Self-Describing and Data Propagation Model for Data Distribution Service

Chungwoo Lee[1], Jaeil Hwang[1], Joonwoo Lee[1], Chulbum Ahn[1], Bowon Suh[1], Dong-Hoon Shin[1], Yunmook Nah[1] and Doo-Hyun Kim[2]

[1] Department of Computer Science and Engineering, Dankook University,
126 Jukjeon-dong, Suji-gu, Yongin-si, Gyeonggi-do, 448-701, Korea
{cman, jihwang, jwlee, ahn555, bwsuh, dhshin, ymnah}@dblab.dankook.ac.kr
[2] Department of Internet and Multimedia Engineering, Konkuk University,
1 Hwayang-dong, Gwangjin-gu, Seoul 143-701, Korea
doohyun@konkuk.ac.kr

Abstract. To realize real-time information sharing in generic platforms, it is especially important to support dynamic message structure changes. For the case of IDL, it is necessary to rewrite applications to change data sample structures. In this paper, we propose a dynamic reconfiguration scheme of data sample structures for DDS. Instead of using IDL, which is the static data sample structure model of DDS, we use a self describing model using data sample schema, as a dynamic data sample structure model to support dynamic reconfiguration of data sample structures. We also propose a data propagation model to provide data persistency in distributed environments. We guarantee persistency by transferring data samples through relay nodes to the receiving nodes, which have not participated in the data distribution network at the data sample distribution time. The proposed schemes can be utilized to support data sample structure changes during operation time and to provide data persistency in various environments, such as real-time enterprise environments and connection-less internet environments.

Keywords: data distribution service, dynamic message reconfiguration, persistency, real-time information sharing

1 Introduction

Recently, real-time distributed data processing requirements are ever increasing in many real-world applications, such as weapon systems, sensor-based embedded systems [1,2], airplane software, flight simulator [3,4], and normal business systems [5]. In the past, such real-time processing techniques were primary concern in the military applications, which have to develop embedded systems for weapon systems. Nowadays, it becomes essential to share and utilize various information and knowledge in real-time, even in the normal business environments. For example, the OLTP data need to be transferred in real-time to enterprise data warehouse for more correct decision making. To realize such real-time information sharing in more generic platforms, it is especially important to support dynamic message structure

changes. To realize such real-time distributed environments, real-time distributed middleware technologies are required. RT-CORBA, which was evolved from CORBA, TMO, which was developed at UCI, and DDS (Data Distribution Service), which is announced as a standard specification by OMG, are representative middleware technologies for such environments. RT-CORBA is a standard proposed by RT-SIG of OMG to allow QoS specification, real-time service and performance optimization, which have not been well supported in CORBA [6,7]. TMO is a natural and syntactically minor but semantically powerful extension of the conventional objects [8,9]. DDS is a publish-subscribe model for real-time environments and was adopted as a middleware standard to develop data distribution services by OMG [10,11,12]. These middleware technologies have some problems to be used in the real-time business environments, because they often require dynamic changes of message structures, as compared to the embedded environments, which seldom require data structure changes. Such changes are required because of database schema changes or XML document structure changes. For the case of IDL (Interface Definition Language), it is necessary to rewrite applications to change data sample structures.

In this paper, we propose a dynamic reconfiguration scheme of data sample structures for DDS and explain the APIs to support such dynamic restructuring of data sample structures in distributed real-time applications. We also describe how to support persistency, which is one of important QoS (Quality of Service) elements of DDS. Instead of using IDL, which is the static data sample structure model of DDS, we use a self describing model using data sample schema, as a dynamic data sample structure model to support dynamic reconfiguration of data sample structures. In our case, we can dynamically support data sample structure changes, because data sample schema can be determined in run-time. We explain how to create and change data sample structures and how to send and receive data samples using data sample schema. We also propose a data propagation model to provide data persistency in distributed environments. We guarantee persistency by transferring data samples through relay nodes to the receiving nodes, which have not participated in the data distribution network at the data sample distribution time. Finally, to show the usefulness and efficiency of our schemes, some experimental results are shortly provided. The proposed schemes can be utilized to support data sample structure changes during operation time and to provide data persistency in various environments. The remainder of this paper is organized as follows. Section 2 describes overview of data distribution service. A dynamic reconfiguration scheme of data sample structures for DDS are proposed in Section 3. Section 4 explains how to support persistency and section 5 provides some experimental results. Finally, section 6 concludes the paper.

2 Overview of Data Distribution Service

DDS is networking middleware that simplifies complex network programming. It implements a publish/subscribe model for sending and receiving data, events, and commands among the nodes. Nodes that are producing information (publishers) create

topics (e.g., temperature, location, pressure) and publish *samples* (data values of topics). DDS takes care of delivering the sample to all subscribers that declare an interest in that topic. DDS handles all the transfer chores: message addressing, data marshaling and de-marshalling (so subscribers can be on different platforms than the publisher), delivery, flow control, retries, etc. Any node can be a publisher, subscriber, or both simultaneously. The DDS publish-subscribe model virtually eliminates complex network programming for distributed applications.

The DDS specification describes two levels of interfaces. A lower DCPS (Data-Centric Publish-Subscribe) level is targeted towards the efficient delivery of the proper information to the proper recipients. According to the conceptual model of DCPS [13], 'Publisher,' 'Subscriber,' 'DataReader,' 'DataWriter,' and 'Topic' are 'DomainEntity.' Also, 'DomainEntity' and 'DomainParticipant' are 'Entity.' 'Entity' has a relationship with 'QosPolicy.' Each 'Publisher' can have multiple 'DataWriters' and each 'Subscriber' can have multiple 'DataReaders.' An optional higher DLRL (Data Local Reconstruction Layer) level allows for a simple integration of DDS into the application layer.

In the network-centric model usually used in previous middleware technologies, the position of receiving node must be specified, like 'Node 1 sends data to Node 2.' Therefore, special treatments were required for sending nodes when positions of receiving nodes are changed or new receiving nodes are inserted. As compared to this, the DCPS of DDS does not specify the position of receiving nodes. Sending nodes just specify topic of data and receiving nodes receive data when they are interested in the topic of current data. For example, node 1 sends data to the DDS network specifying that the topic of that data is 'A.' At node 2, if the topic of the data that the node wants to receive is 'A,' it waits for that topic from the DDS network and receives the data having that topic. As such, the DDS network is extendable and flexible, because position changes of receiving nodes and insertions of new receiving nodes do not affect the network.

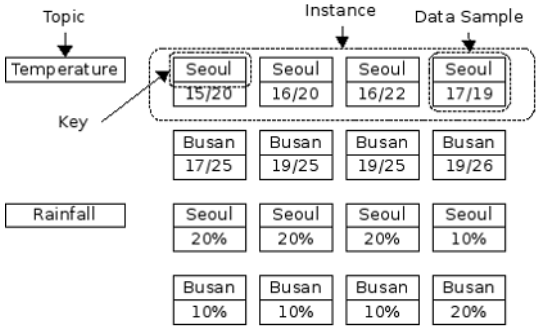

Fig. 1. Instance examples

Figure 1 shows example instances of two topics 'Temperature' and 'Rainfall.' The field values, such as 'Seoul' and 'Busan,' which identify instances, are called *keys*. A group of data having the same key is called an *instance*. Each instance shows the history of data samples having the same key. Each individual data within an instance is called a *data sample*, which is the unit of data transmission in DDS networks. The DDS provide QoS elements, such as USER_DATA, TOPIC_DATA, GROUP_DATA,

DURABILITY, PRESENTATION, DEADLINE, OWNERSHIP, LIVELINESS, PARTITION, RELIABILITY, HISTORY, etc.

3 Self-Describing Model to Support Dynamic Reconfiguration

In this section, we describe a self-describing model and data sample schema to support dynamic reconfiguration of data samples.

3.1 Self-Describing Model

The IDL is used to define data structures of data samples in the DDS standard specification. Therefore, data structures of data samples are fixed and application programs have to be rebuilt to change data structures during system operation. To allow dynamic definition of sample structures, there must exist ways to define data structures and transmit such structures dynamically.

Fig. 2. Schema model

Figure 2 shows the proposed schema model. In our self-describing model of DDS, data schema is first broadcasted as a built-in topic and then data samples, with data structure identifiers attached, are transmitted.

3.2 Data Sample Schema

An entity to define a data structure of data samples is called a *data sample schema*, which is an enumerated list of data types of corresponding data fields in a data sample. This structure must exist in the DomainParticipant before the corresponding data samples are created or interpreted. The structure of internal topic DCPSSchema to transmit schema information is shown in Table 1.

Table 1. The structure of DCPSSchema

Field name	Type	Meaning
key	BuiltinTopicKey_t	DCPS key to identify registration
participant_key	BuiltinTopicKey_t	DCPS key of the participant which make registration of data sample schema
topic_name	string	topic name associated with data sample schema
schema_seq	integer	sequence number of data sample schema
field_count	integer	number of fields of data sample schema
field_types	TypeArray_t	array of field data types

The ‘key’ is used to manage schema changes. The ‘participant_key’ is the DCPS key which make registration of the given data sample schema. Therefore, only data samples created by this participant can reference this schema. In the ‘topic_name,’ the topic of data sample referencing this schema is specified. The sequence number given to this schema is recored in the ‘schema_seq.’

Figure 3 shows an example using data sample schema. The schema S with key value 1 and sequence number 1, created by the participant 1 to send topic T, is broadcasted to the DCPSSchema. The schema S consists of 3 fields, with data types {INTEGER, STRING, FLOAT}. For data transmission, the Publisher writes a sample D1 with the sample schema S. It first writes the key value 1 and sequence number 1 for the sample schema S in the header of the data sample. Then, the values of each fields, 1234(the value of 0th field), 3(the length of the first field), “DKU”(the value of the first field), 0.5678(the value of the second field) are written. The Subscriber receiving this data sample D1 uses the key value 1 and sequence number 1 to identify the data sample schema S and interprets the data sample using this schema.

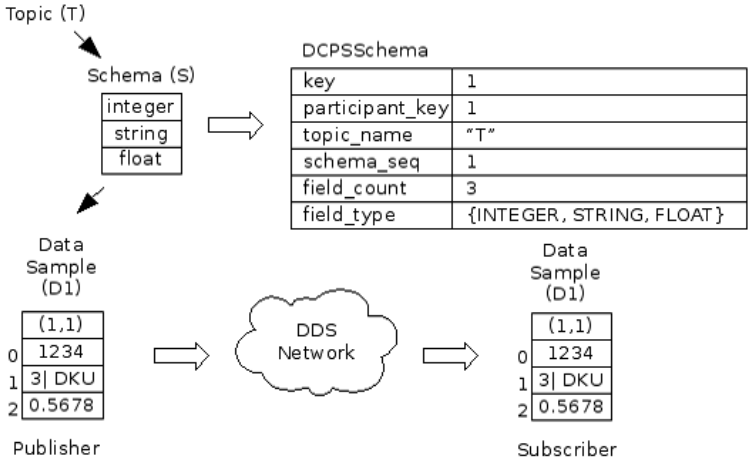

Fig. 3. Example of schema usage

Now, let’s see how data sample structures can be dynamically reconfigured, as shown in Figure 4. Suppose we have data samples D1 and D2, all following the schema S. The key value and schema sequence number in the header information of D1 and D2 is (1,1). Also suppose that the data sample schema S is changed dynamically to S’, having field data types {INTEGER, STRING} and such change is updated into the DCPSSchema. Data samples, such as D3 and D4, which are created after this schema change, will have new header information (1,2), which means the key value is the same but the sequence number is incremented(changed). The Subscriber can now interpret new data samples by new schema S’, because the sequence number in the header of data samples is now 2.

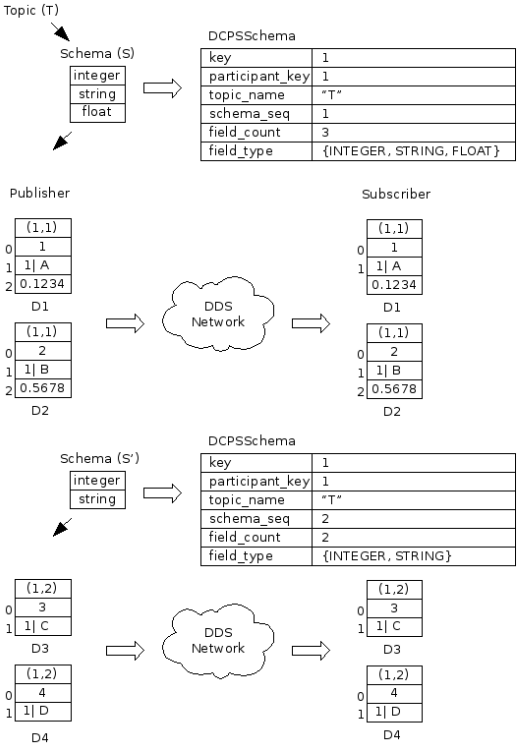

Fig. 4. Example of schema change

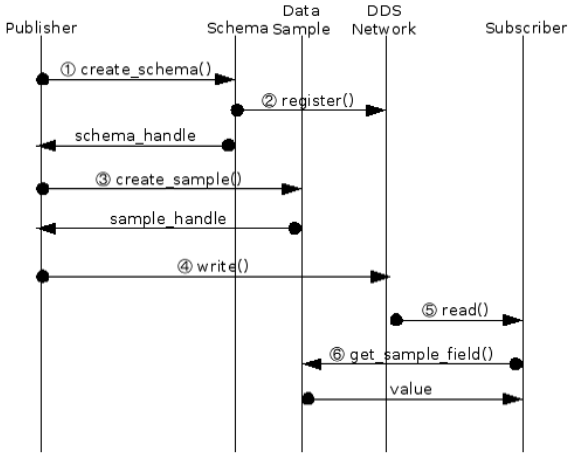

Fig. 5. Schema handling sequence

The workflow to support the dynamic reconfiguration of data sample structures is shown in Figure 5. The Publisher defines the schema(data structure) to send data(①).

Then, this schema is registered to the DDS network(②). The Publisher then creates a data sample regarding this schema(③) and transmits it to the DDS network(④). The Subscriber receives this data sample(⑤) and then looks for the schema referenced by this data sample and interprets this data sample using the corresponding schema(⑥).

We have implemented the following APIs to support dynamic configuration of data sample structures.

- **APIs for schema definition:** create_schema_handle(), delete_schema_handle(), register_schema_handle(), unregister_schema_handle(), insert_schema_fields(), append_schema_fields(), remove_schema_fields(), replace_schema_fields(), get_schema_fields_type(), get_schema_fields_count
- **APIs for instance definition:** set_schema_key_fields(), get_schema_key_fields(), create_key_handle(), delete_key_handle(), extract_key_handle(), set_key_field(), get_key_field(), register_instance(), unregister_instance(), lookup_instance()
- **APIs for data writing:** create_sample_handle(), delete_sample_handle(), write(), read(), set_sample_field(), get_sample_field()

4 Data Propagation Model to Support Data Persistency

The enterprise data should not be lost during transmission in distributed environments. The DURABILITY and RELIABILITY are DDS QoS elements related with data loss. The DURABILITY is related with the persistency of data transmitted by publishers and the RELIABILITY is related with the reliability of communication lines. If the DURABILITY is VOLATILE, data sent by publishers are not saved at all and those subscribers which joined at the network later than the data publishing time can not read the data. If the DURABILITY is TRANSIENT_LOCAL or TRANSIENT, the published data is saved in the publishers' memory for later request and those subscribers which joined at the network later than the data write time can read the data. But, it is impossible to read the data after the corresponding publishers are disconnected from the network. If the DURABILITY is PERSISTENT, all subscribers can read their data at any time, even after the publishers are disconnected from the network.

4.1 Data Propagation Model

To support persistency in DDS network, we propose a data propagation model, as shown in Figure 6. The Subscriber C, which could not receive data from the Publisher A because it did not exist at the broadcasting time, can receive that data from the intermediary node, such as the Node B, even after the original Publisher, such as Node A, is disconnected from the network.

To realize this propagation model, we need intermediary nodes, also called relay nodes, that can propagate data instead of the publishing node. In our method, every participant whose DURABILITY is PERSISTENT can take role of intermediary nodes. Each intermediary node is required to have a persistent repository for data

propagation. Each intermediary node stores its received data in its persistent repository and forwards received data if required. The structure of persistent repository for an intermediary node is shown in Table 2.

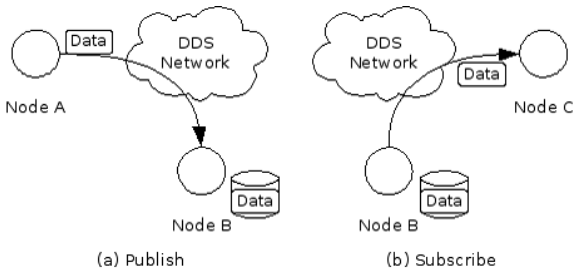

Fig. 6. Data propagation model

Table 2. The structure of persistent repository

Field name	Type	Meaning
participant_key	BuiltinTopicKey_t	DCPS key of the publisher which sent data sample
instance_key	KeyValue_t	instance key of data sample
sample_count	integer	number of stored data samples
samples	Integer	array of {sample_seq, sample}

When a data sample is received by an intermediary node, it is stored in the entry of persistent repository with the matching 'participant_key' and 'instance_key.' The corresponding 'sample_count' is incremented, while the sample itself is stored in the 'samples' array. Because the resources for persistent repositories are limited, we can not allow every data sample to be stored. We have to manage only recent history by limiting the number of history according to the system configuration. We can decide the detail limitation by using the QoS element DURABILITY_SERVICE, as shown in Table 3.

Table 3. The QoS element DURABILITY_SERVICE

attribute	Meaning
service_cleanup_delay	interval to delete all samples in persistent repository
history_kind	store every sample(KEEP_ALL) or recent sample(KEEP_LAST)
history_depth	number of recent samples to be kept, when history_kind is KEEP_LAST
max_samples	maximum number of samples to be stored in the repository
max_instances	maximum number of instances to be stored in the repository
max_samples_per_instance	maximum number of samples to be stored per instance

Figure 7 is an example showing the use of persistent repositories. Suppose the DURABILITY of all nodes A, B and C are specified as PERSISTENT. Also, assume

the ‘history_kind’ is KEEP_LAST and the ‘history_depth’ is 2 for all nodes, meaning that all three nodes keep last 2 data samples.

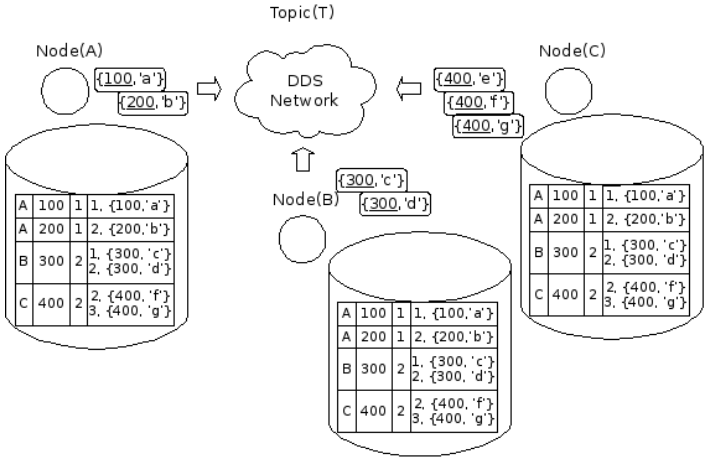

Fig. 7. Example using persistent repositories

Figure 7 shows the status of persistent repositories after the Node A send data samples {100, ‘a’}, {200, ‘b’}, the Node B send data samples {300, ‘c’}, {300, ‘d’}, and the Node C send data samples {400, ‘e’}, {400, ‘f’}, {400, ‘g’}. Here, the values, such as 100 and 200, are key values.

4.2 Data Propagation Protocol

The protocol to receive data which can not be received at the broadcasting time is shown in Figure 8.

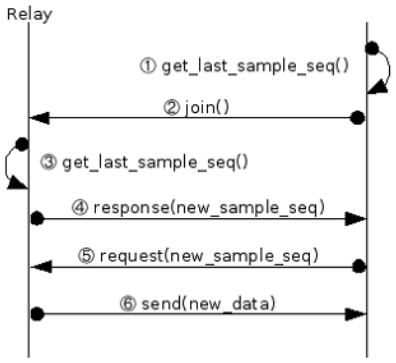

Fig. 8. Data propagation protocol

The newly joined subscriber checks the last sample number from its own persistent repository(①). The new subscriber then sends ‘join’ message to the DDS network(②). Each intermediary node gets its last data sample number(③) and responds this last

number to the subscriber(④). The new subscriber compares sample numbers and send transmission request, if the sample number of itself is less than one of the sample numbers of intermediary nodes(⑤). The intermediary node which receives transmission request sends the required data to the new subscriber(⑥).

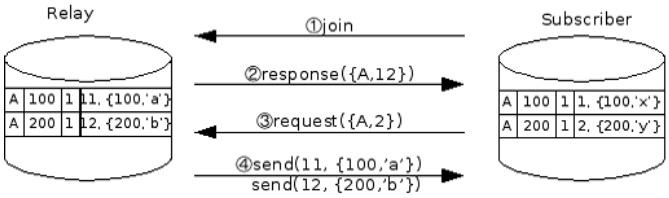

Fig. 9. Case of new subscriber with lower sample number

Figure 9 shows an example case where newly joined subscriber has a last sample number(2) less than one of the last sample number(12) of the intermediary nodes. The relay node send two samples {100, 'a'} and {200, 'b'} whose sample numbers are greater than 2.

5 Experiments

For our experiments, we used the API of the open source project ORTE (Ocera Real-Time Ethernet) 0.3.1[14] which implements the RTPS (Real-Time Publish-Subscribe) protocol [15], the low level communication protocol of DDS. In a distributed environments consisting of 1 publisher and 3 subscribers, we compared turnaround delays of DDS/IDL with DDS/Schema(the proposed method). We also compared such delay for the case implemented by Sun Java System Message Queue 3.7[16], which is a kind of message queue middleware for ESB(Enterprise Service Bus) service.

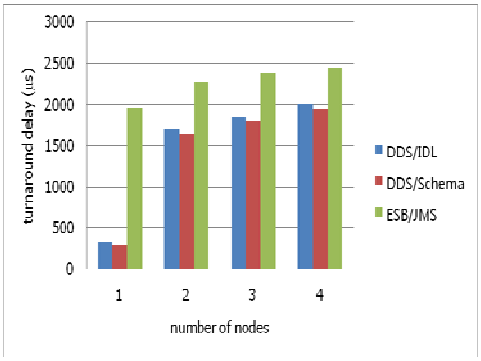

Fig. 10. Turnaround delay time in distributed environments

Figure10 shows the turnaround delay in distributed environments. The turnaround delay for DDS/IDL was 2,012μs, DDS/Schema was 1,956μs and ESB/JMS was

2,448μs. This result shows us that the performance of DDS/IDL and DDS/Schema is almost the same, but the proposed method has better ability allowing dynamic reconfiguration of data sample structures without re-programming overhead.

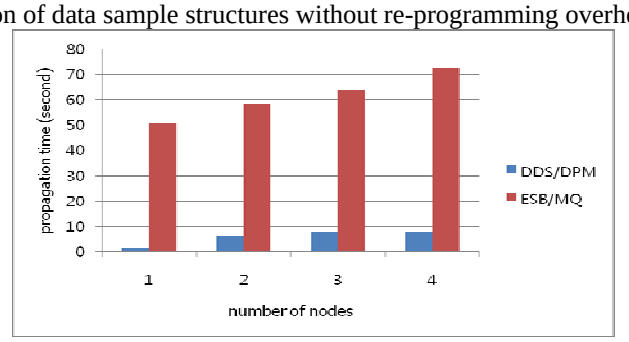

Fig. 11. The initial sample propagation time of subscribers

We also compared the data propagation time in the test system consisting of 4 participants. Figure 11 shows the initial data sample propagation time of subscribers. The propagation time for DDS/DPM(the proposed method) was 8 seconds, while ESB/MQ was 72.1 seconds, which shows us that our persistency mechanism is faster than ESB/MQ.

6 Conclusion

In this paper, we proposed a dynamic reconfiguration scheme of data sample structures for DDS. We also described how to support persistency, which is one of important QoS elements of DDS. Instead of using IDL, which is the static data sample structure model of DDS, we used a self describing model using data sample schema. We also proposed a data propagation model to provide data persistency in distributed environments. We guaranteed persistency by transferring data samples through relay nodes to the receiving nodes, which could not participated in the DDS network at the broadcasting time. Finally, some experimental results to show the usefulness and efficiency of our schemes were shortly provided. The proposed schemes can be utilized to support data sample structure changes during operating time and to provide data persistency in various environments, such as real-time enterprise environments and connection-less internet environments.

Our effort for experimental implementation of the proposed techniques is at an early stage. The detail algorithms to handle node failures and to optimize message passing overheads need to be carefully designed to realize our approach. We believe that both analytical and experimental studies of the communication overhead and performance aspects on massive number of nodes are highly meaningful subjects for future research.

Acknowledgments. This research was supported by the Ministry of Knowledge Economy, Korea, under the Information Technology Research Center support program supervised by the Institute of Information Technology Advancement (grant

number IITA-2008-C1090-0801-0031). This work was also supported by the Korea Science and Engineering Foundation (KOSEF) grant number R01-2007-000-20958-0 funded by the Korea government (MOST).

References

1. Pardo-Castellote, G. and Schneider, S.: The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications. Proc. IEEE International Conference on Robotics and Automation. IEEE Press. (1994) 2870-2876
2. Schneider, S. A., Ullman, M. A., and Chen, V. W.: ControlShell: A Real-Time Software Framework. Proc. IEEE International Conference on Systems Engineering. IEEE Press. (1991) 129-134
3. Kuhl, F., Weatherly, R., and Dahmann, J.: Creating Computer Simulation Systems. Prentice Hall. (1999)
4. Dahmann, J. S. and Morse, K. L.: High Level Architecture for Simulation: An Update. Proc. 2nd International Workshop on Distributed Interactive Simulation and Real Time Applications. (1998) 32-40
5. Khosla, V. and Pal, M.: Real Time Enterprises: A Continuous Migration Approach. Information, Knowledge, Systems Management 3(1). (2002) 53-79
6. Schmidt, D. C. and Kuhns, F.: An Overview of the Real-Time CORBA Specification. Computer 33(6). IEEE Press. (2000) 56-63
7. Cooper, G., DiPippo, L., Esibov, L., Ginis, R., Johnston, R., Kortman, P., Krupp, P., Mauer, J., Squadrito, M., Thuraisingham, B., Wohlever, S., and Wolfe, V.: Real-Time CORBA Development at MITRE, NRaD, Tri-Pacific and URI. Proc. IEEE Workshop on Middleware for Distributed Real-Time Systems and Services. IEEE Press. (1997) 69-74
8. Kim, K. H.: A TMO Based Approach to Structuring Real-Time Agents. Proc. 14th IEEE International Conference on Tools with Artificial Intelligence. IEEE Press. (2002)165-172
9. Kim, K. H.: APIs for Real-Time Distributed Object Programming. IEEE Computer. (2000) 72-80
10. Zieba, B. and Sinderen, M.: Preservation of Correctness During System Reconfiguration in Data Distribution Service for Real-Time Systems(DDS). Proc. 26th IEEE International Conference on Distributed Computing Systems Workshops. IEEE Press. (2006) 30-35
11. Pardo-Castellote, G.: OMG Data-Distribution Service: Architectural Overview. Proc. 23rd International Conference on Distributed Computing Systems Workshops. (2003) 200-206
12. Hugues J. and Pautet, L.: A Framework for DRE middleware, an Application to DDS. Proc. 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing. IEEE Press. (2006) 224-231
13. Data Distribution Service for Real-time Systems, V1.2. OMG. (2007)
14. Smolik, P., Sebek, Z. and Hanzalek, Z.: ORTE-Open Source Implementation of Real-Time Publish-Subscribe protocol. Proc. 2nd International Workshop on Real-Time LANs in the Internet Age. Porto: Universidade de Porto. (2003) 68-72
15. Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification, V.1.0. ICE. (2004)
16. Schmidt, M. T., Hutchison, B., Lambros, P. and Phippen, R.: The Enterprise Service Bus: Making Service-Oriented Architecture Real. IBM Systems Journal 44(4). (2005) 781-797