Applying Dynamic Separation of Aspects to Distributed Systems Security

Miguel García[1], David Llewellyn-Jones[2], Francisco Ortin[1], and Madjid Merabti[2]

[1] Universidad de Oviedo, Asturias, Spain
UO157632@uniovi.es
[2] Liverpool John Moores University, Liverpool, United Kingdom
D.Llewellyn-Jones@ljmu.ac.uk

Abstract. Distributed systems are commonly required to be flexible and scalable, as the number and arrangement of their (potentially mobile) devices may easily change. Security in distributed systems is a complex issue which can produce several problems such as eavesdropping, phishing or denial of service. To overcome these problems, there are various security measures that can be applied. This paper proposes the use of dynamic *Aspect Oriented Software Development (AOSD)* to implement security mechanisms in distributed systems. By applying dynamic separation of concerns using AOSD, it becomes possible to make corrections in the software at runtime. Therefore, it is possible to adapt the security measures of distributed systems at runtime, even when their sizes and arrangements change, without compromising global security. These changes can be applied when a distributed system is running, without requiring its execution to be stopped or interrupted. Using the DSAW dynamic AOSD platform, we have implemented solutions for two common security problems in distributed systems: access control and data flow, and encryption of transmissions. Qualitative and quantitative evaluations of both implementations are presented to estimate the pros and cons of using dynamic AOSD in the development of security measures of distributed systems.

1 Introduction

Distributed systems involve the interaction between disparate and independent entities working towards a common goal [3]. For instance, nowadays mobile device users expect to be able to connect to the Internet from anywhere to check email or use social networks. The creation of *ad hoc* device networks in order to exchange information is also becoming increasingly feasible, to be used in different scenarios such as at big events (concerts, shows, competitions) or even in emergency situations. These examples of networks must be flexible and scalable, as the number and arrangement of the devices can easily vary. Moreover, in general, the devices that make up these networks are heterogeneous as they may come from different kinds of services and clients. In any case, it is necessary to exchange information in a secure and controlled way.

Security is one of the main challenges concerning distributed systems is security. The complexity of this issue relies on the different vulnerabilities existing in distributed systems. Information sources, transmissions, the intermediate nodes or the final receivers are the first points that need to be protected in any distributed system. In systems with a single device or in private networks, most of the security threats can be easily controlled using static hardware or predeployed software, but this solution is not suitable in distributed systems.

Several problems can arise, intentionally or not, in distributed systems. Examples include eavesdropping, denial of service (DoS), spoofing or repudiation of transactions. The vast majority of measures used to prevent or solve these problems are performed using software. Encryption, digital certificates, access control lists or monitoring are some of the solutions used. Since there are different techniques used in order to solve security problems, in this paper we analyse the benefits on using the *Aspect Oriented Software Development (AOSD)* [16] paradigm. This paradigm allows developers to make good use of the *Separation of Concerns* (SoC) principle [15] when developing applications. If the AOSD platform employed offers dynamic aspect weaving [29], it is possible to dynamically adapt distributed systems in order to include new security concerns when a new threat appears at runtime. Security measures can be modified, inserted or removed with no need to stop the distributed system, and without changing the software or devices of which it is comprised. Corrections performed once the distributed system is running are motivated by two main causes: changes to the environment including new vulnerabilities (not taken into account at design time) or performance (the software has security measures, but they are disabled for efficiency reasons).

In this paper we use an AOSD approach to solve two common problems in distributed systems security: access control and data flow, and encryption. These solutions are based on AOSD, using the *Dynamic and Static Aspect Weaving (DSAW)* [38] platform for their implementation. DSAW is an AOSD platform that supports static and dynamic code weaving, which among other things allows the modification of application functionality at runtime. As shown below, applying this technology it is possible to control the flow, access and encryption of data dynamically. In this way, the distributed system is able to adapt to security measures when required, and can vary in size and arrangement without compromising security. The results of these use cases might also be applied to other distributed systems security problems.

The remainder of this paper is structured as follows. Section 2 describes and analyses the main problems and possible solutions in distributed systems security. Static and Dynamic AOSD and the DSAW platform are presented in sections 3 and 4 respectively. An aspect-oriented implementation of the proposed solutions to the specific problems of data access and data flow and encryption are described in Section 5. In Section 6 we evaluate the advantages and performance costs of our proposal. Section 8 presents the conclusions and future work.

2 Distributed Systems Security

The security of a distributed system is founded on its ability to protect itself. As far as security is concerned, two main questions must be taken into account. 1) What must be protected. In general, distributed systems need to protect the data, services and system resources being used. 2) What must be protected from. There are several ways to attack systems, as there are different ways to repel these attacks. In the case of distributed systems, one of the main threats is unauthorised users attempting to exploit vulnerabilities to access the system. Although this is the greatest threat, other problems such us eavesdropping or DoS are apparent as well.

The security measures of distributed systems primarily try to prevent, detect and correct threats that endanger the integrity of the system. The issues related to security in distributed systems can be divided into three general areas [6]: confidentiality, availability and integrity. Both security measures and threats fall into one of these categories. To consider a distributed system as secure, it has to be ready for possible problems in each of these categories.

2.1 Confidentiality

Confidentiality covers everything related to access of information by users (whether this access is authorised or not). Based on attempts to break confidentiality of distributed systems, the main attacks are traffic analysis, eavesdropping, spoofing, unauthorised access, MitM (man in the middle) and re-injection attacks [26]. Security measures to prevent these attacks try to ensure that only authorised users can access data and services. In the same way, it should be guaranteed that users with access do not constitute a new threat. It must also be ensured that a user with appropriate permissions is able to access the system without being mistakenly identified as dangerous. The main security measures used to ensure confidentiality are: authentication [40], authorisation [3], audit [41], data flow control/analysis [20, 42] and encryption [24].

2.2 Availability

Availability has to do with the need for both data and services to be available at any time. The main attacks against availability try to disrupt the proper working of services by flooding them with a lot of requests. These attacks are known as *Denial of Service (DoS)* attacks. Common measures used to protect against these attacks include *firewalls* and *load balancers* [5].

2.3 Integrity

In distributed systems, as in any system that exchanges data, it is necessary to ensure that during transmission the data reaches its destination without being manipulated. In this case, attacks may simply corrupt the message through

interference to make the information unreadable to the recipient. Other more complex attacks alter the content of the messages without the recipient being aware that the information has been changed (MitM). The most common techniques for these purposes are *message authentication codes (MAC)* (also referred to as *hash values*) and *digital signatures* [3].

2.4 Software approaches to implement security measures

As we have seen before, there are a variety of vulnerabilities that can compromise the security of a distributed system. But there are also diverse measures available for the detection and correction of threats. As we can see from the above discussion, in a dynamic system the most effective security measures are often based on adapting the system components. Taking into account how these adjustments are performed, the following classification can be established:

- Rearranging the components. By modifying the topology of the system it is possible to prevent or to correct problems such as restrictions on the data flow or availability of data and services.
- Introducing new components (or removing some of the existing components). To avoid situations such as the *buffer overrun* or *DoS* attacks, *firewalls* or other components capable of solving these problems can be introduced into the system.
- Enclosing one or more of the components using a wrapper. To protect components against attacks or modify their behaviour appropriately, they can be coated with new features to avoid dangerous situations such as adding encryption to communications or adding audit capabilities.
- Reconfiguring one or more of the components. In this case the components are reconfigured to suit the needs of the system, for example if there is a communication cut, the configuration of components is changed to divert traffic so that availability will not be affected.
- Directly adapting the code. The functionality of the components is directly modified to adapt to problems. For example, adding to them a digital signature or access control information.

These adjustments can be made using different software development approaches. Due to the nature of distributed systems (heterogeneous devices, point-to-point connections, mobile devices, etc.) it should ideally be performed in a manner that is as flexible as possible: through a reusable and maintainable approach capable of adapting distributed systems at runtime.

- Reusable and maintainable: In software development there are security issues that are orthogonal and independent to the main concerns of an application. The code for these security concerns therefore end up entangled with and scattered throughout the whole application. By separating the application functional code from these crosscutting concerns, the application source code stays disentangled, making it easier to debug, understand, maintain and modify. Moreover, this high level of abstraction allows the system to reuse and share individual concerns.

– Dynamic. The system should be able to adapt itself at runtime based on the dynamic environment. For example, it can use more or less security measures depending on the confidence of the environment or the capabilities of the component.

Under these assumptions, we propose the idea of applying *Aspect Oriented Software Development (AOSD)* in distributed systems security. As we will discuss in the following sections, with this approach it is possible to perform some of the adjustments mentioned above in a generic way without modifying the source code of the system and without stopping its execution, allowing the system to change its topology, capabilities and component numbers in a flexible and reusable way.

3 Aspect Oriented Software Development

Aspect Oriented Software Development (AOSD) [16] is a concrete approach to implement the principle of *Separation of Concerns (SoC)* [15]. AOSD facilitates a modularisation of different functionalities which cut across the entire system software.

An *aspect* is defined as any piece of software that cannot be encapsulated in a method or procedure, being scattered throughout the source code of an application. Common examples of aspects include transaction control, memory management, threading, persistence or logging [15].

In many cases, important functionalities in a system are not easily modularised. With the classic object-oriented paradigm, the code that deals with these features is often spread out in different parts of the application. AOSD manages the complexity of the software development by separating out functionality that is otherwise commonly tangled with the code of other features. The major benefits of this approach are the high level of abstraction, reuse of functionality, high legibility and improved software maintainability [15].

The final application is the result of combining several modules. On one side there are the modules that contain the basic functionality; on the other side, the aspects with concrete functionalities that usually cut across the system functionality.

In Figure 1, the difference between an object-oriented program and an aspect-oriented one is highlighted. Following the traditional paradigm, the application functionality and the code for specific aspects are mixed, making it difficult to maintain, debug and even comprehend. However, using aspect oriented development; each different functionality can be included in a separate module, facilitating its maintainability.

3.1 Weaving

Once both the basic functionality (components) and aspects are developed, it is necessary to form the whole program. This process is called *weaving*. Traditional

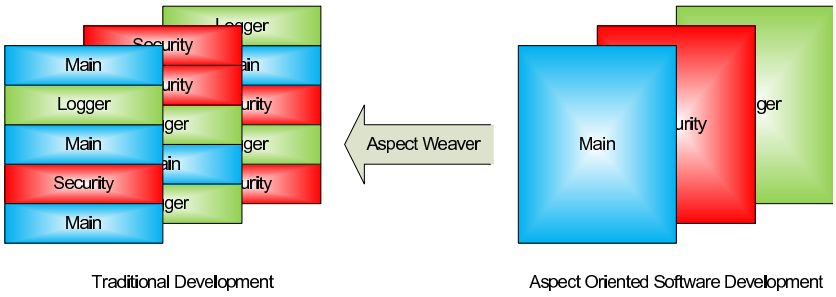

Fig. 1. Traditional development vs Aspect oriented development.

processes to generate a program consist of passing the source code to a compiler, in order to obtain an executable. In AOSD the code must also pass through the weaver, to obtain the program with full functionality. This process can be performed statically (at compile time or load time) or dynamically (at runtime).

To create the program there may be a relationship between the components and the aspects, establishing an interoperation between components and aspects. To allow this interaction, it is necessary to establish the points at which this interaction can occur. These points are called *join-points*: those elements of the programming language semantics which the aspects coordinate with [16]. It is also necessary to describe the mapping between join-points and aspects by means of *pointcuts*. A pointcut is a set of join-points plus, optionally, some of the values in the execution context of those join-points [18].

Static Weaving The majority of existing AOSD implementations are based on static weaving. Static weaving consists of combining the aspects' and components' functionality prior to application execution. This combination consists of inserting calls to *advice* in the components code. An advice is a method-like construct used to define additional behavior at join-points [18]. An advice is part of an aspect.

This type of weaving causes little performance penalty because all the code is combined and statically optimized before its execution. Since the application is woven at compile time, any functionality required to be adapted at runtime requires stopping the application, recompiling, reweaving and starting execution from scratch, often losing persistency in the process. For this reason, this kind of weaving is not used in applications that require a high level of runtime adaptation. Dynamic weaving is used instead.

Dynamic Weaving Although it is not always necessary, there are applications that need to be adapted at runtime in response to changes in the execution environment [29, 43, 30]. So-called *autonomic software* is an example; these systems should be able to repair, manage, optimise or recover themselves [17].

In the case of dynamic weaving the program is compiled in the traditional way and an executable is obtained. This program does not need to foresee which functionalities will be adapted. When the running program needs to be adapted, it can be dynamically woven with new aspects that adapt the behaviour of the application. The original program continues unmodified, and hence it can be reused without the changes added in memory.

The main advantage of this kind of weaving is that it support the dynamic adaptation of programs, maintaining the basic functionality separate from the aspects throughout the entirety life cycle of the software. Therefore, the resulting code is more adaptable and reusable, and both the aspects and the basic functionality can evolve independently [27]. The main disadvantage is that the dynamic adaptation commonly entails a runtime performance cost [7].

4 Dynamic and Static Aspect Weaving: DSAW

Existing dynamic AOSD tools such as AOP/ST, PROSE, DAOP, JAC, CLAW, LOOM.NET, JAsCo or DSAW [38] have been previously used to add new functionalities (not foreseen at design time) to existing systems. In order to evaluate the appropriateness of dynamic and static weaving, a platform with both approaches weaving will be used.

We have selected the DSAW platform to develop our proposal. DSAW (Dynamic and Static Aspect Weaving) [37, 25] is an aspect oriented software development platform that supports homogeneous static and dynamic weaving. Its main features are:

1. Full dynamic weaving: DSAW allows runtime (un)weaving of aspects, even at join-points that were not woven before the application was executed.
2. Platform independence: It is designed over the .NET virtual machine reference standard [10].
3. Language independence: DSAW performs the adaptation of software at the level of the Intermediate Language (IL) of the virtual machine (executable files and libraries).
4. Weave-time independence: The platform not only allows static and dynamic weaving, but can also take advantage of the same implementation of the components and aspects, independently of the scenario chosen.
5. Wide range of join-points: DSAW offers a wide and flexible set of join-points to facilitate the adaptation of applications for both dynamic and static scenarios.

Any existing .Net application, regardless of the high-level programming language used to develop it, can be adapted by DSAW. Figure 2 shows the steps that occur in the development of an application in DSAW. The Join-Point Injector (JPI) takes the application binary code (assembly) and, prior to its execution, performs instrumentation of the code in memory. If the weaving is static, a pointcut specification file must be passed as a parameter. In that case, the application

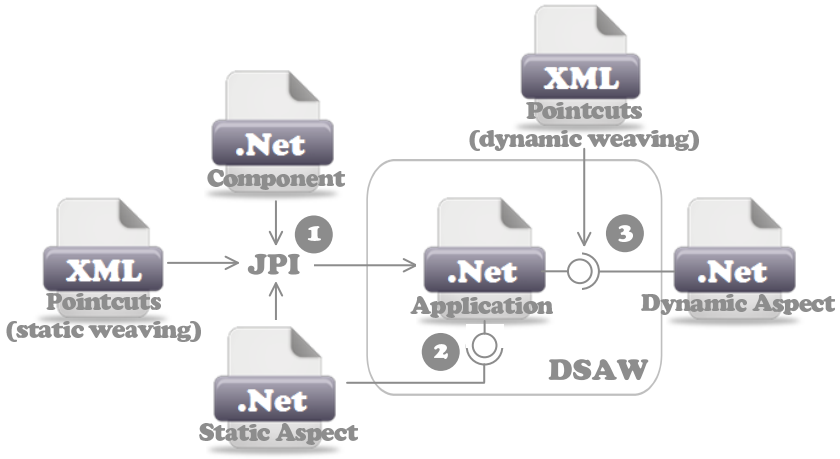

Fig. 2. Dynamic weaving steps.

is modified with calls to specific methods of the appropriate aspect specified in the static pointcut file.

Figure 3 shows an example pointcut specification file. In this case two pointcuts are specified: `before executing` the `SendMessage` method of **any** class, the `AddAuthorisationLevel` method of the `AccessControlAndDataFlowAspect` class of the `AccessControlAndDataFlow.dll` assembly must be executed, and `after executing` the `ReceiveMessage` method of **any** class, the `VerifyAuthorisationLevel` method of the `AccessControlAndDataFlowAspect` class of the `AccessControlAndDataFlow.dll` assembly must be executed too. If no such XML is passed to the JPI, all of the join-points will be available at runtime to weave dynamically.

In case dynamic weaving is required, the JPI instruments the application with more code. Since it cannot be known in which join-points the developer of a dynamic aspect could be interested in, the JPI instruments the application so that any join-point can be intercepted at runtime.

At runtime, a new aspect could be required to be woven. For this purpose, it is necessary to specify the pointcut with another XML document. Figure 4 is an example of a dynamic pointcut specification file. In this case it is specified: before calling to the `Connect` method of the `Chat` class, the `ChangePort` method of the `ChangeConnectionPortAspect` class of the `Aspect.dll` assembly must be executed. Dynamic weaving is performed by copying the aspect and the XML pointcut description file into the application execution directory. At this time, the code added to the application by the JPI makes the application dynamically invoke the aspect code when the execution reaches an intercepted join-point. In this way, the behaviour of the application is dynamically adapted, because when the execution of the application reaches the specified join-point, the code indi-

```
<aspect_definitions>
 <pointcut_definition id="c1">
  <time>before</time>
  <joinpoint_type>
   <methodexecution>
    <method_signature>
     <return_type><type_name>*</type_name></return_type>
     <qualified_method_name>
      <qualified_class>
       <namespace><type_name>*</type_name></namespace>
       <class><identifier_name>*</identifier_name></class>
      </qualified_class>
      <name><identifier_name>SendMessage</identifier_name></name>
     </qualified_method_name>
    </method_signature>
   </methodexecution>
  </joinpoint_type>
 </pointcut_definition>

 <pointcut_definition id="c2">
  <time>after</time>
  <joinpoint_type>
   <methodexecution>
    <method_signature>
     <return_type><type_name>*</type_name></return_type>
     <qualified_method_name>
      <qualified_class>
       <namespace><type_name>*</type_name></namespace>
       <class><identifier_name>*</identifier_name></class>
      </qualified_class>
      <name><identifier_name>ReceiveMessage</identifier_name></name>
     </qualified_method_name>
    </method_signature>
   </methodexecution>
  </joinpoint_type>
 </pointcut_definition>

 <advice_definition idTypeOfInjection="StaticInjection">
  <assembly>AccessControlAndDataFlow.dll</assembly>
  <type>AccessControlAndDataFlowAspect</type>
  <behaviour>AddAuthorisationLevel</behaviour>
  <pointcut_definitionRef idRef="c1"/>
 </advice_definition>

 <advice_definition idTypeOfInjection="StaticInjection">
  <assembly>AccessControlAndDataFlow.dll</assembly>
  <type>AccessControlAndDataFlowAspect</type>
  <behaviour>VerifyAuthorisationLevel</behaviour>
  <pointcut_definitionRef idRef="c2"/>
 </advice_definition>
</aspect_definitions>
```

Fig. 3. Static pointcut specification.

cated in the pointcut will be called. To unweave the aspect, it is only necessary to remove the XML file previously passed.

5 Use cases

The main objective of this work is to show how aspect oriented development may be an aid to developing flexible security mechanisms in distributed systems.

```
<aspect_definitions>
 <pointcut_definition id="c1">
  <time>around</time>
  <joinpoint_type>
   <get>
    <field_type><type_name>*</type_name></field_type>
    <qualified_field_name>
     <qualified_class>
      <namespace><type_name>*</type_name></namespace>
      <class><identifier_name>*</identifier_name></class>
     </qualified_class>
     <name><identifier_name>swSender</identifier_name></name>
    </qualified_field_name>
   </get>
  </joinpoint_type>
 </pointcut_definition>

 <pointcut_definition id="c2">
  <time>around</time>
  <joinpoint_type>
   <get>
    <field_type><type_name>*</type_name></field_type>
    <qualified_field_name>
     <qualified_class>
      <namespace><type_name>*</type_name></namespace>
      <class><identifier_name>*</identifier_name></class>
     </qualified_class>
     <name><identifier_name>srReceiver</identifier_name></name>
    </qualified_field_name>
   </get>
  </joinpoint_type>
 </pointcut_definition>

 <advice_definition idTypeOfInjection="DynamicInjection">
  <assembly>ChangeCommunication.dll</assembly>
  <type>ChangeCommunicationAspect</type>
  <behaviour>ChangeCommunication</behaviour>
  <pointcut_definitionRef idRef="c1"/>
  <pointcut_definitionRef idRef="c2"/>
 </advice_definition>
</aspect_definitions>
```

Fig. 4. Dynamic pointcut specification.

In order to do this, we have developed in DSAW security mechanisms for two specific scenarios: access control / data flow and communications encryption.

5.1 Access control/Data Flow

Distributed systems do not have a defined network topology and it is very usual that the information has to pass through intermediate nodes to travel through the network (an approach sometimes referred to as multi-hop routing [8]). This is not a problem when any node has rights to access the information that travels through the system. However, there may be nodes that have no privilege to see certain information, causing a security problem. In that case, it is necessary to restrict the access to the information, in order to prevent nodes without the appropriate permission from accessing unauthorised data. *Access control*

establishes who can access the information at any time, and data flow determines how information flows through the network.

Traditional mechanisms like CORBA [19] solve these problems by considering only "one-to-one" relationships, without taking into account that devices are part of a distributed system. This solution is effective in client-server networks or those with fixed topologies. However, in point-to-point networks or those with fuzzy topologies, the usage of these mechanisms may imply restrictions on the data flow. If a node is not authorised to send data to another one, but the latter is the only way to get to a third one, data transmission to the third node is not possible, even without a specific restriction on the system to do it. Other solutions analyse network topology in order to identify potential problems in the distribution of the information and to establish restrictions on access and data flow[23]. To do this, they use security policies with different levels of authorisation for each node. These kinds of security policy are widely used in military systems or for catastrophy management, where access control of information is essential[23]. Nodes can only send information to those nodes that have an authorisation level greater or equal to theirs, but they need to know the authorisation levels for all nodes they are connected to. Figure 5 shows an example scenario.

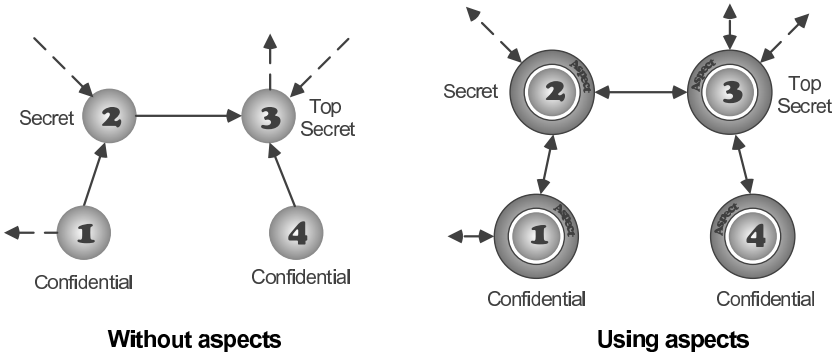

Fig. 5. Distributed system with different authorisation levels.

Nodes 1 and 4 may send information to any other node because the `confidential` level is the lowest. Node 2 can only send information to node 3, since `secret` is less restrictive than `top secret`. Finally, node 3 cannot send information to anyone because its level is the highest. The problem is that this approach lacks flexibility; whenever the network changes its number of nodes or topology, it is necessary to reanalyse the whole system and update the policies with the new restrictions. As in the previous case, in diffuse topologies, this solution may be very costly, and data flow restrictions can arise, preventing two nodes from exchanging information. For example, nodes 1 and 4 in Figure 5

have the same access level, but they cannot exchange information because node 3 cannot relay messages to nodes 4 or 1.

In distributed systems scenarios, these kinds of solutions are difficult to implement, mainly because they are very flexible networks, where the number of nodes and topology change frequently. Therefore, flexible security mechanisms are required. It should be allowed to control the access to the information at any time without interfering with the data flow, regardless of the number of nodes and shape of the system.

We have used the static weaver of DSAW to implement a distributed system with a security policy that guarantees the secure transmission of information over changing topologies, allowing both the tagging of data with the authorisation levels of nodes and the encryption of the information. Applications are built relying on the classical *send* and *receive* operations, and aspects intercept these messages to include the following functionalities:

1. Authentication to grant the user the appropriate authorisation level.
2. Encryption of information to avoid unauthorised access to it.
3. Data tagging to determinate how information flows across the network and to control access to it.

When a node sends the information, an aspect encrypts and labels it with the authorisation level of the source node. At destination, another aspect receives and decrypts the information and verifies whether the authorisation level of the node is high enough to access the data. If the level suffices, the aspect passes the data to the node; otherwise, it is discarded. With this mechanism, the information travels encrypted and labelled with the authorisation level in a totally transparent way throughout the system.

Although access is controlled, the problem of restricting data flow has not been avoided yet. To solve it, aspects can take into account information about the source node. When an intermediate node receives data to relay, it is sent with the authorisation level of the source node and not with its own. As is shown in the right side of Figure 5, with this approach all nodes can send data to any node, regardless of the authorisation level they have, because the aspects restrict the access to information and control the data flow. Node 1 can now send data to node 4 through nodes 2 and 3. If an intermediate node alters the data in any way, the information will automatically have the same level of authorisation and the aspects will restrict its dissemination.

To implement this solution, it is necessary to determine the join-points used in the implementation of this distributed system. Using the DSAW platform, we can use its static weaver to modify both the values of the parameters of a method just before its execution and the result just afterwards. This way, we can introduce data tagging and encryption when we send the information, and decrypt and verify the authorisation levels when we receive it. Figure 6 shows a simple example of C# code that receives and sends data over a network. The code to be adapted is that which is in charge of carrying out the transmissions. The ideal places to do this are the `SendMessage` and `ReceiveMessage` methods.

These methods are responsible for sending and receiving messages. To send and receive information over a connection, data has to be serialised and deserialised. The `connection` field is the connection used to exchange information with another node. The `swSender` and `srReceiver` fields are the streams responsible for sending and receiving information through the connection. `SwSender` and `SrReceiver` are the properties created to encapsulate the access to these fields.

One of the major benefits of using the AOSD paradigm is that all points in the program responsible for sending or receiving data can be tackled simultaneously, by applying the aspect to all instances of these methods throughout the program. However, from a security perspective, care must be taken to ensure that all network calls used are covered by the pointcut specification (for example, not just those based on the `TcpClient` class as shown here).

```
public class Chat {
 private TcpClient connection;
 private StreamWriter swSender;
 public StreamWriter SwSender{
  set {swSender = value;}
  get {
   if(swSender == null)
    swSender = new StreamWriter(connection.GetStream());
   return swSender;
  }
 }
 private StreamReader srReceiver;
 public StreamReader SrReceiver {
  set {srReceiver = value;}
  get {
   if(srReceiver == null)
    srReceiver = new StreamReader(connection.GetStream());
   return srReceiver;
  }
 }
 private Data ReceiveMessage() {
  String received = SrReceiver.ReadLine();
  Data message = Deserialize(received);
  return message;
 }
 private void SendMessage(Data message) {
  String toSend = Serialize(message);
  SwSender.WriteLine(toSend);
  SwSender.Flush();
 }
}
```

Fig. 6. Sample of C# code for sending and receiving messages.

In Figure 3 we show the pointcuts used to do this implementation. The `AddAuthorisationLevel` method is invoked in order to encrypt and introduce data tagging. We also call the `VerifyAuthorisationLevel` method, which is responsible for decrypting and verifying that the authorisation level of the node allows it to access the data. Figure 7 shows the code of the aspect used to develop our solution. In this aspect, using the `AddAuthorisationLevel` method, the

data is labelled with the authorisation level of the code just before encrypting and sending it. Attention must be paid to the fact that whether or not an intermediate node includes new (or altered) information, the authorisation level held will be the one of the source node, not the one of this intermediate node.

```
public class AccessControlAndDataFlowAspect {
 private static DataWrapper receivedData;
 private int AuthorisationLevel() {
   //... Method that obtains the authorisation level of the node.
 }
 private DataWrapper Encrypt(DataWrapper data) {
   //... Method that encrypts a DataWrapper.
 }
 private DataWrapper Decrypt(DataWrapper data) {
   //... Method that decrypts a DataWrapper.
 }
 public static object AddAuthorisationLevel(object ResultVal, Param[] par,
     ...) {
  int nodeAuthorisationLevel = AuthorisationLevel();
  Data dataToSend = (Data) par[0];
  if (receivedData != null && receivedData.Data.Equals(dataToSend))
   nodeAuthorisationLevel = receivedData.AuthorisationLevel;
  receivedData = null;
  DataWrapper dataWrapper = new DataWrapper(dataToSend,
      nodeAuthorisationLevel);
  par[0] = Encrypt(dataWrapper);
  return 1;
 }
 public static object VerifyAuthorisationLevel(object ResultVal, Param[] par,
      ...) {
  int nodeAuthorisationLevel = AuthorisationLevel();
  DataWrapper dataWrapper = (DataWrapper)ResultVal;
  receivedData = Decrypt(dataWrapper);
  if (nodeAuthoritationLevel >= receivedData.AuthoritationLevel)
   ResultVal = receivedData.Data
  else
   ResultVal = new Data();
  return 1;
 }

 class DataWrapper : Data {
  private AuthorisationLevel authorisationLevel;
  private Data data;
  public AuthorisationLevel AuthorisationLevel {
   get { return authorisationLevel; }
   set { authorisationLevel = value; }
  }
  public Data Data {
   get { return data; }
   set { data = value; }
  }
  public DataWrapper(Data data, AuthorisationLevel authorisationLevel) {
   Data = data;
   AuthorisationLevel = authorisationLevel;
  }
 }
}
```

Fig. 7. C# code of the aspect to tag and verify the data.

The first step is to obtain the authorisation level of the source node. Using the `AuthorisationLevel` function, it is possible to obtain the authorisation level of a node. Then, if the source node has previously received some data and it is equal to the data to send (*i.e.* the node is relaying a message), the authorisation level to use is the one of the data received. To label the data with the authorisation level, the value of the parameter `par[0]` (that is the `message` parameter of the `SendMessage` method provided by DSAW) is changed for a wrapper composed of the data and the obtained authorisation level. The `DataWrapper` class inherits from the `Data` class, making the `SendMessage` and `ReceiveMessage` methods works correctly, even if its argument is changed. Finally, the `dataWrapper` is encrypted.

When the aspect receives the data, the `VerifyAuthorisationLevel` method decrypts the data and verifies that the authorisation level of the node is sufficient for accessing the received data –otherwise it discards the data. As in the previous case, it is necessary to obtain the authorisation level of the node. It is also necessary to extract the authorisation level of the data using the `ResultVal` (provided as a parameter by the platform), which is the result of the `ReceiveMessage` method execution. Authorisation levels are checked; if the level is sufficient, the method extracts the original data from the wrapper and returns it; otherwise, the data is discarded.

This way, the behavior of the nodes is adapted when the send and receive messages are sent. Data is labeled before being sent and before being accessed the permissions for reading it are checked. Using this approach, the distributed system can vary in number of nodes and topology without compromising safety, since the access control and the data flow is controlled at all times, and there is no need to analyse the network or update the policies. The entire process works in a transparent way with respect to both the nodes and the system. The functionality of the application is modularised apart from the communication and security concerns, implemented as separate aspects.

5.2 Encryption

Communications encryption is a security measure the main objective of which is to prevent unauthorised users from accessing the information exchanged between the nodes of the system. This is typically done using algorithms that transform data into unreadable messages. One problem is that the encryption process can be resource-heavy, because it is necessary to encrypt data before sending it and decrypting it by the recipient. In distributed systems composed of mobile devices, this process is rather costly and has a great impact on battery consumption. To prevent overloading of the devices, it is usual to use specific nodes of the system, with greater computing power, to encrypt and decrypt the information. A sample scenario could be various mobile devices exchanging data through a public insecure Wi-Fi network. If two nodes need to exchange confidential information, they can use a more secure UMTS [13] connection. This channel, although slower, has a higher level of security. Once the transmission of this data has finished, the mobile devices could reuse the original Wi-Fi connection.

At the top of the Figure 8 shows the difference between the two scenarios. In the first scenario, there is a direct communication between nodes 3 and 4. In the second one, since there is an insecure zone between these two nodes, the communication is passed through two E/D (encrypt/decrypt) nodes. The advantage of the second scenario is that devices are freed of this work, but a delay between sending and receiving messages is introduced. It is necessary to send information to these special E/D nodes, encrypt the information, exchange it between them, decode the data and deliver it to the destination node. To overcome this delay, it is better to use encryption only when it is necessary. For example, when communications are likely to be heard (if it is suspected that there are intruders listening to the network). But if the channel is in a controlled and trusted environment, the encryption would not be required as it would introduce an unnecessary overhead and delay in the message transmissions.

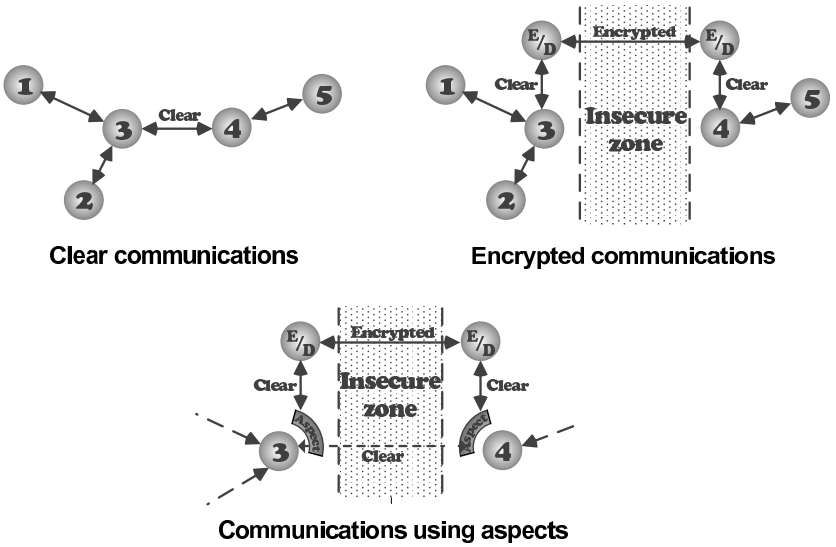

Fig. 8. Communications with and without encryption.

We have developed this scenario where, under these circumstances, the system is able to dynamically change communications in a distributed system. Thus, when information must be encrypted, transmissions are forwarded to the nodes that encrypt and decrypt it (*Clear communications* in Figure 8), and when it is not necessary, the original communication routes are used (*Encrypted communications* in Figure 8). Using dynamic weaving it is possible to dynamically change the behaviour of nodes 3 and 4. To change the communication channels, the aspect to be injected establishes a new connection between the source node and a E/D node. Afterwards, it is necessary to divert the traffic through this new

connection. Finally, when the aspect is unwoven, the original communication is re-established.

The connection with the encryption and decryption E/D nodes takes place when the aspects are woven. At runtime, the nodes 3 and 4 are connected to an E/D node that are in turn connected to each other. This way, as shown at the bottom in Figure 8 a new secure connection between nodes 3 and 4 is established.

Once connected, using the same aspects, the functionality of the nodes 3 and 4 is altered in order to forward the information. All traffic is diverted through the secure connection, keeping the nodes unaware of that fact. Then, when nodes 3 and 4 send information, it will go to its corresponding E/D node, which encrypts and retransmits it to another E/D node. Once received by the second E/D node, the data is decoded and transmitted to the destination node using clear communication.

The disconnection takes place when the aspects are unwoven. Then, the original communication is automatically re-established. Since the aspect does not interfere when the nodes access their connection, the information will be sent as it originally was.

As discussed above, whenever the connection is used for both sending and receiving data in an insecure environment, we want an aspect to intervene and replace the connection by a more secure one. To achieve this, the code in Figure 6 intercepts two around join-points at the `SwSender` and `SrReceiver` read properties. Figure 4 shows the pointcut document describing these join-points. The `ChangeCommunication` method of the `ChangeCommunicationAspect` class is woven at both join-points. Thus, when the properties are read, aspect code is run instead.

Figure 9 shows the code of the aspect that modifies the connection. When the aspect is woven, a new connection with a E/D node is created (using a predefined *ip* and *port*). The E/D nodes must be out of the insecure zone, as shown in Figure 8 (Communications using aspects). These nodes encrypt, decrypt and exchange data between each other through the insecure zone. When the `ChangeCommunication` method is invoked, the aspects verify which of the properties is being called (this information is passed as the `member` parameter to the aspect by DSAW) and return the corresponding secure stream in each case.

Therefore, if the pointcut specification document is passed to the platform, the behaviour of the nodes is dynamically changed, as when the `ReceiveMessage` and `SendMessage` methods are invoked (and the properties are used to access the streams) the code of the aspect transparently provides a secure connection.

With this approach, when the encryption of the communications is required at runtime, it is sufficient to pass DSAW the file with the pointcut definition and the aspect. Finally, when it is necessary to stop the encryption, both files should be removed from the platform to recover the original behaviour.

```
public class ChangeCommunicationAspect {
 private static TcpClient connection = Connect();
 private static StreamWriter swSender;
 private static StreamReader srReceiver;
 private static TcpClient Connect() {
  TcpClient connection = new TcpClient();
  connection.Connect(IP_ED, PORT_ED);
  swSender = new StreamWriter(connection.GetStream());
  srReceiver = new StreamReader(connection.GetStream());
  return connection;
 }
 public static object ChangeCommunication(String member ...) {
  if (member.Equals("swSender")) {
   return swSender;
  }
  if (member.Equals("srReceiver")) {
   return srReceiver;
  }
 }
}
```

Fig. 9. C# code of the aspect that modifies the connection.

6 Evaluation

In this section, we evaluate the appropriateness of DSAW to develop security measures, comparing it with the traditional object-oriented programming (OOP) paradigm. Our experimental methodology is outlined first. Afterwards, the benefits of using DSAW are stated and quantitative evaluations performed. Finally, we present a discussion regarding the evaluation obtained.

6.1 Methodology

We highlight the benefits of dynamic separation of aspects in distributed systems security using DSAW. The benefits are those mentioned throughout the paper (detailed in Section 6.2). The quantitative measurements we consider are runtime performance and memory consumption.

The quantitative assessment contrasts runtime performance and memory consumption between the DSAW platform and traditional OOP. We have implemented a distributed system with the security measures of the two use cases (Section 5) using DSAW and OOP. These implementations have been compared using the .Net Framework 2.0 build 50727 for 32 bits, over a Windows 7 x64 operating system. All tests have been carried out on a lightly loaded 2.13GHz Intel Core 2 Duo system with 4GB of RAM. We developed the software in the C# programming language.

In order to compare DSAW with OOP, we have simulated the networks of the two use cases with each implemented distributed system. In the first case (Section 5.1), the distributed system is composed by three single nodes. In the second one (Section 5.2), the system has two single nodes and two E/D nodes. To evaluate runtime performance, we have instrumented the code with hooks to record the value of the processor timestamp counter. We measured the difference

between the beginning and the end of exchanging a set of messages to obtain the total execution time. To suppress the cost of native code generation by the JIT compiler, we first make a single use of the distributed system exchanging a small set of messages between the nodes. This first invocation is not taken into account in our evaluation, in order to ignore the time required to dynamically generate native code by the virtual machine JIT compiler.

All the tests have been executed utilising the Windows 7 performance monitor. We measured the maximum size of working set memory used by the process (the `PeakWorkingSet` property). The working set of a process is the number of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault. The working set includes both shared and private data. The shared data comprises the pages that contain all the instructions that the process executes, including instructions from the process modules and the system libraries.

6.2 Benefits of using DSAW

Based on our experiences from the case studies, we found the following to be the main benefits obtained by using the DSAW platform to apply dynamic separation of aspects to distributed systems security.

1. **Higher abstraction level**. Developers can focus on concrete concerns in isolation. The security measures may be developed by experts in the domain, separately from the other components of the distributed system.
2. **Reuse, maintenance and legibility**. Separation of concerns attains decoupling of modules, allowing the distributed system to reuse and share security concerns. The code is not tangled and scattered throughout the whole application. The code of each security issue is not coupled to the rest of the code. Since the code is less complex, it is easier to understand and maintain.
3. **Increase of application development productivity**. In addition to previously mentioned advantages, the decoupling of security measures and functional code may imply an increase in the efficiency of the software development process.
4. **Dynamic adaptation**. It is possible to adapt distributed systems depending on the runtime environment. For example, if a better performance is needed and the communication channel is secure enough, it is possible to disable some security measure such as encryption. If a new threat appears at runtime, it is possible to dynamically adapt distributed systems in order to include new security measure to solve it. This way, security measures always take into account the necessities of the system.
5. **Platform and language independence**. DSAW is language and platform neutral. Therefore it is possible to develop components and aspects in any .Net language and they can be executed in any .Net implementation. This is an interesting feature for heterogeneous distributed systems, composed of elements using different operating systems. Security measures may be developed in any .Net languages.

6. **Weaving-time concern**. With DSAW the implementation of the components and aspects is independent of the weaving-time concern. Neither applications nor aspects should be modified if the weaving process changes from static to dynamic, or vice versa. This way, the same security measure may be established as dynamic (to be disabled or enabled at runtime) or static (obtaining a better runtime performance).
7. **Join-points**. Unlike most tools with both kinds of weaving, DSAW offers the same wide set of join-points for the two scenarios.
8. **Binary code injection**. Since DSAW performs the adaptation at the virtual machine level, the application and aspect source code is not required. Thus, it is possible to adapt legacy distributed systems, injecting security measures developed by others.

6.3 Quantitative evaluation

To evaluate the cost of using static and dynamic weaving in DSAW, we have measured runtime performance and memory consumption. For both use cases (Section 5), we have developed a distributed system using the traditional object-oriented programming paradigm. In both cases, we have extended the developed system adding the following security measures, using OOP: access control and data flow (in the first use case), and encryption (in the second one).

In the control access and data flow scenario, we have used static weaving to inject the aspect in the original developed system; and in the encryption case, dynamic weaving was used. Thus, we have compared the performance and memory consumption for OOP and AOSD approaches. For each scenario, we have sent and received different numbers of messages between the distributed system nodes. The first call has not been included to rule out the cost of JIT compilation. Tables 1 to 4 show the results of the four scenarios: Access Control/Data flow (OOP vs. DSAW Static) and Encryption (OOP vs. DSAW Dynamic); execution time is expressed in milliseconds and memory consumption in Kbytes of a growing number of messages (columns one to five).

Access Control/Data Flow		Number of Messages				
		1,000	**3,000**	**6,000**	**10,000**	**15,000**
Execution Time	**OOP**	3,565.9	12,028.2	24,764.5	41,405.3	62,324.2
	DSAW Static	3,825.7	12,476.4	25,525.5	42,808.6	64,285.3

Table 1. Execution time of the Access Control / Data Flow scenario.

Discussion In both scenarios, performance costs did not depend on the number of messages sent and received. The standard derivation was 6.72% for the static scenario and 2.82% for the dynamic one. Memory consumption is the same regardless of the number of messages exchanged in both scenarios.

Access Control/Data Flow		Number of Messages				
		1,000	**3,000**	**6,000**	**10,000**	**15,000**
Memory Consumption	**OOP**	30,221	29,556	29,368	30,049	30.085
	DSAW Static	30,495	30,988	30,156	30,212	30,192

Table 2. Memory consumption of the Access Control / Data Flow scenario.

Encryption		Number of Messages				
		50,000	**100,000**	**150,000**	**300,000**	**500,000**
Execution Time	**OOP**	5,579.4	11,913.9	18,380.3	36,433.4	60,382.4
	DSAW Dynamic	9,067.1	18,785.5	29,101.6	58,587.5	97,688.9

Table 3. Execution time of the Encryption scenario.

Encryption		Number of Messages				
		50,000	**100,000**	**150,000**	**300,000**	**500,000**
Memory Consumption	**OOP**	29,524	28,904	28,764	28,460	29,089
	DSAW Dynamic	45,020	44,936	45,164	45,148	45.080

Table 4. Memory consumption of the Encryption scenario.

Figure 10 shows a comparison of runtime performance and memory consumption between OOP and DSAW approaches for both scenarios. All values are shown relative to OOP implementation (values were divided by the values of OOP). The first major discussion could be related to the cost of weaving. In the static weaving case, aspects involved a 4.12% and 1.41% cost of runtime performance and memory consumption respectively. For the encryption scenario, where dynamic weaving was used. The runtime performance penalty was 59.2% and the increase of memory consumption was 55.96%.

In order to perform a deeper analysis of the runtime performance penalty, we used a .Net profiler obtaining the results shown in the Tables 5 and 6. The first column of each table is, for each operation measured by the profiler, the percentage of the total execution time of the OOP implementation. The second one shows the execution time increase of the DSAW approach relative to the OOP one. We measure the following parts of `SendMessage` and `ReceiveMessage` methods, because it is where the distributed system is adapted:

1. JPI: this value has been obtained by adding the execution time of all the code introduced by the JPI during the instrumentation of the application.
2. Method call: is the execution time required to call the send and receive method.
3. Method execution: is the time required to execute the code into the send and receive methods. This value is obtained by subtracting the whole execution time of the method call and the previous execution time value.

In the access control and data flow scenario, Table 5, the JPI increase is only 0.56% for both sending and receiving. The send and receive method call

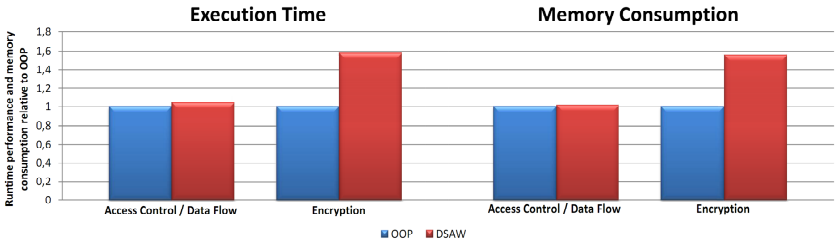

Fig. 10. Execution time and memory consumption relative to OOP.

Access Control / Data Flow

		% Execution Time OOP	DSAW penalty
Send	**JPI**	0.00%	0.56%
	Method calls	0.35%	1.73%
	Method execution	0.12%	0.02%
	Serialize	48.92%	1.27%
Receive	**JPI**	0.00%	0.56%
	Method calls	0.43%	0.67%
	Method execution	0.27%	0.02%
	Deserialize	42.34%	0.33%

Table 5. Detailed Access Control / Data Flow runtime performance penalty.

penalty is 1.73% and 0.67% respectively (both, added up, imply a 58.3% of the total penalty). The performance penalty in the method execution is only 0.02%. Finally, we have included the `Serialize` and `Deserialize` methods in the analysis because there are significant increases, 1.27% and 0.33% respectively. These penalties represent around 40% of the total cost. The OOP version uses `Data` objects to exchange the information, but in the AOSD implementations it is necessary to serialise and deserialise `DataWrapper` objects, which contain the original `Data` object and its associated authorisation level. This difference causes the runtime performance cost.

Table 6 details the profiler output for encryption scenario. In this case, the JPI increase is 17.03% sending and 8.66% receiving messages. For the method call the penalty is 20.7% and 11.14% respectively. In the OOP implementation, it implies almost no cost because it is only necessary to access the `SwSender` and `SrReceiver` properties. Finally, the increase of the method execution of the aspects regarding the original application execution time is 0.72% (around 1% of the total cost).

After presenting the data in Figure 10 and Tables 1 to Tables 6, three issue are highlighted. The first one is related to the cost of static weaving. In the access control and encryption scenarios, the runtime performance and memory consumption penalties are remarkably low. The second discussion is the difference between the two kinds of weavers. In both cases, JPI and method call showed the highest cost. In the static scenario, these times represent a penalty

Encryption			
		% Execution Time OOP	DSAW penalty
Send	**JPI**	0.00%	17.03%
	Method calls	0.01%	20.70%
	Method execution	2.88%	0.38%
Receive	**JPI**	0.00%	8.66%
	Method calls	0.01%	11.14%
	Method execution	1.28%	0.34%

Table 6. Detailed Encryption runtime performance penalty.

of 3.52%, that is 85.5% of the total cost. In the dynamic case, the increase is 57,53%, 97.19% of the total. The dynamic penalty is significantly higher than the static one (3.52% versus 57.53%) because in the dynamic scenario it is necessary to use reflective techniques in order to invoke both platform and aspects [25]. The final issue is related to memory consumption. In the static weaving scenario it is around 1%, but it is increased to 55.96% in the dynamic case. The DSAW dynamic weaver injects a join-point table to (de)activate the join-points in the application, causing this memory increase [25].

7 Related Work

There are some existing studies and pieces of work that implement and introduce security mechanisms using AOSD. Viega [36] propose an extension of the C programming language to support aspects. This extension allows the definition of security policies apart from the application code. Subsequently, the policies are transformed into code, which are woven together with the main application source code. Using this approach, automatic security checks can be added, including protection against buffer overflow, logging or encrypted connections. When new threats are detected, the security policies are updated with different suitable measures. Although this work is presented as an extension of the C programming language, the authors state that it might be applied to other languages.

AspectJ has also been applied to security issues. Huang [14] present a generic and reusable library to introduce security mechanisms in Java developments. This library provides reusable and generic aspects in AspectJ, as practical software components and a prototype implementation of a common security-relative API for AOSD. These aspects encapsulate complicated security-relevant code and the process of invoking Java security packages (such as JCE or JAAS). Developers can reuse the security aspects through aspect recomposing or overriding, building applications easily to modify and maintain.

Kim and Lee [34] also use AspectJ, discussing how authorisation capabilities could be added to existing well-structured, object-oriented systems. They consider the use of design-time weaving and stress the need for the developer to be fully aware of the AOP requirements throughout the development process.

Similarly Shah and Hill [31] found in their application of AOSD for security that customisation of an application tended to be required, necessitating a design-time approach. However, in both cases the intention of the work is to present a high-level view, and more detailed explanations of the approaches are not provided.

In the network area, there are works that propose the utilisation of aspects to solve security issues. Lopes and Kiczales [21] present a language framework called D. This framework untangles the implementation of synchronisation schemes and remote data transfers from the implementation of the components using aspects. Truyen [35] propose a reflective architecture. This architecture allows making dynamic reconfiguration of non-functional requirements such as real-time, reliability, availability and security. For each non-functional requirement a specific AOP language is defined. The application of AOP security to Web applications has also been considered in some detail [39, 9] for the purposes of authentication and authorisation. In the case of Chen and Huang's work [9] access control is added at the data-level in order to provide more fine-grained control; however the work relies on applications adopting Apache Struts, since pointcuts are chosen relative to the *user action classes* of this framework.

A number of authors have taken a slightly different approach, proposing new pointcuts that have been especially designed for the purposes of weaving security aspects into an application. Mourad [22] propose the addition of *GAFlow* and *GDFlow* pointcuts, which allow the closest ancestor joinpoint on all ancestor paths, and the closest child joinpoint reached by all paths starting from the pointcut, to be defined respectively. These pointcuts, specified based on the control-flow graph of an application, allow security capabilities to be added in a way that ensures they are initialised and deinitialised appropriately. New primitives to allow data to be passed between pointcuts are also needed to support this. Pointcuts for the tracking of data flow through an application have also been proposed [2, 4] as useful additions from a security perspective. For example, by creating *get* and *set* pointcuts that allow the control and monitoring of variable access.

As far as the authors are aware, PROSE and JBoss AOP are the only dynamic AOSD platforms that have been used for adapting security issues at runtime. PROSE was proposed to make dynamic adaptation of software architectures where applications can be adapted depending on the runtime environment [28]. In this way, it is possible to integrate specific concerns (*e.g.* access control or authorisation) in software for existing distributed systems such as *spontaneous* networks [11]. Each node of the system must initialise a remote weaver service. The distributed system has a central repository of aspects, which acts as a client and discovers all the active weaver services in the network. Depending on the properties of each node and the runtime environment, a customised set of aspect instances is remotely sent to the weaver services by the repository. PROSE uses the Java Virtual Machine Debugger Interface (JVMDI) [33] to pause the execution at the join-points and then calls the behaviour defined in the received aspects resuming its execution. The immediate effect is that the crosscutting

concerns modify the behaviour of the node. This work is only preliminary, not detailed yet. Song [32] also use PROSE to apply security dynamically at runtime. The application to Web Services makes it particularly relevant, with the authors discussing a *Web Service eXtension Environment* (WSXE) that's used to apply access control and logging as examples. However the weaving is only applied to individual services rather than multiple services interacting across the network. Finally Hermosillo [12] use JBoss AOP also for the dynamic weaving of security aspects, in this case to protect against SQL-injection and XSS attacks on Web applications. The use of dynamic weaving is intended to allow the aspect to evolve independently of the application, but for the purposes of Web applications it's not necessary to consider aspects applied to multiple network components simultaneously.

This related work suggests considerable scope for the use of AOSD for the purposes of security hardening. However, more research is needed to considers the dynamic application of security aspects in a coordinated manner across multiple interacting network components. We feel the language independent nature of DSAW with its strong support for dynamic weaving makes it particularly applicable in this case. Our contribution therefore lies in providing an effective implementation that demonstrates the practicality of applying aspects across multiple networked components, enhancing security beyond that which is possible by concentrating on just individual components in isolation.

8 Conclusions

This paper analyses how dynamic and static AOSD can be used to separate common security concerns of distributed systems. Distributed systems have vulnerabilities that jeopardise their safety. We have developed a classification of security issues that can arise in distributed systems, the majority of them can be solved by adapting the system components. In this paper, we propose the use of *Aspect Oriented Software Development* to implement these adaptations. Flexible security mechanisms in distributed systems can be implemented with AOSD, obtaining higher maintainability, reusability and dynamic adaptability –aspects can be (un)woven without stopping the system. Thus, distributed systems can vary in number of nodes and topology without compromising their security. We have used the DSAW platform to implement two scenarios of applying AOSD to security issues of distributed systems: one scenario of communications encryption, and another one of access control and data flow. Both solutions were introduced to the system in a transparently way, and they are able to solve the problems without interfering with the system. Depending on the dynamic environment, the solutions may be introduced or removed at runtime.

The assessment of runtime performance has shown that real applications developed with Static DSAW have entailed a performance cost of 4.12%, comparing static weaving with the traditional object-oriented development.

In the dynamic weaving scenario, comparing Dynamic DSAW with the object-oriented paradigm, the runtime performance penalty is 59.2%. The detailed anal-

ysis shows that a significant part of this penalty is due to using reflective techniques, in order to obtain the dynamism.

With respect to the memory consumption, the penalty is around 1% in the static weaving scenario and 55.96% in the dynamic one. This difference is due to the dynamic weaver's use of a special table to dynamically manage the joinpoints.

The immediately future work will be focused on applying this approach to develop other security measures such as *Intrusion Detection* or *Load Balancers*. A more involved question concerns how the approach can be suitably generalised in distributed "systems-of-systems" [1] scenarios. There are two major limitations with current AOSD methodologies that must be overcome in order to achieve this, both related to how pointcuts are specified.

First is that current methods for defining pointcuts presume a prior understanding of the way an application operates and has been written. For example, by assuming a knowledge of the way a method is named, or the parameters it takes, rather than based on its functionality. However, for the purposes of imposing security on existing services, a more refined *introspective* technique is required. This might, for example, perform an analysis of the functionality of an application's methods before deciding where to inject the appropriate aspects.

Second is that aspects are currently intended to be applied to just a single executable at a time. In a system-of-systems, where injecting an aspect into one executable may cause dependencies that require aspects to be injected into another, a richer semantics for the description of pointcuts across multiple system is required. Both of the case studies presented here are examples of this: code to encrypt data sent from one application will cause problems unless there is a respective decryption functionality provided at the receiving end.

We aim to address both of these challenges in future work by considering more flexible means of defining pointcuts, such as by allowing pointcuts to be defined in a reactive manner, taking into account the results of analysis of multiple interacting applications within a network.

Current documentation and implementation of this work can be freely downloaded from `http://www.reflection.uniovi.es/dsaw/download/2010/ietsw/`

9 Acknowledgments

This work has been funded by the Department of Science and Technology (Spain) under the National Program for Research, Development and Innovation; project TIN2008-00276 entitled *Improving Performance and Robustness of Dynamic Languages to develop Efficient, Scalable and Reliable Software.*

Bibliography

[1] R.L. Ackoff. Towards a system of systems concepts. *Management Science*, 17(11):661–671, 1971.

[2] D. Alhadidi, N. Belblidia, M. Debbabi, and P. Bhattacharya. λ_SAOP: A security AOP calculus. *The Computer Journal*, 52(7):824–849, Oct 2009.

[3] A. Belapurkar, A. Chakrabarti, H. Ponnapalli, N. Varadarajan, S. Padmanabhuni, and S. Sundarrajan. *Distributed Systems Security: Issues, Processes and Solutions*. Wiley, 2009.

[4] N. Belblidia, M. Debbabi, A. Hanna, and Z. Yang. AOP extension for security testing of programs. *2006 Canadian Conference on Electrical and Computer Engineering*, pages 647–650, 2006.

[5] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 12th conference on USENIX Security Symposium-Volume 12*, page 2. USENIX Association, 2003.

[6] M. Bishop. *Introduction to computer security*. Addison-Wesley Professional, 2004.

[7] K. Böllert. On weaving aspects. In *Proceedings of the Workshop on Object-Oriented Technology*, page 302. Springer-Verlag, 1999.

[8] J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, page 97. ACM, 1998.

[9] K. Chen and C. Huang. A practical aspect framework for enforcing fine-grained access control in web applications. *Information Security Practice and Experience*, pages 156–167, 2005.

[10] T.C. Ecma. TG3. Common Language Infrastructure (CLI). Standard ECMA-335, 2005.

[11] L.M. Feeney, B. Ahlgren, A. Westerlund, et al. Spontaneous networking: an application oriented approach to ad hoc networking. *IEEE Communications Magazine*, 39(6):176–181, 2001.

[12] G. Hermosillo, R. Gomez, L. Seinturier, and L. Duchien. AProSec: an aspect for programming secure web applications. *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pages 1026–1033, 2007.

[13] H. Holma, A. Toskala, et al. *WCDMA for UMTS: Radio access for third generation mobile communications*. Citeseer, 2000.

[14] M. Huang, C. Wang, and L. Zhang. Toward a reusable and generic security aspect library. *AOSD: AOSDSEC*, 4, 2004.

[15] W. Hürsch and C.V. Lopes. Separation of concerns. *Northeastern University, February*, 1995.

[16] J. Irwin, G. Kickzales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, and J. Loingtier. Aspect-oriented programming. *Proceedings of ECOOP, IEEE, Finland*, pages 220–242, 1997.
[17] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
[18] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. *ECOOP 2001?Object-Oriented Programming*, pages 327–354, 2001.
[19] U. Lang and R. Schreiner. *Developing secure distributed systems with CORBA*. Artech House Publishers, 2002.
[20] D. Llewellyn-Jones, M. Merabti, Q. Shi, and B. Askwith. Analysis and Detection of Access Violations in Componentised Systems. In *2nd Conference on Advances in Computer Security and Forensics (ACSF 2007), Liverpool, UK*, pages 12–13. Citeseer, 2007.
[21] C.V. Lopes and G. Kiczales. D: A language framework for distributed computing. *Xerox PARC, TR SPL97-010 P*, 9710047.
[22] A. Mourad, A. Soeanu, M. Laverdire, and M. Debbabi. New aspect-oriented constructs for security hardening concerns. *Computers & Security*, 28(6):341–358, 2009.
[23] National Computer Security Center NCSC. Trusted network interpretation environments guideline, 1990.
[24] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):999, 1978.
[25] F Ortin, L Vinuesa, and J.M. Felix. The DSAW Aspect-Oriented Software Development Platform. *International Journal of Software Engineering and Knowledge Engineering*, To be published.
[26] C.P. Pfleeger and S.L. Pfleeger. *Security in computing*. Prentice Hall, 2003.
[27] M. Pinto, M. Amor, L. Fuentes, and JM Troya. Run-time coordination of components: Design patterns vs. componentaspect based platforms. In *ASoC workshop (Advanced Separation of Concerns)*, pages 18–22. Citeseer, 2001.
[28] A. Popovici, T. Gross, and G. Alonso. AOP support for mobile systems. *Paper at the OOPSLA*, 1.
[29] A. Popovici, T. Gross, and G. Alonso. Dynamic weaving for aspect-oriented programming. In *In Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, pages 141–147. ACM Press, 2002.
[30] M. Ségura-Devillechaise, J.M. Menaud, G. Muller, and J.L. Lawall. Web cache prefetching as an aspect: towards a dynamic-weaving based solution. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, page 119. ACM, 2003.
[31] V. Shah and F. Hill. An aspect-oriented security framework. *Proceedings DARPA Information Survivability Conference and Exposition*, 2:143–145, 2003.
[32] Haitao Song, Yingyu Yin, and Shixiong Zheng. Dynamic aspects weaving in service composition. *Sixth International Conference on Intelligent Systems Design and Applications*, pages 1003–1008, 2006.

[33] Sun Microsystems. Java Virtual Machine Debugger Interface Specification. http://java.sun.com/products/jdk/1.2/docs/guide/jvmdi. 2000.

[34] Kim Taeho and Lee Hongchul. Establishment of a security system using aspect oriented programming. *2008 International Conference on Control, Automation and Systems*, pages 863–866, 2008.

[35] E. Truyen, B.N. Jørgensen, and W. Joosen. Customization of component-based Object Request Brokers through dynamic reconfiguration. In *Technology of Object-Oriented Languages and Systems–TOOLS*, volume 33, pages 181–194. Citeseer.

[36] J. Viega, JT Bloch, and P. Chandra. Applying aspect-oriented programming to security. *Cutter IT Journal*, 14(2):31–39, 2001.

[37] L. Vinuesa and F. Ortin. A Dynamic Aspect Weaver over the. NET Platform. *Metainformatics*, pages 197–212, 2004.

[38] Luis Vinuesa, Francisco Ortín, José M. Félix, and Fernando Álvarez. Dsaw - a dynamic and static aspect weaving platform. In *ICSOFT (PL/DPS/KE)*, pages 55–62. INSTICC Press, 2008.

[39] Hao Wan, Guo-an Zhao, Ze-hua Gao, Feng Gao, and Nan Wang. Universal design of web security based on AOP. *2009 International Symposium on Computer Network and Multimedia Technology*, pages 1–4, 2009.

[40] T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.

[41] Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, page 283. ACM, 2000.

[42] Z. Zhao and W. Li. Dynamic reconfiguration of distributed data flow systems. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 2, 2007.

[43] J.A. Zinky, D.E. Bakken, and R.E. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, 1997.