Abstract. Object Management Group (OMG)-Data Distribution Service (DDS) middleware, based on data centric publish-subscribe mechanism, is widely used in developing distributed software. However, many concerns in DDS are hard to localize in a single module. This impedes maintenance, reusability and other quality factors. Unfortunately, this has not been explicitly considered in the DDS domain. In this paper, we identify the key cross cutting concerns related to DDS and provide solutions to them.

Keywords: Aspect Oriented Software Development, AspectJ, DDS, OpenDDS, Monitoring, Distributed Software, DDS-AST.

1 Introduction

Designing and implementation of a distributed software system is complicated. It has many challenges such as sending and acquiring data with the required quality of services and coping with lots of software components that will communicate with each other.

One of the commonly used ways of developing distributed software is the use of OMG-DDS based middleware. In many DDS nodes, system has publishers and subscribers that communicate with each other. All of these nodes deal with the data to be published (send) and subscribed (received). So all of the nodes specify settings related with publishers, subscribers, data writers, data readers and other DDS concepts. Also quality of service (QoS) of these DDS concepts shall be set consistently; otherwise communication cannot be established between nodes.

Although use of a DDS based middleware provides many features to develop a distributed system in a modular and robust fashion, the software systems developed in this way has many cross-cutting concerns. Since distributed software may have hundreds of nodes that publishes and subscribes data, the cross-cutting concerns might be scattered through these software components.

Aspect Oriented Software Development (AOSD) provides solutions for the cross-cutting concerns. Cross-cutting concerns can be gathered as aspects. Use of aspects in a distributed application provides many advantages such as better modularization, more readable, reusable and manageable code.
In this paper, we developed an aspect support tool (DDS-AST) for DDS based distributed software systems using AOSD concepts. OpenDDS [4] is a popular open source software based on OMG-DDS. AspectJ [5] is selected as the AOSD tool. As a case study we developed radar and sonar nodes to demonstrate a DDS based command and control software.

In this work, firstly we used Object-Oriented Design Patterns in order to have more modular, maintainable and robust code. Then we used aspects to cope with the difficulties coming from cross-cutting concerns and also to add new functionalities that would be difficult to implement otherwise.

Remainder of this paper is organized as follows: Section 2 introduces background information about DDS. Section 3 provides details of our object-oriented design and design patterns used. Cross-cutting concerns in the software are identified in Section 4. Section 5 gives the aspect oriented implementation details of these cross-cutting concerns and new functionalities added using AOSD techniques. Related work is given in Section 6. Section 7 discusses further the effects of AOP on the case study and lessons learned. Finally, we conclude the paper in Section 8 with a summary and future work.

2 DDS

Data Distribution Service (DDS) is a middleware standard released by Object Management Group (OMG). Usage of OMG-DDS as the backbone of communication infrastructure enables software engineers to design and develop software components according to a widely accepted standard.

DDS is implemented as an “infrastructure” solution such that it can be added as the communication interface for any software application. DDS consists of Data Centric Publish and Subscribe (DCPS) and optional Data Local Reconstruction Layer. Layered architecture of DDS is given in Figure 1. DDS specification standardizes the software application programming interface by using DCPS as the communication mechanism [1].

Publish/Subscribe systems are nowadays considered a key technology for information exchange. Each participant in a publish/subscribe communication system can publish or subscribe information. Publishers produce information in form of events, which are consumed by subscribers. Subscribers can declare their interest on information issuing subscriptions. Subscriptions are used to filter the events produced by publishers [9].

Publish-subscribe applications are generally distributed applications. Software program elements within DDS communicate one another through sending (publishing) as well as getting (subscribing) data. The publisher need not know of the subscriber from the data it transmits as well as the other way around. The only thing to know may be the particular data type that is being communicated.
DDS introduces Quality of Service (QoS) parameters to specify several communication parameters like rate of subscription, rate of publication, how long the data is valid, etc. DDS does not exchange data within messages. Instead, the data structures to be exchanged are modeled with Interface Definition Language (IDL). DDS communication scheme is given in Figure 2 [10].

**Figure 1 – DDS Layered Architecture**

**Figure 2 – DDS Communication Scheme [10]**

DDS concepts that constitutes DCPS layer will be explained briefly:

**Domain**: The domain is the basic construct used to bind applications together for communication. All Data Readers and Data Writers may communicate within this domain. A distributed application may have one or more domain.

**Domain Participant**: The Domain Participant object enables a developer to specify default QoS parameters for all data writers, data readers, publishers and subscribers in the corresponding domain.

**Data Writers**: Data Writers are the primary access point for an application to publish data into a DDS data domain.

**Publishers**: Publishers are used to group together Data Writers. This allows you to configure a default set of QoS parameters and event handling routines that will apply to all the Data Readers in that Subscriber’s group.
Data Readers: Data Reader is the primary access point for an application to access data that has been received by a Subscriber.

Subscribers: Subscribers are used to group together Data Readers. This allows you to configure a default set of QoS parameters and event handling routines that will apply to all the Data Readers in that Subscriber’s group.

Topic: Topics provide the fundamental connection point between publishers and subscribers. The Topic of the publisher on a single node must match the Topic of the related subscriber. If the Topics do not match, communication will not be possible. A Topic consists of a Name and a Type. Topics must be uniquely defined within any one particular domain.

3 Object-Oriented Design

The object-oriented design of DDS-AST is described below. We prefer to omit some low level classes that just concern DDS applications for simplicity.

In order to demonstrate the concepts of a command and control software we developed a demonstration program with 2 sub-system integration unit (SIU) nodes (C++) and 2 user-interface nodes (Java). Similar SIU and UI nodes/services might be added. All of C++ and Java nodes publish and subscribe data (SIU-to-UI and UI-to-SIU). Figure-3 shows demonstration program’s communication scheme.

![Figure 3 –Communication Scheme of the Demonstration Program](image)
Since C++ aspects are not in the scope of this work details of C++ nodes are not given. User Interface (UI) components are developed in Java and each component may publish/subscribe data to/from DDS Domain. Figure-4 shows how a radar component updates its GUI according to the subscriptions and publishes.

3.1. Model View Controller (MVC) pattern:

MVC pattern is used to isolate domain information from the user interface. It separates UI login, input logic and business logic providing loose coupling between them [11]. We apply MVC to our UI nodes. In radar component, RadarTrackControlPanelControl is our Controller for MVC Design Pattern, SuiAosdRadarGuiData(Publisher), SuiAosdRadarModel, SuiAosdRadarStatusData,
SiuAosdSubscribeManager, SiuAosdRadarPublishManager are our Model and RadarTrackControlPanel, RadarMainFrame, RadarTrackListPanel, SiuAosdRadarUI are our View. Model component communicate with C++ publisher/subscribe codes. View component displays radar system component and its published/subscribed data. Control component acts like mid-layer between model and view components.

![MVC Pattern](image)

Figure 5 – MVC Pattern

3.2. Bridge Pattern:
Bridge pattern is used to decouple abstraction from its implementation so that two can vary independently [11].

We applied Bridge Pattern to our components view packages of UI nodes so that Radar and Sonar UI classes are implemented from abstract classes to minimize the static code and avoid any conflicts such as cyclic reference occurrence.

3.2.3. Singleton Pattern:
Singleton pattern is used to restrict the instantiation of a class to one object [11].

We applied Singleton Pattern to our MVC’s controller class (RadarTrackControlPanel) so that it’s guaranteed that RadarTrackControlPanel has only instance.

4 Identifying Aspects

Although applying object oriented design patterns improve software quality in terms of modularity, maintainability, reusability, etc some cross-cutting concerns is still scattered and tangled in the code.

In our case, these concerns arise not only from the inherent object-oriented paradigm problems but also from the nature of the distributed architecture. For example, all the communicating components/projects of DDS-AST shall have publishers and subscribers. So communication and timing concerns are scattered in the object-oriented program as well as Quality of Service concern.

Following cross cutting concerns are observed in the object oriented code:

4.1. DDS Monitoring Concern

Almost all DDS based command and control software systems are distributed applications and consist of many nodes. So all the publishers, subscribers etc exists in all nodes. So listing the parameters of DDS concepts such as publishers, subscribers,
etc requires visiting all nodes. Listings of these parameters are extremely important especially in development phase. If the parameters of publisher and subscriber of the same topic is not matched, no communication takes place although compiler gives no error. So finding these kinds of problems in huge code is a great problem.

4. 2. QoS Control Concern

Similar to DDS Properties, QoS parameters of publishers and subscribers shall match. In order to see the QoS of nodes, all publishers and subscribers shall be visited. If the QoS of publisher and subscriber of a topic are not consistent, there will be no communication. Again, this is a huge problem in development.

4. 3. Switching Language Concern

A command and control program shall support working in different languages. In order to make program have multi-language support, all the labels, widgets, graphical components, etc., in the user interface shall be acquired and changed in the user interface.

Especially, if multi-language support is required to be implemented in a legacy code this concern is scattered thorough all user interface components and might need a big change.

5  Aspect-Oriented Programming

In section 4, we stated some cross cutting concerns in the object oriented design. In this section we will describe the solutions of these cross cutting concerns implemented in AspectJ.

5. 1. DDS Monitoring Concern

In order to acquire the properties of DDS concepts in a command control software system all the DDS concepts such as publisher and subscribers shall be visited and the required values of them shall be listed if we only use object oriented design techniques. This is an obvious cross cutting concern in the design. However, using aspect oriented design techniques getting the required properties of all publishers, subscribers in a single aspect is possible.

In order to cope with this problem we developed several aspects to list the properties of publishers and subscribers separately. Using these aspects it is possible to list the properties of all publishers and subscribers is possible within each aspect. This functionality is especially important in the development phase of a command and control system, because it is possible to have publishers and subscribers with mismatching properties resulting in no communication. These kinds of problems are hard to detect since compiler does not give errors for these problems.

Four different aspects are developed to cope with this problem. Although it is possible to develop these in a single aspect instead of four different, it is not preferred because these aspects can be used for different purposes to list different settings so in this way it is more modular. Additional information about four aspects developed to cope with Listing of DDS properties are given below:
5. 1. 1 ListPublishers Aspect

Using ListPublishers aspect it is possible to list the properties of Publishers in the domain. Pointcut and advises of this aspect is given in Appendix – A, Figure 7.

5. 1. 2 ListSubscribers Aspect

Similarly, using ListSubscribers aspect is used to list the properties of Subscribers in the domain. Since algorithm of subscribers and publishers are different, their corresponding aspects differ. Pointcut and advises of ListSubscribers aspect is given in Appendix – A, Figure 8.

5. 1. 3 GetTypeOfPublishedData Aspect

Not only publisher settings but also type and value of published data are important to monitor. So another aspect is developed for this purpose. Since the data is captured as a java object, it is possible to identify its class, name and type of its attributes and the value of its attributes. So, this aspect is extremely important in development. Pointcut and advises of GetTypeOfPublishedData aspect is given in Appendix – A, Figure 9.

5. 1. 4 GetTypeOfSubscribedData Aspect

Similar to the GetTypeOfPublishedData, this aspect captures the data that is being subscribed as well as its class, attributes and values. In this way it is possible to compare not only the settings of the publishers/subscribers but also the values of the published/subscribed data. So, this aspect is useful to detect the errors originating from invalid data settings. Pointcut and advises of GetTypeOfSubscribedData aspect is given in Appendix – A, Figure 10.

5. 2. EnforceQoS Aspect

In communicating in DDS environment, QoS settings of publishers and subscribers are very important. If QoS settings do not match there will be no communication between publisher and subscriber. In a typical environment QoS settings are typically:

- CONTINUOUS_QoS: mainly used for continuous data,
- RELIABLE_QoS: mainly used to keep data for late subscribers,
- ALARM_QoS: used for alarm type of events
- PERSISTENT_QoS: used to keep data in DDS’s database.

Also other QoS settings are available in OMG-DDS standard.

In developing a DDS based software monitoring QoS settings of publishers and subscribers are important to ensure communication. So, EnforceQoS aspect is developed. As an example, this aspect checks the QoS settings of publishers and subscribers. If it is not RELIABLE_QoS it prompts a warning message and changes it to RELIABLE_QoS whatever the developer writes and ensures communication. Pointcut and advises of EnforceQoS aspect is given in Appendix – A, Figure 11.

5. 3. Switching Language Concern

In order to have multi-language support in software, generally different resource files containing strings in different languages are used. But if a legacy command and control program is required to support many languages, then all the strings of the user...
interface components such as labels, widgets, etc. shall be accessed and changed. This is a time consuming process to follow.

However, using AOSD, it is possible to make a legacy code (even a huge command and control software) to support many languages. SwitchLanguage aspect is developed for this purpose. In this aspect, static cross cutting is used to add required attributes and methods to the existing classes. So, according to the user selection, it is possible to use legacy code in desired language without changing the legacy code at all.

Pointcut and advises of SwitchLanguage aspect is given in Appendix – A, Figure 12.

6 Related Work

Coping with dynamic evolution of the software and implementing quality factors have been addressed by many researchers. The design patterns for software have been introduced to improve the quality and reusability by Gamma et al. AOSD was introduced by Kiczales, et al. in 1997. A part from above work, many DDS producers developed auxiliary tools that aim to help DDS developers in distributed applications. PrismTech that develops OpenSplice DDS also produces many such tools: OpenSplice Tester, OpenSplice Tuner, OpenSplice Configurator, OpenSplice Modeler, etc. Similarly Real Time Innovations (RTI), who played important role together with THALES, standardizes OMG DDS and develops RTI-DDS. RTI also produces tools to support DDS development. One of the most popular open-source DDS implementation is OpenDDS developed by Object Computing Inc. (OCI). OCI just released a prototype monitoring tool for DDS development.

Also MiSOFT Company who produces MiSOFT DDS developed auto code generator and SPY tool to support DDS development. Although some of the above tools are used for the purpose of monitoring DDS properties and they are very helpful for fast development none of them provides capability such as helping in multi language support or to overcome difficulties arising from cross cutting concerns in distributed systems.

7 Discussion

We developed Aspect Support Tool (DDS-AST) to support implementation of DDS based systems. In order to demonstrate the capabilities of DDS-AST, we developed an object oriented distributed software using DDS.

We applied several object-oriented design patterns in the base code in order to have a more reusable, maintainable and modular code. Applications of object-oriented patterns do not require new programming languages and/or tools so they can be adopted easily.

Also, we applied AOSD techniques to cope with problems arising from cross cutting concerns in DDS. Although our demonstration software includes C++ DDS nodes that publishes and subscribe data, we did not demonstrate implementation of design patterns in C++. Similarly, we did not implement aspects in C++ since these concepts are out of our scope in this paper keeping in mind that it is possible to capture data published by a C++ node through Java subscriber and similarly the data that will be subscribed by C++ node is published from a Java node.
Since AspectJ uses reflection mechanisms in implementing aspects, performance problems might be expected. But, we did not encounter any performance bottleneck in the development of DDS-AST. This point shall be considered in development of larger systems.

Many of the aspects developed in this paper might be used in larger distributed systems that use OpenDDS with a small or no modification. But this property shall be tested well.

8 Conclusion
In this paper, firstly we showed the application of some design patterns in a distributed application to improve the quality and understandability of the code.

Then we implement several aspects using AOSD techniques to cope with cross-cutting concerns which cannot be modularized by using traditional OO techniques. We concentrate on distributed applications, especially; DDS based applications since DDS has many inherent cross cutting concerns.

We also demonstrated how to add new functionalities to a legacy code using AOSD techniques readily.

One of the open issues is the evaluation of performance quality factor. Although performance is important in distributed application it is not in the scope of this paper.

As a result we developed an efficient tool to support implementation of DDS based systems DDS-AST. This tool may handle many cross cutting concerns. Also it helps developers to develop their systems fast enabling fast error detection in the development phase.

9 Acknowledgments
This paper was prepared in collaboration with Mr. Bedir Tekinerdoğan, Assistant Professor at the Department of Computer Engineering of Bilkent University.

10 References
4. OpenDDS: http://www.opendds.org
5. AspectJ: http://eclipse.org/aspectj
9. Angelo Corsaro, Quality of Service in Publish/Subscribe Middleware, Universita di Roma La Sapienza, April 2006.
APPENDIX - A

pointcut pblshr(Publisher p, String s, QosType q) : execution (Publisher.new(String[],String,QosType)) && args(String[],s,q) && target(p);

after(Publisher p, String s, QosType q): pblshr(p,s,q)
{
    String str = "------------------------\n";
    str += "--- ListPublishers ASPECT --\n";
    str += "new publisher - class name: " + s + "\n";
    str += "new publisher - type name: " + p.typeName + "\n";
    str += "new publisher - qos type: " + q.toString() + "\n";
    Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
}

Figure 7 – ListPublishers Aspect

pointcut pblshr(Subscriber sb, String s, QosType q) : execution (Subscriber.new(String[],String,*,QosType)) && args(String[],s,CallBackDelegate,q) && target(sb);

after(Subscriber sb, String s, QosType q): pblshr(sb,s,q)
{
    String str = "------------------------\n";
    str += "--- ListSubscribers ASPECT: \n";
    str += "new subscriber - class name: " + s + "\n";
    str += "new subscriber - type name: " + sb.typeName + "\n";
    str += "new subscriber - qos type: " + q.toString() + "\n";
    Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
}

Figure 8 – ListSubscribers Aspect
pointcut p(Object obj): execution(* *.publishData(Object)) && args(obj);

after(Object obj): p(obj)
{
    String str = "";
    str += "------------------------" + "n";
    str += "--- publishing: " + obj.getClass().getSimpleName() + " is publishing..." + "n";
    for (int i = 0; i < obj.getClass().getFields().length; i++)
    {
        Object bl = new Object();
        try
        {
            if ((obj.getClass().getFields()[i].getType().toString()) == "boolean")
                bl = (boolean) (obj.getClass().getFields()[i].getBoolean(obj));
            else if ((obj.getClass().getFields()[i].getType().toString()) == "int")
                bl = (int) (obj.getClass().getFields()[i].getInt(obj));
            else if ((obj.getClass().getFields()[i].getType().toString()) == "double")
                bl = (double) (obj.getClass().getFields()[i].getDouble(obj));
            else
                bl = obj.getClass().getFields()[i].get(obj);
        }
        catch (Exception e)
        {
            str += e.toString() + "n";
        }
        str += obj.getClass().getFields()[i].getType().toString() + "n";
        str += "- " + obj.getClass().getFields()[i].getName() + "=" + bl + "n";
    }
    Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
}

Figure 9 – GetTypeOfPublishedData Aspect
pointcut p(GenericDataReaderListener g, String s, CallBackDelegate c):
execution(GenericDataReaderListener.new(String, CallBackDelegate)) && args(s, c) && target(g);

after(GenericDataReaderListener g, String s, CallBackDelegate c): p(g, s, c) {
    String str = "";
    str += "--- subscribing:
    s += "--- Call Back Class Name: " + c.getCallBackClass().getSimpleName() + "n";
    try {
        s += "--- Call Back Method: " + c.getCallBackMethod().getName() + "n";
    } catch(Exception e) {
        s += "Exception: " + e.toString() + "n";
    }
   Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
}

Figure 10 – Get Type Of Subscribed Data Aspect

pointcut pblshr(QosType q) : execution (Publisher.new(*, *, QosType)) && args(String[], String, q) &&
within(com.aselsan.vatoz.libdds.*);

void around(QosType q): pblshr(q) {
    String str = "";
    str += "--- Enforce QoS ASPECT: " + q.toString() + "n";
    if(q != QosType.ReliableQos) {
        str += "do not use " + q.toString() + "type QoS\n"
        str += "instead use RELIABLE QoS\n"
        Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
        q = QosType.ReliableQos;
        proceed(q);
    } else {
        str += q.toString() + "type QoS is true \n"
        Monitoring.textArea.setText(Monitoring.textArea.getText() + str);
        proceed(q);
    }
}

Figure 11 – Enforce QoS Aspect
privileged public aspect SwitchLanguage
{
    declare parents: TrackControlPanelAbstract implements ActionListener;
    private JCheckBox TrackControlPanelAbstract.switchLanguageCheckBox;
    public void TrackControlPanelAbstract.actionPerformed(ActionEvent arg0) {
        if(switchLanguageCheckBox.isSelected()){
            isSendAllowedCheckBox.setText("Gonderme Acik");
            isOffsetModeCheckBox.setText("Ofset Modu");
            isControl1CheckBox.setText("Kontrol");
        } else{
            isSendAllowedCheckBox.setText("Send Allowed");
            isOffsetModeCheckBox.setText("Offset Mode");
            isControl1CheckBox.setText("Control");
        }
    }
}

after(TrackControlPanelAbstract r) : execution
(TrackControlPanelAbstract.new(..)) && this(r) {
    r.switchLanguageCheckBox = new JCheckBox("Switch Language");
    r.switchLanguageCheckBox.setBounds(20,100,120,25);
    r.add(r.switchLanguageCheckBox);
    r.switchLanguageCheckBox.addActionListener(r);
}

Figure 12 – SwitchLanguage Aspect