A Domain-specific Middleware Layer using AOSD: Next-Generation Digital News Publishing

Steven Op de beeck, Dimitri Van Landuyt,
Eddy Truyen and Pierre Verbaeten
DistriNet, Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium
{steven.opdebeeck,dimitri.vanlanduyt,
eddy.truyen,pierre.verbaeten}@cs.kuleuven.be

ABSTRACT

In this demonstrator we illustrate the design and development of a domain-specific middleware layer for supporting a publishing infrastructure. The design of the middleware layer is based on an *aspect-oriented architecture* which has been iteratively defined by applying state-of-the-art aspect-oriented methods. The demonstrator shows the implementation and deployment of the middleware layers on top of JBoss, using the JBoss AOP framework.

Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Distributed applications;
C.2.11 [Software Architectures]: Domain-specific architectures

Keywords

Domain-specific Middleware, AOSD

1. BACKGROUND

The way in which news is produced and distributed is changing significantly. The business landscape of publishing is drastically evolving towards a more competitive environment with a lot of new players emerging from unexpected sides such as Google News [3]. This has put a lot of pressure on the traditional publishing companies to evolve to more competitive news offerings.

There has been a trend towards *digital publishing platforms* that support multiple service offerings which are bundled depending on the targeted group of customers. A digital publishing platform offers news in different media formats and sizes, to be delivered through different telecommunication channels, and to be displayed on different end user terminals and devices. In this way, the daily (on-line) newspaper is complemented with an aggregation of differentiated services. These services may also offer personalized

content and advertisements, tailored to the interests of individual customers, their location, situation and behaviour. Although it may not be clear yet whether this kind of flexible service offering represents the future of news publishing, it is certain that publishing companies will gain a competitive advantage if their ICT infrastructure can rapidly evolve towards supporting these new kinds of offerings.

Figure 1 depicts the system architecture of a typical digital publishing platform. At the top-left side of the figure, the different output channels are presented: the publishing platform is able to deliver news in real-time to mobile users connecting *on the move* using a mobile device (through a telecom infrastructure). Also, the news services are offered to home users that connect via the publisher website. The publishing platform is even backwards-compatible in the sense that classical delivery mechanisms like printing and physical delivery of the newspaper are still supported. At the center of the figure, the publishing server is presented, together with a replica. These nodes consist of the actual services offered by the publisher. At the bottom side of the figure, the accounting system is represented, connecting the publishing servers to financial institutions (e.g. bank servers) to trigger payment transactions. At the right-hand side of the figure, two separate back-end servers are represented, one for each of the main types of data. The news content consists of articles, advertisements, advertisement reservations, and their meta-data (tags, etc). The user profile information consists of all customer information, together with user interests that are both statical (defined by the user himself) and dynamical (inferred by the publishing system).

2. PROBLEMS ADDRESSED

To implement and deploy such a digital publishing architecture in a realistic setting, we have to take into account that the existing ICT infrastructure of publishing companies have evolved over the years: they are not developed from scratch. Various concerns in publishing systems (such as authentication, user profiling, verification, rights management, feedback, etc.) were implemented differently in each iteration. Moreover, a lot of these concerns are known to cut across the existing workflow of the publishing company (see Figure 2). Evolving these crosscutting concerns is a very complex matter because it requires invasive changes to multiple software components of the existing ICT infrastructure. These intrusive and crosscutting modifications have set back the overall quality and maintainability of the publishing in-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Middleware'08 Companion December 1-5, 2008 Leuven, Belgium.
Copyright 2008 ACM 978-1-60558-369-3 ...\$5.00.

Figure 2: Concentra Media: News Composing and Exchange System

Figure 1: Deployment view of distributed architecture.

frastructure.

In this demonstrator we illustrate the design and development of a domain-specific middleware layer for supporting a publishing infrastructure. The design of middleware layer is based on an *aspect-oriented architecture* which has been iteratively defined by applying state-of-the-art aspect-oriented methods[2]. The demonstrator shows the implementation and deployment of the middleware layers on top of JBoss, using JBoss AOP framework [4]. This illustrates the benefits of employing AO component models on top of thin (rather than fat) middleware containers with respect to evolvability.

The architectural idea behind the design of this publishing middleware leverages aspects as modular service extensions or features. In this way, extensions such as user tracking and personalization of news services can be optionally included in a particular deployment of the system. This kind of variability allows the creation of customized news service offerings that are tailored to a particular group of customers.

3. AOSD AND MIDDLEWARE

Traditional container-based middleware platforms are typically equipped with several services that support different non-functional requirements such as transaction management, asynchronous messaging and security (authentication and authorization). The implementation of these middleware platforms is however often monolithic because there is a high coupling between the middleware services and the middleware container. As a negative consequence, there is often no easy way to adapt the implementation of a particular middleware service. Nor is it possible to remove unneeded services from the container, or install new (third-party) middleware services. For example, in a classical J2EE container like the GlassFish application server [1], it is not possible to remove *transaction handling*, even when such functionality is not needed.

This lack of adaptability often limits the reusability of the middleware services. When a certain middleware service does not comply to the exact requirements of the application, the application developer has no option left than to fully implement his own specific requirements on top of the middleware, without being able to reuse parts of the middleware service implementation. Suppose for example that the built-in authorization service offers role-based access control, while the application requires also mandatory

access control. In an ideal situation, the application developer would implement the enforcement of the mandatory access control as an incremental replacement of the role-based access control service. However, currently, the application developer must fully implement mandatory access control on top of a “fat” middleware container, which is considered rather useless from a security perspective.

These downsides can be tackled using aspect-oriented software development techniques. AOSD considerably improves the modularization of infrastructural middleware services. For example. In version 4 of the JBoss application server, some mission-critical services such as transaction management, asynchronous messaging and security (authentication and authorization) are implemented using the JBoss AOP framework [4]. These aspects are however still “under the hood” and therefore not open for extension. At the level of a programming interface, annotations or deployment descriptors are still being used. As such the real benefits of aspect-orientation cannot really made available to the programmer. If one wants to adapt the middleware, the programmer needs to master various skills and knowledge to delve deep inside the code. The architecture of middleware is still a black-box because of the complexity of various API’s. Similar, the application of reflective meta-object protocols to middleware has led to similar obstacles due to the sheer complexity.

What is missing is a coherent top-down view on the architecture of the middleware. By treating the development of middleware implementation as a full-life cycle process, all the way from requirements, to architecture and deployment, developers can better identify, compose and trade-off various design decisions in middleware.

Another challenge is the modularization of crosscutting concerns that become visible during build-time or thereafter. Currently all server-side components of the publishing system run on a single node. A real deployment of the system typically executes on top of a distributed system of multiple nodes. However, JBoss AOP’s support is limited to specifying aspect-oriented compositions within a single node. As a result, aspect-oriented compositions that crosscut across N different nodes (e.g. suppose the Accounting component must be invoked from N nodes) can not be cleanly captured. A workaround in our current architecture would be to deploy N separate JBoss AOP aspects, one for each node of the system. This considerably complicates the build process, however, and obstructs the kind of feature-oriented variability that we have demonstrated before. The advantage of Distributed AOP platforms such JAC [7], AWED [6] and DyMAC [5] is that these platforms allow specifying such N-to-1 compositions in a single unit of deployment. As a result, using one of these platforms, the build scripts could be kept oblivious from this particular deployment-time concern.

4. DESCRIPTION OF DEMONSTRATOR

The “fil rouge” of the demonstrator is an evolution scenario where an existing publishing infrastructure (depicted in Figure 3 as Iteration 1) is extended with digital publishing functionalities (depicted in Figure 3 as Iteration 2).

Iteration 1.

A first version of the publishing infrastructure is constructed using classical component-based, non-AO techniques. This first version is a very basic newspaper website, where the media consumer is allowed to fetch news for free.

Also, advertisers can submit advertisements. This is the only source of revenue; i.e. revenue generated by advertising. The result of this first phase is a working and deployed publishing infrastructure, built and documented throughout the software life-cycle.

Iteration 2.

In Iteration 2, the publishing infrastructure undergoes a major evolution towards supporting next-generation publishing characteristics. The new requirements include more thorough user authorization and user tracking, which are in turn enablers for additional requirements such as content personalization, advertisement personalization, and accounting. In this second iteration, a re-factoring process is first performed, to analyze which concerns crosscut the first version of the software, and to separate these using AOSD-Europe’s approach for aspect-oriented architecture [2]. This leads to an aspect-oriented architecture. Then, this aspect-oriented architecture is extended with the features that support the requirements for *personalization* and *user tracking*.

Figure 3: Overview of the evolution scenario adopted in the demonstrator.

After having completed and tested our implementation, we are ready to deploy the system. Deployment involves mapping the different implementation modules to units of deployment. Because of the specific structure of this mapping, AOSD provides good support for *feature-oriented variation points* that are typically desired in a software product-line. This is elaborated below:

- **Features.** Specific features in the product line are represented in the form of aspects. Each aspect is an extension to a base architecture, extending the existing base functionalities with crosscutting functionalities, such as *security*, *accounting*, *personalization* and *user tracking*.
- **Variation points.** Variability points are certain points in execution where variation —the addition of certain features— is expected. These points are

Figure 4: A software product line view on the publishing middleware architecture.

represented by plain java annotations, such as `ServiceUsage` annotation. These are points in the base architecture that are exposed to the outside world, and open for extension.

5. DEMO

In the demonstration we will show that features such as user tracking and personalization of news services can be deployed into the system dynamically. This opens the way for selling these features as additional services that can be turned on or off for individual users. This in turn produces additional business value because the run-time variability supports characteristics of an ideal news product that allows direct marketing to targeted groups of customers.

We will make use of a screen-cast video that shows the application through a number of usage scenarios (including the runtime variability). A live running version of the application will also be present for those with a deeper interest. We will also provide a number of high-level development documentation artifacts to help explain the design of the application.

Hardware and presentation requirements

A beamer and projection screen. Laptops will be provided by us.

Acknowledgements

This research is supported by European Commission FP6 Grant AOSD-Europe: European Network of Excellence on AOSD (IST-2-004349), the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

6. REFERENCES

- [1] Glassfish; <https://glassfish.dev.java.net/>.
- [2] B.Tekinerdogan, A. Garcia, C. Sant'Anna, E. Figueiredo, M. Pinto, and L. Fuentes. Approach for modeling aspects in architectural views. *AOSD-Europe Deliverable D77*, 2007. <http://www.aosd-europe.net/deliverables/d77.pdf>.
- [3] Google inc. Google news; <http://news.google.com>.
- [4] Red Hat inc. Jboss.com; <http://www.jboss.com/>.
- [5] B. Lagaisse and W. Joosen. True and Transparent Distributed Composition of Aspect-Components. In

Proceedings Middleware'06, volume 4290 of *Lecture Notes in Computer Science*, 2006.

- [6] Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, Bruno De Fraine, and Davy Suvée. Explicitly distributed aop using awed. In *Proceedings of AOSD 2006*.
- [7] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin. JAC: A flexible solution for aspect-oriented programming in Java. In *Reflection 2001*, volume 2192 of *Lecture Notes in Computer Science*, 2001.