A Consistency Framework For Dynamic Reconfiguration in AO-Middleware Architectures

Bholanathsingh Surajbali,[1], Paul Grace[2] and Geoff Coulson[3]

[1] Smart Research Development Centre, CAS Software AG, Karlsruhe, Germany
b.surajbali@cas.de
[2] IT Innovation, University of Southampton, Southampton, UK
pjp@it-innovation.soton.ac.uk
[3] School of Computing and Communication, Lancaster University, Lancaster, UK
geoff@comp.lancs.ac.uk

Abstract. Aspect-oriented (AO) middleware is a promising technology for the realisation of dynamic reconfiguration in distributed systems. Similar to other dynamic reconfiguration approaches, AO-middleware based reconfiguration requires that the consistency of the system is maintained across reconfigurations. AO middleware based reconfiguration is an ongoing research topic and several consistency approaches have been proposed. However, most of these approaches tend to be targeted at specific narrow contexts, whereas for heterogeneous distributed systems it is crucial to cover a wide range of operating conditions. In this paper we address this problem by exploring a flexible, framework-based consistency management approach that cover a wide range of operating conditions ensuring distributed dynamic reconfiguration in a consistent manner for AO-middleware architectures.

1 Introduction

A fundamental challenge for distributed systems is their need to support dynamic reconfiguration in order to maintain optimal levels of service in diverse and changing environments. In response to this challenge, aspect-oriented AO-middleware [1], [2], [3], [4] has recently emerged as a suitable architecture to build reconfigurable distributed systems. The core concept of AO-middleware is that of an aspect: a module that deals with one specific concern and can be changed independently of other modules. Aspects are made up of individual code elements that implement the concern (advices) which are deployed at multiple positions in a system (join points).

Dynamic reconfiguration of distributed systems requires assurances that the reconfiguration does not leave the system in an inconsistent state that can potentially lead to incorrect execution or even complete system failure. In AO-middleware environments reconfiguration inconsistencies arise from a range of characteristic sources - for example, if an encryption mechanism is added to the source of a communication channel, a corresponding decryption mechanism must be added to the sink of the channel; a given system must be reconfigured transactionally such that a given change is applied either to all of a specified set of targets, or to none; or a given system must be reconfigured

such that it must not expose more security vulnerabilities than it was exposed to initially. In general, avoiding these sources of inconsistency is a difficult task due to the diversity of distributed applications (e.g. centralised/decentralised, static/mobile, small scale/large scale etc) and also because of diverse application-specific factors (e.g. varying dependability requirements, or varying trade-offs between consistency and scalability). Relying on the application developer to ensure the consistency of the system is not feasible under such heterogeneous conditions. Moreover, a one-size-fits-all approach to consistency management is not feasible either. Instead, multiple consistency strategies should be supported within a framework-based approach so that appropriate strategies can be applied to each set of arising circumstances.

This paper therefore focuses on this latter perspective: that of identifying and mitigating the numerous incidental threats that can lead to inconsistent reconfigurations in AO-middleware systems. To address this perspective we present a novel distributed consistency framework, named COF for AO middleware environments that maximises the probability of consistent dynamic reconfiguration in the face of incidental factors. A key contribution is our approach itself is highly configurable and reconfigurable, as the frameworks mechanisms for detecting and repairing threats are themselves composed of dynamically woven aspects.

The rest of the paper is organised as follows. Section 2 presents, a threat taxonomy of the various threats to consistency to AO-middleware architectures prone. Then, in Section 3, we describe the COF framework, followed by Section 4 evaluating COF performance overhead. Finally, Section 5 discusses related work and we offer our conclusions in Section 6.

2 Threat Taxonomy

In this section, we present a list of threats which may jeopardise the consistency of a critical distributed system due to dynamic reconfigurations. To illustrate the "big picture" of our approach we present Fig.1 a generalised system model of an AO-middleware platform; this illustrates one AO-middleware instance for simplicity but the model is repeated across nodes in the distributed system. The model consists of five core entities: (i) the *reconfiguration agent* representing the entity initiating reconfiguration requests; (ii) a *configurator*, which acts on the reconfiguration request; (iii) an *AO-middleware platform* providing the necessary abstraction to support the composition and reconfiguration of distributed aspects to underpin the distributed application services; (iv) a set of *infrastructure servers* providing a set of infrastructure services to the system, such as hosting the system repositories (containing aspect software) and (v) the *communication service* providing exchange of messages and events among the different address spaces (referred as nodes) in the distributed environment. Also within the model, we identify a set of core join points (numbered 1 to 5 in Fig.1) at which aspects can be woven within a given instance of the AO-middleware deployed at each node. Hence, these are the points where the consistency framework solutions (in the form of threat aspects) are deployed to ensure consistency is achieved.

Compositional Threats These relate to conflicting dependencies of reconfiguration resulting in negative interactions between system entities. For instance, some aspects

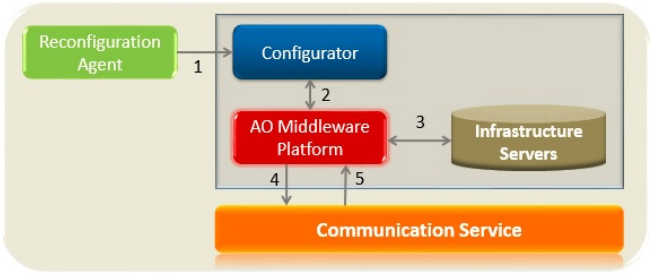

Fig. 1. Generalised model for AO-middleware platforms

are inherently dependent on each other such as a decryption aspect is dependent on the corresponding encryption aspect. Therefore, the order in which aspects are woven is crucial: e.g., encryption must be put in place before its associated decryption. Further, "remote aspects" [1] which are used by several distributed client nodes can be a source of inconsistency; for example, if a cache implemented as a remote aspect is removed without the consent or even the awareness of its client nodes, errors can arise when clients attempt to communicate with the cache. Finally, semantic conflicts can occur due to incompatibilities of the reconfigured aspect with the rest of the system as may arise in the deployment of logging and privacy aspects [10]. Moreover, the composition order in which aspects are woven can also affect their interactions, for example, if a cache advice is executed before an authentication advice, clients may be able to get access to resources without first authenticating themselves.

Operational Threats The inherently unstable characteristics of the networks and nodes employed in the scenario increase the chances that a reconfiguration will be compromised. For example, application nodes may fail to apply a requested reconfiguration if: i) the node is overloaded or has crashed; ii) the node's local policy forbids it to make the requested change; iii) aspects may still be performing computations when an attempt is made to remove or recompose them. Such factors can clearly lead to parts of the intended reconfiguration not being carried out, and consequent inconsistency. Further, aspects to be reconfigured into the system are typically stored in infrastructure service repositories which may get congested with requests, or themselves crash, meaning that aspects may not be available to be deployed in some cases or at some times. Additionally, different repository instances may have different versions of the aspects: e.g. different versions of the encryption aspects may be produced over time, so that different nodes configure different versions and be inconsistent with each other. Finally, if reconfiguration-related messages are lost, re-ordered, duplicated or delayed, the consistency of the reconfiguration can be compromised. For example, a fragmentation aspect may be deployed but not the corresponding reassembly aspect.

User Threats. These refer to threats introduced to the AO-middleware system model by the reconfiguration agent; this can be the developer/administrator, or software runtime code initiated by some authority manager (e.g. in self-managed systems). For example, if a reconfiguration request is not properly checked, it may proceed while containing errors (for example wrongly formed declarative reconfiguration specifications) which

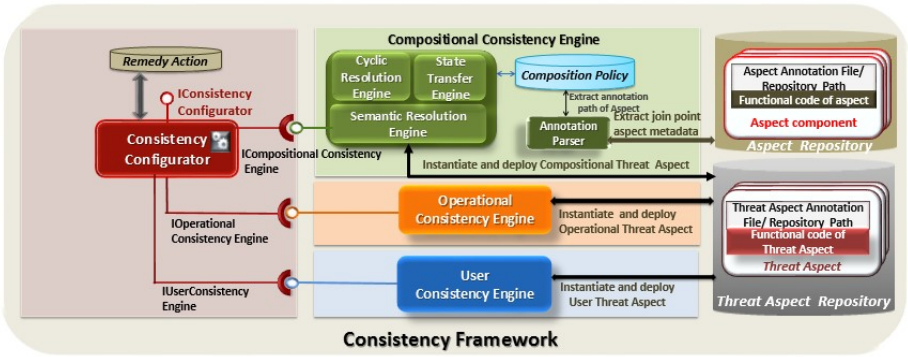

Fig. 2. Consistency Framework (COF)

may lead to incorrect actions and system inconsistency when the reconfiguration is applied. Similarly, a reconfiguration request may be unauthorised or reconfiguration messages may be spoofed by malicious nodes in an attempt to compromise consistency. In addition, reconfiguration requests may arise simultaneously in the system so that reconfiguration-related messages relating to distinct requests may be interleaved and potentially received in different orders at different nodes. For example, one request may ask to replace the fragmentation aspect with a different algorithm, while another asks for it to be removed. There will clearly be different outcomes depending on the execution order of these two requests and furthermore the outcomes might be different at different nodes.

3 Consistency Framework (COF)

The consistency framework (COF) as shown in Fig.2 addresses the reconfiguration threats defined in the threat taxonomy. Importantly, COF defines a canonical set of threat aspects that mitigate the threats found in the taxonomy, and an associated set of join point strategies to guide the application of the threat aspects within diverse AO-middleware implementations.

3.1 Consistency Configurator

The Consistency Configurator acts as a unit of autonomy making decisions about when and how to perform consistent reconfiguration. The Consistency Configurator is connected to the Remedy Action repository providing appropriate remedy actions to the Consistency Configurator for each reconfiguration. The Remedy Action uses a "condition action" approach that evaluates the reconfiguration request and instructs the Consistency Configurator to deploy appropriate threat aspects using the three main consistency engines. The consistency engines each evaluate the corresponding join points if they already have the required threat aspects. If the join points are present, an acknowledgement is returned to the Consistency Configurator, otherwise threat aspects are loaded from the Threat Aspect Repository and deployed at the defined join points.

On receiving a reconfiguration request with consistency threat aspects, the Consistency Configurator checks the aspect threat specification, associated with the reconfiguration script with the Remedy Action. The list of aspects required to be deployed for the reconfiguration is returned to the Consistency Configurator, which then sends to each consistency engine the list of threat aspects required at the join points. Each of the consistency engines then checks using the Aspect Repository if the threat aspect is present at the AO-middleware platform join point. If the threat aspect is present, the consistency engine returns an acknowledgement back to the Consistency Configurator for the reconfiguration to proceed. If no threat aspect is woven at the join point, then the consistency engines requests the instantiation of the threat aspect from the Threat Repository. The threat aspect instances, as well as the join point where the threat aspect needs to be woven are sent to the AO-middleware platform weaver. In case, a threat aspect is already woven and needs to be replaced, the Consistency Configurator first ensures that the reconfigured threat aspect is not performing any computation.

3.2 Compositional Consistency Engine

The Compositional Consistency Engine (CCE) addresses compositional threats in AO-middleware architectures by encapsulating and deploying:

- a *coordination protocol* such as Necoman protocol [5] and a *transaction protocol* encapsulated as an aspect and woven as a "before" advice at the top of the communications stack at join points 4 and 5 to address *dependency inconsistencies*.
- a *caretaker aspect* that proxies the aspect being reconfigured at join point 2 to address *unsynchronised unbinding of distributed aspects*; such that on receiving a message from a client the caretaker instructs the client that the aspect has been removed.
- *semantic reasoning and resolution engine (SRE)* [6] to query and resolve possible sources of inconsistency at join points 1 and 4 to detect semantic conflicts from incoming reconfiguration requests (from the reconfiguration agent) or from reconfiguration requests sent from the network.
- the *R*esolving *Cycl*ic Dependencies *E*ngine (ReCycle) [7] to detect cyclic inconsistencies from incoming reconfiguration requests from the reconfiguration agent or from incoming requests from the network by encapsulating and weaving ReCycle as aspect at join points 1 and 4.

3.3 Operating Environment Consistency Engine

The operating environment consistency engine component addresses the various distributed operating environment reconfiguration threats by encapsulating and deploying:

- a *transaction aspect* at the communication interface (join point 5) to detect *local node disruptions* and provide consensual decision making on what to do when these occur (e.g. accept the partial failure or roll back).
- *replication* [8] and *load balancing strategies* [9] aspects at the interface to the infrastructure services (join point 3) to detect*infrastructure service failures*.
- a *reliability* threat aspect at join points 4 and 5 to create a reliable communication service to handle *communication failures*.

3.4 User Consistency Engine

The user consistency engine component addresses the various user defined reconfiguration threats by encapsulating and deploying:

- a *reconfiguration validator aspect* to validate the reconfiguration script against policies to ascertain the correctness of the reconfiguration operation at join point 1 to resolve *badly formed requests*.
- an *authentication aspect* as "before" advice at the AO-middleware platform's communication interface at join points 4 and 5 to address *unauthorised reconfigurations*. This ensures only authentic users can adapt the system. Furthermore, in an un-trusted environment, additional *encryption and decryption aspects* can be woven at the communication interfaces (i.e. join point 4 and 5 respectively) of the sender and receiver (e.g. public or private cryptography algorithms can be used).
- a *distributed concurrency aspect* at join point 1 so that each reconfiguration request is isolated within a critical section addressing *simultaneous reconfigurations* inconsistencies.

4 Performance of COF

We now assess the performance characteristics of COF in two AO-middleware platforms we have considered (i.e. AO-OpenCom [4], and the JBoss AOP version of DyReS [10]). For this we use an experimental setup based consisting of a small network of four standalone workstations employed as shown in Fig.3a: a 1.8 GHz Core Duo 2 PC with 3GB RAM (node A); a 3.4 GHz Pentium IV PC with 1GB of RAM (node B); and a 2.8GHz Pentium IV PC with 1 GB of RAM (node C); a 1.33 GHz Core Duo 2 laptop with 2GB of RAM (node D). All of these are connected via a 100Mbps local area network. Each evaluation machine was installed with the AO-OpenCom and DyReS framework which was executed on a Java 1.7 virtual machine (VM). Based on this setup, the different threat aspect are represented in Fig.3b and the reconfiguration we perform is to *dynamically weave a symmetric AES [11] encryption/decryption aspect across each of the nodes*. The overhead results are shown in Table 1. It should be pointed that we do not claim that these results are in any sense definitive. Rather, they are indicative of the order of magnitude of overhead to be expected of COF deployments. In particular, the numbers are specific to our implementations.

Table1: Reconfiguration of COF with AO-OpenCom and DyReS

	Overhead Using COF (ms)		**Steady State Latency Time (ms)**	
	AO-OpenCom	*DyRes*	*AO-OpenCom*	*DyRes*
Without COF	1994	5311	1724	5852
With COF	2995	7241	1724	5860

We can see that the base time to perform the reconfiguration without COF varies considerably across the two platforms: AO-OpenCom is fastest, with DyReS taking 2.66 times longer. The longer time taken by DyReS over AO-OpenCom is attributed mainly to the former's use of the NeCoMan coordination protocol [12], which seems

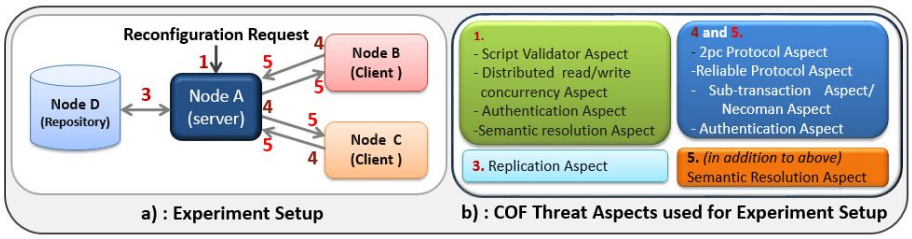

Fig. 3. Experimental setup to evaluate COF

to incur a high degree of inter-node chattiness. In terms of the COF-induced overheads, AO-OpenCom and DyReS respectively take 1.25 and 1.36 times longer than their respective without-COF baselines, indicating that the overheads of COF are stable across all two implementations. Furthermore, the fact that the with-COF case for AO-OpenCom takes less time than DyReS indicates that COF overheads seem to be well within acceptable ranges.

5 Discussion and Related Work

Threat aspects are not completely orthogonal - in particular, the order in which they are composed is important, and executing aspects at some common join point in a "wrong" order could lead to problems (e.g. situations in which a message needing to be processed by a particular aspect has already been consumed by another). This ordering issue is particularly important for join points at the top of the communication stack (join point 4, 5) at which point numerous aspects are woven; for example, where both the consensus and reliability threat aspects are woven, the reliability aspect should come first to ensure that the consensus protocol uses a reliable communications service. In general, COF mandates a particular order for the weaving of the threat aspects and enforces this order using attributes attached to each aspect.

Few AO middleware platforms have addressed the challenges of performing consistent dynamic reconfiguration. DyMac [1], FAC [13] and CAM/DAOP [2] are component and aspect-based middleware frameworks that take a more principled approach to distribution by offering distributed aspects. They both support distributed aspect composition but no support for consistency and dynamic reconfiguration. Damon [3] is a distributed AO-middleware offering dynamic reconfiguration with remote pointcut and remote advice capabilities similar to AO-OpenCom and DyMac. However, the approach does not provide any consistency mechanisms for use during reconfiguration. Both DJasCo [14] and ReflexD [15] use a consistency protocol to ensure that whenever an aspect is woven at a specific host, mirrors are also woven at other involved hosts. However, they do not consider any other consistency threats as discussed in the threat taxonomy. Lasagne [2] offers semantic consistency support to prevent dangerous combinations of aspects, and offers atomic weaving of aspects. It also checks for unauthorised clients requesting aspect composition. However, it does not offer solutions for operating-environment threats and several other threats.

6 Conclusions and Future work

In this paper we have presented a framework-based approach to consistency maintenance over dynamic reconfiguration operations for AO-middleware platforms. We believe that our threat taxonomy is representative of the type of threats that should be considered by all dynamic AOP platforms. Importantly, COF applies an aspect-oriented approach to consistency management, so the solutions it identifies are described in terms of "threat aspects" and can be applied using the native compositional model of the target AO-middleware platform. Furthermore, the evaluation result show COF: i) ability to handle reconfiguration threats; ii) flexibility of the framework as applied to two AO-middleware platforms; and iii) overheads are acceptable. In future we plan to investigate embedding our approach in a self-managing, autonomic environment in which reconfiguration requests are initiated by the platform itself as opposed to the user.

References

1. Lagaisse, B., Joosen, W.: True and transparent distributed composition of aspect-components. In: Proc. International Conference, Middleware, Springer (2006)
2. Loughran, N., Parlavantzas, N., Colyer, A., Pinto, M., Sánchez, P., Webster, M.: Survey of aspect-oriented middleware. (2005)
3. Mondejar, R., Garcia, P., Pairot, C., Urso, P., Molli, P.: Designing a distributed aop runtime composition model. In: Proc. of ACM Symposium on Applied Computing, ACM (2009)
4. Surajbali, B., Grace, P., Coulson, G.: Ao-opencom: An ao-middleware architecture supporting flexible dynamic reconfiguration. In: 17th ACM Sigsoft Conference on Component-Based Software Engineering, ACM (2014)
5. Janssens, N., Joosen, W., Verbaeten, P.: Necoman: middleware for safe distributed-service adaptation in programmable networks. Distributed Systems Online (2005)
6. Surajbali, B., Grace, P., Coulson, G.: A semantic composition model to preserve (re) configuration consistency in aspect oriented middleware. In: Proceedings of the 8th International Workshop on Adaptive and Reflective Middleware, ACM (2009)
7. Surajbali, B., Grace, P., Coulson, G.: Recycle: Resolving cyclic dependencies in dynamically reconfigurable aspect oriented middleware. (2010)
8. Beloued, A., Gilliot, J.M., Segarra, M.T., André, F.: Dynamic data replication and consistency in mobile environments. In: Proc. Doctoral Symposium on Middleware, ACM (2005)
9. Minson, R., Theodoropoulos, G.: Adaptive support of range queries via push-pull algorithms. In: Principles of Advanced and Distributed Simulation, IEEE (2007)
10. Truyen, E., Janssens, N., Sanen, F., Joosen, W.: Support for distributed adaptations in aspect-oriented middleware. In: In Proc. of the 7th International Conference on AOSD, ACM (2008)
11. Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M.: Report on the development of the advanced encryption standard (aes). Technical report, DTIC Document (2000)
12. Truyen, E., Joosen, W.: Run-time and atomic weaving of distributed aspects. In: Transactions on Aspect-Oriented Software Development II. Springer (2006)
13. Pessemier, N., Seinturier, L., Duchien, L., et al.: A component-based and aspect-oriented model for software evolution. Journal of Computer Applications in Technology (2008)
14. Navarro, L., Benavides, D., Südholt, M., al.: Explicitly distributed aop using awed. In: Proc. 5th International Conference on AOSD, ACM (2006)
15. Tanter, É., Toledo, R.: A versatile kernel for distributed aop. In: Distributed Applications and Interoperable Systems, Springer (2006)