Architecture Description Language for COTS Integration in Aspect Oriented Software Development

Bereket Tolcha
berekettolcha@gmail.com

Mesfin Kifle
Department of Computer Science, Addis Ababa University, Ethiopia
kiflemestir95@gmail.com

Abstract

Commercial Off-The-Shelf systems or components are being popular in large system developments to save time and money to support reusability. Usually, COTS are assumed thoroughly tested in many situations and environments. They are relatively safe to integrate them to other systems. But the problem is that their integration is not easy since their internal logic is not exposed, they exist as a black box.

Hence, integrating COTS into large systems needs careful design at the architecture stage. Architecture Description Languages come into picture for this purpose. Nevertheless, the existing ADL do not support integration of COTS into Aspect Oriented Software Development approach.

This paper proposes a solution to model interaction of a COTS component with another respective software component. The paper also evaluates the proposed solution by taking two systems and integrating COTS components into them. A prototype is developed to demonstrate the proposed solution in two levels: at design time ADL specification and code level implementation.

The proposed solution uses Aspectual ACME as a base ADL by using its basic elements in representing architecture of a system to design the new artifact because it is an already developed ADL for Aspect oriented software development methodology.

Keywords: Architecture Description Language; Aspect Oriented Software Development; COTS System; Integration

1. Introduction

In Aspect Oriented software development approach, cross cutting concerns are considered as *aspects* and not just objects. Cross cutting concerns are those objects that touch many objects of the system.

There are many efforts in Aspect Oriented Software Development to manage the non-functional requirements and the tangling code problems properly [1]. The main advantage of this approach is implementing the known software development concept “separation of concern”, which dictates everything is modularized based on their concern.

Architectural Description Languages (ADL) is a language that provides primitives to specify components and connectors. A standard notation, in ADL, for representing architectures helps to promote mutual communication, the embodiment of early design decisions, and the creation of a transferable abstraction of a system. ADLs result from a linguistic approach to the formal representation of architectures, and as such they address its shortcomings. Also important, sophisticated ADLs allow for early analysis and feasibility testing of architectural design decisions.

There are different ADLs, in which some are for general purpose and some others for specific purpose. Most ADLs are developed for a specific software development process like OO Software Development, Aspect Oriented Software Development or others.

To represent the architecture of software in common and understandable way, many architects have used Architecture Description Languages (ADL) for many years [2].

The building blocks of an ADL are components, connectors, ports, roles and architectural configurations or properties. Components and connectors may have associated interfaces, types, semantics and constraints, but only explicit component interfaces are a required feature for ADLs. A component's interface is a set of interaction points between itself and the external world. It specifies the services (messages, operations and variables) a component provides and the services it requires of other components. Connectors model interactions among components and specify rules that govern those interactions. A connector's interface specifies the interaction points between the connector and the components and other connectors attached to it. It enables proper connectivity of components by exporting as its interface those services it expects of its attached components. Configurations define architectural structure and how components and connectors are connected [3].

The AspectualACME is a more generic ADL designed for aspect oriented software development by enhancing the traditional ADL called ACME which is a general purpose ADL. It also supports the representation of integration of Commercial Off-the-Shelf (COTS) software using Aspect Oriented development method. But it is more general and it doesn't represent some specific integration concerns.

Currently available AspectualACME represents the integration of third party software like COTS by considering them as another component of the system. But the integration of COTS should be handled carefully and thoroughly because of their black box nature for changeability and maintenance.

Basically acquiring COTS components should ease up the development process of a new software system quite a lot, because the development team doesn't have to "reinvent the wheel" for some particular features anymore. However, it is really rare that a COTS component can be just installed straight away to the existing production environment [4].

For the purposes of this paper, the term COTS means "a software product, supplied by a vendor, that has specific functionality as part of a system - a piece of prebuilt software that is integrated into the system and must be delivered with the system to provide operational functionality or to sustain maintenance efforts and it is a black box" [5].

More and more software projects are using COTS components. Using COTS components brings both advantages and risks. To manage some risks in using COTS components, it is necessary to increase the reusability of the glue-code so that the problematic COTS components can easily be replaced by other components. Aspect-oriented programming (AOP) claims to make it easier to reason about, develop, and maintain certain kinds of application code [1].

With AOP, each aspect can be expressed in a separate and natural form, and can then be automatically combined into a final executable form by an aspect weaver. As a result, a single aspect can contribute to the implementation of a number of procedures, modules, or objects. It therefore helps to increase reusability of the source code. AOP brings several new ideas and definitions

The main benefit of using AOP is that, all the changes are centralized in the aspect without scattering everywhere in the whole system. AOP users should ensure that the cross-cutting concerns in the glue-code are homogeneous [5]. If cross-cutting concerns in COTS glue-code can be separated into aspects, it will be easier to understand and change the system [6].

When integrating COTS components, the internal implementation may cause mismatches between these COTS components. Therefore, glue-code may be needed to integrate these COTS components and make them work together.

Hence, the following are the problems addressed in the paper: how the existing ADL for Aspect Oriented Software Development represents aspectual concerns and its limitations, how the existing ADL for Aspect Oriented Software Development represents integration of COTS, and what should be done to fill

the gap of representing the integration of COTS using an ADL.

The methodology used in this research is mostly exploration and investigation by reading the available papers about the working ADLs and after gathering enough information about the existing ones, the new solution will be designed using the basic notations and semantics used by the traditional ADL called ACME. The designed solution is tested by using two medium sized software using AspectualACME syntax and code level implementation.

2. Related Work

There are two different ADLs reviewed for this paper. One is designed as a new ADL based on traditional ones to represent all concerns of Aspect Oriented Software Development and it is called AC2-ADL [2]. The other one is a traditional ADL which is for general purpose and made some enhancements to make it fit for Aspect Oriented and it is called AspectualACME. The modification is to add a new artifact for representation of cross-cutting concerns of the Aspect Oriented Method [7] and their interaction with other components of the system. Because the second option is closest to the proposed solution, its detail is discussed below.

2.1 AspectualACME and how it works

The basic elements of ACME are not enough to properly modularize and compose crosscutting concerns (or features) with other system concerns. Crosscutting concerns are concerns that get scattered and tangled with other concerns realized by the system components. AspectualACME extends ACME in order to modularly represent crosscutting concerns at the architectural level.

It proposes modeling crosscutting concerns as regular components, and enriching composition mechanisms of ACME to support the definition of crosscutting relations. AspectualACME introduces the Aspectual Connector (AC), a special connector that encapsulates the crosscutting component interactions.

2.2 Aspectual Connector

An aspectual component is a component that represents a cross-cutting concern in a crosscutting interaction. The traditional connector is not enough to model the crosscutting interaction because the way that an aspectual component composes with a regular component is slightly different from the composition between regular components only. A crosscutting concern is represented by providing services of an aspectual component and it can affect both provided and required services of other components which can be, in turn, regarded as structural join points at the architectural level.

Aspectual Connector (AC) is an architectural connection element that is based on the connector element but with a new kind of interface. The purpose of such a new interface is twofold: to make a distinction between the elements playing different roles in a crosscutting interaction, i.e., affected base components and aspectual components; and to capture the way both categories of components are interconnected. The AC interface contains: (i) base roles, (ii) crosscutting roles, and (iii) a glue clause.

The base role may be connected to the port of a component (provided or required) and the crosscutting role may be connected to a port of an aspectual component. The distinction between base and crosscutting roles addresses the constraint typically imposed by many ADLs about the valid configurations between provided and required ports. An aspectual connector must have at least one base role and one crosscutting role. The composition between components and aspectual components is expressed by the glue-clause. The aspectual glue specifies the way an aspectual component affects one or more regular components.

3. The Proposed Solution

Based on the detailed study being conducted on articles related to ACME, Architectural structure is described in ACME with components, connectors, ports, connections and systems. The interaction

between those components will be represented by attachments.

The Aspectual Connector in AspectualACME specializes on the conventional connector abstraction to support the description of interactions among components that have a crosscutting impact and other components.

There is no special way of representing the integration of plug-in software mainly COTS type in the traditional ACME or on the modified version of it called AspectualACME. The integration of COTS is represented in the architecture simply by normal components and connections available in the traditional ACME.

The artefact which will be used to facilitate the integration of COTS component into a system that is developed using Aspect Oriented Software Development method is designed using the basic architectural elements of AspectualACME ADL (connector and role).

The new artifact is designed by making the following considerations:

- It uses the wrapper to abstract the black box behaviour of COTS based on the comparison of the three methods in Table 1. This will make the integration changeable and maintainable because it hides the detail complexities of the COTS components and it is better than the other two options because we can detach it including the COTS component from the main system without affecting it.
- The COTS component will be considered as another component of the system using its wrapper to communicate with other components of the Aspect Oriented system.

So based on these considerations, the new artefact is a connector (named COTS_Connector) with some modifications from the basic connector element of ACME to represent the specific connection characteristics related to COTS integration.

3.1 Architecture of the Solution

The newly designed connector called COTS_Connector will contain two roles. The first role is a normal AspectualACME role in the System component side (R) and the other is from the COTS component side (CR) which will be abstracted through the wrapper role WR.

Table 1: Comparison between three techniques of COTS integration

Method	*Wrapper*	*Glue-ware*	*Proxy*
Hides complexity of the COTS component	Yes	Yes	Yes
Makes the COTS component as a system component	Yes	No	No
Easy to change the COTS component	Yes	No	Yes

Because the wrapper abstracts the existence of the COTS component, it should expose the services offered by the COTS component using COTS Role (CR) through the Wrapper Role (WR). The COTS_Connector will make use of the Wrapper Role to access the COTS_Role to connect to the COTS components services which makes the COTS component abstract so that there is no direct interaction between system components and COTS component. This will make the architecture more flexible for maintenance and change of COTS component for many reasons like if new functionality is required or the previous COTS is updated and needs to be replaced by the updated one.

Table 2: The syntax difference between ACME connector and COTS connector

Regular connector in ACME	*COTS_Connector in AspectualACME*
Connector Connector1 = { Role aRole1; Role aRole2; }	COTSConnector Connector1 = { Role R; Wrapper_RoleWR; COTS_Role CR; }

Table 2 compares the COTS connector with a normal component connector without aspectual concerns. But the COTS connector can also connect an aspectual component with COTS component. In this case the COTS definition will change accordingly to reflect the characteristics of the aspectual component.

Table 3: The syntax difference between AspectualACME connector and COTS connector

Aspectual connector in AspectualACME	*COTS_Connector in AspectualACME*
AspectualConnector Connector1 = { Role AspectualRole1; Role Role1; }	COTSConnector Connector1 = { Role AspectualR1; Wrapper_Role WR1; }

Wrapper_Role WR1 = {

systemRole SR1;

COTS_Role CR1;}

- Where systemRole represents the role of the larger system which integrates COTS.

The basic difference between COTS connector and the other two connectors (normal and Aspectual), as shown in table 3, is that in case of COTS connector it has another additional role for the wrapper to refer to the COTS role and the wrapper role will have its own definition to combine its role with the COTS role. This way of integrating COTS components will guarantee the abstraction of the black box nature of COTS. So there will be a very minimum work required in case of COTS replacement or maintenance.

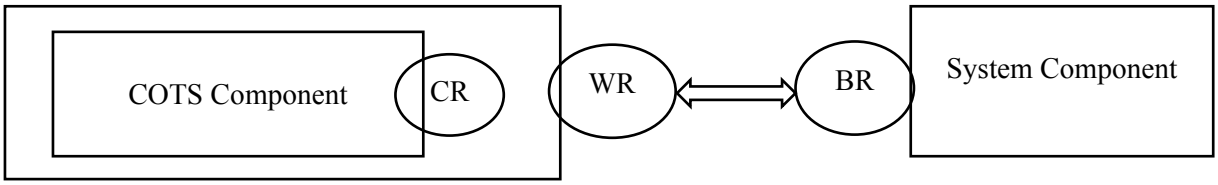

Figure 1: How COTS_Connector connects the system component and the wrapped COTS component

In figure 1, the major characteristic of COTS_Connector in addition to the basic characteristics it inherits from ACME Connector is that the connector has two sides. One points to the system component, it could be aspectual component or normal component, and the other side points to the wrapper component which also connects to the COTS component.

We have now three connectors in AspectualACME including the newly designed one to represent (i) normal connection between components, (ii) to represent aspectual concerns between normal component and aspectual component, and (iii) the new one to represent the connection between normal or aspectual component and the COTS component.

There are some basic differences between these three connectors irrespective of their common characteristics like properties and attachments, as described below:

ACME Connector: is the basic connector defined by ACME and it only connects two normal components without aspectual concerns

AspectualACME Connector: is the modified version of the above connector so it works for normal components and in addition it connects normal components with aspectual components by defining the crosscutting concerns connection

COTS Connector: is also the modified version of the ACME connector so it did work for connecting normal components and in addition it connects normal and aspectual components with COTS component through a wrapper.

To evaluate the designed solution, two systems are considered with two COTS components for each system. Both are used aspect oriented method in the implementation level.

i. Vehicle Management System (VMS) which is Developed for Addis Ababa Transport Bureau

ii. Integrated Market Actors Management Information System (IMAMIS) which is designed for Ethiopian Commodity Exchange Authority

The first system is explained below because it is the one used for prototyping.

Vehicle Management System

The Vehicle Management System is a medium sized software system designed to facilitate all the services given by transport bureau related to vehicles for owners and other stakeholders like Customs Authority to make sure whether the owner paid stamp duty or not and many others. The system is developed using aspect oriented software development method.

From the list of services provided by the system, we will explain temporary registration service and its prototype will be presented next. It contains a list of sub-services to accomplish the complete task. The sub-services or features are:

- Register Vehicle Information
- Register Owner Information
- Collect Service Payment
- Issue Temporary Plate

The COTS have the functionality to Register Vehicle Information sub-service, which has a list of activities in it:

- Connect to customs database based on the given connection information
- Check for vehicle availability based on a given vehicle motor no or chassis no
- Collect and provide vehicle information based on a given vehicle motor no or chassis no

When an owner is registered there are steps to be taken before saving the owner information. The workflow for owner registration is listed below:

1. Check if the vehicle exists
2. Collect the vehicle information from Customs and save it to Transports database
3. Using the registered vehicles code, register owner related to that specific vehicle

The COTS component will also integrate with the aspectual component (security) of the system. The integration syntax could be as follows:

```
COTSConnector Connector1 = {
                    Aspectual_Role
                    AspectualR1;
                    COTS_Wrapper_RoleCWR1
                    ;}
```

In the above syntax, the COTS connector is defined by two roles which are Aspectual_Role to represent the aspectual component's role and the COTS_Wrapper_Role to represent the wrapper role which holds the COTS wrapper inside it.

4. Discussion

The designed solution is evaluated and it is proven that the new artifact will solve or minimize the complexity of integrating COTS components into aspect oriented systems. It is also tested that the new artifact will make the changeability and maintenance of the COTS component because its integration is abstracted by the wrapper component. The integration of the system is being made directly with the wrapper component and not with COTS and the wrapper will communicate with the COTS. So to change or maintain the COTS component it will be just to modify the wrapper class without affecting the actual system.

5. Implementation

To demonstrate the real life applicability of the proposed solution, a prototype is designed and implemented in two scenarios for the Vehicle Management System. The first scenario is using AspectualACME syntax to demonstrate the proposed connector whether it is usable to represent the integration of COTS into an aspect oriented software. The second scenario will demonstrate the code level implementation of the architecture model represented by the ACME ADL syntax in the first scenario. This second scenario integrates two types of COTS into a system to show the flexibility of the proposed solution in changing the COTS. The two COTS types will

have a pre-compiled .dll library and a "web service". Both types will have the same functionality but in different internal ways of implementation.

The implementation of the model will be using Microsoft Visual Studio 2010 and Microsoft SQL Server 2005 products for coding the functions and developing the database backend respectively.

The components to fulfill Vehicle registration in VMS are RegisterVehicle (COTS Component), RegisterOwner (System Component) and the Wrapper component which hides the complexities of the COTS and present available and required features for VMS. The VMS by the help of the COTS will check if there is vehicle information available from the customs authority database and bring the information to the transport authority database if it exists when searching by vehicles Motor Number or Chassis Number which are unique identifiers of any vehicle worldwide. Then VMS will register the owners for the registered vehicle using RegisterOwner component developed in-house.

Through the process of checking information availability and registering of vehicle information is accomplished using the COTS component, the VMS shouldn't communicate to the COTS directly because it will make the integration so tight that any modification, maintenance or replacement of the COTS component will affect the VMS as a whole significantly. So to handle this integration problem, we implement wrapper component which mediates the COTS component and VMS.

To accomplish the above stated task, the wrapper component will have its own configuration document which made the mapping of features between the COTS component and VMS possible. Managing the mapping of features in configuration file will make the integration easy to maintain in case of any maintenance or complete change of the COTS component for any reason. There could be a different means of designing the wrapper component to fulfill the requirements other than using configuration file but in this demonstration the configuration file is chosen. To simulate and demonstrate the applicability of the wrapper component using its configuration file, the scenario described above will be implemented as follows.

Some parts of the configuration file of the wrapper component is presented below.

// here we have application setting required to map the location of the COTS component and different function names provided in the COTS component. The function names are required to create function mapping from the COTS and the complete system without changing the wrapper code in case of any function name or other change occurs in the COTS component. If any such change occurs in the COTS, it will be changed only on the configuration document without affecting the wrapper code.

```
<settingname="COTSLocation"serializeAs="String">
<!--this setting will be used if the above setting
    'COTSType' is set to 'dll'-->
<value>C:\Users\btolcha\Documents\Visual Studio
    2010\Projects\ThesisPrototype\RegisterVehicleInf
    ormation\bin\Debug\RegisterVehicleInformation.d
    ll</value>
<!--this setting will be used if the above setting
    'COTSType' is set to 'webservice'-->
<value>http://localhost:8732/RegisterVehicleService/
    RegisterVehicle.svc</value>
</setting>
<!--the following three settings are used in the
    wrapper to create delegate(representative) function
    based on their name in the COTS-->
<settingname="ConnectToDatabaseWrapper"serialize
    As="String">
<value>ConnectToDatabase</value></setting>
<settingname="IsVehicleInfoAvailableWrapper"serial
    izeAs="String">
<value>IsVehicleInfoAvailable</value></setting>
<settingname="GetVehicleInfoWrapper"serializeAs=
    "String">
    <value>GetVehicleInfo</value></setting>
```

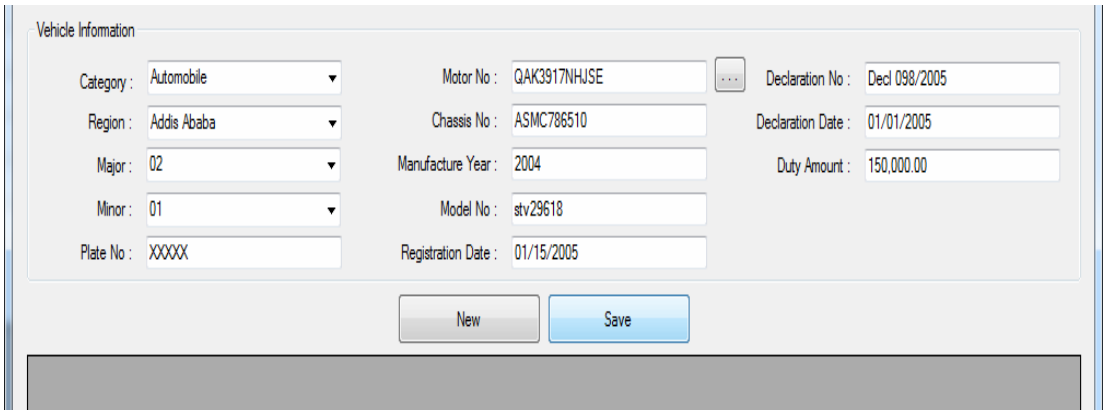

Figure 2: How the window looks when vehicle information is registered after completing vehicle information in addition to the information displayed using search from the customs database

After filling the rest of the information in addition to the information found from customs database including the owner information, the system will look like figure 2.

The new button in Figure 2 is used to search the vehicle from customs database using the COTS function through the use of wrapper components function. After the search is completed if the vehicle is available in the customs database by motor number or chassis number, it will be displayed in the space provided. Like for example, the 'Manufacture Year", "Model No", "Declaration Date" are displayed in the text area. The rest of the information will be filled by the system user and using 'Save' button on the 'vehicle information' section, the user can save the vehicle information into the transport authority database.

6. Contribution and Future Work

The main motivation of this paper is to design a way to integrate COTS components in an efficient and cost effective way in time of development and also in time of maintenance or change of the COTS component. The change may be required for many reasons like new requirement came up from the users or just the already existing COTS don't have the required feature.

The designed artifact is evaluated using two medium sized systems with aspect oriented implementation and it works well in hiding the direct contact between the COTS component and the system component. And also it doesn't require additional syntactic knowledge other than basics of ACME ADL elements.

As a future work, it would make the solution complete if the integration concept is handled for COTS with crosscutting concerns (aspectual components).

References

[1] Axel Anders Kvale, et al., "A Case Study on Building COTS-Based System Using Aspect-Oriented Programming", 2004.

[2] Wen Jing, et al., "AC2-ADL: Architectural Description of Aspect-Oriented Systems", 2009.

[3] Thaís Batista, et al., "Reflections on Architectural Connection: Seven Issues on Aspects and ADLs", 2005.

[4] Heikki Kontio, "Current Trends in Software Industry: COTS Integration", 2008.

[5] M. Morisio1, et al., "Investigating and Improving a COTS-Based Software Development Process", 1999.

[6] E. Kesseler, "Assessing COTS Software in a Certifiable Safety-Critical Domain", Information Systems Journal (18) 2008, pp 299 - 324.

[7] Alessandro Garcia, et al., "On the Modular Representation of Architectural Aspects", 2006.