely on a quicker calculation, resulting in a possibly more pessimistic deadline.

We are aware that other middleware solutions to real-time data distribution are emerging in standards bodies. We are closely following the OMG's development of the Data Distribution Service. It is our intent to eventually work with a mature implementation of this service to provide the scheduling and data temporal consistency support that we have provided in our current Event Service-based implementation.

As mentioned in Section 5.1, the current off-line implementation involves specifying the system in a somewhat ad hoc way, and then creating a model in RapidRMA to do the analysis. We are currently working on developing tool support for automating the entire up-front modeling and analysis phase of the implementation. We are examining model-integrating computing tools like GME [26] that will allow us to model the system once, and then insert our own code to perform the JITDD algorithm as well as the real-time analysis.

7 References.

[1] A. Uvarov, V. Fay-Wolfe, Towards a Definition of the Real-Time Data Distribution Problem Space, *Proceedings of the First International Workshop on Data Distribution for Real-Time Systems* (at ICDCS'03), May 2003.

[2] P. Peddi and L. DiPippo, A Replication Strategy for Distributed Real-Time Object-Oriented Databases, *Proceedings of The 5th IEEE International Symposium on Object-oriented Real-time distributed Computing*, Washington, D.C., April 2002.

[3] First International Workshop on Data Distribution for Real-Time Systems, In conjunction with International Conference on Distributed Computing Systems, May 2003.

[4] OMG, Data Distribution Service for Real-Time Systems Specification, OMG document ptc/03-07-07.

[5] M. Karakaya, O. Ulusoy, Evaluation of a Broadcast Scheduling Algorithm, *Lecture Notes in Computer Science*, Springer-Verlag, v. 2151, 2001.

[6] P. Xuan, S. Sen, O. Gonzalez, J. Fernandez, K. Ramamritham, Broadcast on Demand: Efficient and Timely Dissemination of Data in Mobile Environments, *Proceedings of the Fourth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'97)*, 1997.

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

[8] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, D. Estrin, Data-Centric Storage in Sensornets, *First Workshop on Hot Topics in Networks (HotNets-I) 2002.*

[9] F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks, *MOBICOM'02*, September 23-28, 2002, Atlanta, GA.

[10] Y. Yao, J. Gehrke, Query Processing for Sensor Networks, *Proceedings of the 2003 Conference on Innovative Data Systems Research*, Jan. 2003.

[11] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, 2001.

[12] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, In *HICSS '00*, January 2000.

[13] B. C. Lu, B. M. Blum, T. Abdelzaher, J. A. Stankovic, T. He, RAP: A Real-Time Communication Architecture for Large-Scale Wireless Networks, *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02),* 2002.

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

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

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

[17] S. Li, S. H. Son, J. A. Stankovic, Event Detection Services Using Data Service Middleware in Distributed Sensor Networks, *Proceedings of the International Workshop on Information Processing in Sensor Networks (IPSN'03),* April 2003, Palo Alto, CA.

[18] Real-Time Innovations, www.rti.com.

[19] Thales Netherland, www.thales-nederland.nl/.

[20] J. H. van 't Hag Data-Centric to the Max - The SPLICE Architecture Experience, *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, May 19 - 22, 2003.

[21] TriPacific Software, Inc. www.tripac.com.

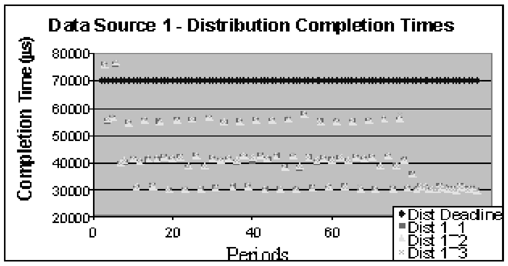
[22] J. Liu, Real-Time Systems, Prentice-Hall, June 2000.

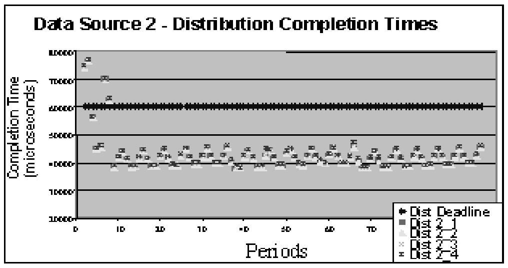
[23] TAO, http://www.cs.wustl.edu/~schmidt/TAO.html

[24] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt, The Design and Performance of a Real-time CORBA Event Service, in *Proceedings of OOPSLA '97*, Atlanta, GA, October 1997, ACM.

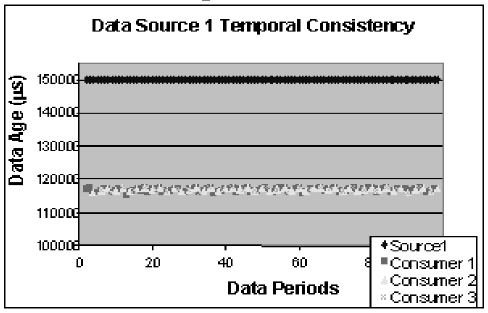
[25] M. Murphy, K. Bryan, Corba 1.0 Compliant Static Scheduling Service for Periodic Tasks Technical Documentation, *URI Technical Report TR04-297*, Jan. 2004.

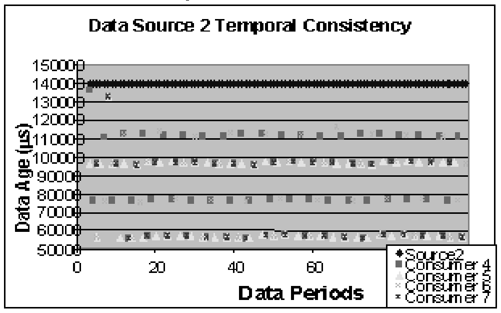
[26] A Ledeczi, M Maroti, A Bakay, G Karsai, J Garrett, C Thomason IV, G Nordstrom, J Sprinkle, P Volgyesi: The Generic Modeling Environment, *Workshop on Intelligent Signal Processing*, Budapest, Hungary, May 17, 2001.



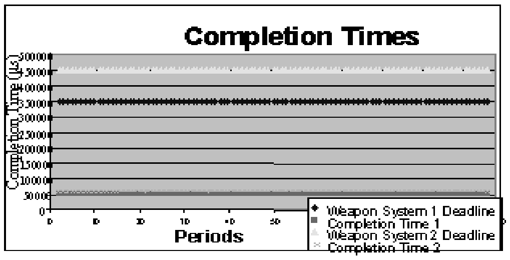


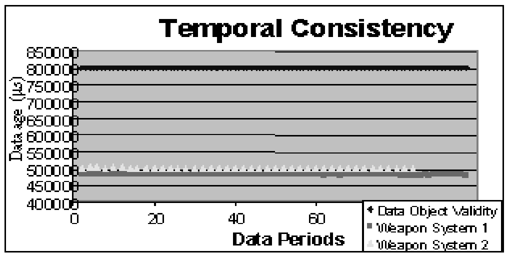
Figures 8 and 9 – Scenario 1 Distribution Completion Time vs. Deadline





Figures 10 and 11 – Scenario 1 Temporal Consistency of Data Sources





Figures 12 and 13 – Navy Weapon Alignment Simulation Results