Preservation of Correctness During System Reconfiguration in Data Distribution Service for Real-Time Systems (DDS)

Bogumil Zieba
Thales Nederland B.V,
AWS Department
Haaksbergerstraat 49, Hengelo (O), 7554 PA,
The Netherlands
bogumil.zieba@nl.thalesgroup.com

Marten van Sinderen
University of Twente,
Department of Computer Science, Centre for Telematics and Information Technology,
PO Box 217, 7500 AE, Enschede,
The Netherlands
m.vansinderen@utwente.nl

Abstract

This paper addresses dynamic reconfiguration of distributed systems that use a publish/subscribe (pub/sub) middleware. The objective of dynamic reconfiguration is to evolve incrementally from one system configuration to another at run-time in order to e.g., ensure the reliability of the system. The correctness notion of a distributed system is introduced that assures that the system parts that interact with entities under reconfiguration do not fail because of reconfiguration. We analyse the OMG specification of pub/sub systems - DDS (Data Distribution Service for Real-Time Systems) with respect to its support for the correctness preservation during reconfiguration. We notice that the DDS specification defines such an architecture and behaviour of the pub/sub system that automatically preserves correctness. This differentiates the DDS from other middleware technologies that require that the correctness preservation is guaranteed on application level or by reconfiguration manager/controller. We give several examples of automatic correctness preservation supported by the DDS.

1. Introduction

The reliance on software systems imposes restrictions on the possibility of restarting them or taking them off-line. It is usually not acceptable, e.g., for economical or safety reasons, to cause major interruptions in the service these systems provide. They have high availability, adaptability and maintainability requirements and they have to cope with advances in technology, modifications of their operating environment and ever-changing human needs [1, 2]. The aim of **dynamic reconfiguration** is to allow a system to evolve at run-time [2, 3], as opposed to design-time, while introducing little (or ideally no) impact on the system's execution. In this way, systems do not have to be taken off-line, rebooted or restarted to accommodate changes. Performing reconfiguration on a running system is an intrusive process. Reconfiguration may imply, for example, addition, removal, migration or replacement of reconfigurable entities and interference with ongoing interactions between entities. Reconfiguration management must assure that system parts that interact with entities under reconfiguration do not fail because of reconfiguration [1, 2].

Publish/subscribe (pub/sub) systems have recently gained significant attention because their computational model fits well when dealing with real-time, distributed data-centric applications [4, 5]. Pub/sub systems feature a data-centric communication pattern, where applications publish (supply or stream) large amount of "data" samples, which are then available to remote applications that are interested in them. It uses an interaction model that consists of information publishers, which publish events to the system and information subscribers, which subscribe to events of interest within the system. An event can be seen as a special message sent by an information publisher and (implicitly) addressed to the set of information subscribers, which issued a subscription that matches the event [6]. A participant may simultaneously publish events and subscribe to the other events.

DDS (Data Distribution Service for Real-Time Systems Specification) is a recent OMG (Object Management Group) specification for interoperable

pub/sub middleware. The purpose of this specification is to offer standardised interfaces and behaviour of pub/sub systems [7]. In this paper we analyse the DDS with respect to its support for the correctness preservation during reconfiguration.

The rest of the paper is structured as follows. Section 2 presents an overview of the DDS architecture. Section 3 introduces a notation of correctness preservation during reconfiguration. Section 4 outlines the DDS systems architecture and behaviour during reconfiguration with reference to notation presented in section 3. Section 5 discusses related work. Finally, section 6 presents our conclusions.

2. Overview of DDS

We focus the analysis of the DDS on one part of the specification – Data Centric Publish-Subscribe (DCPS). The DCPS specification covers the lower level API for applications to communicate with other applications and the pub/sub infrastructure that is responsible for efficient events delivery. It consists of the following entities [7] (figure 2): **Domain** – creates a 'virtual network' of pub/sub participants. Only participants that belong to the same domain can communicate. The domain separates participants allowing several independent distributed applications to coexist in the same physical network without interfering or even being aware of each other (figure 1).

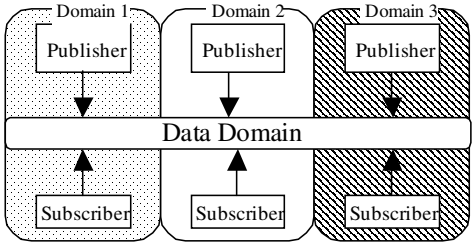

Fig 1. Domains in the DDS

DomainParticipant – container for participants in the domain. It acts as a factory for the Publishers, Subscribers and Topics. **Publisher** – container for a group of DataWriters that acts as a factory for them. QoS (Quality-of-Service) can be assigned to the Publisher that will be collectively attached to corresponding DataWriters within the Publisher. **DataWriter**– this is the main access point for applications publishing data samples. **Subscriber** – container for a group of DataReader that acts as a factory for them. QoS can be assigned to the Subscriber that will be collectively attached to corresponding DataReaders within the Subscriber. **DataReader** – the main access point for applications for receiving data samples. **Topic** is the most basic description of the data to be published and subscribed to. A Topic is identified by its name, which must be unique in the whole Domain. In addition, it fully specifies the type of the data that can be communicated when publishing or subscribing to the Topic.

The DDS relies on the use of QoS. A QoS is a set of characteristics that controls some aspects of the behaviour of the DDS Service. QoS may be associated with all entities in the system such, as Topic, DataWriter, DataReader, Publisher, Subscriber and DomainParticipant.

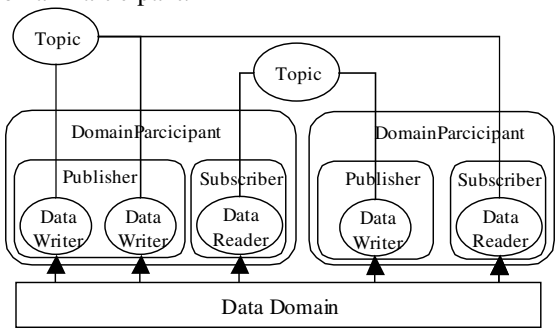

Fig 2. The DDS architecture showing relationships between entities

3. Correctness preservation during reconfiguration

For any distributed, middleware-based system consistency preservation during reconfiguration is a major issue. A system can become useless in case the preservation consistency is ignored. The system under reconfiguration must be left in a "correct" state after reconfiguration. In order to support the notion of correctness of a distributed system, three aspects of correctness requirements are identified [1, 3, 8]:

1) The system must comply with **structural integrity requirements**. Structural integrity requirements constrain the structure of a system in terms of the relationships between entities and the ways in which these entities might be put together. For example, in terms of CORBA it is satisfying the interface definition of the original object and reference to new reconfigured object.

2) Entities in a distributed system need to be in **mutually consistent states** if they are to interact successfully with each other. Entities are said to be in mutually consistent states, if each interaction between them, on completion, results in a transition between well-defined and consistent states for the parts involved. Interactions are the only means by which entities can affect each other's state. In order to provide

an example, we can consider that object A invokes an operation on B. Objects A and B are said to be in mutually consistent states if A and B have the same assumptions on the result of the interactions between them. To be more specific, either both of them perceive that an invocation has occurred successfully or both of them perceive that the invocation has failed. Suppose the change manager decides to replace B by B' after A initiated an operation invocation on B. For the resulting system to be in a consistent state, either (i) the invocation has to be aborted, A is informed and synchronization is maintained; or (ii) B receives the request, finishes processing it and sends the response and then is replaced by B'; or, (iii) B is replaced by B' and B' has to honour the invocation, by processing the request and sending a response to A. In case none of these alternatives occur, A might be kept waiting for a response forever.

3) The application **state invariants** are predicates involving the state of (a subset of) the entities in a system. The preservation of safety and liveness properties of a system depends on the satisfaction of these invariants. For example, let us consider an object that generates unique identifiers. An application-state invariant could be "all identifiers generated by the object are unique within the lifetime of the system". In order to preserve this invariant, the new version of the object must be initialised in a state that prevents it from generating identifiers that have been already used by the original object.

4. The DDS support for correctness preservation during reconfiguration

In this section we present the DDS architecture and QoS-controlled behaviour with reference to the notation of correctness preservation, presented in previous section.

4.1. Structural Integrity Requirement

The following DDS properties: **decoupling** between publishers and subscribers, **symmetric design** of architecture and QoS-controlled behaviour influence the accomplishment of structural integrity requirement. The decoupling, that is the essential characteristic of any pub/sub system, can be detailed in: **Space** - interacting parties do not need to know each other; **Time** - parties do not need to be actively participating in the interaction at the same time; **Flow** - asynchrony of the model. Interacting participants do not directly reference to each other but through the pub/sub infrastructure, which acts as a broker for communication. Therefore the responsibility for realizing the structural integrity requirement is shifted from application level to pub/sub middleware infrastructure.

The design of the DDS architecture is symmetric. That means that in the DDS pub/sub system there are no centralized, single points of failure or privileged participants (like e.g., in CORBA - Name Server). The identical pub/sub middleware infrastructure is run on every node taking part in a communication. Each node has a **global knowledge** of all topics, publishers and subscribers within the same domain.

4.1.1 QoS-controlled behaviour during subscriber reconfiguration

The DDS provides QoS policies that determine the pub/sub system behaviour in case of subscriber reconfiguration (unavailability). Appointed data samples to the 'off-line' subscriber can be either discarded or stored for 'late-joining' subscribers (subscribers that do not exist at the moment of data production, but may appear in future). Such pub/sub system behaviour may be realized using the following QoS:

1) **Durability** QoS - expresses if data should 'outlive' their writing time. It has the following parameter values: *volatile* (publisher does not need to keep any data samples on behalf of any subscriber that is not known by the publisher at the time the data sample is written), *transient* (keep some samples so that they can be delivered to any potential 'late-joining' subscriber; it depends on other QoS such as History and Resource Limits) and *persistent* (data samples are kept on permanent storage, so that they can outlive a system session). 2) **History** QoS – specifies the total number of samples that are stored per instance of publisher for 'late-joining' subscribers. The maximum number of instances can be specified in the Resource Limits. 3) **Resource** Limits QoS – the amount of resources reserved for storing data samples. 4) **Reliability** QoS - indicates the level of reliability of data delivery. These levels are ordered, from *best_effort* (unreliable delivery, without data retransmission) to *reliable* (fully reliable data delivery). Subscriber sends an acknowledgment of receipt of each data sample. The DDS durability service stores unacknowledged data samples for later delivery. We present the use-case of subscriber reconfiguration in figure 3. The assumptions are that the subscriber requires reliable data delivery and data samples have set persistent durability QoS.

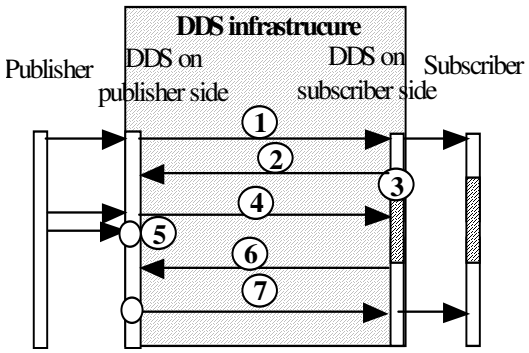

Fig. 3. Use-case of subscriber's reconfiguration.

1) The publisher produces data samples that are delivered to the subscriber. 2) DDS on the publisher side receives the acknowledgement of each delivered data sample. 3) The subscriber goes off-line for reconfiguration e.g., migration from one computer node to another. 4) The publisher produces data sample, but the DDS infrastructure at publisher side does not receive acknowledgments of data delivery. 5) The durability service at the publisher side stores unacknowledged data samples on permanent storage. 6) The reconfiguration finishes and the subscriber resume operation. It broadcasts a data sample containing its subscription (DCPSSubscription topic - built-in topic in DDS). 7) DDS at the publisher side receives the subscription and it sends out data samples that are stored on permanent storage.

4.1.2 QoS-controlled behaviour during publisher reconfiguration

In many mission-critical systems (e.g., Naval Command and Control Systems) an additional and redundant publisher is introduced in order to provide continuous publication of data samples during reconfiguration of the publisher. It takes over the role of the 'main publisher' for the time of reconfiguration. The DDS defines the Ownership QoS that determines ownership of data samples. Certain types of data samples (defined as Topic) can be either updated (owned) by many publishers (Ownership value is set to *shared*) or by one instance of the publisher (Ownership value is set to *exclusive*). When two or more publishers publish data samples that have exclusive ownership, data samples from only one of them are delivered and those from others are discarded. The preferred publisher (owner of data samples) is determined based on the parameter value *ownership strength*. The publisher with highest value of the ownership strength is the preferred publisher. The DDS provides the parameterised mechanism to discover and keep track of the presence of publishers in the domain. It allows assigning to participants or data samples **Liveliness** QoS that enforces entities to send an "alive" signal every period of time specified by the liveliness value. This QoS is used by the DDS to determine the owner of data samples.

A crucial requirement for the pub/sub systems is to accomplish continuous data publication. We present the use-case of publisher reconfiguration in figure 4. Data samples have set exclusive ownership. A second publisher is introduced in order to take over the role of the 'main publisher' for the reconfiguration time. Both of them publish identical data samples.

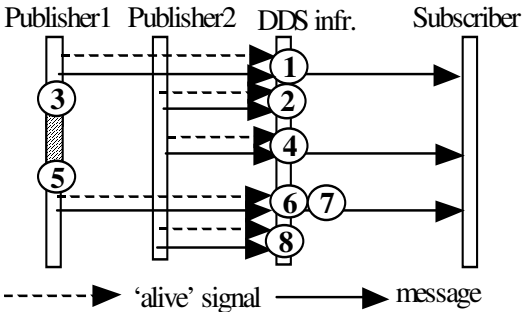

Fig. 4. Use-case of publisher reconfiguration

1) The DDS infrastructure delivers data samples from publisher 1. 2) The DDS infrastructure discards data samples from publisher 2 due to the higher value of ownership strength of publisher 1. 3) The reconfiguration of the publisher 1 starts. 4) The DDS infrastructure does not receive 'alive' signals from publisher 1 and establishes publisher 2 as the owner of data samples. Data samples from publisher 2 are accepted and delivered to the subscriber. 5) Publisher 1 resumes operation. 6) The DDS infrastructure receives the 'alive' signal from publisher 1 and re-establishes it as the data owner due to the higher value of the ownership strength. 7) Data samples from publisher 1 are delivered. 8) Data samples from publisher 2 are discarded.

4.2 Mutually consistent state requirement

A design of effective distributed applications, that uses the pub/sub communication model, shall take into account the type of data flow that those applications exchange. For example, the **signal data flow** generated by real-time sensors has the following properties: values may change continuously, have short persistence, is time-critical (updates are useless when they are old), idempotent (repeated updates are acceptable), last-is-best (new information is more

important than a missed sample). Another example of a data flow is the **command data flow** that requires instructions to be delivered in a sequence, reliable and precisely-once [9]. The differentiation between the properties of data flows requires dealing differently with the mutual state consistency problem. One approach is to tolerate some data samples to be lost during subscriber reconfiguration since retransmissions are useless, like in the example of signal data flow. Another approach is to strongly enforce each data sample delivery and order of delivery. Therefore in this case the publisher retransmits all undelivered data samples after subscriber reconfiguration. The system designer, through assignment of the Reliability QoS to different entities (DataReader, DataWriter and Topic), determines the behaviour of the DDS pub/sub system in case of a participant's unavailability. The reliability QoS set to value reliable automatically enforces mutually state consistency (see example in figure 3).

4.3. Application State Invariants

The pub/sub communication model creates the illusion of a shared "global data space" populated by data samples that applications in distributed nodes can access via simple read write operation (see figure 5). The DDS introduces the notation of **data objects** that are data samples uniquely identified by: Topic (introduced in the previous section) ; Key – the field in the message that uniquely determines this message within the Topic

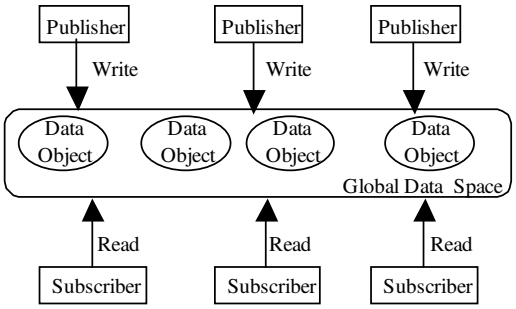

Fig. 5. The pub/sub system presented as "global data space".

For example, table 1 presents the Topic Track. The first field is the key field that uniquely determines this message.

Table 1 The example of track topic definition

```
Struct Track {
Longtrack  id                          //key
position pos; }
```

If two or more data objects share the same key value, the more recent instance overwrites the other. If no key field is defined, no fields indicate uniqueness and all the data objects are overwriting each other.

Applications periodically or state-change-driven publish/subscribe their internal state to the "global data space" as data objects. The state of the system consists of a collection of data objects codified as the most current values representing the state of each application in the system. The values of data object, representing the state of the system, are not stored on one node, but distributed across the system. Because the system state is distributed, an application that goes on-line can publish/subscribe to its most recent internal state from the DDS infrastructure (figure 6). 1) A participant saves/writes its internal state to DDS infrastructure. 2) The participant goes off-line for reconfiguration. 3) Reconfiguration finishes and the participant resumes its operation. 4) The participant loads/reads its internal state from DDS.

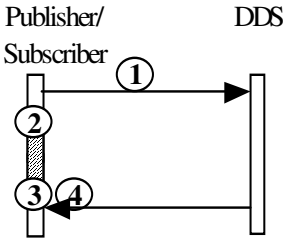

Fig. 6. Applications save/load their internal state to/from the DDS infrastructure.

5. Related Work

An alternative approach for reconfiguration in pub/sub middleware is presented in [10.] The dynamic reconfiguration is defined informally as the ability to rearrange the routes traversed by events in response to changes in the topology of the network of dispatchers and to do this without interrupting the normal system operation. This is contrary to our approach in which we assume changes in the components allocation in the fixed topology of the network. The Lira infrastructure for managing dynamic reconfiguration applies and extends the concepts of network management to component-based, distributed software systems [11]. Lira is designed to perform component-level reconfigurations through Reconfiguration Agents associated with individual components and the latter through a hierarchy of managers. Reconfiguration Agents are programmed on a component-by-component basis to respond to reconfiguration requests appropriate for that component. Managers embody the logic for monitoring the state of one or more

components, and for determining when and how to execute re-configuration activities [11]. The taken approach does not discuss anything about: the correctness preservation, state consistency during reconfiguration, the impact of reconfiguration infrastructure on the system performance, and components reliability.

6. Conclusions

In this paper we analysed the DDS specification with respect to its support for the correctness preservation during reconfiguration. The analysis considered three aspects of correctness, as presented in [1]: structural integrity, mutually consistent state and application state invariants. The DDS pub/sub architecture and QoS-controlled behaviour automatically ensure correctness preservation during reconfiguration. This differentiates the DDS from other middleware technologies that require that the correctness preservation is guaranteed on application level or by a reconfiguration manager/controller e.g., like in the [3]. This makes the DDS specification well suited for a dynamic environment, where dynamic reconfiguration and automatic discovery of participants are major concerns. Unlike Jini, CORBA and other client-server technologies, the DDS does not rely on centralized nodes e.g., name servers and is therefore highly resilient to partial failures in the network. However, we notice that the DDS architecture is based on broadcast messages and a 'global knowledge' assumption, which may be a reason of scalability problems, when applied to large-scale networks.

A partial evaluation of this research may be found in [12]. In that paper, we propose a new dynamic reconfiguration service for a pub/sub middleware that enables dynamic reallocation of components in order to achieve predictable and reliable system behaviour and fulfil deployment requirements. We have built a prototype that validates our research on existing DDS conformant pub/sub system implementations (Splice2v2 from Thales Naval Nederland).

Acknowledgment

This work was partly supported by European Research Programme: Marie Curie Host Fellowship under contract number- HPMI-CT-2002-00221.
This work was partly supported by the European Union under the E-Next Project FP6-506869.
We want to thank colleagues at Thales Nederland for the valuable comments and their advices regarding this research: Erik Hendriks and Wojciech Mlynarczyk.

References

[1] Moazami-Goudarzi K. - Consistency preserving dynamic reconfiguration of distributed systems. PhD thesis, Imperial College, London, March 1999.

[2] Kramer J, Magee J. - Dynamic configuration for distributed systems. IEEE Transactions on Software Engineering 11(4), pp. 424-436, April 1985.

[3] Wegdam M. – Dynamic Reconfiguration and Load Distribution in Component Middleware. PhD thesis, University of Twente, CTIT Ph.D-thesis series, No. 03-50, ISSN 1381-3617; No. 03-5, 26 June 2003.

[4] Pardo-Castellote G. – OMG Data Distribution Service: Real-Time Publish/Subscribe Becomes a Standard – RTC Magazine (www.rtcmagazine.com) – January 2005.

[5] Skowronek J., van't Hag H. - Evolutionary software development. NATO ESD conference 2001.

[6] Cugola G., Jacobsen H.A - Using Publish/Subscribe Middleware for Mobile Systems. ACM SIGMOBILE Mobile Computing and Communications Review archive,;Volume 6 , Issue 4 (October 2002), pp. 25 –33.

[7] Data Distribution Service for Real-Time Systems Specification - ptc/03-07-07 - May 2003.

[8] Almeida, J.P.A., Wegdam, M., Sinderen, M. van, Nieuwenhuis, L. - Transparent dynamic reconfiguration for CORBA. Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01), Rome, Italy, September 17-20, 2001, pp. 197-207.

[9] Pardo-Castellote G., Schneider S., Hamilton M. - NDDS: The Real-Time Publish-Subscribe Middleware - White paper - Real-Time Innovations, Inc. http://www.rti.com.

[10] G. Cugola, G.P.Picco, A.L. Murphy- Towards Dynamic Reconfiguration of Distributed Publish- Subscribe Middleware. In Proceedings of the 3rd International Workshop on Software Engineering and Middleware (SEM02), co-located with the 24th International Conference on Software Engineering (ICSE02), May 2002, Orlando (FL), USA, A. Coen- Porisini and A. van Der Hoek eds., Lecture Notes on Computer Science vol. 2596, pp. 187-202, 2003.

[11] M. Castaldi, A. Carzaniga, P. Inverardi, A.L. Wolf –A Lightweight Infrastructure for Reconfiguring Applications - B. Westfechtel, A. van der Hoek (Eds.): SCM 2001/2003, LNCS 2649, pp. 231–244, 2003. Springer-Verlag Berlin Heidelberg 2003

[12] Zieba B., Glandrup M., Sinderen M. van, Wegdam M. - Reconfiguration Service for Publish/Subscribe Middleware (bzieba.info/publications.html).