DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS

Ruben Mondejar Andreu

ISBN: 978-84-693-5426-1
Dipòsit Legal: T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Rubén Mondéjar Andreu

DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS

PHD THESIS DISSERTATION

Advised by Dr. Pedro García López
and Dr. Carles Pairot Gavaldà

Department of Computer
Science and Mathematics

Architecture and Telematic
Services Research Group

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Departament d'Enginyeria
Informàtica i Matemàtiques
Avinguda dels Països Catalans, 26
43007 Tarragona
Tel. 977 55 96 79
Fax. 977 55 97 10

Pedro García López, professor titular del Departament d'Enginyeria Informàtica i Matemàtiques de la Universitat Rovira i Virgili,

CERTIFICO:

Que aquest treball, titulat "Distributed AOP Middleware for Large-Scale Scenarios", que presenta Rubén Mondéjar per a l'obtenció del títol de Doctor, ha estat realitzat sota la meva direcció al Departament d'Enginyeria Informàtica i Matemàtiques d'aquesta universitat.

Tarragona, 10 de Febrer de 2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Simplicity, carried to an extreme, becomes elegance.

Jon Franklin

Before software can be reusable it first has to be usable.

Ralph Johnson

Knowing is not enough, we must apply. Willing is not enough, we must do.

Bruce Lee

As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.

Dave Parnas

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Acknowledgements

Deseo mostrar mi más sincero agradecimiento a todas aquellas personas sin las cuales, este trabajo no habría visto la luz:

Mi primer agradecimiento es para Pedro García, quien me inició en el mundo de la investigación. Como director ha sido alguien fundamental, ya que sin su apoyo, consejos, guía, y en general, sin toda esa ayuda y tiempo dedicados, esta tesis no habría sido posible.

En segundo lugar, quiero agradecer, en igual medida a Carles Pairot, co-director de esta tesis, de quien también he aprendido mucho y no dejo de hacerlo diariamente. Su legado y su ininterrumpida ayuda en todas las fases de esta tarea se han convertido en la segunda columna que sostiene esta tesis.

Tampoco quiero olvidarme del resto de integrantes del grupo de investigación AST, especialmente de los que han pasado por el Laboratorio 144 durante todos estos años. Sin duda, su compañerismo y consejo han sido muy importantes.

Furthermore, I would like to acknowledge to all members of the Score Team for the excellent stage that I realized at INRIA-Nancy (France). Particulary, I am very grateful to Pascal Molli for his great help and support, and for the other researchers with whom I had the great pleasure to work with. I have great memories of those months.

Que los familiares, amigos y demás compañeros, que han compartido o visto al menos parte de la gestación de esta tesis, reciban también mi agradecimiento. Dada la suerte de haber contado con tanta gente para apoyarme o darme consejo, me es imposible escribir una lista completa o mencionar a alguien

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

por encima de los demás. Así que, si alguna vez leéis esto, sabed que vuestro nombre también esta aquí.

Mi último agradecimiento y el más pronunciado es para mi novia Rocío, quien sabe mejor que nadie todas las implicaciones y esfuerzo dedicados para realizar y finalizar esta tesis. Su apoyo y ayuda no han cesado en ningún momento y han marcado un antes y un después en mi vida.

Muchas gracias a todos, Thank you all too much,

Rubén Mondéjar Andreu
Febraury 2010

This dissertation has been written during my time at the Laboratory 144 of Architecture and Telematic Services Research Group, and has been supported by a grant from the Universitat Rovira i Virgili in Tarragona (`http://www.urv.cat`).

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Abstract

Large-scale environments in which distributed applications are executed experiment continuous changes. These changes include those resulting from workload variations, resource unavailability, or host failures, among others. Developing wide-area applications that can be aware of such situations may, in many cases, be not cost effective. In contrast, middleware services can be provided to applications, thus permitting a transparent way to overcome such problems.

Many solutions are focused in adding some mechanisms to typical architectures (e.g., web servers). However, these mechanisms are usually ad-hoc solutions (e.g., add-ons) that cannot be applied easily to other versions, or related software. In addition, these changes are not suitable for runtime environments without re-deploying existing applications.

Composition is the technique that helps us to design and implement adaptive software. It can provide static and dynamic software composition to achieve new goals which were not predicted during the design, load-time, or runtime phases. Therefore, modern programming paradigms like component-oriented programming, reflective computing, or separation of concerns appeared to improve adaptive software development.

The separation of concerns principle, for instance, addresses a problem where a number of concerns should be identified and completely separated (without dependencies). Aspect Oriented Programming (AOP) is a modern paradigm that increases modularity by allowing the separation of crosscutting concerns. In addition, dynamic AOP allows less interdependence between the

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

aspects of software architectures in runtime. However, these solutions do not take into account separation of distributed concerns (e.g., load-balancing).

Distributed AOP is a novel and promising paradigm that introduces distributed interception in these scenarios. It defines many new concepts like remote pointcuts, which are similar to traditional remote method calls, since the execution of interception code is performed remotely; component-aspects, which try to merge the component-oriented and aspect-oriented worlds; and aspect group notions. Thus distributed AOP establishes a context where aspects can be deployed in a set of hosts. Nevertheless, as far as we are concerned, there exist no approaches in distributed AOP that fulfill large-scale requirements satisfactorily.

In this setting, the trend seems to focus on decentralization. Examples of decentralization scenarios include peer-to-peer (P2P) networks, which are a serious alternative to traditional client-server systems for some application domains. These models take advantage of the computing at the edge paradigm, where resources available from any computer in the network can be used, and are normally made available to their members.

However, the development of distributed applications in decentralized and large-scale environments has always been a complex task. Developers are usually faced with the same typical technique implementation over and over again, including distribution, location, load-balancing, replication, or caching, just to name a few. It is not practical to address these challenges every time we want to develop an application of this scope. For these reasons, a middleware architecture that provides the necessary abstractions and mechanisms is required to construct distributed applications in these kinds of networks.

In this dissertation, we present a distributed AOP middleware proposal for large-scale development. Our main motivation is to enable **distributed concerns** in a transparent way to applications which were not specifically designed for **large-scale environments**. Our approach benefits from a P2P substrate and a dynamic AOP framework to implement its services in a decentralized, decoupled, and efficient way. It provides a scalable deployment platform where distributed aspects are deployed and activated in individual or grouped hosts. Moreover, we introduce a distributed composition model that envisages sepa-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

ration of distributed concerns, taking the necessary features from component models, like distribution facilities and connectors, and from computational reflection, like introspection and meta-levels. Our composition model is recursive and fully distributed, allowing low dependency and high cohesion among distributed aspects. This model reduces the development complexity and enables interesting services like recomposition at runtime. Finally, we present an implementation prototype of this middleware proposal, called Damon, which has been tested in a real large-scale network.

Damon has a wide applicability due to the distributed interception benefits in large-scale scenarios. This is demonstrated by two main use case approaches. First, we have developed a collaborative wiki application based on structured P2P systems, called *UniWiki*. This distributed application is activated in a group of hosts around the network. A set of distributed aspects enable distribution and replication of wiki pages, and supervision of data consistency under concurrent modifications. Secondly, we have integrated Damon with a web server in order to create an adaptive web platform: *SNAP*. This platform offers distributed deployment and management of web applications, applying several distributed concerns, like dynamic load-balancing, session tracking, or self-activation of applications.

In summary, in this thesis we aim to support distributed concerns in large-scale scenarios using distributed AOP paradigm.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems; D.2.7 [Software Engineering]: Software Architectures; J.8 [Internet Applications]:Middleware

General Terms: Design, Languages, Performance, Experimentation

Keywords: Distributed AOP, Peer-to-Peer, Middleware, Adaptive Software, Separation of Concerns, Composition, Large-Scale Networks

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Resumen

Los entornos de gran escala donde las aplicaciones distribuidas son ejecutadas experimentan continuos cambios en su comportamiento. Estos cambios incluyen los derivados en las variaciones de la carga de trabajo, la indisponibilidad de recursos o fallos en los servidores, entre otros. El desarrollo de aplicaciones de área extensa puede ser consciente de tales situaciones pero, en muchos casos, no resulta rentable. Por el contrario, los servicios middleware son proporcionados a las aplicaciones de forma trasparente, permitiendo superar estos problemas.

Muchas soluciones se centran en la adición de algunos mecanismos a las arquitecturas típicas (e.g., servidores web). Sin embargo, estos mecanismos son generalmente soluciones ad-hoc (e.g., add-ons) que no puede aplicarse fácilmente a otras versiones, o sistemas similares. Además, estos cambios no son adecuados en tiempo de ejecución, dada la necesidad de volver a cargar de nuevo el sistema en estas soluciones.

La composición es la técnica que nos ayuda a diseñar y aplicar software adaptativo. Dicha técnica puede proporcionar composición de software estática y dinámica para alcanzar nuevas metas que no se han previsto durante las fases de diseño, tiempo de carga, o tiempo de ejecución. Por lo tanto, los paradigmas de programación modernos como la programación orientada a componentes, reflexión computacional, o la separación de preocupaciones aparecen para mejorar el desarrollo de la composición de software adaptativo.

El principio de la separación de preocupaciones se basa en solucionar un problema, diferenciando e identificando una serie de preocupaciones que es-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

tán completamente separadas (sin dependencias entre ellas). La programación orientada a aspectos (AOP) es un paradigma moderno que incrementa la modularidad, permitiendo la separación de preocupaciones transversales. Además, el AOP dinámico reduce la interdependencia entre aspectos en las arquitecturas software en tiempo de ejecución. Sin embargo, estas soluciones no tienen en cuenta la separación de preocupaciones distribuidas (e.g., balanceo de carga).

De esta forma el AOP distribuido llega a ser un nuevo y prometedor paradigma que introduce intercepción distribuida en este tipo de escenario. Este define muchos conceptos nuevos como pointcuts remotos, que son similares a las tradicionales llamadas remotas de métodos, ya que está implícita en la ejecución de una máquina remota; componentes-aspectos, que tratan de combinar los mundos orientados a componentes y aspectos, y la noción de grupo de aspectos. Así el AOP distribuido establece un contexto en que los aspectos se despliegan en un conjunto de máquinas. No obstante, por lo que a nosotros se refiere, no existen aproximaciones en AOP distribuido que cumplan satisfactoriamente con las necesidades de los sistemas de gran escala.

En este contexto, generalmente se tiende hacia la descentralización. Ejemplos de estos sistemas descentralizados incluyen el paradigma peer-to-peer (P2P), siendo una seria alternativa a los sistemas tradicionales cliente-servidor en algunos ámbitos de aplicación. Estos modelos se aprovechan del paradigma de la computación en los extremos, donde los recursos disponibles en cualquier máquina de la red están a disposición de todos sus miembros.

Sin embargo, el desarrollo de aplicaciones distribuidas, tanto en entornos descentralizados como de gran escala, siempre es una tarea compleja. Los desarrolladores normalmente se enfrentan con las típicas problemáticas una y otra vez, como pueden ser la distribución, la ubicación, el balanceo de carga, replicación, o el almacenamiento en caché, por nombrar algunos ejemplos. Por lo tanto, no es práctico hacer frente a los mismos retos cada vez que queremos desarrollar aplicaciones en este ámbito. De ahí que se requiera una arquitectura middleware que ofrezca las abstracciones y los mecanismos necesarios para poder construir aplicaciones distribuidas en este tipo de redes.

En la siguiente tesis presentamos una propuesta de middleware de AOP distribuido para el desarrollo de aplicaciones de gran escala. Nuestra prin-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

cipal motivación es permitir que las **preocupaciones distribuidas** puedan integrarse de forma transparente en aplicaciones que no fueron diseñadas especficamente para **entornos de gran escala**. Nuestro enfoque se beneficia de los sustratos de P2P y AOP dinámico para implementar estos servicios de manera descentralizada, desacoplada y eficiente. Esta arquitectura middleware proporciona una plataforma escalable de despliegue donde los aspectos distribuidos han sido introducidos en la red y activados en máquinas individuales o en grupos de máquinas.

Además, introducimos un modelo de composición distribuido que contempla la separación de preocupaciones distribuidas, adoptando las características necesarias de los modelos de componentes, como los mecanismos de distribución y conectores, y de reflexión computacional, como la introspección y los meta-niveles. Nuestro modelo de composición es recursivo y totalmente distribuido, y a su vez permite niveles bajos de dependencia y altos de cohesión entre los aspectos distribuidos. Por otro lado, este modelo reduce la complejidad en el desarrollo de aplicaciones distribuidas gestionando las interacciones, y habilitando interesantes servicios como la recomposición en tiempo de ejecución. Por último, aportaremos un prototipo de la implementación para nuestra propuesta de middleware, llamado *Damon*, el cual ha sido probado en una red real de gran escala.

Damon tiene una amplia aplicabilidad debido a los beneficios de la intercepción distribuida en entornos de gran escala. Esto se demuestra con la propuesta de varias aproximaciones de casos de uso. En primer lugar, hemos desarrollado una aplicación wiki colaborativa, llamada *UniWiki*, basada en sistemas P2P estructurados. Esta aplicación forma un grupo de instancias a lo largo de la red que replican parcialmente las páginas wiki y que mantienen la consistencia de los datos en un entorno con modificaciones concurrentes. En segundo lugar, hemos integrado Damon con un servidor web, con el fin de crear una plataforma web adaptativa: *SNAP*. Esta plataforma ofrece un entorno de despliegue y de gestión de aplicaciones web totalmente distribuido. En ella se aplican varias preocupaciones distribuidas, como el balanceo de carga dinámico, el mantenimiento de sesión, o la activación de aplicaciones automática.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

En conclusión, en esta tesis se pretende dar soporte a las preocupaciones distribuidas en redes de gran escala, haciendo uso del AOP distribuido.

Categorías y Descriptores de Asunto: C.2.4 [Redes de Comunicaciones]: Sistemas Distribuidos; D.2.7 [Ingeniería del Software]: Arquitecturas Software; J.8 [Aplicaciones de Internet]: Middleware

Términos Generales: Diseño, Lenguajes, Rendimiento, Experimentación

Palabras Clave: AOP distribuido, Peer-to-Peer, Middleware, Software adaptativo, Separación de preocupaciones, Composición, Redes de gran escala

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Contents

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Figure List

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Table List

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 1

Introduction

The development of distributed applications in decentralized and large-scale environments has always been a complex task. Developers are usually faced with the same problems over and over again, including distribution, location, load-balancing, replication, or caching, just to name a few. It is not practical to address these challenges every time we want to develop an application of this scope.

For these reasons, a middleware architecture that provides the necessary abstractions and mechanisms is required to construct distributed applications in these kinds of networks.

In order to provide a complete solution, we present a distributed AOP middleware proposal for large-scale development. Our main motivation is to enable **distributed concerns** in a transparent way to applications that were not specifically designed for **large-scale environments**.

Our approach benefits from a P2P substrate and a dynamic AOP framework to implement its services in a decentralized, decoupled, and efficient way. It also provides a scalable deployment platform where distributed aspects are deployed and activated in individual or grouped hosts.

Moreover, we introduce a distributed composition model that envisages separation of distributed concerns, taking the necessary features from component models, like distribution facilities and connectors, and from computational reflection, like introspection and meta-levels. Our composition model is recursive and fully distributed, allowing low dependency and high cohesion among

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

distributed aspects.

Our middleware solution reduces the complexity of distributed application development managing interactions, and enabling interesting services like recomposition at runtime.

In this introduction we first explain the problem statement and goals of this thesis scenario. Subsequently, we set the objectives that a successful approach must fulfill in this research line. Finally, we enumerate our contributions, and we give an overview of this dissertation.

1.1 Problem Statement

Distributed computing studies the coordinated use of physically separated computers. Furthermore, the increase in computing capacity, the reduction of hardware and communication costs, and the massive use of wide-area networks, have been changing the way distributed applications are being developed.

However, the development of distributed applications in large-scale environments has always been a complex task. Such complexity is determined by several factors like distributed application development, deployment or management.

The *client-server* approach is the classic and most used model, because it is the easiest scenario to deal with these issues. There are only two types of different entities: the server, which offers all services, and the client, which uses them. In this model, all services, like message dissemination or data persistence, are offered by the server itself, being responsible to propagate and store information.

When a server is unable to withstand system load, more servers can be added, adopting clustering strategies or server federation techniques, mainly depending on the local or remote machines location. In either case, adoption of these strategies implies an economic overhead, charged to the institution which hosts the servers.

The alternative to the previous model is *decentralization*. The idea is that all components in the distributed system have the same responsibilities acting both as clients and servers. However, this kind of solutions increases the

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

complexity of the system, implying new problems to address.

Nowadays, **large-scale scenarios** already suffer three important challenges of distributed systems:

- **Scalability** [62]: is not easy to establish the criteria that a distributed system must fulfill to be considered scalable. By definition, a scalable system is the one that grows in the numerical and the geographical dimensions:
  - *Numerical dimension* is the capacity of the system to continue to function efficiently when the size of the network is increased as well as the number of elements. A system possesses three main countable elements: the number of hosts, the number of data pieces (e.g., objects) or resources, and the number of services. Thereby, a big number of hosts produce a high amount of communication, and it thus affects the load on specific zones of the network. For these reasons, we need to take some measures to avoid bottlenecks, promoting decentralization.
  - *Geographical dimension* is the ability to perform efficiently communication tasks in wide-area networks. However, one of the reasons why it is currently hard to scale existing distributed systems is because they are based on one-to-one and synchronous communication.

- **Availability** [12]: refers to the fact that access to any resource must be guaranteed at all times. In traditional client-server architectures, the majority of resources for an application execution are managed and hosted on the main server. However, the idea of availability usually comes in the form of resource replication.

  So redundancy guarantees an improvement of availability. This idea is not new, since several already existent applications have exploited this goal (e.g., SETI@Home [7]). Moreover, we consider this like a requirement that must be fulfilled by a distributed system, since we cannot rely on resources located in a single remote server.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- **Transparency** [90]: means that any form of distributed system should hide its distributed nature from its users, appearing and functioning as a typical centralized system. This property can be studied in may ways, like location, access, persistence, or replication transparency.

  Location transparency is when any form of distributed system should hide its distributed nature from its users, appearing and functioning as a local system. Regardless of how resource access and representation has to be performed on each individual computing entity, with *access transparency* the distributed system users should always access resources in a single, uniform way.

  Persistence transparency is successful when the user is unaware of the final resource storage (e.g., volatile memory or disk device). This kind of transparency also enables to store data in different systems like simple tables, or complex data store (e.g., databases) or in different serialized formats (e.g., XML). In *replication transparency* the different copies of a resource should appear to the user as a unique single resource. The system is the responsible to handle these copies, updating them on modification operations, and controlling the uniqueness of resources, and the concurrent accesses.

  Finally, although full transparency is an important goal, it is hard to achieve due to the inherent problems of distributed systems. As an example, there will always be more latency on accessing remote resource than local resources.

As a matter of fact, a distributed system for large-scale scenarios should be scalable numerically and geographically, it should make its resources (e.g., data or services) easily accessible, and it should hide its distributed nature.

1.2 Goals

Indeed, it is not trivial to develop distributed applications on top of a large-scale network, since no middleware infrastructure is available. Specially, when developers need to re-implement common mechanisms over and over again,

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

thus wasting precious development time, which could be dedicated to other tasks. In this dissertation, we aim to propose a middleware architecture suitable and flexible enough to allow large-scale application development.

In this line, the goals of this dissertation are to facilitate the creation and development of distributed applications in large-scale scenarios. To achieve this, we require a middleware approach that abstracts all common services needed by developers, so that implementing a distributed application on top of a large-scale substrate is as easy as possible.

Therefore, in this dissertation we plan to achieve the following goals:

- Definition of a layered architecture that enables the modularization of distributed concerns on large-scale scenarios during the application development phases. Thus, this architecture should simplify the design and implementation of applications, as well as provide a uniform access to middleware services in a transparent way.

- Definition of a development model for distributed concerns completely abstracted from the complexities of the underlying layers. In addition, this model should allow the necessary mechanisms like discovery, location, deployment, or activation in the middleware proposal.

- Implementation of the proposed generic model by two main layers that include a *distributed composition model* and a *scalable deployment platform* for large-scale distributed concerns.

- Finally, our ultimate goal will be to demonstrate the viability and applicability of our proposal. Consequently, we will validate the platform with novel distributed applications (i.e., proofs of concept) in large-scale scenarios. As a consequence, these applications will have scalability, availability, and transparency properties.

1.3 Contributions

On the basis of the goals listed above, we plan an approach which defines a complete *middleware architecture* proposal, abstracting the underlying layer

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

complexities, and even allowing the distributed concerns encapsulation, deployment, and composition. Furthermore, we outline the next contributions:

- The distributed composition model allows local and remote interactions among distributed concerns. The innovative contributions of this model are :
  - **Our first contribution** is the *encapsulation* of *distribution concerns* from distributed applications in completely separated and modulated true distributed entities. Providing a detailed architecture description language and grammar that clarifies entity definition and connections. The descriptor also allows the definition of a recursive entity, and promotes third party development.
  - **Our second contribution** is the definition and implementation of a *distributed meta-level* model for distributed concerns. This model enables a distributed meta-concern entity and its meta-level connections. These meta-connections are able to intercept the local and remote interactions among running distributed concerns. Thus, its contribution allows a new level of transparency in our middleware.
  - **Our third contribution** is the *event-based* nature of our middleware that allows *runtime reconfiguration* capabilities, through its decoupled connection model, and reflection techniques. Therefore, this contribution enables dynamic composition of entities and/or meta-entities, and to change their connections without stopping the system execution.
- Although the composition model is generic, and can be applied to other scenarios (e.g., mobile networks [31]) in this dissertation we are fully focused on the large-scale arena. Therefore, we need a deployment platform layer that provides the necessary abstractions and services for the upper layer, and satisfying the requirements of a large-scale distributed system (Section 1.1) :
  - **Our fourth contribution** is the *decentralized container* that offers location and discovery services, and provides the distributed con-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

cern life cycle. *Each host of the system owns a part of the container.* This location service allows distributed concerns to be located and inserted into our distributed generic platform. This decentralized approach is similar to traditional container systems, but it extends them with availability and scalability properties.

- **Our fifth contribution** is a set of *decentralized functionalities and abstractions* that are based on asynchronous and synchronous communications mechanisms for one-to-one and one-to-many (groups) scenarios. Moreover, the abstractions provide new ways and more flexible manners to perform distributed concerns functionalities.

- **Our final contribution** is the demonstration of the *viability* and *applicability* of our approach.

  - An implementation prototype, which follows the model that we have defined in this work. Moreover this prototype is validated via experimentation in a real large-scale network.
  - Several distributed concerns in different proof-of-concepts. Concretely, we present two proof-of-concepts, a collaborative application (i.e., wiki) and a web deployment platform that benefits directly from our proposal to integrate new distributed concerns.

1.4 Thesis Structure

The structure of this dissertation is summarized as follows:

- **Chapter 2. Background.** This chapter presents the background on the two main areas that a distributed middleware requires: an adaptive and reflective substrate and a large-scale network infrastructure. In addition, we focus our research on implicit middleware, focusing on distributed interception techniques.

- **Chapter 3. Distributed AOP Middleware for Large-Scale Scenarios**. In this chapter, we describe our middleware proposal to develop

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

distributed applications in large-scale environments. We outline the viability of our proposal by presenting our practical implementation in the form of a prototype (Damon). To conclude this chapter, we include some experimentation results to verify the effectiveness of our proposal.

- **Chapter 4. Building a Scalable Collaborative Wiki Application**. In this chapter, we introduce the first proof-of-concept scenario that uses our generic model features. Concretely, we present a distributed collaborative application (wiki) that uses three distributed concerns (distribution, replication, and consistency) and it is suitable for large-scale scenarios.

- **Chapter 5. Enabling Web Applications over Wide-Area Networks**. Subsequently, in this chapter we present the second proof-of-concept for our middleware proposal. In this case, we present an adaptive large-scale web system that is composed by a web server and the necessary distributed concerns, like load-balancing and session tracking.

- **Chapter 6: Conclusions**. This chapter presents the conclusions that ensue from this thesis. We finish the chapter by describing some other prospective future uses for our large-scale middleware proposal.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 2

Background

In this chapter we will be studying the state of the art in some different disciplines that can help us to achieve the goals of this dissertation. As we shall explain below, we can differentiate three important properties in distributed systems: scalability, availability, and transparency.

For this reason, we focus our study in three related research lines : *distributed middleware* related to scalability, *adaptive middleware* for availability, and *implicit middleware* for transparency.

Therefore, in the following sections we will introduce the background of each research line.

2.1 Distributed Middleware

Middleware sits above the network layer and below the application layer and abstracts the heterogeneity and complexity of the underlying environment. It provides an integrated distributed environment whose objective is to simplify the task of implementing distributed systems, and also to provide value-added services such as location or persistence to enable distributed application development. Middleware is about integration and interoperability of application and services running on heterogeneous computing and communication devices.

Distributed middleware is a mature technology for developing distributed applications. Its popular acceptance in distributed settings has led to a consid-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

erable level of sophistication and support. In order to build scalable systems we explore two kind of distributed middleware in this section:

- Peer-to-Peer Middleware : to solve scalability problems (e.g., bottlenecks) of our middleware platform.
- Event-Based Middleware : to manage asynchronous communication and decoupled architectures.

We start explaining the P2P networks, and later we will focus on event-based systems, specially on the P2P based.

2.1.1 Peer-to-Peer Networks

Any communication substrate which is intended to be used for large-scale scenarios needs to be responsible for routing messages between network nodes in an efficient and fault tolerant way. It is important that this routing substrate is as autonomous as possible so as it can handle node failures, arrivals, departures and other exceptional events in a transparent way to the upper layers.

Nowadays, the way Internet applications tend to be organized is in a relatively small number of powerful servers, which provide service to many client nodes. In fact, this is the standard operation way of the World Wide Web (WWW): a centralized client-server architecture. Once the application hosting server becomes overwhelmed with requests from many clients, it clearly becomes a bottleneck. Moreover, if such server crashes, the application becomes unusable, unless redundant servers take care of this unavailability issue. Therefore, the centralized architecture seems to be a non-adequate alternative for low-cost large-scale fault tolerant massive application accessibility.

Some large-scale successful applications (e.g., eMule [88]) which support high numbers of concurrent connected users have advocated the use of peer-to-peer (P2P) technologies to solve the scalability problem. These applications use the decentralization paradigm in order to avoid bottlenecks. Consequently, there is not only one unique server holding all application data, but a bunch of nodes which support the application working together. If a service node goes

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

down, another one can take its place and continue serving requests. The same happens when trying to load balance requests: if there exist more servers than just one for serving these requests, load can be balanced throughout all available servers. This approximation driven to the extreme is the P2P philosophy: all nodes are treated as equal peers.

However, P2P architectures tend to be non-reliable. This non-reliability comes from the fact that there can be constant joins and leaves, and that resources have to be relocated on the fly. The wide diversity of node capacities, operating systems and system architectures which conform the network, give to P2P this heterogeneity factor. In order to support these particular features, P2P networks must be self organizing and self repairing, as well as fault tolerant. Their typical objective is to make good use of the shared distributed resources (e.g., CPU time, bandwidth, storage capacity, etc.) among all nodes.

P2P networks can be classified in a wide variety of ways. In this line, one of the principal challenges of such systems is how to locate any particular resource. Since this can be a very complex problem, several approaches have been taken to overcome it. They are, in chronological order, the central index location scheme (e.g., Napster [58]), and the unstructured location scheme, also known as unstructured P2P networks (e.g., Gnutella [2, 77]). However, resource location is non-deterministic, because a resource could not be found efficiently although it is in the network. Lastly, the key-based routing [89] or distributed hash table schemes [48], based on structured P2P networks, appear.

2.1.1.1 Structured Peer-to-Peer

The premise of this kind of networks is : *if a specific resource is into the network, it should be found in a determined number of hops.* For this purpose these networks start becoming structured node groupings. Nodes are arranged in a structured fashion, typically following ring [89] o tree [80] formations. The objective is to assign particular nodes to store particular content. When a node looks for a resource, it must be redirected to the node which is supposed to hold it.

The challenges of such structured P2P networks are as follows:

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- Bottleneck avoidance in particular nodes, thus distributing responsibilities *evenly* among the existing peers.

- Adaptation to nodes joining or leaving (or failing). As a consequence, it is logical to give new responsibilities to joining nodes, and redistribute responsibilities from leaving nodes.

These challenges match perfectly the idea of a Distributed HashTable (DHT), where the key is hashed to find the resource responsible peer node, obtaining data and load balancing across nodes (see Figure 2.1). In traditional hash tables, each data item is associated with a key. The key is hashed to find its corresponding bucket in the hash table. Each bucket is expected to hold *#items/#buckets* items. In order to map such data structure to the distributed problem, it is considered that nodes are the buckets in the global Distributed Hashtable.

As a consequence, we can define Distributed Hashtables as a class of decentralized distributed systems that partition ownership of a set of keys among participating nodes, and can efficiently route messages to the unique owner of any given key. Each node is analogous to a bucket in a hash table. DHTs are typically designed to scale to large numbers of nodes and to handle continual node arrivals and failures. This infrastructure can be used to build more complex services, such as distributed file systems, P2P file sharing systems, cooperative web caching, multicast, anycast, and domain name services.

Even though this approach seems to solve the problems introduced with both central index and unstructured P2P network schemes, it also brings several issues to be taken care of, dynamicity and size.

Dynamicity: when we use a hash function, virtually every key will change its location whenever a node is added or removed. In order to solve this problem, a method called **consistent hashing** [42], and adopted by the Chord [89] routing algorithm, is currently used by the major DHT designers. Consistent hashing implies defining a fixed hash space where all hash values fall within, and do not depend on the number of peers. As a consequence, each key falls into the peer closest to its ID in hash space, according to some proximity metric. Such concept is further detailed when the Chord routing algorithm is

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

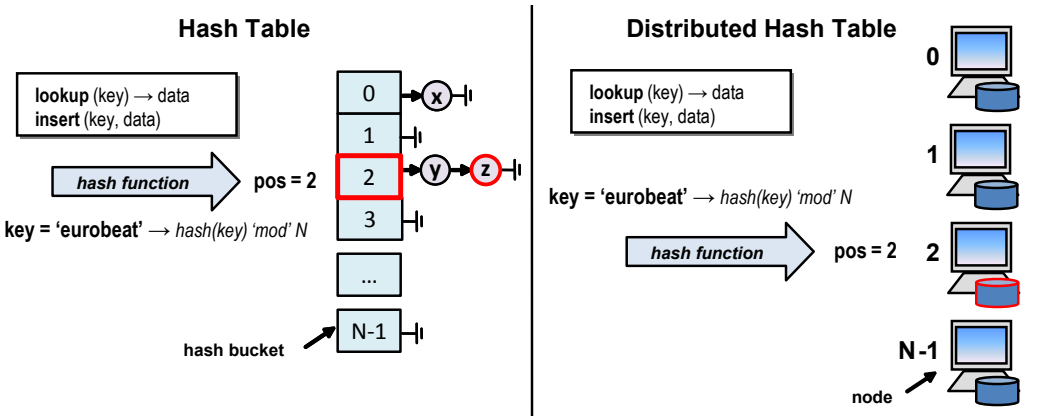

Figure 2.1: Distributed Hashtable Abstraction. In a normal hash table, hash buckets are stored in local memory. However, in a DHT, hash buckets correspond to network physical nodes, and (key,value) pairs are stored on them.

explained in Section 2.1.1.2.

Size: all nodes must be known to insert or lookup data. Such approach works well with small and static server populations. Nevertheless, when talking about wide-scale P2P networks, it is impossible to assume that every single node is to be connected to all others, since the maintenance overhead would kill the entire network. The only possible solution is to allow each peer to know only a few neighbours. Messages are therefore routed through neighbours via multiple hops, using an **overlay routing** scheme.

When designing an efficient DHT, hosts configure themselves into a structured network such that mapping table lookups require a small number of hops. The DHT abstraction provides a minimal access interface, which is mainly data-centric. It naturally supports a wide range of applications, because it imposes very few restrictions: keys have no semantic meaning, and values are application dependent. Therefore, DHTs can be used as a decentralized data insertion and location facility.

It is important to note that DHTs provide the mechanisms to insert and locate data in a decentralized fashion, by using its principal programming interface: **put (key, value)** and **get (key) → value**.

This kind of structured P2P overlay networks are often called Key Based Routing (KBR) substrates, since message routing depends upon node identi-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

fiers. The relatively new structured P2P protocols appeared during the last years seem to provide a solid base for supporting many P2P future developments. This is the reason why we consider structured P2P key-based routing substrates a very interesting alternative for being the basis for our proposed generic model. These substrates provide neat features like self-organization, self-healing, fault tolerance, efficient message routing, and many others, thus fulfilling some of the requirements we had in mind: scalability, dynamicity, fault tolerance, etc. There exist many systems which adopted such scheme, as for example Chord [89] or Pastry [80]. Such protocols are described in the following sections.

2.1.1.2 Chord

Chord [89] is a KBR substrate approach. The Chord protocol specifies how to find the locations of keys, how new nodes join the system, and how to recover from the failure (or planned departure) of existing nodes. At its heart, Chord provides fast distributed computation of a hash function, and mapping keys to nodes responsible for them. It uses *consistent hashing* [42], for assigning *key, value* pairs to their hash buckets, which correspond to physical nodes.

With high probability the hash function balances load (all nodes receive roughly the same number of keys). Also with high probability, when an N^th node joins (or leaves) the network, only an *O(1/N)* fraction of the keys are moved to a different location : this is clearly the minimum necessary to maintain a balanced load. Chord improves the scalability of consistent hashing by avoiding the requirement that every node knows about every other node. A Chord node needs only a small amount of *routing* information about other nodes. Because this information is distributed, a node resolves the hash function by communicating with a few other nodes. In an N-node network, each node maintains information only about $O(\log\ N)$ other nodes, and a lookup requires $O(\log\ N)$ messages. Chord must update the routing information when a node joins or leaves the network; a join or leave requires $O(\log^2\ N)$ messages.

The *consistent hash* function assigns each node and key an m-bit *identifier* using a base hash function such as *Secure Hash Algorithm 1* (*SHA-1*). A node identifier is chosen by hashing the node IP address, while a key identifier is

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

produced by hashing the key. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible.

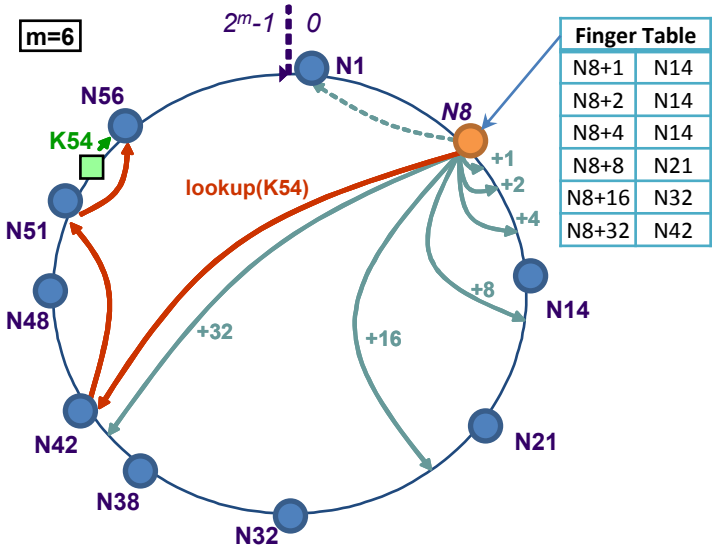

Figure 2.2: A Chord ring consisting many nodes. Notice how the finger table is organized and how K54 is looked up following Chord algorithm.

Consistent hashing assigns keys to nodes as follows. Identifiers are ordered in an *identifier circle* modulo 2^m. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the *successor node* of key k, denoted by *successor(k)*. If identifiers are represented as a circle of numbers from 0 to 2^m - 1, then *successor(k)* is the first node clockwise from k. Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node n joins the network, certain keys previously assigned to its successor now become assigned to n. When node n leaves the network, all of its assigned keys are reassigned to its successor.

Each node maintains information about only a small subset of the nodes in the system in its routing table, called *finger table*. The search for a node moves progressively closer to identifying the successor with each step. A search for the successor of f initiated at node r begins by determining if f is between r and the immediate successor of r. If so, the search terminates and the successor

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

of r is returned. Otherwise, r forwards the search request to the largest node in its finger table that precedes f; call this node s. The same procedure is repeated by s until the search terminates.

Chord includes this procedure in a simple form and known as stabilization protocol. This protocol is fault resilient, self-organizing and self-healing, and with an acceptable performance even in the face of concurrent node arrivals and departures. Nevertheless, this stabilization protocol simplicity is also one of its biggest problems, since it involves too much communication between nodes.

2.1.1.3 Pastry

Pastry [80] is a structured P2P network routing substrate that is defined as a self-organizing overlay network of nodes, where each node routes client requests and interacts with local instances of one or more applications.

Each node in the Pastry P2P overlay network is assigned a 128-bit node identifier (*nodeId*). The nodeId is used to indicate a node position in a circular nodeId space, which ranges from 0 to 2^{128} - 1. The nodeId is assigned randomly when a node joins the system. It is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit nodeId space. For instance, nodeIds could be generated by computing a cryptographic hash of the node public key or its IP address. As a result of this random assignment of nodeIds, with high probability, nodes with adjacent nodeIds are diverse in geography, ownership, jurisdiction, network attachment, etc.

Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\log_2 bN$ steps under normal operation (b is a configuration parameter with typical value of 4). Despite concurrent node failures, eventual delivery is guaranteed unless $|L|/2$ nodes with *adjacent nodeId* fail simultaneously ($|L|$ is a configuration parameter with a typical value of 16 or 32). Therefore, Pastry routes to any node in the overlay network in $O(\log\ N)$ steps in the absence of node failures, and it maintains routing tables with $O(\log\ N)$ entries.

For the purpose of routing, nodeIds and keys are thought of as a sequence of digits with base 2^b. Pastry routes messages to the node whose nodeId is

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

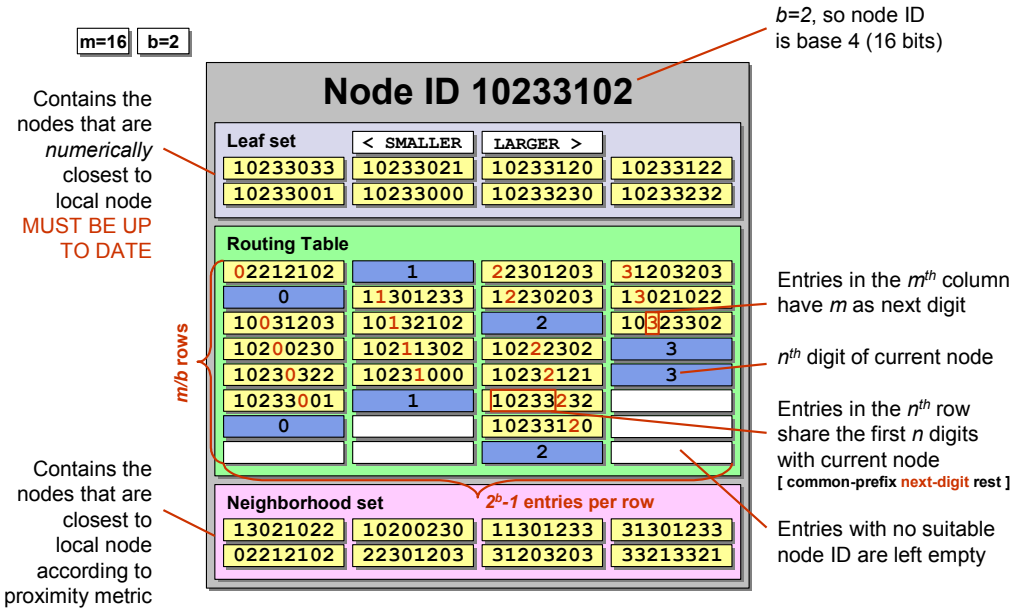

Figure 2.3: State of a hypothetical Pastry node. With nodeId 10233102, b = 2. All numbers are in base 4. The top row of the routing table is row zero. The shaded cell in each row of the routing table shows the corresponding digit of the present node nodeId. The nodeIds in each entry have been split to show the common prefix with 10233102 - next digit - rest of nodeId. The associated IP addresses are not shown.

numerically closest to the given key. This is accomplished as follows. In each routing step, one node normally forwards the message to another node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node id. To support this routing procedure, each node maintains a *routing table*, a *neighborhood set* and a *leaf set*.

One important feature about Pastry is its **locality awareness**. This feature guarantees that the route chosen for a message is based on the proximity metric. Pastry notion of network proximity is based on a scalar proximity metric, such as the number of IP routing hops or geographic distance. It is assumed that the application provides a function that allows each Pastry node to determine the *distance* of a node with a given IP address to itself. A node

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

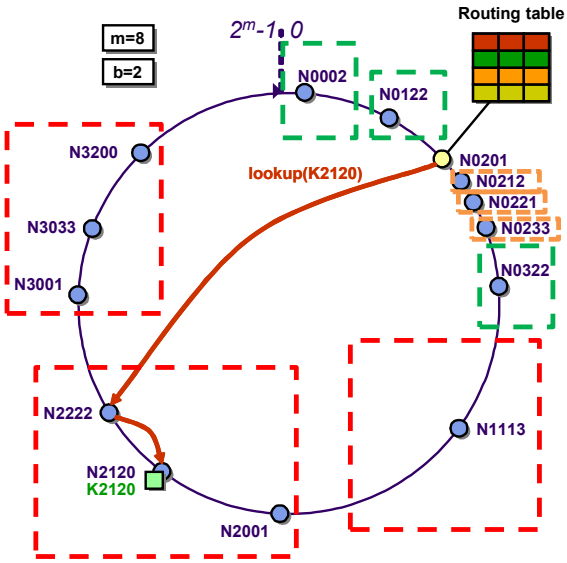

Figure 2.4: Pastry State and Lookup. For each prefix, a node knows some other node (if any) with the same prefix and different next digit. When multiple nodes are available, the topologically-closest is chosen, thus maintaining good locality properties.

with a lower distance value is assumed to be more desirable. An application is expected to implement this function depending on its choice of proximity metric, using network services like traceroute or Internet subnet maps, and appropriate caching and approximation techniques to minimize overhead.

The original version of FreePastry (i.e. Pastry implementation) was shipped with a minimal API which allowed programming of several applications like PAST [22] and Scribe [17]. This API was extended to support the Common API for structured P2P overlay networks [20].

2.1.1.4 Common API

In this scenario, the complexity of P2P applications development and interoperation was increased, since several structured P2P systems were designed, and many applications was implemented on top of them. To use these P2P applications on top of different structured overlays without changing their implementations, a Common API [20] was proposed.

The Common API for structured P2P overlays attempts to identify the

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

fundamental abstractions provided by structured overlays and to define APIs for the common services they provide. As the first step, a *Key-Based Routing* (KBR) *API* is defined, which represents basic (tier 0) capabilities that are common to all structured overlays. The KBR is easily implemented by existing overlay protocols and makes possible to efficiently implement higher level services and a wide range of applications. Thus, the KBR is the common denominator of services provided by existing structured overlays. In addition, a number of higher level (tier 1) abstractions are identified and it is shown how they can be built upon the basic KBR. These abstractions include *Distributed HashTable* (DHT), *group anycast* and *multicast* (CAST), and *Decentralized Object Location* and *Routing* (DOLR).

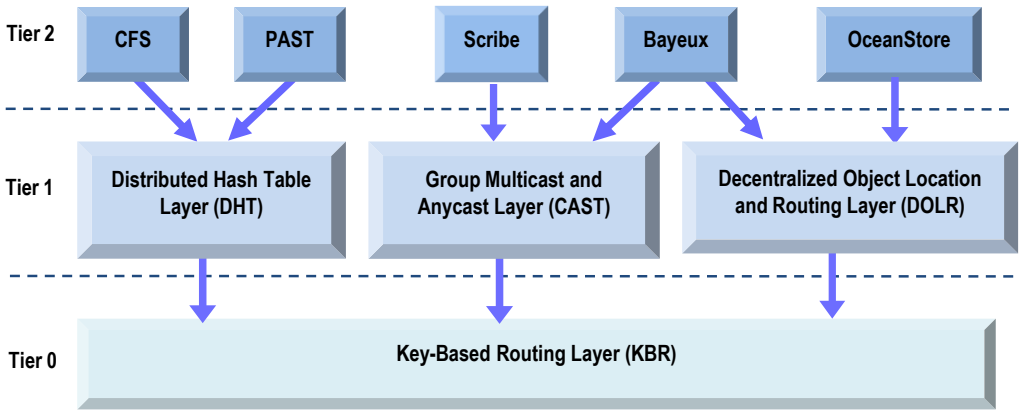

Figure 2.5: Common API Diagram Basic abstractions and APIs, including Tier 1 interfaces: distributed hash tables (DHT), group anycast and multicast (CAST), and decentralized object location and routing (DOLR).

Figure 2.5 illustrates how these abstractions are related. Key-Based Routing is the common service provided by all systems at tier 0. At tier 1, we have higher level abstractions provided by some of the existing systems. Most applications and higher-level (tier 2) services use one or more of these abstractions.

The **DHT** abstraction provides the same functionality as a traditional hash table, by storing the mapping between a key and a value. This interface implements a simple store and retrieve functionality, where the value is always

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Method signature	Description
void route (key: K, msg: M, NodeHandle hint)	forwards a message M, towards the root of key K. The optional hint argument specifies a node that should be used as a first hop in routing the message.
void forward (key: K, msg: M, NodeHandle nextHopNode)	is invoked at each node that forwards message M, including the source node, and the key root node. This call informs the application that message M with key K is to be forwarded to nextHopNode.
void deliver (key: K, msg:M)	is invoked on the node that is the root for key K upon the arrival of message M.
NodeHandle[] local_lookup (key: K, int: num, boolean:safe)	produces a list of nodes that can be used as next hops on a route towards, key K such that the resulting route satisfies the overlay protocol bounds on the number of hops taken.
NodeHandle[] neighborSet (int: num)	produces an unordered list of neighbours (nodehandles) of the local node in the ID space. Up to num nodehandles are returned.
NodeHandle[] replicaSet (key: K, int: max_rank)	returns an unordered set of nodehandles on which replicas of the object with key K can be stored.
void update (NodeHandle: n, boolean: joined)	is invoked to inform the application that node n has either joined or left the neighbour set of the local node as that set would be returned by the neighborSet call.
boolean range (NodeHandle: N, rank: r, key: lkey, key: rkey)	provides information about ranges of keys for which node N is currently a r-root.

Table 2.1: Brief description of KBR interface methods

stored at the live overlay node(s) to which the key is mapped by the KBR layer. Values can be objects of any type.

The **CAST** abstraction provides scalable group communication and coordination. Overlay nodes may join and leave a group, multicast messages to the group, or anycast a message to a member of the group. Because the group is represented as a tree, membership management is decentralized. Thus, CAST can support large and highly dynamic groups. Moreover, if the overlay that provides the KBR service is proximity aware, then multicast is efficient and anycast messages are delivered to a group member near the anycast originator.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

The **DOLR** abstraction provides a decentralized directory service. Each object replica (or endpoint) has an *objectID* and may be placed anywhere within the system. Applications announce the presence of endpoints by *publishing* their locations. A client message addressed with a particular objectID will be delivered to a nearby endpoint with this name.

DHT	DOLR	CAST
put (key, data)	publish (objectId)	join (groupId)
remove (key)	unpublish (objectId)	leave (groupId)
value = get (key)	sendToObj (msg, objectId, [n])	multicast (msg, groupId)
		anycast (msg, groupId)

Table 2.2: Summary of all DHT DOLR CAST interface methods

The Table 2.2 shows the Common API Tier 1 API All services defined at tier 1 require interfacing with the lower key-based routing API layer (tier 0), which is the core all structured overlay network implementations must provide.

As we can clearly observe, the Common API provides the upper levels with three interaction layers which perfectly fit into the layers we have defined throughout this section: a large-scale routing layer (KBR), an application-level multicast layer (CAST), and an object persistence layer (DHT).

2.1.2 Event-Based Systems

In event-based middleware architectures, applications essentially communicate through the propagation of events, which have some data. The main advantage of event-based systems is that applications are inherent loosely coupled. In this way, what makes these middleware solutions important is the distribution transparency degree that they provide.

Then, event dissemination has typically been associated with the publish/subscribe systems [26]. In this kind of systems, subscribers (or consumers) express their interest in a specified content by subscribing to it. From the moment of the subscription, they will start receiving events from publishers (or producers) on the content.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

2.1.2.1 Publish/Subscribe

Nowadays, the architecture of distributed systems is dominated by client/server platforms relying on synchronous request/reply. This architecture is not well suited to implement event-based applications (e.g., dissemination of auction bids) due to the coupled nature of synchronous communication.

In contrast, publish/subscribe performs the intrinsic behaviour of event-based applications, where asynchronous communication is indirect and initiated by producers of events (i.e., publish notifications) and these are delivered to subscribed consumers by the support of a notification service (e.g., JMS [40]).

In this line, subscribers express interest in one or more kind of messages, receiving only the messages that are of interest, without knowledge of publishers. This abstraction between publishers and subscribers allows dynamic, decoupled, and scalable network infrastructures.

A publish/subscribe event system can be classified by the different ways of specifying how to subscribe to and publish particular content:

- *Topic-based*: Participants publish notifications and subscribe to specific subjects, which are represented by keywords.
- *Type-based*: The name-based topic classification scheme is replaced by other filtering events according to their type. This enables the language and the middleware to be more closely integrated.
- *Content-based*: A subscription scheme based on the properties of the notifications is used. In other words, events are not classified according to some pre-defined external criterion (e.g., topic name), but according to properties of the events themselves.

Content-based system is the most expressive one because it allows to evaluate filter predicates over the whole content of a notification. This advantage compared to the other mechanisms results in increased flexibility facilitating extensibility and change.

Moreover, this kind of middleware provides a clear benefit : *decoupling*. Publishers are loosely coupled to subscribers, and need not even know of their

existence. With the topic being the focus, publishers and subscribers are allowed to remain ignorant of system topology. Each can continue to operate normally regardless of the other. In the traditional tightly-coupled client-server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running.

In this setting, the synergy of publish/subscribe systems and P2P networks starts with approaches like Bayeux [105] and Scribe [17]. These topic-based event-systems are built like a CAST layer over a KBR substrate (see Section 2.1.1.4).

Finally, content-based P2P-based systems [3] imply more complexity, being a current research area.

2.1.2.2 Scribe

Scribe is a decentralized event system that is built on top of the Common API, and uses the Pastry KBR substrate for its underlying route management and host lookup. Moreover, it can be considered a topic-based publish/subscribe system. Clients create topics to which other clients can subscribe. Once the topic has been created, the owner of the topic can publish new entries under the topic which will be distributed in a multicast tree to all of the Scribe nodes that have subscribed to the topic.

The system works by computing the hash of the topic name, and it is used as a Pastry key, and the publisher then routes packets to the node closest to the key using Pastry routing protocol to create the root node of the topic on that node. Clients then subscribe to the topic by computing the key from the topic and publisher name and then using Pastry to route a subscribe message to the topic towards the root node. When the root node receives the subscribe message from another node it adds the node ID to its list of children and begins acting as a forwarder of the topic.

Decentralization is accomplished through having all nodes in the network snoop on subscribe messages going past them on their way to the topics root node. If the topic is one to which the current node subscribes, it will stop forwarding the packet toward the root node and add the node trying to subscribe

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

as one of its children. In this way a treelike structure is formed with the root node at the top sending out to the first few subscriber nodes, and then each of these nodes forwarding the messages on to their children, and so on.

Because packets from random nodes on the Pastry network destined for the same node often end up traveling along the same path very soon in their journey, they end up attaching to whatever part of the tree is nearest to them in the Pastry network. Since each hop along a Pastry route represents what is locally the best route according to the routing metric in use, the subscribe message seeks out the closest portion of the tree and attaches itself there.

2.2 Adaptive Middleware

An adaptive system has the ability to change its behaviour and functionality. Adaptive middleware is software whose functional behaviour can be modified dynamically to improve its performance depending on the scenario conditions or requirements. The primary requirements of an adaptive system [81] are measurement, reporting, control, feedback, and stability.

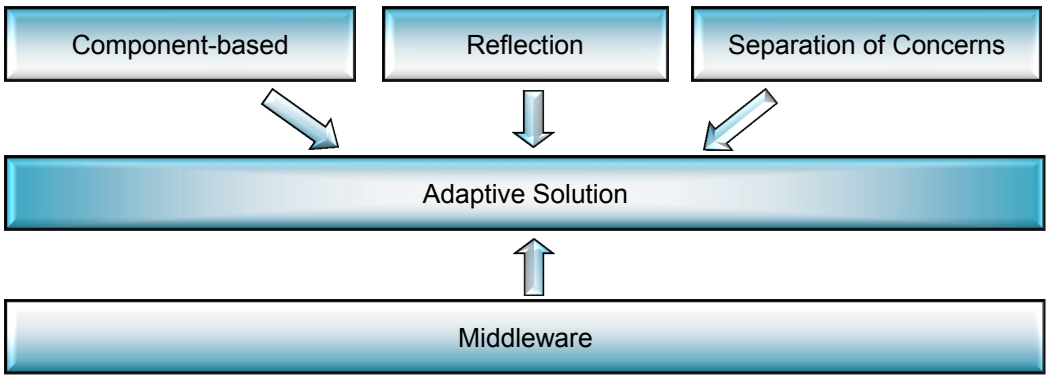

Figure 2.6: Adaptive Solutions Diagram.

In the distributed domain, we may suffer unpredictable situations like workload variations, host failures, and resource unavailability, among others. Adaptive software architectures seem to be a good solution to address these problems. However, there is no complete solution in this area. In summary, the advances in programming paradigms [67, 51, 45, 30] have also contributed to

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

the design of adaptive middleware, complementary to the foundation provided by the design and implementation of traditional middleware solutions.

In [53] the authors distinguish three basic techniques for software adaptations (Figure 2.6): computational reflection [15], component-based models [25], and Aspect-Oriented Programming (AOP) [44]). These solutions have been used during years, and they have their own advantages and disadvantages (listed in Table 2.3).

Although many important contributions have been made in this area, these three paradigms play key roles in supporting adaptive middleware. Each is discussed as follows.

Solutions	Advantages	Disadvantages
Computational Reflection	has the ability to inspect itself, and adapt its behaviour.	has not yet proved to be able to manage the complexity of large-scale distributed systems.
	meta-level concept allows separation of adaptations at different levels.	applying reflection to a broad domain of applications is yet to be done.
Component-based Models	support adaptation through composition techniques.	components are less independent (cohesion) than we can expect initially.
	systems may either be configured statically at design time, or dynamically, at load and/or runtime.	solutions that are constructed with component frameworks, are not fully transparent and can be intrusive.
Separation of Concerns (AOP)	separate system or application code in crosscutting concerns.	is a novel paradigm that needs more research to evolve in this area.
	is able to perform powerful interception mechanisms.	has not been successfully applied in large-scale system development yet.

Table 2.3: Summary of advantages and disadvantages of adaptive solutions

2.2.1 Computational Reflection

Reflection [15] is the capability of a system to reason about itself, act upon this information, and adjust to changing conditions. A common definition of reflection is a system that provides a representation of its own behaviour

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

that is suitable to inspection and adaptation and is causally connected to the underlying behaviour it describes [51].

In a reflective architecture, a computational system has two sides : the object model side, and the reflective side. The aim of object-oriented programming is to solve problems, and return information about an external domain, while the reflective part returns information about the system itself.

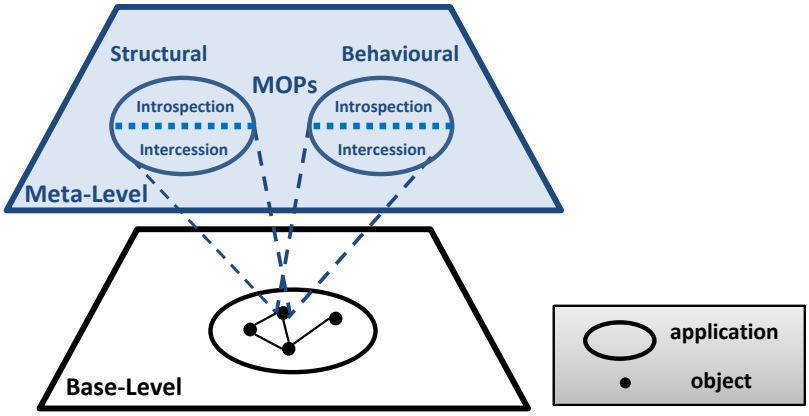

Figure 2.7: Relationship between base-level and meta-level objects.

In [45] work, Kiczales et. al. combine the reflection and object-oriented paradigms in the form of a meta-object protocol (MOP). One innovative notion of this work was the separation of the system into two levels : base-level and meta-level [52].

This concept can be extrapolated to other entities. Then, the entities that deal with the self-representation and the application reside at these two different software levels: the meta-level and the base-level, respectively. Entities that deal with the functionality of the application are at the base-level. Similarly, entities that deal with the application self-representation reside in the meta-level. Both levels are related in such a way that changes at the base level are reflected at the meta-level, in a causal connection way. The meta-level has access to the information at the base-level, but the base level does not have any knowledge about the meta-level.

In middleware platforms, two styles of reflection have emerged. Structural reflection is concerned with the underlying structure of objects or components, therefore, it is possible to inspect interface information, and adapt software

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

architecture topology. Behavioural reflection is concerned with activity in the underlying system, as for example, in terms of the arrival and dispatching of invocations.

Research in the reflective arena is close related to middleware area. We also note that several reflective middleware solutions [14, 10, 92] have been proposed to support development of distributed systems and reflective middleware. As we can observe, these works use reflection within middleware to give developers the way to resolve the challenges of adaptive middleware.

2.2.2 Software Components

Software components are software units that can be independently implemented by third parties. Components are self-contained: components clearly specify what they require and what they provide.

Component models [25] extend the object oriented paradigm by adding new abstractions and concepts that express composition relationships between system components. A component oriented environment emphasizes the definition of standard interfaces which indicate how their components must be used. These interfaces define the component as a collection of methods invoked whenever a service is required.

The use of components is based on the plug-and-play concept: that is, we can connect a component as a part of an application without needing to change it for it to start working. This idea applies to many commercial products, and eases the building of configurable applications whose functionalities depend on their aggregated components. Normally, component-based software is built on top of frameworks, which provide the life cycle services required by components. These frameworks may also manage component activation and passivation, persistence, naming, etc.

One of the most popular approaches is Component-based software development (CBSD), which tries to settle the basis for design and development of reusable software component-based distributed applications. Such discipline has gained increasing interest from the academic as well as business point of view.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Traditional distributed component-based architectures are mainly client-server based. There are a number of component-oriented architectures that have been developed over the years. Clear examples are the EJB [87], or Corba CCM (Component Connection Model) [79] those have proven to be successful in the adaptive field.

- The EJB component model supports adaptation by automatically supporting services such as transactions and security for distributed applications.

- The Corba CCM supports adaptation by enabling injection of new connections among components. Therefore, component themselves remain intact, and component functions can be used directly by other components without additional preparations.

As a conclusion, component models support adaptation through the composition of their components, services, and connections. In particular, connections [84] are specially useful to deal with adaptation in dynamic scenarios.

2.2.3 Aspect Oriented Programming

Separation of concerns [67] means decomposing an application into distinct parts (i.e., concerns): cohesive areas of functionality. Thereby, each programming paradigm supports some level of encapsulation of concerns into separate, independent entities that represent these concerns. However, there exist some concerns that cannot be cleanly decomposed from the rest. This type of concerns is known as *crosscutting concerns* and they can be scattered over different parts of their code, and tangled with other scattered concerns.

Aspect Oriented Programming (AOP) [44] is an emerging paradigm that presents the principle of separating crosscutting concerns, allowing less interdependence, and more transparency. Thereby, an aspect is a module that encapsulates a crosscutting concern, and it is composed of pointcuts and advice bodies. The interception of an aspect is performed in a *join point* (a point in the execution flow), and defined inside a *pointcut* (a set of join points).

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

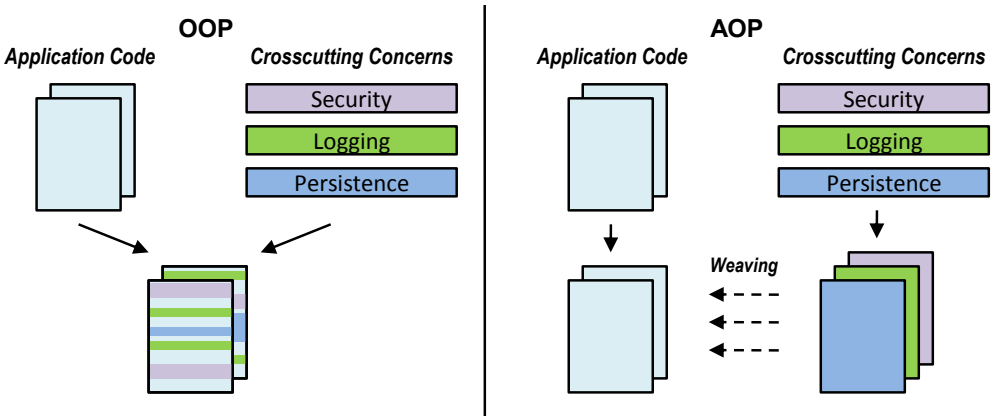

Figure 2.8: Example of crosscutting concerns in OOP and AOP scenarios.

Whenever the application execution reaches one pointcut, an *advice* (namely a callback) associated with it is executed.

This process allows the addition of new behaviours with a clear separation of concerns, where developers weave (i.e., merge) different aspects into a complete application. As for example security, logging, or persistence crosscutting concerns (Figure 2.8).

Dynamic AOP [75] promotes the same benefits as AOP, but without weaving precompiled aspects. A number of dynamic AOP tools have been developed, with different weaving techniques like efficient bytecode rewriting or dynamic proxies. In addition, crosscutting concerns can be reconfigured (i.e., weaved) at load-time (e.g., JBoss AOP [38]) or at runtime (e.g., AspectWerkz [8]).

Naturally, these benefits are important to adaptive middleware. Moreover, dynamic AOP enables factorization and separation of crosscutting concerns from the middleware core, which promotes reuse of crosscutting code and facilitates adaptation. Using dynamic AOP, customized versions of middleware can be generated for application-specific domains.

Nowadays, many developed ad-hoc solutions [74, 33, 34] support dynamic AOP into the adaptive middleware arena, since such methodology provides easier extension and reusability than others. Some of these research lines propose the use of dynamic AOP substrate for policy-based adaptive systems, or coordination support for distributed changes (DyReS [95]).

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

2.3 Implicit Middleware

An implicit middleware [87] is defined as a tool that allows developers to remain unaware of the middleware services during application development. In implicit middleware, required services are defined in separate code, supported by annotations and/or descriptor files. As a consequence, code execution is governed through a binding mechanism which glues application code with middleware services.

Indeed, implicit middleware enables transparent interaction between the original system (e.g., web server) and other new functionalities (e.g., load-balancing). In order to build this implicit middleware, a common resource for binding is used: the *interceptor*. Schmidt et al. [82] defines the interceptor like the architectural pattern, which allows services to be added transparently to a framework, and triggered automatically when certain events occur. Thereby, by means of using interceptors, developers enable a clean distinction between application code and middleware code.

By using such approach, we benefit from the following set of actions, which make it easier to :

- **Code** : connection with the middleware services is transparent, allowing developers to focus on the application code.
- **Maintain** : the separation between application and middleware services is clean and understandable. In addition, changing middleware services does not require changing application code.
- **Support** : developers can change needed middleware services by modifying the correspondent binding via annotations or descriptor files.
- **Reuse** : middleware services are reusable among other applications, or versions of the same one, in a simple way. Normally, developers only need to modify the connection point.

There exist different approaches on implicit middleware, the generic but normally more intrusive wrapping techniques [13], or ad-hoc interception solutions provided explicitly by the own platform. Examples of this second case, are Enterprise JavaBeans (EJB) 3.0 [87] or Fractal [16] component models.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

2.3.1 Distributed Interception

If we move these ideas to distributed settings, we logically come up with the **distributed interception** concept. We can define this concept like the technique that allows interception mechanisms in distributed scenarios. In addition, distributed interception can benefit from other disciplines like connection-oriented models [84] or network communication approaches (e.g., RPC [94]).

We can find some examples in literature about distributed interception :

- Eternal [59]. The Eternal system can be considered as one of the first contributions in this field. Interception in Eternal works by capturing specific system calls (i.e., IIOP) used by the Corba ORB system [79]. Such calls are mapped onto a multicast communication group.

- Chameleon [19]. In the Message-Oriented Middleware (MOM) setting [24], Chameleon uses message handlers (i.e., interceptors) to extend its behaviour by means of filter mechanisms.

- Dermi [66]. Dermi is a P2P remote object middleware that includes an inner distributed interception functionality suitable for large-scale scenarios.

One of the most important limitations of these approaches is that they are only focused into providing ad-hoc distributed interception. As a consequence, interception have to be implemented with the provided framework mechanisms. Therefore, distributed interception is not suitable to be framework service, meaning that these solutions are not focused to applying interception to other systems.

As an example, in a MOM scenario we have to implement its own filters for intercept messages, but we are not able to use any of these previous frameworks to intercept or handle the messages of other platforms transparently.

We can solve such limitation by using interception solutions or the Aspect Oriented Programming (AOP) paradigm. They offer generic and non-intrusive local interception to any parts of the system, as well as to other external systems.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Going one step further, distributed AOP is an emerging and advanced implementation of distributed interception. We are going to introduce this new paradigm in the next section.

2.3.2 Distributed AOP

The term **distributed aspect** usually refers to crosscutting software modules that are designed to work transparently, but reside either in multiple computers connected into a network or in different processes and/or threads inside the same host. In this way, one aspect sends a request from a local event (e.g., pointcut) to another aspect in a remote host/process/thread in order to execute some routine (e.g., advice).

However, the term could be confused with one of the extensions of the AOP concept used in the context of distributed computing, such as aspects using distributed mechanisms or aspectualization of component models:

- Soares et al. [86] propose the use of Java Remote Method Invocation (RMI) [78] and AspectJ [75]. They report that they use AspectJ for improving the modularity of their RMI-based programs, splitting code and remote object logic into local aspects.

- The use of AOP into component-based models tries to settle the crosscutting concerns for designing and developing reusable software component-based distributed applications. Examples of this *aspectualization* of components or component containers are [70, 71, 5]. This process means that crosscutting concerns of components models (e.g., location or deployment) are separated in local aspects.

Nevertheless, although they could be considered approximations of AOP in distributed system area, neither of these works is not considered distributed AOP [93]. Since they use traditional AOP (i.e. local interception) to separate crosscutting concerns of distributed systems but in a local way.

For this reason, it is assumed by the literature that the first solution was the work [63], which introduces the remote pointcut mechanism. This abstraction

is considered the starting point for distributed AOP, and other works have extended this idea in some way [69, 61, 50, 93, 55].

2.3.2.1 State of the Art

This section focuses on existing distributed AOP technologies that support distributed aspects with a notion of remote pointcut and remote advice.

DjCutter [63] presents a new AOP language similar to AspectJ [75], but in a more specific scope. Furthermore, DjCutter main innovation was the remote pointcut concept. This work defines a remote pointcut like a function for identifying join points in the execution of a program running on a remote host. Thus, it allows developers to code aspects modularizing crosscutting concerns without explicit network code. The advice bodies in all aspects are executed in a unique host in the network, thus making this approach inappropriate for large-scale domains.

JAC [69] or Java Aspect Components, is not a language, is a framework where the main entity is the Component-Aspect. JAC dynamic AOP framework is extended in order to support distributed pointcuts. Distributed pointcuts enable definitions of crosscutting structures that are not necessarily located on a single host. JAC simulates the semantics of remote advice by executing local advices on a local copy of the aspect (aspects are replicated on each host). Finally, a consistency protocol makes sure that whenever aspects are deployed on one specific host, the same aspects are also deployed on the other involved hosts. However, JAC does not support any group or context abstraction, neither remote activation of distributed aspects.

AWED [61] is a declarative language for distributed aspects with syntax based on AspectJ. It provides remote pointcuts on selected hosts, including support for remote sequences. It also offers distribution (asynchronous and synchronous) of advice execution. It introduces the group notion in this distributed aspect area as well. AWED uses this group notion for the deployment, instantiation and state sharing of aspects. Although other solutions work with a single or a set of host scopes, AWED was the first in using a group communication infrastructure to perform it (JGroups [1]). Nevertheless, its group abstraction forces activation of the same distributed aspects in each host (total

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

replication).

ReflexD [93] is a kernel for distributed AOP. This kernel consists in a general framework for the implementation of distributed AOP languages. The main entity is composed of three concepts : distributed cut, action, and binding. The cut refers to the execution points of an application, and the remote effect is the action. The binding is an explicit entity, which can be manipulated at runtime. Finally, it includes an initial approximation for a distributed control flow mechanism based on RMI.

DyMAC [50] is a Component-Aspect based middleware. Its main entity defines remote pointcuts, remote advices, and distributed joinpoint infrastructure. DyMAC also introduces an extended set of activation and instantiation scopes. Furthermore, the composition is supported via a set of descriptors. The component (dependency definition similar to EJB), the application (relations among component-aspect entities), and the deployment descriptor.

2.3.2.2 Comparative Criteria

Having described all the background, we analyzed the related work on existing distributed AOP systems. As we have seen in the previous section, there exist some different works in the distributed AOP area. In some way, each of these approaches implements different distributed mechanisms inside this paradigm.

In addition, comparing our goals in this dissertation with this previous, we establish the five criteria for comparison as follows.

1. *Scalability Requirements* when the system scales in respect to its size maintaining its performance and reliability in a wide-area network.

2. *Reflection Capabilities* offer introspection (observation) and intercession (modification) of the system structure and behaviour.

3. *Adaptive Composition* allows dynamic reconfiguration of the system and its components in runtime.

4. *Access and Location Transparency* are allowed thanks to the hiding capacity of resource discovery and management process.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

5. *Persistence and Replication Transparency* hide the underlying storage and replica management mechanisms.

2.3.2.3 Evaluation

In this section, we analyze each criterion from these approaches focus. Table 2.4 summarizes the rank of what we believe each approach provides in regard to each of the criteria analyzed (legend is as follows: ϕ : not supported; ν : supported in some way).

Table 2.4: Summary of considered requirements in the state-of-the-art

Approaches Approaches	Scalability Requirements	Reflection Capabilities	Adaptive Composition	Acc. & Loc. Transparency	Per. & Rep. Transparency
DjCuttter	ϕ	ϕ	ϕ	ϕ	ϕ
JAC	ϕ	ϕ	ϕ	ν	ϕ
DyMAC	ϕ	ϕ	ν	ϕ	ϕ
ReflexD	ϕ	ν	ϕ	ν	ϕ
AWED	ϕ	ϕ	ϕ	ν	ν

Scalability Requirements : The main fact that limits these works is their network infrastructure. Indeed, most of the current work in distributed AOP has been based on remote method invocation using a remote object framework. This infrastructure limits these works to provide only one-to-one primitives, and makes the construction of group services even harder. Exceptionally, AWED is based on a communication group approach (JGroups [1]) that offers a one-to-many group abstraction. Nevertheless, these solutions construct their inner services (e.g., naming or registry) in a centralized way, which eventually becomes a bottleneck.

On the other hand, not only the construction, but the maintenance of the platform and their installed resources is needed in this dimension. However, neither fault-tolerance nor churn-resilient mechanisms can be found on any of the analyzed frameworks. Only ReflexD contemplates solving the binding failure problem, but as a possible future work line.

In order to guarantee performance and efficiency in the geographical dimension it is highly recommended to perform an effective use of asynchronous and proximity-based communication services. In this line, AWED is the unique

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

solution that provides asynchronous mode for its remote pointcut mechanism. However, although some of these works provide new scopes for remote connections, none of these approaches takes care of the latency in their communication systems.

When the system grows, host communication may be unreliable, specially in hosts that are in large size groups. As we commented on before, most of these works have been namely on non-scalable infrastructures, based on one-to-one primitives. As a consequence the construction of group services on top of them is complex. In this sense, AWED is based on a communication group infrastructure that provides a one-to-many service, but it is not designed for large-scale scenarios.

Reflection Capabilities : In general, it seems that these works are not designed to offer reflection capabilities to their developers. Therefore, both introspection and intercession requirements are not easy to achieve. Only ReflexD, which is based on a reflective platform, provides some basic functionality. Specifically, this reflective approach models its entities like meta-objects, and offers reification of the involved hosts and its aspect links.

However, any of these solutions offer a complete interface of introspection and intercession of its system to enable runtime observation and modification of its structure and/or behaviour.

Adaptive Composition : Some solutions establish a clear notion of the distributed aspect entity. However, only DyMAC presents a set of descriptors, in order to enable separation between entity implementation and its composition declaration.

On the other hand, popular dynamic AOP implementations [75] (e.g., AspectJ or JBossAOP) have the load-time pointcut definition restriction (i.e., pointcut instances in runtime are not allowed). However, some of these related works use their own mechanisms to allow runtime interception, as for example JAC with wrappers, or ReflexD using reflection.

Nevertheless, reconfiguration seems not possible in runtime. Because, none of these works allow explicit mechanisms like distributed aspect hot redeployment, decoupled connection mechanisms, or the provision of inner interception techniques.

Access and Location Transparency : In this kind of distributed middleware, the transparency when we are accessing or locating resources must be guaranteed. One important question is how resources (e.g., hosts or aspects) are identified. JAC and AWED represent hosts as plain strings. AWED supports explicit constructs in its pointcut language to specify on which host an advice should be executed. Also, execution in groups of hosts is transparent.

In addition, AWED allows the use of wildcards to avoid the explicit location address where the advice must be executed. On the other hand, ReflexD offers a pair name and address (server:port) for instantiation/localization of distributed aspects. However, none of them allow to move aspects while they are in use, being this location change process not possible transparently.

Persistence and Replication Transparency : In this last criterion, only AWED presents a degree of persistence transparency in its state sharing mechanism. However, this mechanism explicitly needs an auxiliary distributed aspect, generated at deployment phase, to be able to perform global state sharing via synchronization tasks. This problem is due to state sharing follows a centralized solution where all the instances of the same entity would rely on one specific host that holds the shared field.

AWED replicates data in all hosts, since all hosts have the same content in AWED (i.e., activated distributed aspects), although is possible that only a few use them. In addition, AWED does not introduce any mechanisms to take advantage of this implicit total replication.

2.3.3 Conclusions

As a main conclusion, the most promising paradigm to support distributed concerns seems to be the distributed AOP. Since this kind of approach is able to provide a non-intrusive, distributed, and adaptive middleware solution (See Table 2.3).

After some works that combine component models and AOP facilities [70], the remote pointcut primitive [63] finally established the distributed AOP starting point. Remote pointcuts are an adaption of pointcuts in a distributed way, since they invoke the execution of an advice on a remote host. More-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

over, these pointcuts may be propagated in a group of hosts [61], by using, for example, a common identifier.

Indeed, remote pointcuts and remote advices are nowadays the differential fact for this kind of frameworks. As a consequence, the main effort of these works is to create the most complete remote pointcut and remote advice model.

In addition, we observe that most of these works have interesting functionalities or services like distributed control-flow [93], support for remote sequences and state sharing [61], or the distributed joinpoint context-aware infrastructure [50].

Nevertheless, as we have seen in this chapter, none of these approaches fully complies with the requirements that we have described in Section 2.3.2.2. As a consequence, none of them can achieve the goals of this dissertation.

Even though some of these works express that can be suitable for large-scale systems, we believe this is not possible, because none of them was designed with the scalability requirement in mind. As a consequence, most of these works have been built on top of remote object frameworks. This fact limits them to provide only one-to-one primitives, and makes the construction of group services even harder.

One special case is AWED, which obtains some good results in this comparative thanks to the asynchronous and group communication infrastructure (i.e., JGroups). Nevertheless, its group scope force it to have exactly the same aspects deployed on each host (total replication of host content). In addition, it is well-known that JGroups is not designed for large-scale scenarios. As a consequence, AWED seems to be more focused into clustering solutions (e.g., the cache problem [61]) that imply a small number of hosts per group, where each host has a similar behaviour.

One of the current trend in distributed computing is to head towards scalable and decentralized models. These models benefit more from the computing at the edge paradigm, where resources available from any computer can be used and are normally made available to their members. Middleware architectures play an important role in achieving such task, and abstracts developers from the underlying layer issues like persistence, fault tolerance, and load balancing, among others. Therefore, there is a need for a middleware platform that can

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

be used to develop worldwide oriented distributed applications. This middleware must be scalable, must provide availability guarantees, and must offer a good degree of transparency in its inner mechanisms and services.

The next chapter introduces our distributed AOP middleware proposal for large-scale development. Our main motivation is to enable **distributed concerns** in a transparent way to applications which were not specifically designed for **large-scale environments**. Our approach benefits from a P2P substrate and a dynamic AOP framework to implement its services in a decentralized, decoupled, and efficient way. It provides a scalable deployment platform where distributed aspects are deployed and activated in individual or grouped hosts. Moreover, we introduce a distributed composition model that envisages separation of distributed concerns, taking the necessary features from component models, like distribution facilities and connectors, and from computational reflection, like introspection and meta-levels.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 3

Distributed AOP Middleware for Large-Scale Scenarios

In the previous chapter, we have studied the different solutions in distributed middleware related to the scalability, availability and transparency properties. We concluded that the most promising solution to provide these properties is the distributed AOP paradigm. Nevertheless, none of current approaches in this area are designed to support distributed aspects in large-scale scenarios.

In this chapter we describe our whole **distributed AOP middleware for large-scale scenarios** proposal. We introduce the proposed middleware solution and its innovative services, and describe how it fits in with the different layers so that all of its functionalities are provided to the upper layers. Two distributed middleware disciplines can be considered the foundations of our model. There are the P2P middleware and the event-based middleware. These middleware solutions help us to construct distributed systems in a decoupled and scalable way.

In addition, it is also important to note that what is described here is a generic proposal for a large-scale AOP middleware approach. It means that it is not necessarily tied to any specific underlying technology. It is clear that when we implemented our prototype we had to make certain decisions. However, these were only design decisions, since the generic nature of our approach means that other foundation layers (e.g., interception logic) could

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

have been chosen.

The structure of this chapter is as follows: we describe the main innovative contributions our distributed AOP middleware provides, describing the distributed composition model, the scalable deployment platform, and the foundation layers, following a top-down design strategy. Later, we include an overview of prototype implementation called Damon, as well as experimentation and empirical evaluations.

3.1 Introduction

Nowadays, the increase in computing capacity, the reduction of hardware and communication costs, and the massive use of wide-area networks, have been changing the way distributed applications are being developed. However, scalability, availability, and transparency still remain strong significant issues for distributed approaches. Our distributed AOP middleware for large-scale scenarios proposal aims to provide these properties to existent or new applications without modifying them (i.e., non-intrusive).

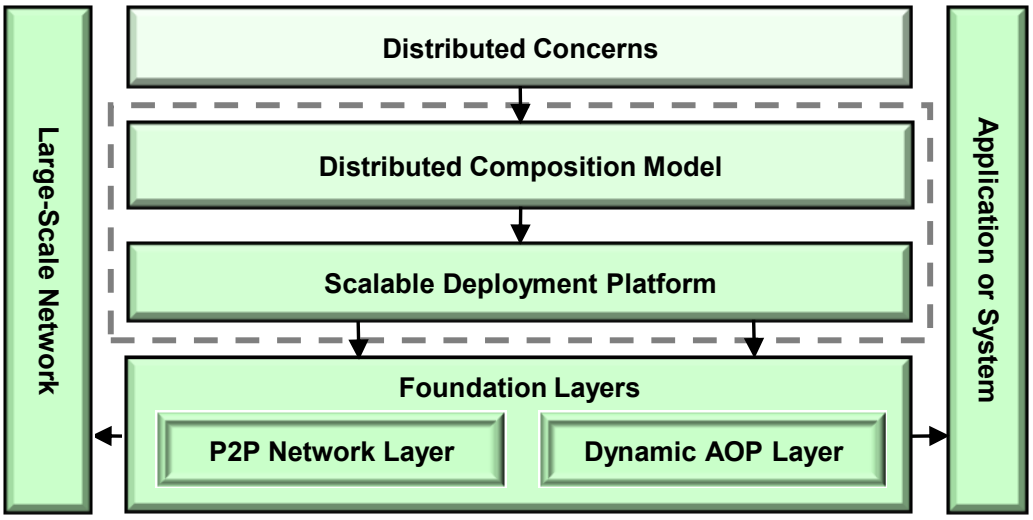

Figure 3.1: Proposed Generic Distributed Middleware Architecture.

Our view is to provide two complementary middleware layers, consisting of a distributed composition model layer, constructed over a scalable deployment

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

platform layer. As seen in the architectural diagram (Figure 3.1), both layers sit on top of the same common building blocks, and its order determines its programming complexity. In addition, both these layers are built on top of the two main pillars of distribution and interception.

We have adopted a **top-down strategy** in order to describe our proposal. It starts from the highest level and breaks down the middleware architecture to gain insight into its middleware layers. Therefore, access to lower-level layers is abstracted and made transparent to the application developer.

Each layer and its associated services include the most important contributions (see Section 1.3) of this dissertation:

- **Distributed Composition Model**: is the upper-layer of our proposal. This model envisages separation of distributed concerns, taking the necessary features from component models, like distribution facilities and connectors, and from computational reflection, like introspection and meta-levels.

- **Scalable Deployment Platform**: provides the necessary functionalities and services (e.g., reflection or life-cycle) to the upper layer, and the deployment for deploy distributed aspects in large-scale scenarios. Therefore, this platform offers a complete life-cycle on the distributed aspect container.

- **Foundation Layers**: is composed by two main pillars, the P2P network layer, which includes the routing, the messaging, and the persistence layers, and the dynamic AOP layer that consists of the hot deployer and the aspect weaver.

All these layers allow innovative contributions on our distributed AOP middleware suitable for large-scale application development. Some examples are: distributed aspect composition, new abstractions for remote interactions, and powerful runtime reconfiguring mechanisms. Finally, these middleware features will be further extended throughout this chapter.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

3.2 Distributed Composition Model

A distributed component model connects local and/or remote software components that will be used to build the system. These components may be designed from scratch for the new system, or may be brought in from other projects and/or third party vendors. Components are high level aggregations of smaller software pieces, and provide a black box building block approach to software construction.

In this section, we introduce a distributed component model for distributed AOP, which is the upper level contribution of our middleware (Figure 3.1). Basically, our model supports interaction in heterogeneous scenarios, by offering new composition features. Some of these features come from component models, like distributed connectors or descriptors, and from computational reflection, like introspection and meta-level services. Furthermore, our model is designed to provide flexible recomposition techniques, enabling an efficient and transparent reconfiguration at runtime.

An important question arises regarding when the composition can be performed. Such process can only be executed in any of these stages:

- **Design**: defining the distributed entity, and its properties and connectors (local and remote), in order to generate the descriptors.

- **Load-time**: using the descriptors to deploy and activate the associated distributed aspects in the specified host or host group.

- **Runtime**: allowing redefinition of activated distributed aspects, without restarting the system. We stress out the runtime mechanisms since we use a novel distributed meta-level approach.

Some key challenges in composition models like recursivity (distributed aspects as part of other distributed aspects) or interoperability (supporting third party composition) have been considered in our model. In the rest of the section we explain in detail the composition model focusing on the entities, connectors, and descriptors.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

3.2.1 Distributed Aspect Entity

We are mostly interested in distributed concerns where distributed aspects are executed in a remote host or simultaneously in multiple hosts. Examples of this kind of concerns are fault-tolerance, load-balancing, replication, synchronization, among others.

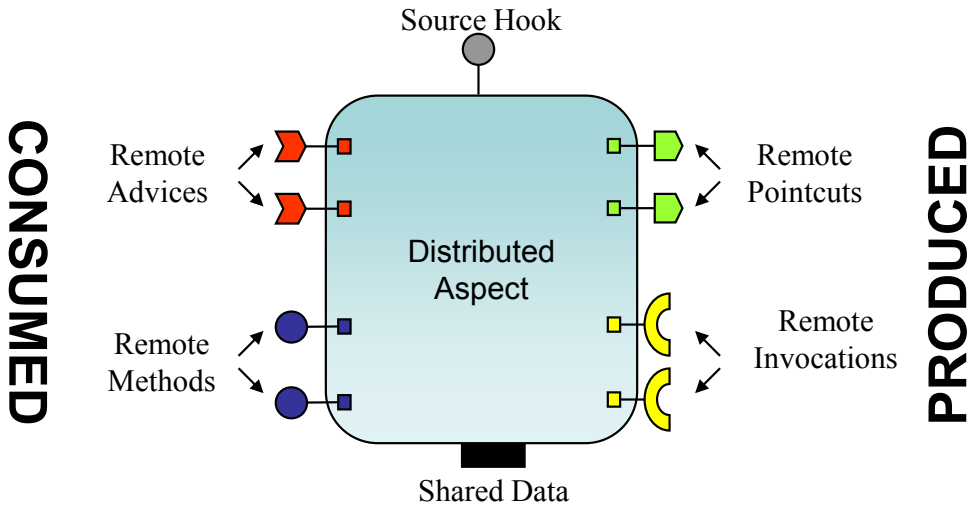

Figure 3.2: Distributed Aspect Diagram.

Indeed, distributed AOP presents a different philosophy than traditional solutions like remote object or component models. When developers design a new application, they firstly obtain or implement the *raw* application without any distributed concerns in mind. They may simply design the main concern by *thinking in local*, and later implementing the rest of the distributed concerns, designing the necessary connectors (e.g., remote pointcuts), which conform the distributed AOP application.

Previous works (e.g., [86]) force developers to take care of communication issues, mixing them with local aspects techniques. In order to improve this development, a distributed concern must be modelled as a true distributed entity.

Therefore, the key feature is the distributed aspect entity. Our solution, inspired by different distributed disciplines, like connection oriented frame-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

works and composition models, makes it easier to wire entities. As a graphical representation (see Figure 3.2) for our model, we have been inspired by Corba Component Model (CCM) [79] because it provides enough richness to represent all elements we need.

Let us outline the main attributes of a distributed aspect entity: the Source Hook, the Remote Connectors, and the Shared Data.

3.2.1.1 Source Hook

The **Source Hook** is defined as the connector that is responsible for performing local interception in our model. Our Source Hook Interface is inspired by Crosscutting Interfaces (XPI) [35], which provide an abstraction to separate local application code (i.e., advices) from pointcut specific implementations. Summarizing, this mechanism binds transparently the application code with the distributed aspects. Note that this approach is independent from the underlying AOP language and toolkit ((e.g., AspectWerkz [8])).

In Figure 3.3 we present a sample scenario where we aim to intercept (pointcut) the JDBC [39] driver interface, in order to distribute the local updates and queries (advice). This kind of solution has other benefits like a major level of comprehensibility for developers.

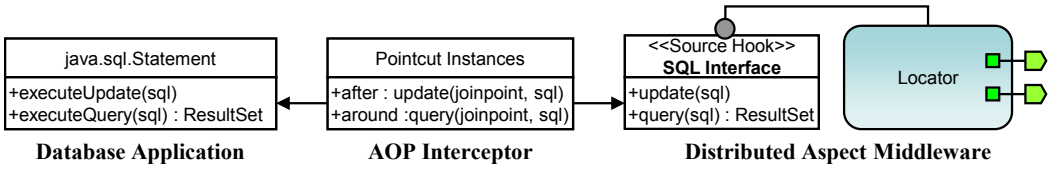

Figure 3.3: Example of Source Hook for JDBC.

On the other hand, an important issue is how to deal with of multiple execution of the same Source Hook. An sample scenario, will be an interception of the dispatcher of a web server when it is dealing with massive requests of a popular website. In order to address this problem, we develop an interesting variation called **Static Source Hook**. This mechanism follows the singleton pattern, allowing only one execution at the same time (mutual exclusion) for a

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

specific Source Hook. The following executions are waiting, are to be executed one by one, in a FIFO order.

3.2.1.2 Event-Based Connection Model

Indeed, remote pointcuts and remote advices are nowadays the differential fact for distributed AOP approaches (Section 2.3.2.1). However, many of the existent solutions used remote invocations to perform remote pointcuts, although this kind of mechanism is not scalable or flexible enough.

For this reason, we have constructed our model over an event-based infrastructure (Figure 3.4). In this model, the event bus is responsible for transmitting to event consumers thrown by producers based on the information contained in these events. Therefore, we model the distributed aspects connectors like remote events, with a connector identifier (e.g., *alarm* or *state*) associated dynamically.

As we have seen, each connector has a unique identifier that defines the topic of the event bus. In addition, the connector is related to a target that specifies the context involving an individual host or group of hosts (i.e., topic subscription identifier). Furthermore, each host declares its individual identifier, or whether it is to join a group dynamically, since this can change at runtime. As a consequence, we avoid the use of static identifiers (i.e., IP addresses) in our host definitions.

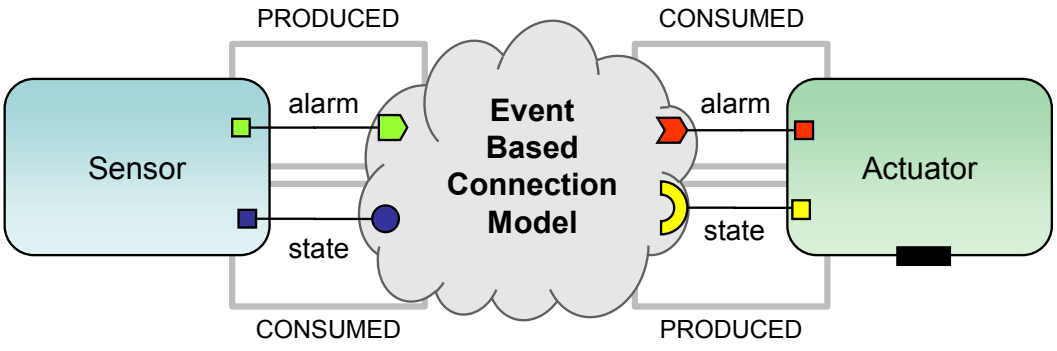

Figure 3.4: Event-Based Connection Model Example.

In our model, all the remote services can use asynchronous/synchronous

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

communication primitives performed by means of using the underlying messaging event-based functionality. We allow loose coupling connections thanks to the use of these asynchronous and event-based mechanisms. These remote connectors can be split into two categories: produced (remote pointcuts or invocations) or consumed (remote advices or methods) :

- **Remote Pointcut and Remote Advice**: Once local interception is triggered, the remote pointcut starts. Thereby, this service us linked to Source Hooks, which remotely propagates associated joinpoints. Distributed aspects can disseminate pointcuts to a single host or a group of them, using the corresponding scope (e.g., one-to-many). Distributed aspects can be linked themselves together by means of a remote advice. The receptacle allows a distributed aspect to declare its dependency to a specific remote pointcut. Thus, a remote advice uses the same identifier to be notified about its desired remote pointcut.

  In the example of Figure 3.4 the Sensor distributed aspect produces remote pointcuts (*alarm*) and the Actuator distributed aspect can receive them. In this way, these distributed aspects work together without being tightly linked, thanks to our event-based connection model.

- **Remote Invocation and Remote Method**: Traditional method invocation is also supported on our connection model. It serves a main purpose of inter-aspect communication on demand. Distributed aspects are scattered among the network, and they need to execute their own methods, as found in traditional AOP, but in a distributed way. In such scenario, methods are dynamically invoked on other hosts.

  As seen on Figure 3.4, we can observe that the Actuator distributed aspect produces remote invocation in order to obtain the *state* of a specific host under demand. In addition, if this connection (*state*) is synchronous, our model allows to the Sensor instance to reply this request into the same communication channel.

Furthermore, these different connectors enable the push and pull models in our proposal. For the push model, remote pointcuts (producers) generate events and actively pass them to the event communication channel. On

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

the other side, remote advices (consumers) wait for events to arrive from the channel. For the pull model, a distributed aspect can actively request events from another entities using remote invocations. The producers wait for these requests to arrive from the channel using remote methods. When these pull requests arrive, event data is generated and triggered.

3.2.1.3 Shared Data

Distributed aspects in our model can be of two types: stateful and stateless. Stateless distributed aspects do not maintain a state while stateful ones do.

On the other hand, the **Shared Data** mechanism allows *stateful* distributed aspects to save and restore their state information from the persistence service. Therefore, this mechanism guarantees data recovery after the hosts that contain distributed aspects or shared data fail. Moreover, this data can be shared by one, many or all members of the group.

For stateful distributed aspects, it is also necessary to provide a persistence mechanism to take care of the total disappearance of instances. Usually, if all instances of the same distributed aspect are gone, their shared state is lost. Therefore, this mechanism follows a persistence strategy where the state is maintained in all cases.

As we can observe in Figure 3.4, we have the Actuator distributed aspect, which can store the historic information received from Sensors. This distributed aspect is declared like a stateful entity, and for this reason its data is saved into the network automatically. If this distributed aspect left the network and returns later, it will also recover its previous state. Moreover, if we have more Actuators in the network, they will be able to share their own information, and work together (i.e., collaborate) with global state information.

3.2.2 Distributed Aspect ADL

In our model, we have developed a specific Architecture Description Language (ADL [54]), which provides the way to define an abstract descriptor for our distributed aspect. Basically, this descriptor must specify the properties, activation details, and local/remote connectors of a distributed aspect. As a

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010


```
distributed-aspect =  name , abstraction , [num] ,
target , state,
                      {remote-pointcut} , {remote-invocation},
                      {remote-advice} , {remote-method} ;
name = string ;
abstraction = string ;
num = digit , { digit } ;
target = string ;
state = 'STATELESS' | 'STATEFUL' ;
remote-pointcut =  sourcehook , id , [target] , abstraction ,
[num] , method , args  , mode ;
remote-invocation = id , [target] , abstraction , [num] ,
                    method , args , mode ;
remote-advice = id , method ;
remote-method = id  , method ;
sourcehook =  interface , method ;
method = string ;
args = {arg} ;
id = string ;
mode = 'ASYNCHRONOUS' | 'SYNCHRONOUS' ;
arg = string ;
interface = string ;
digit = '0' | '1' | '2' | '3' | '4'
      | '5' | '6' | '7' | '8' | '9' ;
string = '"' , { all characters - '"' } , '"' ;
all characters = ? all visible characters ? ;
```

Figure 3.5: Distributed Aspect ADL Grammar.

consequence, the distributed aspect descriptor offers other benefits like the necessary abstraction and transparency with the underlying implementation.

ADL grammar is shown in Figure 3.5 using Extended Backus-Naur Form (EBNF [68]), a meta-syntax notation used to express Context-Free Grammars (CFG [18]) :

- definition of production rules where sequences of symbols are respectively assigned to a non-terminal symbol.

- expressions that may be omitted or repeated can be represented through

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

curly braces.

- can be optionally represented through squared brackets.
- terminals are strictly enclosed within quotation marks.

As we can see, the ADL is the responsible to define the distributed aspect properties, and connections.

The definition of each **entity** starts with the *name*, which obviously has to be unique in the namespace, and needs to be the same as the class name (e.g., feedback.Sensor). We continue with the *target* parameter, which determines the host or the group of hosts where the aspect will be activated. In the following line, the activation *abstraction* is defined, thus establishing if the distributed aspect will be activated in a specific host or hosts of the group. Finally, we can specify the *state*. If it is stateful, the shared data will be maintained transparently.

Moreover, with ADL we can specify each **connector**. The produced (e.g., *remote-pointcut*) or consumed (e.g., *remote-advice*) connectors are defined in a similar manner. We need to define the *id*entifier, *method* name, and *arg*uments. Moreover, the remote pointcut must define the *source hook*, which is composed by its *interface* and *method* names. The *mode* parameter in the produced connectors determines if they are asynchronous or synchronous. The connectors use the *target* of the distributed aspect as a default value, but optionally they can define their own target. Therefore, distributed aspects can produce and/or consume in different namespaces, but they are only activated in its namespace.

In order to clarify this point, we also include two descriptor examples (Figure 3.6), based on the sample scenario shown before in Figure 3.4.

These ADL examples (Figure 3.6) define the *Sensor* and the *Actuator* distributed aspects. As we can see Sensors are activated on all members of the (`feedback.net`) group via a one-to-many (*multi*) abstraction, and the Actuator instance is activated in one member of the group using other (*any*) group abstraction. As we have seen before, Sensors have no state (stateless), unlike the Actuator, which is stateful. Moreover, both of them have two declared connections, the *alarm* remote pointcut and remote advice, and the *state* remote invocation and remote method.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

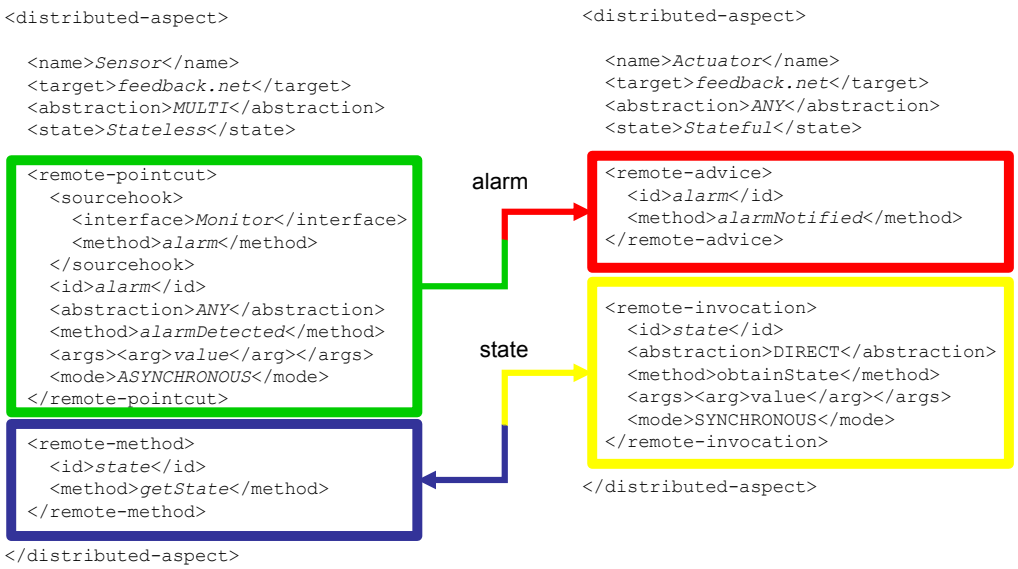

Figure 3.6: Distributed Aspect ADL Examples.

Due to the decoupled properties of our model, if another Actuator instance is activated in this group (i.e., in the same namespace), it will be connected to these instances easily. The only requirement consists of the declaration of a remote advice with the identifier and arguments of the *alarm* service of the Sensor specification.

3.2.3 Distributed Meta-Aspect

One common drawback [75] in current dynamic AOP implementations is the fact that pointcuts are to be declared in load-time. As a consequence, we are forced to declare the necessary hooks in this phase, just to let them be used in execution time.

Our distributed AOP scenario is highly dynamic, where we need to define dynamic recomposition of distributed aspects. This process involves changing between alternative compositions of an application in runtime phase. We consider that this phase starts once distributed aspects are scattered and activated around the network.

For these reasons, we propose in our model a meta-level abstraction for

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

managing distributed aspects in runtime. Such approach introduces a number of advantages: it supports our loosely coupled approach by designing and managing distributed aspects, and enables recomposition across different scenarios involving different hosts.

A distributed meta-aspect is defined as a distributed aspect that works in the meta-level, and can intercept other distributed aspects at runtime. We continue with the previous Sensor and Actuator sample scenario (Figure 3.4), but introducing the Controller distributed meta-aspect entity (Figure 3.7). This entity is able to intercept and redirect the remote connections between Sensor and Actuator distributed aspects. In order to support the ideas of this example, we create two novel mechanisms inside our model: the remote meta-pointcut, and the remote meta-advice.

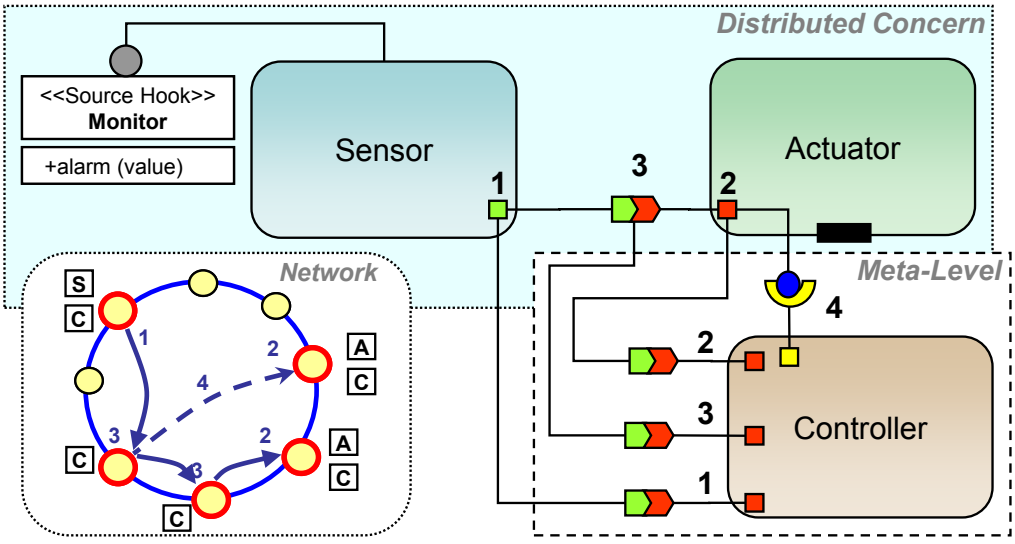

Figure 3.7: Example of Distributed Meta-Aspect Interaction Diagram.

Therefore, our **remote meta-pointcut** mechanism performs the remote interception by specifying a remote pointcut or invocation service. Interception can occur in different moments of the remote service execution (numbers 1, 2, and 3 in Figure 3.7):

1. **Before**: it is performed on the host where the remote service originates, whenever a remote pointcut or remote invocation is triggered.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

2. **After**: when the remote service gets to the destination host(s), in a remote advice or remote method execution, the remote meta-pointcut is invoked. This case is the opposite of the previous one, and it occurs just before the remote service arrives (advice or method execution). Furthermore, reflection information can be accessed from the remote service.

3. **Around**: in such case, interception is raised on any of the travelling hosts between the originator (before) and the destination (after). It is the most complex case, because we need to filter the traffic in the intermediate hosts and analyze the transient messages.

This remote meta-pointcut allows blocking and cancelling of the original service execution and/or routing, depending on the time it is performed. In addition, our distributed meta-aspect service can optionally modify any reflection information provided by the remote join point mechanism, as for instance, the address of the originator host, or the number of visited hosts.

In order to simplify and abstract the model and its implementation, we introduce another connector to manage the interactions from the meta-level (number 4 in Figure 3.7). This connector, called remote meta-advice, provides the way to invoke remote methods or advices of distributed aspects. Furthermore, in this last step developers can perform adaptive techniques. As for example, the Controller in Figure 3.7 injects events to the Actuator, acting as a feedback control [104] mechanism.

```
distributed-aspect = distributed-aspect-previous,
                          {remote-meta-pointcut},
                          {remote-meta-advice ;
remote-meta-pointcut = id , [target] , method , type , [ack] ;
remote-meta-advice = id , [target] , abstraction , [num] ,
                     method , args , mode ;
type = 'BEFORE' | 'AROUND' | 'AFTER' ;
ack = 'TRUE' | 'FALSE' ;
```

Figure 3.8: Distributed Aspect ADL Grammar Extension.

It is clear that in order to implement these mechanisms we need implicit interception on the distributed AOP frameworks. Thereby, such facilities must

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

be provided by the framework itself. In this case, we have two options: to use the observer pattern on the network substrate if it allows message introspection, or, to use AOP facilities in order to intercept it.

In our architecture, we try to combine both these options, by intercepting the inner messaging system [80], and providing an internal source hook interface for message sending, forwarding, and delivery. Subsequently, an internal aspect intercepts these methods, connecting them with the activated remote meta-pointcuts. Our approach has a low computational cost, remains fully transparent to the developer, and abstracts the underlying AOP engine.

Furthermore, we can define distributed meta-aspects in our descriptor definition (Figure 3.8), at the same level of the other produced and consumed remote connectors. Such an example is shown in Figure 3.9, based on the sample scenario in Figure 3.7.

```
<distributed-aspect>
  <name>Controller</name>
  (...)
  <remote-meta-pointcut>
    <id>alarm</id>
    <method>control</method>
    <type>AROUND</type>
  </remote-meta-pointcut>
  (...)
  <remote-meta-advice>
    <id>alarm</id>
    <abstraction>DIRECT</abstraction>
    <method>redirect</method>
    <args>
      <arg>value<arg>
    </args>
    <mode>ASYNCHRONOUS</mode>
  </remote-meta-advice>
  (...)
</distributed-aspect>
```

Figure 3.9: Distributed Meta-Aspect ADL Example.

In this example, the Controller distributed meta-aspect performs redirec-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

tion tasks. In this line, the around meta-pointcut (number 3 in the Figure 3.7) intercepts the (*alarm*) remote pointcut when it is routed to its destination host. On the other hand, a remote meta-advice (number 4) is propagated in substitution of the intercepted remote pointcut.

Other use case scenarios for our distributed meta-aspect can be those of distributed monitoring/profiling, distributed patterns, modifying service abstraction scope, or service propagation to other host(s) or groups.

3.2.4 Composite Distributed Aspect

As final contribution, we propose to extend our solution to a recursive composition model. As a consequence, a distributed aspect can be itself an assembly of distributed aspects. Indeed, such approach eases the provision for dynamic adaptation mechanisms whose goal is to allow sub-entities to be added, withdrawn or replaced, and bindings between distributed aspects to be redefined.

Note that the one important difference of our composition model among others (like CCM or Fractal), is the fact that we can establish connections with the meta-level. Thereby, a composite distributed aspect can interact with distributed meta-aspects. In addition, a composite entity can contain distributed meta-aspects itself. In both cases, the idea follows the black box abstraction, where inner composition remains hidden, and reuse is promoted.

In our recursive model, the composite distributed aspect entity provides a uniform view of an application across the different abstraction levels. The initial (i.e., the lowest) abstraction level is where primitive distributed aspects (e.g., Sensor or Actuator sample entities) reside. On the other hand, the final concerns of a distributed application are situated in the highest abstraction level. Note that a particular contribution of our work is the transparency and high functional cohesion (total separation of concerns) that our solution provides at this final abstraction level.

We can briefly define the difference for composite entities during the lifecycle. When a composite entity is deployed, the system checks the references to the sub-entities, and it deploys them. Therefore, if any of them is a composite distributed aspect, the recursive deployment is performed. The same

recursivity is applied later during the activation or passivation. Such process simplifies the installation of new distributed concerns just performing a single step. By means of using reflection mechanisms developers can choose the most suitable level for recomposition tasks at runtime.

```
<distributed-aspect>
  <name>Monitor</name>
  (...)
  <remote-pointcut>
    <id>alarm</id>
    <ref>Sensor</ref>
    (...)
  </remote-pointcut>
  (...)
  <remote-invocation>
    <id>state</id>
    <ref>Actuator</ref>
    (...)
  </remote-invocation>
</distributed-aspect>
```

Figure 3.10: Composite Distributed Aspect ADL Example.

Finally, the notion of composite distributed aspect can be used at design, load, and runtime phases. By means of an extension of our ADL language, we can define a distributed aspect descriptor for this composed entity. This extension is the name of the distributed aspect (*ref* parameter) in the remote connectors. Figure 3.10 contains an example of this descriptor, where we define the Monitor entity that is composed by the Sensor and Actuator entities. As a consequence, when the Monitor is activated, automatically the other distributed aspects are also activated.

3.3 Scalable Deployment Platform

In this work, we go one step further joining (1) the inherent **distribution** benefits of **P2P** computing, and (2) the powerful **interception** mechanisms of **AOP**. As we can see in Figure 3.1 our middleware proposal is implemented

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

as a multi-tiered architecture. On top of the P2P and AOP foundation tiers, we have built a deployment platform and, on top of it, a composition model. All these tiers are pieces of the system, which fit together, and provide a set of common services to upper-level tiers. Finally, upper-level services are responsible for enabling large-scale distributed aspect development.

As we have stated above, our aim is to provide two complementary middleware layers, consisting of a distributed composition model, and scalable deployment platform. In this section we aim to describe the second one, which is a new middleware layer built on top of the foundation layers.

This platform enables distributed aspect deployment in large-scale scenarios, thus providing :

- **Decentralized Container**: to manage the life-cycle aspects and efficient resource location.
- **Platform Functionalities**: enabling persistence, communication and messaging, reflection abstractions.

Therefore, access to lower-level layers is abstracted and made transparent to the application developer.

3.3.1 Decentralized Container

We find in literature that some distributed AOP works that are managed by an aspect container, which is responsible for creating and configuring the aspects. For this propose, we have designed a **decentralized container** in our model. Note that all hosts that belong to the network are containers, and as such, they can manage many distributed aspects.

Firstly, our container provides the **location and discovery** mechanisms for our middleware platform. In this model, we have adopted an URI-style naming convention (e.g., `p2p://resource.name.net`) to perform decentralized services. These **P2P locators** create an uniform address space enabling service access regardless of its network location (IP address). This mechanism can be used to indicate specific host locations, the identifier of a group of hosts, a specific stored value, etc. For example, for specific host locations, requests

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

are mapped from the P2P locators to the responsible IP addresses (which may change over time). Following the structured overlay design, every host is responsible for a range of locators, and if a host fails, another one replaces it in order to handle those locators.

Secondly, the container is the responsible to manage distributed aspect **life cycle**:

- *Deployment*: entities are serialized and inserted into the network.
- *Activation*: distributed aspects are recovered from the network and installed on specific hosts.
- *Execution*: activated entities are running and using the platform functionalities.
- *Passivation*: distributed aspects are stopped and remain waiting for future activations.

Basically, the first part of the life cycle involves inserting (deploy) and installing (activate) distributed aspects into the distributed system. We briefly describe the deployment and activation of distributed aspects, and how transparent services are provided to handle them correctly.

In the deployment phase, the distributed aspect is coded to run on our platform; next, it is inserted in the platform (persistence service) with an uniquely assigned P2P locator (e.g., `p2p://aspect.class.name`) and making it available to any host. Whenever the aspect is needed, it has to be located using its P2P locator in order to recover the serialized class from the persistence service.

Naturally, if the host containing the deployed distributed aspects fails, recovery would fail as well, as the host that contains this information is missing. To avoid this problem, data replication mechanisms are used in the persistence service (i.e., a platform functionality explained in the next section). When a distributed aspect handle is to be deployed, it is replicated among the nearest hosts to the target host. This way, if the target host fails, information is not lost and the distributed aspect handle can be recovered from any of these closest hosts.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

In the activation phase, distributed aspects are installed in the host(s) specified using a P2P locator (e.g., `p2p://app.group.org`), and using some abstractions (presented in the next section). Briefly the allowed scopes are: a specific host (direct), responsible host for a determinate key (hopped), one host of the group (any), some members of the group (many), or each of the hosts of a group (multi).

Once the aspect is activated, it can make use of the platform functionalities (persistence, messaging and reflection) according to its behaviour(s) and/or requirement(s). Finally, our container is also responsible of the reliability and integrity of activated aspects. In this line, any activated distributed aspect is transparently maintained by the fault-tolerance mechanism provided by our container. Thereby, once a host with any aspect becomes unavailable, and rejoins the network, all its activated distributed aspects, group membership, and persistence data are rapidly restored.

3.3.2 Platform Functionalities

We benefit from the underlying P2P and AOP services to provide a set of **functionalities** for distributed aspect development, available in each of the system hosts.

The **persistence** service involves serialization of information into the network, which replicates this data among the host closest neighbours. Thanks to this persistence service, we can guarantee fault tolerance saving this data into the network in a transparent way. Later, the data is recovered from the network layer, for the same or any other host. Moreover, this service allows developers to create multiple contexts, making easier to separate different application scopes.

The **reflection** service accesses the underlying layers, including system instrumentation tools. It defines a simplified API, where developers can obtain information about aspects, hosts, and network. Thus, examples of that API include obtaining the list of instantiated aspects, host current resource utilization like CPU, memory, or disk space, and network topology information (e.g., group members, or host latencies).

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

The **messaging** service enables efficient routing mechanisms over the routing layer, using unique identifiers (P2P locators) that correspond to specific hosts in a determinate period of time. It also creates an event-based message system on top of the underlying communication layer that allows activation of distributed aspects by means of exchanging messages using a topic-based publish-subscribe mechanism.

A set of **communication abstractions** (see Figure 3.11) are enabled, which allow new remote scopes for the distributed AOP area. Moreover, these communication abstractions allow to the upper layer to create different kind of connectors using them like an attribute.

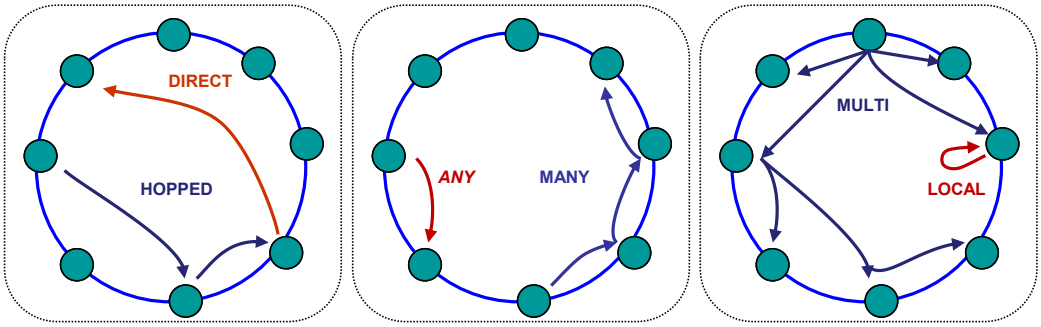

Figure 3.11: Communication Abstractions Diagrams.

3.3.2.1 Hopped Communication

The capabilities provided by the routing substrate are the foundation for what we call the one-to-one hopped abstraction. The advantage of using a key to route messages to the host is that we do not know anything about the destination. Moreover, when the host we are using goes down, the message would automatically route to another host, in a transparent process, and the originator continues to use the same key to route messages.

3.3.2.2 Direct Communication

Using the underlying routing mechanism this abstraction sends a message to the destination that owns the specified locator, doing all the work in only one

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

hop. Therefore, hopped abstractions are not as efficient as direct abstractions. Depending on the used routing substrate, this approach incurs additional overhead, because a message is routed by one or more hosts reaching its destination. This philosophy remains in stark contrast to that of abstraction calls, where the message is moved directly from source to destination. In one-to-one direct abstractions we need to know the destination host URL locator, or directly its real IP address.

3.3.2.3 Multi Communication

On the other hand, one-to-group abstractions are modelled using the messaging system layer by means of disseminating events. Moreover, thanks to the underlying p2p network, these abstractions benefit from a proximity-based topology. To target the destination group we also use P2P URL locators (e.g., `p2p://app.group.net`). Thanks to the utilization of this abstraction we can guarantee that all communications get to all of the group members in an efficient and scalable way.

3.3.2.4 Any Communication

This remote abstraction uses group mechanisms. Specifically, it allows sending a message to the nearest entity (e.g., distributed aspect) within its group, which needs to satisfy a parameterized condition. As a consequence, this is an interesting way to provide remote services that benefit from network locality. If the messaging layer provides us with an efficient anycast primitive, we can use it to create a call to the destination host that belong to the same group. The originator is insensitive to which group provides data; it only wants its request to be served. The idea is to iterate the group members, starting from the closest member in the network. Once a member of the group is found to satisfy the condition, it returns an affirmative result.

3.3.2.5 Many Communication

This abstraction is similar to the **any** abstraction, but sending a message to the n nearest hosts within a group, satisfying a specified condition. It

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

takes advantage of the many abstraction provided by the messaging layer and it therefore sends a message to several group members, continuing to route until it finds enough members to satisfy a global condition. Similar to the any abstraction, when a destination host receives an invocation, it first checks whether it satisfies a local condition and, subsequently, checks whether a global condition is met. The many abstraction is successful when the global condition is met.

3.4 Middleware Foundations

By making efficient use of dynamic AOP facilities and structured P2P substrates, we have defined the underlying layers that are the foundations for large-scale distributed aspect deployment (see Figure 3.12.

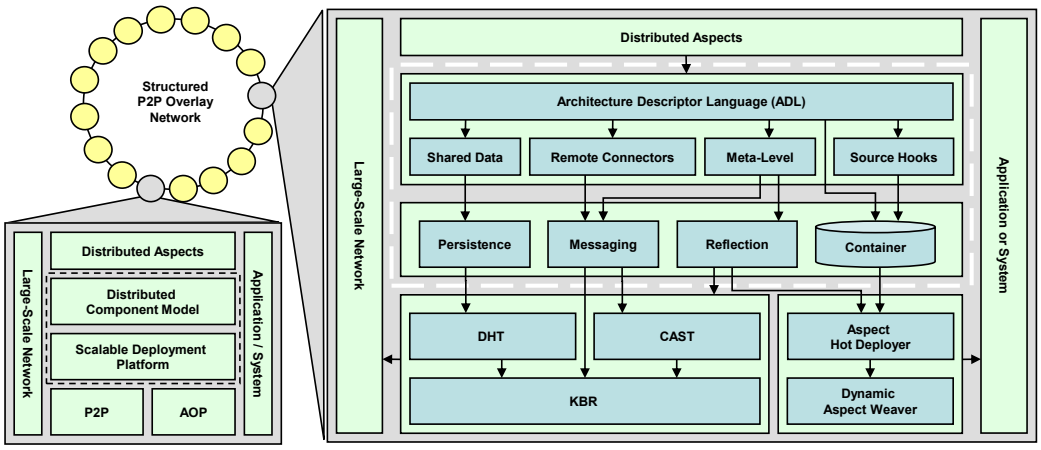

Figure 3.12: Proposed Distributed AOP Middleware Architecture.

First of all, an **AOP engine** is needed to begin with, in order to execute different aspects in each of the system hosts. This layer enables instantiation and execution of aspects at runtime. It also manages the interception mechanisms (pointcuts), interception callbacks (advices), and the interception control accesses (joinpoints).

The first actor in this layer is the deployer, which accepts and transforms the new aspects, or the changes introduced by previous ones. This runtime ability is performed by **hot deployer**s, and is a required characteristic for

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

dynamic AOP engines. Subsequently, the process of integrating deployed aspects, or more precisely, the actions for each of these aspects, is called weaving and is performed by a tool, the aspect weaver.

Therefore, a **dynamic weaver** allows aspects to be woven, unwoven, or replaced at runtime. By using an unified interface (facade pattern), to access these services, we could swap the engine implementation with any other dynamic AOP framework.

One problem is that current dynamic AOP frameworks force to declare in load time the necessary hooks (local pointcuts). However, these kinds of declarations are not usually simple or clear, and may need to be changed in the future. Our solution aims to decouple any local advices from any specific AOP language.

As seen in Figure 3.12, our middleware is built on top of a **P2P substrate** in order to exploit its inherent properties, which include scalability, fault-resilience, self-organization, routing efficiency, network proximity organization, etc. For our middleware proposal we construct a new abstraction layer, which follows the Common API [20] specification. This API specification standardizes the Key-Based Routing (KBR) [89], the application level multicast (CAST) [80], and the Distributed HashTable (DHT) [48] layers

The **KBR** substrate is a common layer found in all structured P2P overlay networks (e.g., Pastry [80] or Chord [89]), which allows for efficient message delivery based on the message key. Therefore, the message moves closer to the destination host following an approaching path that depends on the value of the specified key. Thereby, these KBR routing mechanisms allow the messaging functionality of the upper layer to send message using at most *O(log n)* number of hops, and where n is the number of hosts in the network.

The **CAST** abstraction provides scalable group communication to the upper layer. Overlay nodes may join and leave a group, multicast messages to the group, or anycast a message to a member of the group. Because the group is represented as a tree, membership management is decentralized. Thus, CAST can support large and highly dynamic groups. Moreover, if the overlay that provides the KBR service is proximity aware, then multicast is efficient and anycast messages are delivered to a group host near the anycast originator.

The **DHT** abstraction provides the same functionality as a traditional hash table, by storing the mapping between a key and a value. This interface implements a simple store and retrieve functionality, where the value is always stored at the live overlay host(s) to which the key is mapped by the KBR layer. As a consequence, this service is used by the persistence functionality of the upper layer.

Even though these foundation layers define the necessary services to deal with aspects and P2P networks, they do not fully comply with the requirements for deploying and activating aspects in P2P scenarios. It is clear that these layers provide the primitives required to implement many services, but not the services themselves. This is because such approach is far too low-level for developers. As a consequence, they may find it difficult to deal with aspects in distributed settings due to the lack of specific services and abstractions for distributed aspect development.

3.5 Implementation: Damon

In this section we present our prototype implementation, called Damon. This implementation can be downloaded from `http://damon.sf.net`. Later in this section, we test our prototype in a real large-scale network.

3.5.1 Prototype

Among all available KBR substrates, we chose FreePastry 2.1 [80] for its efficient Java implementation of Pastry (KBR) and Scribe (CAST) [80], and its better awareness of the underlying network topologies.

Scribe, a large-scale decentralized application-level multicast infrastructure built on top of Pastry, is a publish/subscribe message oriented middleware (CAST). We chose Scribe because it provides a more efficient group-joining mechanism than other existing solutions, and it also follows the Common API (Section 2.1.1.4).

Additionally, our preferred KBR substrate choice was Pastry because its routing scheme is efficient, it takes account locality when routing messages,

it is self-organizing and can gracefully adapt to node failures. All this makes Pastry one of the most interesting KBR implementations. Nevertheless, we could have used any other P2P KBR-based overlay network (or even none of them), provided that they share the same basic functionalities.

For the AOP part of our prototype we use the AspectWerkz 2.0 [8] implementation as the dynamic AOP framework. It offers rich semantics allowing runtime access to information about the join point within the advice using regular types without any casting or object array access. Reflective access to runtime information is also possible. Allows weaving (bytecode modification) at compile time, load time and runtime. It hooks in and transforms classes loaded by any class loader except the bootstrap class loader.

Therefore, developers can easily transform any (legacy) application or external library both at runtime and compile time. Finally the availability of a Java implementation at the time this prototype implementation was begun, eased the adoption process. Like the P2P layer, this AOP engine is able to be swapped if another implementation is more suitable for runtime aspect weaver proposes.

3.5.2 Experimentation

In this section, we study the cost property introduced by our middleware infrastructure in the distributed AOP area. We conducted several experiments to measure Damon viability using the PlanetLab testbed [72]. PlanetLab is a globally distributed platform for developing, deploying, and accessing planetary-scale network services. Any application deployed on it can experience real Internet behaviour, including latency and bandwidth unpredictability.

We concentrated on performing general performance tests, and chose more than a hundred hosts from distinct and varied geographical locations, including Canada, Germany, Italy, Spain, Denmark, the UK, China, or the US. We repeatedly ran the tests at different times of day to minimize the effect of momentary node congestion and failures. Before each test, we estimated the average latency between nodes to gauge how much overhead the middleware

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

services incurred.

Table 3.1: Overhead observed of deployment platform tests in milliseconds

Source Host	Destination Host	Lat.	Dep.	Act.
planetlab2.urv.net	planetlab4.upc.es	10	70	97
planetlab-5.princeton.edu	planetlab02.utoronto.ca	73	206	214
planet1.scs.stanford.edu	bonnie.ibds.uka.de	180	520	449
planetlab02.dis.unina.it	planet1.manchester.ac.uk	45	244	192
planet1.cs.rochester.edu	planetlab-2.it.uu.se	108	440	409

3.5.2.1 Deployment and Activation Experiments

The purpose of this experiment is to quantify the overhead imposed by the deployment platform. Specifically, we study the distributed aspect deployment and activation mechanisms. The values shown in Table 3.1 are the median of all of the executed tests. Each test was done using 500 random deployments and activations of random distributed aspects. Basically, the deployment mechanism follows a two-phase protocol (insertion request and acknowledgment). On the other hand, the activation mechanism is performed by sending a request message, which instantiates the container on the destination host. To simplify the test, the aspect class is locally retrieved, since the aspect is activated in the same host where it is deployed. If the aspect class was remotely retrieved, the activation time would be increased by the remote retrieval delay.

Data extrapolated from the table indicates that the normalized incurred deployment overhead is **1.78**, and the one imposed by activation is **3.27**. Seeing these results, we consider that our scalable deployment platform does not impose an excessive overhead.

3.5.2.2 Remote Connections Experiments

In this test scenario we have mainly measured our system reaction to remote connections (remote pointcuts). Each test triggered 500 random remote pointcuts from source host to destination host. Values shown in Table 3.2 are the median of all of the executed tests.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Table 3.2: Overhead observed of remote tests in milliseconds

Source Host	Destination Host	Lat.	Rem. Pointcut
(0) planetlab5.upc.es	(1) planetlab1.diku.dk	113,88	173,22
(1) planetlab1.diku.dk	(2) planet1.berkeley.net	178,46	298,33
(1) planetlab1.diku.dk	(3) pl2.6test.edu.cn	393,67	546,03

As we can observe, results show that our platform does not impose an excessive overhead in this kind of operations, since the normalized incurred remote connection overhead is **1.53**.

3.5.2.3 Meta-Level Experimentation Results

In this experiment we measure the distributed meta-aspect interception performance and overhead. We reuse the previous experiment approach to construct a similar scenario as the one shown in Figure 3.7. Thus, the three actors involved in this scenario are the Sensor, the Actuator, and the Controller. The Sensor is a distributed aspect activated on host 0 (planetlab5.upc.es), and the Actuator is another distributed aspect which resides on host 2 (planet1.berkeley.net). The Sensor triggers remote pointcuts using the *any* abstraction, with the basic condition of finding an Actuator. In addition, these remote pointcuts travel through host 1 (planetlab1.diku.dk).

Our first experiment in this scenario consists of measuring the overhead imposed by the distributed meta-aspect connectors. We aim to measure how scalable is the distributed aspect composition, especially if the introduction of remote meta-pointcut interception can degrade the base performance of the application. Specifically, we measure the *around* case, to demonstrate that whenever there are several interceptors along several hops, they do not reduce the throughput.

The conditions for our first test include the activation of a set of distributed meta-aspect Controller instances on host 1. These Controllers only perform basic tasks like logging, with a minimal (unitary) computation cost. We start the experiment with a constant rate of 100 requests per second. Again, each test was done using 500 random remote pointcuts. We repeat the experiment with variations in the number of activated Controllers. The column values

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

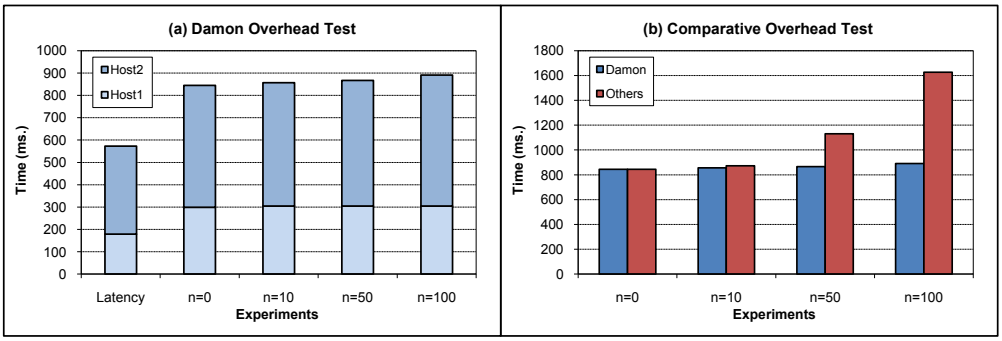

Figure 3.13: Empirical Results obtained in Meta-Level Tests.

shown in Figure 3.13a are the median of all of the tests. The overall overhead average incurred with respect to latency introduced by the Controllers when n=0 is **1,48**, for n=10 is **1,50**, for n=50 is **1,51**, and for n=100 is **1,56**. Note that n is the number of activated Controllers on host 1.

The second test follows the same structure, but in this scenario we change the use of Controllers on host 1 by ad-hoc aspects with explicit pointcuts to the message service. As we have explained in Section 3.2.3, other AOP middleware solutions are able to implement inner interception mechanisms in an ad-hoc way. Thereby, we simulate the behaviour of other approaches with this test scenario. The column values shown in Figure 3.13b are the median of all of the tests run in these experiments. The improvement average incurred for Damon with respect to other solutions when n=10 is **1,03**, for n=50 is **1,47**, and for n=100 is **2,26**. As a conclusion, our solution that provides an inner interception functionality, is able to achieve these lower computational costs.

3.5.2.4 Reconfiguration Experiment

Finally, in our last experiment, we introduce a new Actuator distributed aspect on host 3 (pl2.6test.edu.cn). This time we modify the runtime behaviour of the system by using runtime reconfiguration. While at runtime, a Controller distributed meta-aspect is activated. Nevertheless, this time the Controller performs redirection tasks, where approximately 50% of the remote pointcuts are redirected to host 3. At the beginning of the experiment, the route : (0)

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- (1) - (2) is followed, but just after second 13, the Controller is activated. Since it starts working immediately, the second Actuator (on host 3) begins processing approximately half of the requests. In second 32 the Controller is passivated, returning the system to the initial behaviour. The Figure 3.14 shows the scenario and the described results.

Based on our measurements on the PlanetLab testbed, we have verified that Damon does not impose excessive overhead on distributed aspect deployment, activation; remote connections elegantly fit with the messaging service, and the any abstraction have obtained good results because of the inherent network locality. Finally, we claim that the meta-level overhead of Damon is acceptable for distributed applications, with an elevated number of distributed aspects and meta-aspects running, and performing reconfiguration of the system.

3.6 Summary

In this chapter we have presented our whole distributed AOP middleware proposal. The main contributions of this work include distributed aspect composition, which abstract the developer from the underlying infrastructure, but offering new abstractions for remote interactions, and powerful runtime reconfiguring mechanisms.

The main block of our entire proposal is a distributed composition model, which allows composition at design, load-time and runtime phases. In addi-

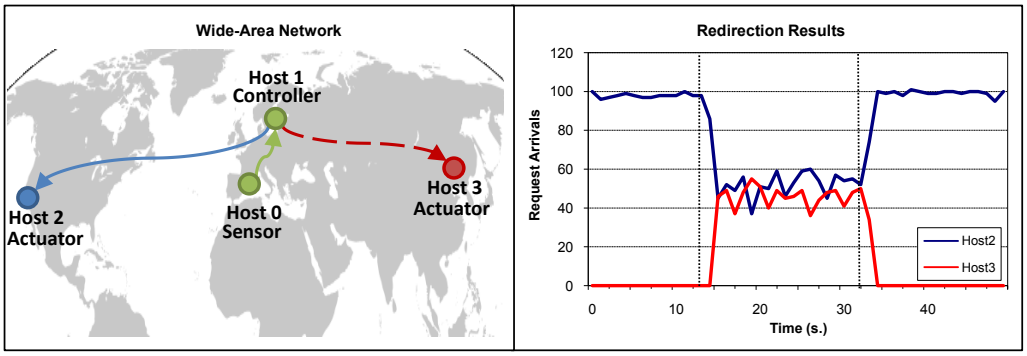

Figure 3.14: Scenario and Results of the Redirection Tests.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

tion, the event-based connection model enables decoupled interaction between remote pointcuts and advices. As consequence, runtime reconfiguration of the connection model is enabled and efficient.

Furthermore, our composition model builds on top of the deployment platform layer, and it provides a higher level of abstraction to application developers by permitting the reusability of distributed aspects through applications.

Our model also allows runtime composition thanks to the use of a distributed meta-aspect solution. The remote meta-pointcut mechanism provides a new way to perform distributed aspect composition at runtime, to achieve new goals that were unexpected during the design or load-time phases.

Other distributed AOP approaches lack some or many of these requirements, making them difficult to apply, since many services are not implicitly provided or are even non-existent. Our proposal is, to the best of our knowledge, the pioneer in envisioning a distributed AOP middleware for large-scale scenarios based on structured P2P and dynamic AOP substrates, which provides scalable deployment and distributed composition facilities.

Our middleware implementation, Damon, is a research prototype that can be downloaded from `http://damon.sf.net`, under a LGPL license. This implementation includes clarifying code examples and tutorials. Experimentation of Damon has been conducted on the PlanetLab testbed. We have proven that our middleware is feasible in large-scale scenarios, and that the system has acceptable performance.

In the next chapters we present two proof-of-concept implementations of a large-scale application that use our distributed AOP middleware. These use case scenarios allow us to stress the benefits of our middleware platform, specially providing new features like scalability, availability, and transparency.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 4

Building a Scalable Collaborative Wiki Application

In this chapter we present UniWiki, a novel large-scale collaborative application. The main motivation of this work is the use of our middleware proposal to apply scalability transparently to non-scalable applications. In addition, we reduce the effort needed for the implementation, and we make our work available to other existing wiki applications.

We have designed UniWiki as an efficient P2P system for transparently distributing the storage of wiki applications, which allows their extension to large-scale scenarios. In the following sections we motivate this application, and we explain the background of this work. Subsequently, we will present our approach, the prototype implementation, and the experimentation. Finally, we will draw some conclusions.

4.1 Motivation

With the current increase of popularity of applications like Wikipedia, Google Docs, Facebook, or Twitter, collaboration has become part of our daily life. These applications have shown how powerful collaboration can be, leading to a new interconnected and collaborative world.

In general, distributed collaboration is essential for any application de-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

signed to help people around the globe to work on a common task. An illustrative example is Wikipedia [98], the collaborative encyclopedia that has collected, until now, over 13,200,000 articles in more than 260 languages. It currently registers at least 350 million page requests per day, and over 300,000 changes are made daily [99].

Currently, collaborative applications commonly use centralized architectures that, in practice, do not necessarily scale. To handle this problem, Wikipedia needs a costly infrastructure [11], for which hundreds of thousands of dollars are spent every year [28].

Another example is Facebook, which appears to be spending on these types of machines and systems (i.e., data centers) amounts to $ 20 million in 2009 [27]. In fact, costs will go further, as they are preparing to start another data center in Virginia that is designed precisely to cover the huge energy demands imposed by the service of millions of users of social networking.

It is clearly a scenario where existing applications need **scalability**. However, achieving the same functionality of centralized architectures on a large-scale system poses numerous challenges. One important problem is related to the form of access to collaborative processes. Hosts can fail or concurrent edition of the same resource, and therefore, a collaborative system should be robust against failures and inconsistent states.

Moreover, because very popular and standardized clients (commonly via web browser or standard application) are continuously evolving, we need to introduce these changes **transparently**.

4.2 Background

Wikis are currently a popular concept, and many mature, fully featured wiki engines are publicly available. Existing approaches to deploy a collaborative system on a distributed network include Wooki [97], DistriWiki [56], RepliWiki [76], Distributed Version Control systems [6, 32, 103], DTWiki [23] and Piki [57].

Several drawback prevent these systems from being used in our target scenario:

- they may need total replication of content, requiring copies of all wiki pages at all hosts,
- they do not provide support for all the features of a wiki system such as page revision,
- they provide only a basic conflict resolution mechanism that is not suitable for collaborative authoring.

As a result of a collaboration stage of four months with the Score Team (affiliated to Nancy-INRIA), we created the UniWiki project. The main goal was to integrate the *WOOT*, (WithOut Operational Transformation) [64] consistency maintenance algorithm into a structured P2P infrastructure.

The WOOT algorithm ensures convergence of content and intention preservation [91], no user updates are lost in case of merging. WOOT is based on the same principles as operational transformation, but sacrificing the breadth of the supported content types to gain simplicity. The WOOT algorithm does not require central servers nor vector clocks, and uses a simple algorithm for merging operations *locally*. Then, every user action is transformed into a series of WOOT operations, which include references to the affected context. Finally, these operations can be exchanged later in order to distribute its behaviour.

Summarizing, the WOOT local behaviour, the common client-server implementation for wiki applications, and the need of structured P2P substrate, suggest an ideal scenario to apply our distributed AOP middleware. The result of this proposal is presented in the next sections, in the form of UniWiki.

4.3 Approach

Over the last few years, many P2P infrastructures (e.g., [80] or [89]) have been released. These systems take advantage of the computing at the edge paradigm, where resources available from any computer in the network can be used and are normally made available. While in the client-server model, only an expensive number of powerful servers provide all the computing and storage power to a much larger network of clients.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Nevertheless, decentralized architectures introduce new issues which have to be taken care of, including how to deal with constant node joins and leaves, network heterogeneity, and, most importantly, the development complexity of new applications on top of this kind of networks. For these reasons, we need a middleware platform that provides the necessary abstractions and mechanisms to construct distributed applications.

In this work, we create a system that can be integrated transparently in existing wiki engines. Our implementation is driven by this transparency goal, and for achieving it, we rely on powerful distributed interception techniques (i.e., distributed AOP). The benefits of this approach will be:

- Full control of the DHT mechanisms, including runtime adaptations.
- Decoupled architecture between wiki front-end and DHT sides.
- Transparency for legacy wiki front-end applications.

For satisfying the transparency and distributed interception requirements, we chose as the basis of our implementation the distributed AOP middleware Damon (Section 3.5). Using this middleware, developers can implement and compose distributed aspects in large-scale environments. Such distributed aspects, activated by local pointcuts (source hooks), trigger remote calls via P2P routing abstractions.

4.4 Implementation

Traditional wiki applications are executed locally on the wiki front-end. This scenario is ideal for applying distributed AOP, because we can intercept the local behaviour to inject our algorithms. Thereby, using Damon, we can model transparently the necessary concerns:

- *Distribution*: refers to dissemination and storage of wiki pages into the system. This dissemination allows a load-balanced distribution, where each node contains a similar number of stored wiki pages. In addition, this concern also guarantees that clients can access always to data in a single and uniform way.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- *Replication*: data is not only stored on the responsible node, since it would become unavailable if this node fails, or leaves the network. Thus, this concern allows to these nodes to copy their wiki pages in other nodes. Moreover, they have to maintain these copies, in order to guarantee a specific number of alive replicas at any moment.

- *Consistency*: when multiple clients are saving and reading concurrently the same wiki pages, data can become inconsistent. This approach proposes to generate operation logs (i.e., patches) of the wiki pages and distribute them. Finally, the final wiki page is regenerated from stored patches.

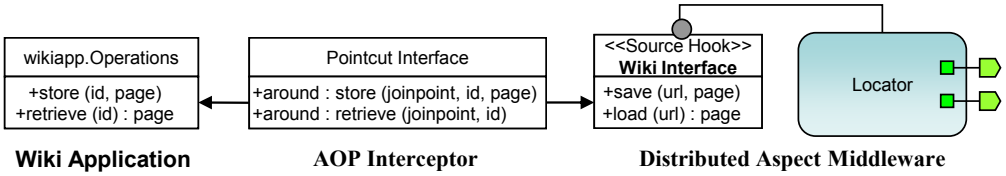

Figure 4.1: Wiki Source Hook Example.

Figure 4.1 presents the UniWiki source hook, where we aim to intercept locally the typical wiki methods of store and retrieve (in this case we use a generic example), in order to distribute them remotely. In addition, the source hook solution helps to separate local interception, aspect code, and the wiki interface. On the other hand, source hooks have other benefits, such as a major level of abstraction, or degree of accessibility for distributed aspects.

In this approach, integration with other wiki applications is quite simple and can be easily and transparently used for third party wiki applications.

We now describe the UniWiki execution step by step as shown in Figure 4.2, focusing on the integration of the algorithms and the interaction of the different concerns. In this line, we analyze the context, and extract three main concerns that we need to implement: distribution, replication and consistency.

In this scenario, the distribution is the basic behaviour of our system, and thus the most important concern. Moreover, this concern is based on key-based routing techniques. Two distributed aspects are used to implement this concern: the **Locator** (front-end) and the **Storage** (back-end).

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

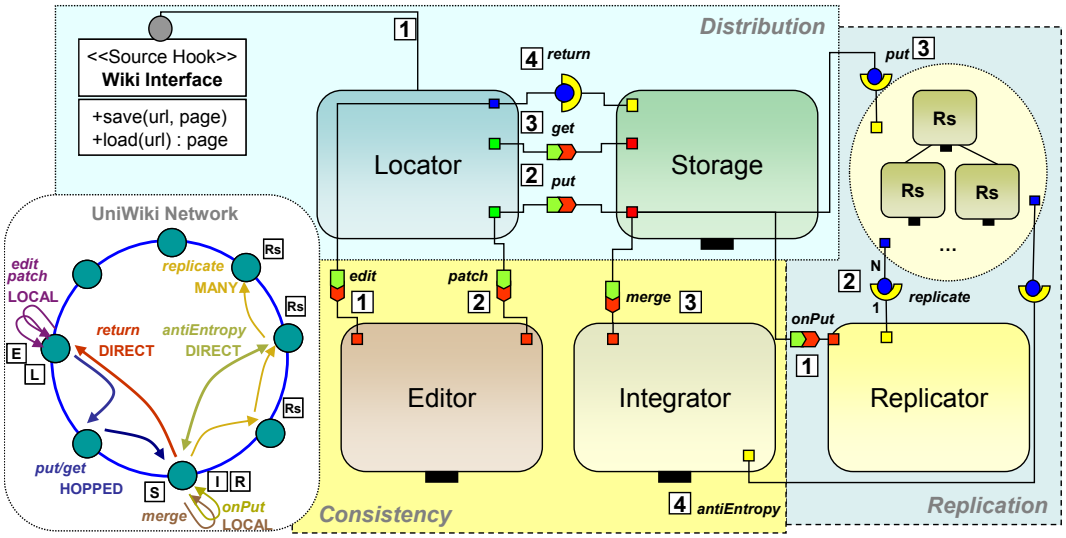

Figure 4.2: UniWiki composition diagram.

Later, the replication concern is also based on P2P mechanisms, following a neighbour replication strategy [49]. Two distributed meta-aspects are used to implement this concern in the back-end: the **Replicator** (intercepting the Storage) and the **ReplicaStore** instances (many per Replicator).

Finally, as explained in the previous section, the consistency concern is centered on the deployed consistency algorithm (i.e., WOOT). In this implementation, it allows edition, patching, and merging of wiki pages, and it performs these operations via distribution concern calls interception. Again, two distributed meta-aspects are used to implement this concern: the **Editor** (front-end) intercepting the Locator, and the **Integrator** (back-end), intercepting the Storage, and interacting with the ReplicaStore.

Distribution:

1. The starting point of this application is the wiki interface used by the existing wiki application. We therefore introduce the **Wiki Interface** source hook that intercepts the save, and load methods. Afterwards, the Locator distributed aspect is deployed and activated on all nodes of the UniWiki network. Its main objective is to locate the responsible node of the local insertions and requests.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

2. These save method executions are propagated using the **put** remote pointcut. Consequently, the remote pointcuts are routed to the key owner node, by using their URL to generate the necessary key.

3. Once the key has reached its destination, the registered connectors are triggered on the Storage instance running on the owner host. This distributed interceptor has already been activated on start-up on all nodes. For request case (**get**), the idea is basically the same, with the Storage receiving the remote calls.

4. Finally, it propagates an asynchronous response using the **return** call via direct node routing. The get values are returned to the Locator originator instance, using their own connector.

Once we have the wiki page distribution running, we may add new functionalities as well. In this sense, we introduce new distributed meta-aspects in order to extend or modify the current application behaviour in runtime. Thereby, thanks to the meta-level approach, we are able to change active concerns (e.g., new policies), or reconfiguring the system in order to adapt it.

Replication:

1. When dealing with the save method case, we need to avoid any data storage problems which may be present in such dynamic environments as large-scale networks. Thus, data is not only to be stored on the owner node, since it would surely become unavailable if this host leaves the network for any reason. In order to address this problem, we activate the Replicator distributed meta-aspect in runtime, which follows a specific policy (e.g., neighbour selection strategy [48]). The Replicator has a remote meta-pointcut called **onPut**, which intercepts the Storage put requests from the Locator service in a transparent way.

2. Thus, when a wiki page insertion arrives to the Storage instance, this information is re-sent (**replicate**) to the ReplicaStore instances activated in the closest neighbours.

3. Finally, ReplicaStore distributed meta-aspects are continuously observing the state of the copies that they are keeping. If one of them detects that the original copy is not reachable, it re-inserts the wiki page, using a remote meta-advice **put**, in order to replace the Locator remote pointcut.

Consistency:

Based on the WOOT framework, we create the Editor (situated on the front-end side) and the Integrator (situated on the back-end side) distributed meta-aspects, which intercept the DHT-based calls to perform the consistency behaviour. Their task is the modification of the distribution behaviour, by adding the patch transformation in the edition phase, and the patch merging in the storage phase.

1. The Editor distributed meta-aspect owns a remote meta-pointcut (**edit**) that intercepts the return remote invocations from Storage to Locator instances. This mechanism stores the original value in its own session data. Obviously, in a similar way, the Integrator prepares the received information to be rendered as a wiki page.

2. Later, if the page is modified, a save method triggers the put mechanism, where another remote pointcut (**patch**) transforms the wiki page into the patch information, by using the saved session value.

3. In the back-side, the Integrator instance intercepts the put request, and **merge**s the new patch information with the back-end contained information. The process is similar to the original behaviour, but replacing the wiki page with consistent patch information.

4. In this setting, having multiple replicated copies leads to inconsistencies. We use the *antiEntropy* technique [21], in order to recover a log of differences among each page and its respective replicas. Using the **recover** remote invocation, the Integrator sends the necessary patches to be sure that all copies are consistent.

Finally, we summarize the UniWiki connections (network scheme in Figure 4.2) among the distributed aspects and meta-aspects:

- Distribution:
  - The **put** and **get** pointcuts are forwarded to the host responsible for the value associated with the current key, using the *hopped* abstraction.
  - The **return** of the requested value from the key owner to the frontend is done as a *direct* abstraction.
- Consistency:
  - The **patch** and **edit** meta-pointcuts of the Editor aspect are executed *locally*, on the frontend serving a wiki request.
  - Upon receiving a patch, the **merge** meta-pointcut of the Integrator aspect is executed *locally* on the node responsible for the key.
- Replication:
  - At the same moment, the **onPut** meta-pointcut of the Replicator aspect is executed *locally* on the node responsible for the key, and:
  - Forwards the new content using the **replicate** remote invocation as a *direct* call to all the *neighbours* responsible for the same key.
  - When a new host joins the network, or recovers from a problem, the Integrator aspect running on it will re-synchronize with a randomly selected replica, using the **recover** remote invocation as a *direct* call.

4.5 Validation

We have conducted several experiments to measure the viability of our UniWiki system. We have used Grid'5000. The Grid'5000 platform is a large-scale distributed environment that can be easily controlled, reconfigured and monitored. The platform is built with 5000 CPUs distributed over 9 sites in France. Every site hosts a cluster and all sites are connected by a high speed network.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

In this sense, we conducted the experiments using 120 real nodes from the Grid'5000 network, located in different geographical locations, including Nancy, Rennes, Orsay, Toulouse, and Lille.

When analyzing our large-scale system we have three main concerns: load-balancing in data dispersion among all nodes (distribution), failed hosts that do not affect system behaviour (replication), and the operation performance (consistency).

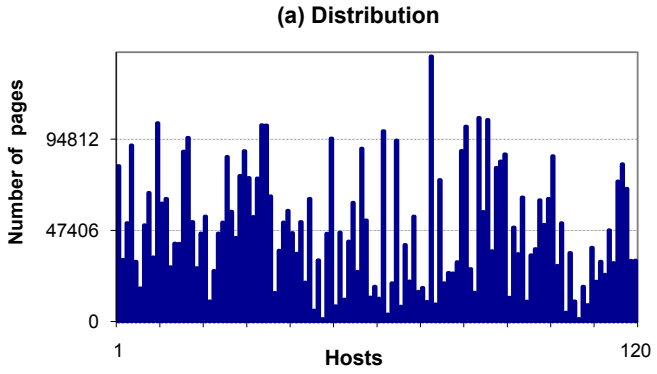

Figure 4.3: Empirical results - Distribution.

For distribution, we study the data dispersion in the network.

- *Objective*: demonstrate that when our system works with a high number of hosts, is able to store a real large data set of wiki pages, and that all the information is uniformly distributed among them.
- *Description*: create a network of 120 hosts, and using a recent Wikipedia snapshot, we introduce their 5,688,666 entries. The idea is that data distribution is uniform, and each host has a similar quantity of values (wiki pages).
- *Results*: we can see in Figures 4.3 that we have a system working and the results are as expected. Thereby, the distribution of data trends to be uniform. Results indicate that each host has an average of 47,406 stored wiki pages, and using an approximate average of space (4.24 KB) per wiki pages we have 196 MB, with a maximum of 570.5 MB (137,776 values) in one node.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- *Why*: Uniform distribution of data is guaranteed by the key-based routing substrate, and by the hash function (SHA-1) employed to generate the keys. However, with this number of hosts (120) the distribution values show that a 54.17% are near the average, and a 37.5% are over the double of the average. Furthermore, we can see similar results in simulations, with a random distribution of 120 nodes. For these simulations we have used PlanetSim [73]. PlanetSim is an object oriented simulation framework for overlay networks that also follows the Common API (Section 2.1.1.4).

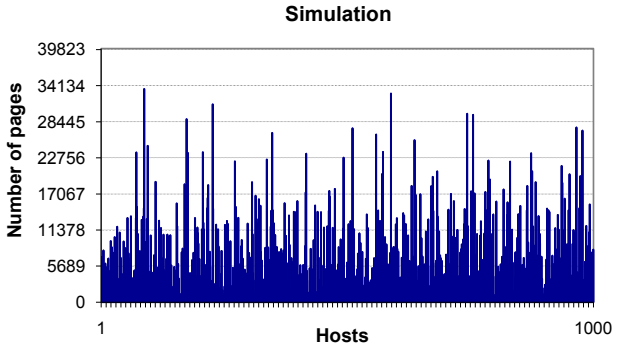

Figure 4.4: Wikipedia data distribution.

We make another simulation with 1000 nodes using the same DHT simulator. The results of this last simulation (Figure 4.4) shows that the values are: 63.7% over the average and 20.6% over double of the average, because with a high number of nodes the uniformity is improved.

Secondly, we study the fault-tolerance of our system platform.

- *Objective*: demonstrate that in a real network with failures, our system continues working as expected.

- *Description*: In this experiment we introduce problems on a specific fraction of the network. In this case, each host inserts 1000 wiki pages, and retrieves 1000 randomly. The objective of this test is to check persistence and reliable properties of our system. After the insertions, a fraction of

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

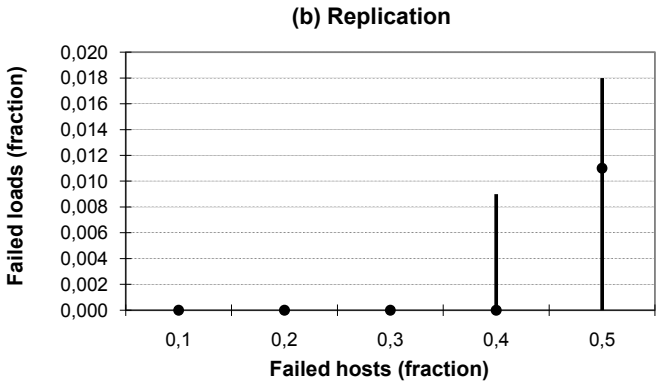

Figure 4.5: Empirical results - Replication.

the hosts fail without warning, and we try to restore these wiki pages from a random existing host.

- *Results*: We can see the obtained results in Figure 4.5. Even in the worst case (50% of network fails at the same time), we have a high probability (average of 99%) to activate the previously inserted wiki pages.

- *Why*: The theoretical probability that all the replicas for a given document fail is $\prod_{i=1}^{n} r_i = r_1 * r_2 * ... * r_n = (r)^n$ where n is the replication factor and r the probability that a replica fails. For instance, when r = 0.5, and n = 6, the result is $(0.5)^6 \approx 0.015 \approx 1.5\%$.

Finally, for the consistency concern we study the performance of our operations.

- *Objective*: demonstrate that our routing abstraction is efficient in terms of time and network hops.

- *Description*: similar to the previous experiment, we create a 120 hosts network, and each host inserts and retrieves 1000 times the same wiki page. The idea is that the initial value is modified, and retrieved concurrently. In this point, we make this experiment twice. The first time with consistency mechanisms disabled (only the put call), and the second time with these mechanisms enabled (patch and merge). In the last step, the consistency of the wiki page is evaluated for each host.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

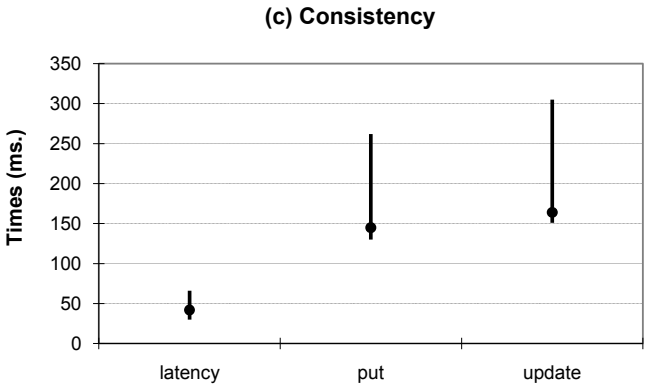

Figure 4.6: Empirical results - Consistency.

- *Results*: for this experiment, we found that the consistency is guaranteed by our system. We can see the operation times in Figure 4.6, and for each operation the number of hops has an average of 2. Therefore, the wiki page put operation has an average time of **145** ms, and an overall overhead of: **3.45**. For update operations the value is logically higher: **164** ms. with an overhead of **3.90**. Finally, the update operation overhead respects put operation, when the consistency operation is performed, is **1.13**.

- *Why*: Due to the nature of the Grid experimentation platform, latencies are low. Moreover, we consider that the operation overhead is also low. Finally, theoretical number of hops is logarithmic respect the size of the network. In this case, log (120 hosts) = 2 hops.

4.6 Summary

In this chapter we have presented the prototype of an efficient P2P system for transparently distributing the storage of wiki applications, which allows their extension to large-scale scenarios, called UniWiki.

In this setting, some decentralized wiki engines have been proposed such as DistriWiki [56], Wooki [97], DTWiki [23], Piki [57], or RepliWiki [76]. However, these approaches have one or more of the following drawbacks or requirement.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Table 4.1: Summary of drawbacks in P2P Wikis

Approaches	Drawbacks		
	total replication	ad-hoc client	consistency
DistriWiki	ν	ν	ν
Wooki	ν	ϕ	ϕ
DTWiki	ϕ	ν	ν
Piki	ϕ	ϕ	ν
RepliWiki	ν	ϕ	ϕ

- *total replication*: means that all hosts will contain an exact copy of all the contents of the network. Total replication is simple to implement, but is not scalable [85]. Therefore, wiki content is fully replicated on every host, which is not acceptable in the context of a huge wiki.

- *consistency*: some of them propose an unsatisfactory solution to concurrent modifications problems by either creating two distinct versions of the wiki page and delegating the merging task to users, or, by choosing a transactional approach, and rejecting unelected concurrent contributions.

- *ad-hoc client*: contributors install a specific rich client application in order to physically join and participate in the P2P network, and those clients have to use this application instead of a standard web browser to contribute or consult any wiki content.

The total replication and consistency drawbacks affect to scalability of the wiki application, and the ad-hoc client drawback to the transparency (access) of the system. Table 4.1 summarizes these drawbacks (legend is as follows: ϕ : not affected; ν : affected in some way).

In our solution, the scalability and transparency is ensured by a completely distributed and decoupled architecture, where each component is totally abstracted from the real wiki application and the client (i.e., web browser). Then, the distributed AOP middleware over P2P networks ensures the communication with the wiki application, which provides the presentation and business logic.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

At the storage level, we combine two intensively studied technologies, each one addressing a specific part: DHTs distribution and replication mechanisms, and the consistency algorithm (e.g., WOOT) ensures that concurrent changes are correctly propagated and merged for every replica.

Indeed, this proof-of-concept use all the services and benefits of our distributed AOP middleware, like source hooks to intercept wiki applications transparently, the meta-level approach for dynamic reconfiguration, or dynamic composition for encapsulate problems in three fully decoupled distributed concerns: distribution, replication, and consistency.

Validation of UniWiki has been conducted on the Grid'5000 testbed. We have proved that our solution is viable in large-scale scenarios, and that the system has acceptable performance. Our experiments were conducted with real data [100] from Wikipedia which include almost 6 million entries.

The initial prototype, freely available at `http://uniwiki.sf.net/`, was developed as a proof-of-concept project, using a simple wiki engine. We are currently working on refining the implementation, so that it can be fully applied on wikis with more complex storage requirements, such as XWiki [101] or JSPWiki [41].

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 5

Enabling Web Applications over Wide-Area Networks

In this chapter we introduce our second use case for the proposed distributed AOP middleware architecture. Particularly, this work introduce a solution for the challenge of availability provision in decentralized scenarios.

In order to address this problem, we present SNAP, which aims to be a large-scale web application deployment framework, for easy transitioning from client-server model to large-scale environments.

Our approach, instead of being a traditional cluster with replicated servers; it is an effectively large-scale platform where each server holds different applications running on top of it. As a consequence, our solution aims to be as generic as possible, thus supporting more dynamic environments.

Moreover, we have designed our architecture using distributed AOP, which transparently intercepts the most significant server methods. By using such a solution we achieve more elegant, modular, and suitable mechanisms than traditional alternatives.

5.1 Motivation

Web applications are currently associated by the general public to those applications which are accessible throughout the Internet. Most people may

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

consider that any web page is a large-scale application. However, this is not totally true. Even though the World Wide Web (WWW) itself is an application whose success transcends to the whole globe, but it mainly follows the centralized client-server model. This means that in most cases, only one server is backing up the whole web page or application, meaning that whenever the server comes down, access to the web page or application is impossible.

In this domain, we may suffer unpredictable situations like workload variations, server failures, and resource unavailability, among others. For example, when we are navigating the web, we may find the *Server is not responding* error message. We often retry again in a few minutes, and if we are lucky, we will be able to continue navigating. However, if we were in the middle of a transaction (e.g., filling a form), or later using an active session, we unfortunately would observe that our data is vanished. In this case, we can observe that one of the most used applications of the Internet is not fault tolerant.

Hence, it is true that the WWW is accessible to the whole world, but in a sense, if the whole world tries to use it at the same time, it is not accessible anymore. In this scenario, servers may stop serving requests if their network bandwidth is exhausted or their computing capacity is overwhelmed. Large-scale applications should be accessible efficiently at any time, and anywhere, by a massive number of concurrent users. As a consequence, *scalability* and *availability* (Section 1.1) are also two main challenges of web systems.

One way to deal with these challenges is to have several identical servers and give the user the option to select among them. This approach is simple, but it is not transparent to the client. An alternative is to rely on an architecture that distributes the incoming requests among these servers in an unobtrusive way. A usual solution to this problem comes in the form of clustering or federation of servers. Following a distributed pattern, servers are made redundant so as when one becomes unavailable, another one can take its place. Nowadays, many important websites operate in this way, but these replicated server alternatives are normally expensive to achieve and maintain.

As a matter of fact, the actual trend is to head toward decentralization like P2P or cloud computing [37] solutions. These models take advantage of the computing at the edge paradigm, where resources available from any computer

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

in the network can be used and are normally made available to their members.

5.2 Background

There exist many different non-AOP solutions to introduce availability in web environments. Some examples in this area include server mods or plugins, servlet filters, or ad-hoc frameworks.

- In this setting, there are a variety of server mods (e.g., load-balancers) that directly depend on the server implementation. Usually, these mods are difficult to bind to a specific server.

- Regarding servlet filters and server mods, the Java Servlet (specification version 2.3 [83]) introduces a new component type, called filter. A filter dynamically intercepts requests, before a servlet is reached. Responses are additionally captured after the servlet is left. This interception mechanism may transform either a request or a response content.

- Finally, one clear example of ad-hoc framework is WADI Application Distribution Infrastructure (WADI [96]). This approach aims to solve problems regarding the state propagation in clustered web servers. Thus, WADI provides several services useful for clustering on Java EE platforms. Nevertheless, its main drawback is that it needs wrapping extensions for each different server implementation and forth-coming versions.

Nevertheless, none of these proposals are suitable for a large-scale scenario. In addition, our proposal manages dynamic content and goes further than other approaches for P2P web hosting (e.g., YouServ [9]) or structured P2P content distribution networks (e.g., Coral [29]) that handle static content.

Other related projects, like YouServ, offer web hosting and content sharing over a P2P network of personal web servers. Although they focus on static content publishing, they also provide a lightweight plugin architecture for constructing small applications. However, YouServ provides a limited proprietary model for dynamic web applications since they are focused on static content distribution.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

On the other hand, Coral is a structured P2P content distribution network, which allows a user to run a web site that offers high performance and meets huge demand. It uses a P2P DNS layer that transparently redirects browsers to participating caching proxies, which in turn cooperate to minimize load on the source web server.

5.3 Approach

We foresee promising cross-fertilizations of distributed interception and web models in the next years. Although both models are already influencing each other, there is still a lack of seamless integration between them in order to achieve constructive synergies. Our approach aims to bring all the benefits of the distributed AOP to the mature and standardized world of web applications and services.

In this line, our infrastructure envisages a decentralized structured P2P network in which every peer hosts a lightweight web server. Using standard web application models (e.g., Java EE [87]), we permit distributed deployment of web applications and services in the network of peers.

5.3.1 SNAP 1.0

Application development over a P2P system is a complex challenge. The first approach [65] of the SNAP Project [`http://snap.objectweb.org/`] was born to achieve the convergence between P2P and WWW models. This previous SNAP version is a decentralized platform designed to provide Java EE applications in wide-area scenarios. Thus, it presents a novel proposal over a structured P2P network where all nodes are heterogeneous and they also dynamically join and leave the system. Thanks to this model we are able to easily transform a traditional client-server web application into a SNAP application with minimal changes providing worldwide scalability.

Following such lightweight scheme, all nodes host a modified copy of a lightweight server, which acts both as a P2P network client and server simultaneously. Therefore, clients can connect through their favourite web browser

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

to any node of the SNAP infrastructure for accessing any deployed web applications. In SNAP, every time an application is requested by any client, it is automatically downloaded from the P2P network, deployed and instantiated on the local web server. All accesses are local to that lightweight server. However, this only happens whenever no available instances of that web application are found already running on the network. In such case, the client is automatically redirected to the closest web server which hosts that application.

5.3.2 SNAP 2.0

Obviously, the previous project was a novel promising solution, but it has some limitations, that already exist in Java EE clusters (Section 5.2) like intrusivity (i.e. modification of web server implementation), stateless solution (i.e., no session tracking implemented), or static mechanisms (i.e., clustering based on a fix number of members).

For this reason, we have redesigned the project applying our distributed AOP middleware proposal in order to provide the necessary distributed aspects into the web model, which for itself is not suitable for large-scale scenarios. This approach provides a sample use case for our distributed AOP proposal. We emphasize how the services provided by our model are used to create this new distributed aspect application. In summary, this proposal consists of a web system which is to be transformed into a large-scale distributed system.

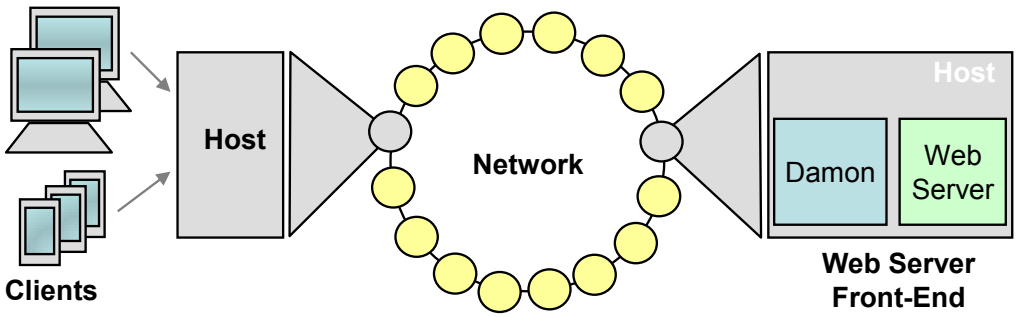

Figure 5.1: Adaptive Web System Infrastructure Overview.

As we can see in Figure 5.1, our idea is to have a large-scale network of web servers that are transparently interconnected via our middleware. In this sce-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

nario, clients are able to access any of the web servers, and use any desired web application. Our main aim is to avoid having identical server instances with a total replication of content (applications, user data, etc.). As a consequence, our objective is to apply more adaptive and dynamic techniques, more similar to the emergent cloud computing paradigm [37]. In this setting, web servers that belong to our infrastructure, initially do not have any predefined set of already deployed applications running on them. Instead, these applications are activated dynamically on demand [85].

5.3.3 Application Life-Cycle

Typically, the web server container is the responsible of web application life-cycle. In our approach, each host contains a web server and its corresponding container, but all of them work following a decentralized logic. In this section, we explain the life-cycle of a web application in the SNAP platform, phase by phase.

We will illustrate this section with a simple example of a dynamic web application. This sample is a bid application called *eshop* where sellers can publish their ads, and buyers can bid/buy the products.

5.3.3.1 Deployment

The deployment phase starts when uploading the web application into SNAP. In this phase, all hosts join the system main group (i.e., `p2p://snap.objectweb.org`) in order to keep connected. The process where web applications are inserted into the network occurs in a transparent way. This process intercepts the local web upload deployment application running on each host. This way, deployed web application data is not only stored on one unique host, but on several of them, by using Damon underlying persistence/replication DHT-based service. Therefore, applications are automatically replicated for fault tolerance.

5.3.3.2 Application Location

Once applications are deployed, any client, with a standard web browser, connects to any of the web servers that are currently available. In addition, clients access the desired web application, by using the corresponding SNAP application locator (e.g., `p2p://eshop.app.net`)). In the uniform web application location phase, SNAP internally redirects requests to its applications to the real IP addresses (which can change over time if nodes fail, new ones join, and so on). P2P locators make the address space uniform; they also make the application access independent of its real location (i.e., IP address) and the service provider. Once the deployed web application is located, the next step is to activate it.

5.3.3.3 Activation on Demand

Every time a client requests a new application, SNAP automatically downloads it from the network (i.e., Damon persistence service), deploys it, and instantiates it on the originator web server locally. If the client requires other services, they are also activated on demand. This activation means the deployment of the web application in the web server, and the creation of a new P2P group for this application.

Note that this only happens when SNAP is not able to find available, active instances of the web application already running on the network. This activation on-demand phase becomes more complex when the network already has active web application instances. Moreover, each application forms its own group of instances (e.g., `p2p://eshop.app.net`).

5.3.3.4 Application Execution

In this last phase, where the application is running, new concerns are involved. We focus on three important concerns related to the availability property of the system. Particularly, the main distributed concerns that we aim to solve are: the workload distribution for service availability, and the session tracking and the global context for data availability.

Taking the sample of *eshop*, the application is deployed and activated on

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

the SNAP platform, but we need to guarantee its availability on the system. If an instance has problems or is too popular, it can become overwhelmed. Therefore, we need mechanisms to load balance the workload of its instances.

This decentralization affects the data of the clients and applications. First, each client owns session with its validated profile and its shopping cart. If they are redirected to other server, a mechanism to restore transparently the corresponding session data is needed. Secondly, each application in SNAP has its own context data, as for example, in *eshop* each ad has its own counter that calculates all the views of this ad with a global value (the same for all the distributed instances). Again, a decentralized concern related to availability appears.

Finally, we have developed the solution for these concerns in the next section.

5.4 Implementation

In order to comply with the explained requirements (i.e., availability provision), we have designed an adaptive large-scale web system, which uses the services provided by our distributed AOP middleware (Chapter 3).

As we can see in Figure 5.2 we separate the Host entity presented in Figure 5.1 in three tiers: server, distributed concern, and meta-level. This logical layout allows only interconnection with the closest tier, where the server is intercepted by distributed aspects, and distributed aspects are intercepted by distributed meta-aspects.

We have mainly composed three distributed concerns:

- *Workload Distribution*: that means **load-balancing** for client requests, dynamic **activation** of web applications, and **management** of distributed applications running on specific web servers.
- *Session Tracking*: which is performed via **distribution** and **replication** of session attributes of each client-application domain.
- *Global Context*: it is the **distribution**, **replication**, and **caching** of the application scope attributes of the global servlet context.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

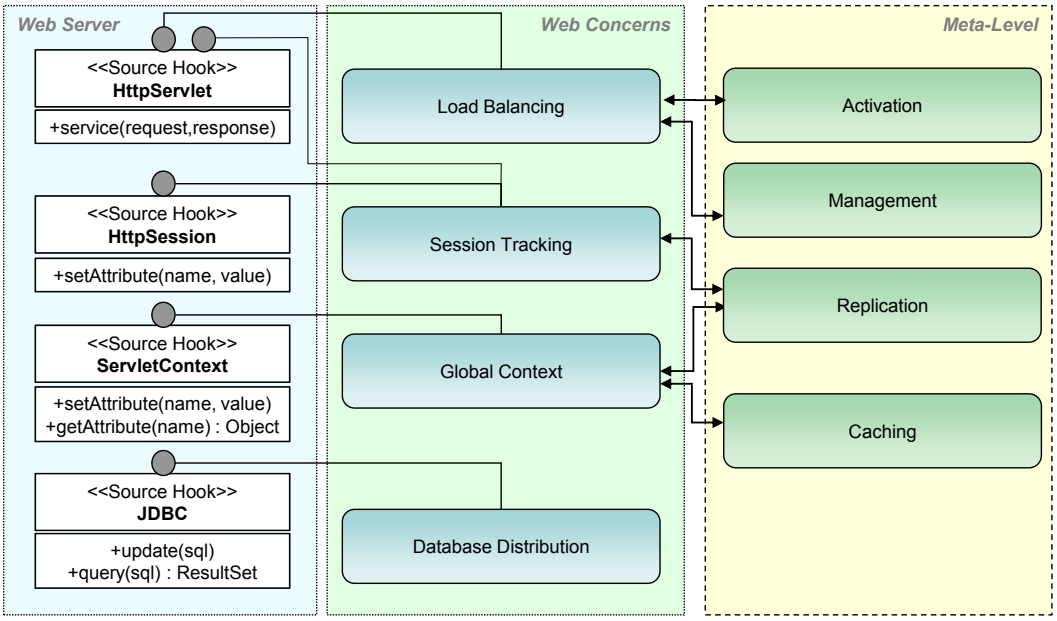

Figure 5.2: SNAP Distributed Aspects Layout.

Note that our approach design covers the tier of the application servers that includes the database tier. However, we do not provide an elaborated solution to scale the database tier, which is another active and prolific area of research [46]. The previously presented example in chapter 3 of database distribution SourceHook (Figure 3.3) and the descriptor of the Locator distributed aspect (Figure 3.6) shown the way to apply new distributed concerns, leaving the door open to possible future work in this area.

In conclusion, by implementing these distributed concerns modelled as distributed aspects, we are able to provide access to web applications in a large-scale scenario. Moreover, we also obtain a solution with the transparency, decoupling, and reconfiguration benefits. In the rest of this section we describe in detail how this web system has been implemented, and how it benefits from the services of our implicit middleware solution.

5.4.1 Workload Distribution

It is well known that a web server has limits to support client demands. Thus, a web server has a well-defined workload threshold, because it may handle only

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

a limited number of concurrent client connections per address. Additionally, servers are able to provide a certain maximum number of requests per second depending on, for example, its own settings, HTTP request type, static or dynamic content, caching, or server system limits. Whenever a web server is near to or over its limits, it becomes overloaded and thus unresponsive.

Examples of the symptoms of an overloaded web server are: requests fulfilled with long delays, HTTP specific errors returned to clients, or client connections being refused or reset before any content is sent to them.

Load balancing is a technique to distribute process and communication activity evenly across a computer network in order to improve the global performance and no single device is overwhelmed. Such feature is especially important for distributed systems where it is apparently difficult to predict the number of requests that will be issued to any server. For instance, busy web sites typically employ two or more web servers in a load balancing scheme. If one server starts to get swamped, requests are forwarded to another server with more capacity.

5.4.1.1 Solution Proposal

Figure 5.3 shows how the composition of our adaptive web system maps onto the complex interactions on the network, and among the distributed aspects. We can also observe how the Load-Balancing entity is a composite distributed aspect that consists of two primitive distributed aspects. The first entity (Redirector) is responsible for capturing and redirecting client requests, and the second one (Monitor) performs host state and application instance monitoring. This is our web system main building block as a distributed concern.

Nevertheless, our system would not be as adaptive as desired if it were not for the other distributed concerns working together, namely Activation and Management. Without such concerns, our load balancing system would not be able to decide *whether* and *how* to redirect client requests to other hosts.

However, such decisions would be rather limited because of the inherent lack of knowledge about the already running instances, and their current state. As a consequence, we add these complementary distributed concerns, as distributed meta-aspects.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

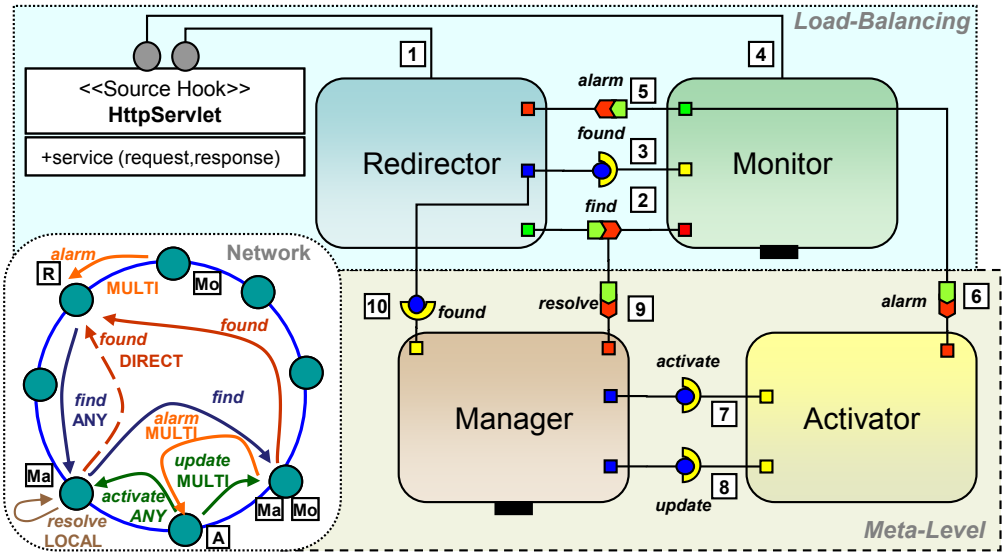

Figure 5.3: Workload Distribution Diagram.

We use the meta-level approach in this scenario, because these distributed concerns (Activation and Management) are linked to the distribution concern (Load-Balancing). In addition, this approach allows a completely decoupled integration between previous and new concerns, among other benefits (see Section 3.2.3).

Therefore, the first of these meta-level entities is the Activator, responsible for maintaining the adequate number of running web application instances in order to satisfy their current demand. The second entity is the Manager, which helps selecting the best candidate (host) among the activated instances depending on some policy.

We now proceed to describe system execution step by step, focusing on the services provided by our distributed AOP middleware.

The first actor involved is the **Redirector** distributed aspect, which is responsible for listening to and redirecting client requests. Redirector instances intercept client requests through the (1) **HttpServlet** SourceHook. Note that such interception is declared as a Static Source Hook since its mission is to intercept all servlet requests, and such task could suffer high demand peaks. If we did not filter these requests, each of them would trigger a remote point-

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

cut. By using this approach, we avoid unnecessary overflow problems in the middleware services.

After this phase, the request is propagated through the (2) **find** remote pointcut. Since all instances of a specific web application form a group, we use the *any* abstraction to propagate the call to the closest group member. Its task is to resolve if another host can satisfy the client's demand. This request is completed when any host of the group satisfies such condition. Afterward, this host notifies the source host through the (3) **found** remote invocation.

Obviously, in the initial case where no instances of the application are active, this request fails. The Redirector forces a local application activation, just to satisfy the current client's demand. Therefore, subsequent requests for such application will be able to find at least, this already running instance.

Additionally, there is a **Monitor** distributed aspect running on each host. Its main task is to observe and evaluate the state of the host, thus obtaining resource information (CPU, memory, number of threads, etc.) through reflection capabilities, and to analyze request demand using the (4) **HttpServlet** Source Hook on servlet requests. The Monitor continuously calculates a numerical estimation. If this estimation surpasses an established threshold, it then throws an (5) **alarm** remote pointcut. This alarm notifies all web application groups the host is member of in order to avoid any further redirections. Additionally, once this invocation occurs on the same host, the local Redirector starts migrating client requests (the *whether* condition) to other web servers.

Until now, this approach offers a basic functionality. Similar to a clustering solution, which involves total replication. In this scenario, our approach can work flawlessly having a fixed instance number per application group. Nevertheless, in dynamic environments, application instances need to be activated and passivated on demand to preserve resources and provide scalability (e.g., cloud computing). In this setting, we need to add more distributed concerns in a transparent way by means of the meta-level mechanisms explained in the Section 3.2.3. From the meta-level perspective, we are able to capture local and remote interactions between Redirector and Monitor distributed aspects, and transparently map these calls onto new interactions.

In dynamic load-balancing solutions, it is also necessary to be able to

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

change policies at runtime. Examples of these policies are the random, round-robin, weight-based, least recently used (LRU), last access time, or minimum load. These policies can be established mainly by the Redirector decisions. However, while the *whether* decision is made using the Monitor information, the *how* decision is made by using the proximity-aware policy by default. For this reason we introduce a **Manager** entity, which is able to change this fixed strategy and decide the best candidate among application instances, following a specific policy (e.g., the host with minor workload).

An **Activator** distributed meta-aspect is activated in at least one host of each application group, in order to be aware of host state changes. For this reason, the Activator is subscribed to the (6) **alarm** propagation from the Monitor. Following a specific activation policy and using the knowledge of the hosts state, the Activator is able to modify the number of instances for each web application. As a consequence, when an Activator decides to activate or passivate any application in any specific host, it sends the (7) **activate** remote invocation to the Manager on this target. Finally, this activation activity is notified through the (8) **update** remote invocation to each Manager on the application group using the *multi* abstraction.

Concurrently, the Manager intercepts the Redirector **find** requests using the (9) **resolve** around remote meta-pointcut. In this process, the Manager evaluates the requests using the collected global knowledge. If the evaluation is positive, and the Manager obtains a reliable candidate, it cancels the original request. Finally, in substitution to the original response, it uses the (10) **found** remote meta-advice to send the host information to the Redirector on the initial host. The Redirector remains unaware of this process, and obtains the requested information with the best match possible and in only a few hops.

5.4.2 Session Tracking

Sessions are commonly provided in client-server environments, either to impose security restrictions, or to encapsulate other runtime state information, or for both these reasons. The problem is to identify which requests belong to which session, since the HTTP protocol for web access is not connection

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

oriented. This means that each request is independent of any previous or following requests that actually do belong to the same session.

Moreover, in cluster environments, HTTP sessions from a web server are frequently replicated in other few servers. Session replication is expensive for an application server, because session request synchronization usually is expensive. Therefore, the problem we want to solve is session tracking for stateful applications.

Certainly, we need to use session migration when a host has been shutdown (e.g., it crashes) or a load balancer decides to redirect the client to a different host. A first approximation to manage sessions may be having one active session in each application server instance. Nevertheless, there is an imposed limitation on the stored cookies that a client may have: browsers are expected to support 20 cookies for each web server, totalling 300 cookies of 4 KB each. Therefore, the browser capacity limits the number of simultaneous accesses to applications which hold several instances in different servers. Also, this solution is causes inefficiency on the server, because it has to manage several inactive sessions.

In a second approximation, we may have a SNAP instance running in our localhost (i.e., applying a proxy pattern). Consequently, our server invokes servlet services in other server instances while maintaining the session state itself. This is an easier method but it forces us to join the P2P network as a server. Such approach may be acceptable since we are using the P2P paradigm, and therefore we are compelled to share our resources. Nevertheless, if we are to provide support for any lightweight devices or low-powered machines, it is preferable that they do not need to join the P2P network obligatorily.

5.4.2.1 Solution Proposal

As a consequence, we have designed a more compact and generic solution that overcomes such limitation. This solution is based on Damon and it also uses distribution and replication concerns, and URL rewriting strategy [47]. For session DHT persistence we use its session ID to identify it. This approach has a structural problem though, because session ID is only considered to be unique in the original host, but this is not applicable to whole network. Therefore, we

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

need to replace the session ID generator by means of intercepting the session creation code.

Nevertheless, this solution forces us to intercept web server specific code. It also invalidates our decoupled and server-abstracted architecture (see Figure 5.2). For this reason, we decided to attach the server host name to the session ID when we store and retrieve its data on/from the network. Therefore, this new ID will not collide with any other one in the global space ID.

Once our session data is accessible throughout the network, we need a way to re-store it whenever a new server becomes responsible for that client. The idea is to have meta-information that identifies the session directly embedded into the URL. This technique is known as URL rewriting, and is used by many systems like content distributed networks (e.g., YouTube [102] or Akamai [4]) to identify its resources.

Usually, it is used for a variety of purposes, as for example, making URLs more compacted and compressible, or preventing undesired hot linking behaviours. Moreover, URL rewriting was further used in the past for stateful applications to replace the cookie mechanism for those browsers which do not support or accept cookies.

We mainly use URL rewriting to report the client session ID to other servers. URLs are modified before fetching the requested item, attaching the session ID like a usual request parameter. For instance: `http://hostname:8080/appdomain/index.jsp`?JSESSIONID=08445a31a78661b5c746feff39a9db6e4e2cc5cf.

Algorithm 1 shows the SessionTracking Distributed Aspect behaviour. Before requests are made, the servlet service method is executed. The *manageSession* method checks whether the session ID is among the request parameters in order to restore previous session information from the network. If it is found and it is not the initial server, such remote session data is recovered via the *recover* remote-pointcut into the new local server session. Lastly, if there is no session parameter with an active session, then this method attaches the JSESSIONID parameter via the *sendRedirect* servlet method. With this mechanism, the request is self-redirected to the same host but completing its URL. Note that getSession method returns the current session associated with this request

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Algorithm 1 *manageSession*
Input: *req* /* *HttpRequest* */
Input: *res* /* *HttpResponse* */

```
1: sid ← req.getRequestedSessionId()
2: param ← req.getParameter('JSESSIONID')
3: if sid = null then
4:    sid ← param
5: end if
6: if sid ≠ null then
7:    if param = null then
8:       url ← req.getRequestedURL()
9:       url ← attachSessionId(url, sid)
10:      res.sendRedirect(url)
11:   end if
12:   session ← req.getSession(false)
13:   if session = null then
14:      session ← req.getSession()
15:      data ← invoke(sid)
16:      restoreSession(session, data)
17:   else
18:      if sid ≠ session.getId() then
19:         data ← invoke(sid)
20:         restoreSession(session, data)
21:      end if
22:   end if
23: end if
```

or, if there is no current session and create is true, returns a new session.

Finally, we have to take some considerations into account, as for example, if the client leaves the session and comes back via a bookmark or link, the session information may be lost or expired. In such case, there is no problem because the stored session information will timeout and will be deleted from the system, therefore starting a brand new session.

5.4.3 Global Context

The last distributed concern that we want to resolve in this chapter is the global context problem. In this decentralized scenario, we found that the

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

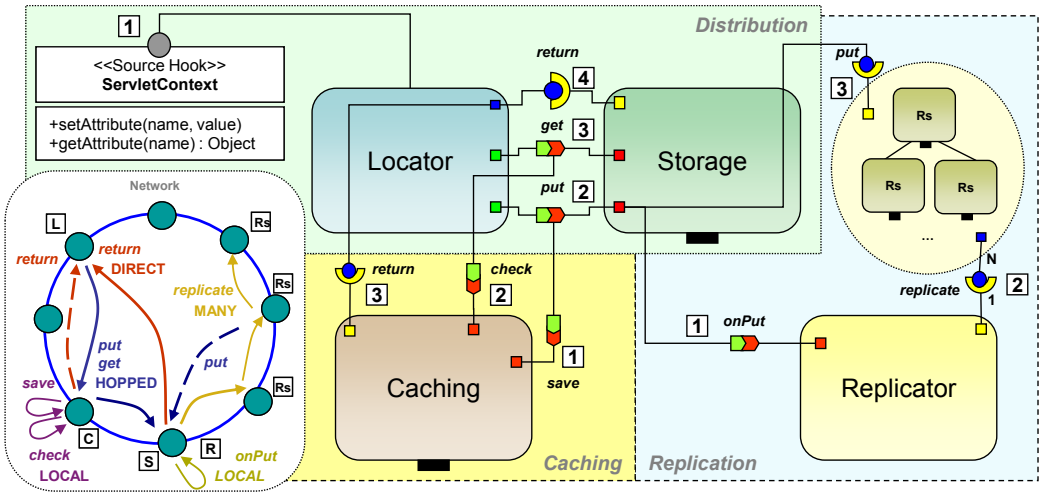

Figure 5.4: Global Context Diagram.

application-level data must be disseminated to the group of servers. In order to move these data to a global context scenario, we apply some distributed concerns, namely distribution, replication, and caching.

The idea is to intercept the context methods (i.e., setAttribute/getAttribute). These methods basically store/retrieve an attribute locally. In addition, attribute names should follow the same conventions as package names.

For this purpose, we have implemented three decentralized crosscutting concerns, namely distribution, replication, and caching. Thanks to our modular architecture we could partially reuse the distribution and replication concerns (Section 4.4). Following such approach we simplify the integration with any application. We now describe the global registry execution step by step as shown in Figure 5, focusing on the services provided by our distributed AOP composition model. Distribution

1. The starting point of this application is the servlet context interface used by the web servers locally. We therefore introduce a Source Hook that intercepts the setAttribute and getAttribute methods. The interface defined by the source hook follows the same structure.

2. Afterwards, the Locator distributed aspect is deployed and activated on all members of the P2P group of the application. Its main objective is

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

to locate the responsible node of the local insertions and requests.

3. These local executions are propagated as remote pointcuts (locateAttr). Consequently, the remote pointcuts are routed to the key owner node, by using their name to generate the key through the hopped abstraction.

4. Once the key has reached its destination, the registered remote advices are triggered on the Storage distributed aspect running on the owner host. This distributed aspect has already been activated on start-up on all members of the registry group.

5. For requester methods (getAttribute), the idea is basically the same, with the Storage receiving the remote pointcuts. However, it propagates later an asynchronous response using the method invocation (returnAttr) with the direct abstraction.

6. Finally, the attribute values are returned to the Locator originator instance, using its corresponding remote method.

Once we have the application running, we may add new functionalities as well. In this sense, we introduce distributed meta-aspects in order to extend and modify the current application behaviour in load-time or in runtime. More specifically, these distributed meta-aspects add the mechanisms of replication and caching concerns, as shown right away.

1. When dealing with the setAttribute method case, we need to avoid any data storage problems which may be present in such dynamic environments as large-scale networks. Thus, data is not only to be stored on the owner node, because if this host leaves the network for any reason, its data would surely become unavailable. For this reason, we activate the Replicator distributed meta-aspect in runtime, which following a specific policy, tries to address this problem.

2. The Replicator has an after remote meta-pointcut (replicate) that intercepts the Storage requests from the Locator service in a transparent way.

3. Thus, when an object insertion arrives to the Storage entity (locateAttr), this attribute value is to be replicated in the closest Replicator instances by using the many abstraction.

4. Replicators are continuously observing the state of the copies that they are keeping.

5. Eventually, if one of them detects that the original copy is not reachable, it reinserts the object again, using a remote meta-advice (insertAttr) in order to replace the original remote pointcut (locateAttr).

The Caching distributed meta-aspect is activated on-demand in the host where there are a high traffic of put/get remote-pointcuts (between Locator and Storage distributed aspects). This is because the main function of this entity is to accelerate the system request. In this case, once the Caching distributed meta-aspect is activated, it dynamically monitors the remote service activity. Afterwards it stores the required information about service interactions.

1. The Caching instance owns an around remote meta-pointcut (*capture*) that intercepts the *put* remote pointcut from Locator to Storage distributed aspects. The idea is that Caching entities store cache values on the most transited hosts, using its own generated reports about traffic on key-value routing paths. In this way, Caching obtains the value insertions that travel throughout its host. In this manner, a temporary copy can stay in cache during a specific time frame.

2. Subsequently, Caching also intercepts the get remote pointcuts, though the *check* remote meta-pointcut. During the *check* execution, it can decide if the original *get* remote pointcut can continue to be routed or not. Thereby, it is here where it verifies if the key parameter of a request has a cached value available.

3. Finally, if this query is satisfactory then the *get* remote pointcut stops being routed, and the result is sent back directly via remote meta-advice (*return*) that substitutes the original *return* remote method from the Storage distributed aspect.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

5.5 Summary

In this chapter, we have presented another proof-of-concept for our distributed AOP middleware: the SNAP framework. SNAP provides scalability and availability of Java EE compatible applications in large-scale environments.

This approximation is a clear example at how a large-scale application can be modelled by adding non-intrusive distributed concerns to the original application or system. Moreover, we can observe that all the potential benefits introduced in previous chapters are present in this proof-of-concept too. For example, composite distributed aspects that encapsulate high-level concerns (e.g., load-balancing), or distributed meta-aspects (e.g., management) that are working behind the scene, and can be changed in runtime. This hot reconfiguration process occurs painlessly without needing to alter the system behaviour.

Regarding reusability, it is important to outline that our contributions may be applicable to other distributed web platforms. Although context, session, and workload distribution solutions are somehow related to our distributed AOP middleware implementation (Damon), we have designed it as a generic and portable infrastructure for web systems.

In addition, the client's experience and usability when browsing web applications is improved, since load balancing, activation, replication, and the other concerns run in the background transparently. As a consequence, clients remain unaware of server problems like server saturation or session loss.

Finally, we are researching new distributed concerns in web scenarios. As we have explained, distributed aspects can easily be installed into the SNAP network, thus we can apply new concepts to this field. These concerns can be transversal services like transactions, security, or synchronization, among others. Moreover, these additional mechanisms could be implicitly distributed and composed since they benefit from the distributed AOP middleware inherent properties.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 6

Conclusions and Future Work

The main contribution of this dissertation is the design, implementation and experimentation of distributed AOP middleware based on P2P and dynamic AOP substrates. Other contribution is the support for *scalable*, *available*, and *transparent* distributed applications and middleware on large-scale scenarios. We have designed, implemented, and validated several software engineering and execution platforms that include composition techniques, application development, and middleware platforms.

In this chapter, we will draw the conclusions, and discuss the contributions of our work. Finally, we suggest the new directions in which research on large-scale distributed application development could evolve.

6.1 Conclusions

The development of a distributed implicit middleware platform for large-scale scenarios is a complex task. The separation of concerns principle, for instance, addresses a problem where a number of concerns should be identified and completely separated (without dependencies). Aspect Oriented Programming (AOP) is a modern paradigm that increases modularity by allowing the separation of crosscutting concerns. In addition, dynamic AOP allows less interdependence between the aspects of software architectures in runtime. However, these solutions do not take into account separation of distributed concerns

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

(e.g., load-balancing). This dissertation presents the design and implementation of a novel distributed AOP middleware that support the development of distributed concerns in large-scale scenarios.

Distributed AOP is a novel and promising paradigm that introduces distributed interception in these scenarios. It defines many new concepts like remote pointcuts, which are similar to traditional remote method calls, since the execution of interception code is performed remotely; component-aspects, which try to merge the component-oriented and aspect-oriented worlds; and aspect group notions. Thus distributed AOP establishes a context where aspects can be deployed in a set of hosts.

Nevertheless, as far as we are concerned, there exist no approaches in distributed AOP that fulfill large-scale requirements satisfactorily. Thus, necessary services are not implicitly provided or even are inexistent in similar approaches. In this dissertation we have analyzed already existing solutions which try to accomplish this objective, and we have stated that none of them elegantly achieve their goal.

With our work, the main goal is to provide an application with new abilities in the form of distributed concerns in the easiest and transparent possible way. Therefore, our proof-of-concepts have obtained scalability and availability properties in a transparent way though distributed aspects, which can be deployed and executed in large-scale scenarios.

6.1.1 Contributions Revisited

In this section we will show a digest summary of our contributions (presented in Section 1.3) in this dissertation as follows:

- Distributed composition model for distributed aspects:
  - First contribution is the *encapsulation* of *distribution aspects* from distributed applications in completely separated and encapsulated true distributed entities.
  - Second contribution: is the definition and implementation of a *distributed meta-level* model, which enables a distributed meta-aspect

entity, and its remote meta-pointcut and remote meta-advice connections.

– Third contribution: is the *runtime reconfiguration* of distributed aspects, which take advantages from the decoupled nature of the event-based connection model and the reflection techniques.

- Deployment of distributed aspects in large-scale environments:

  – Fourth contribution: is the *decentralized container* that offers location and discovery services, and provides the distributed aspect life cycle.

  – Fifth contribution: is a set of *decentralized functionalities and abstractions* for distributed aspects communication, persistence, and reflection.

- Viability and applicability of our middleware proposal:

  – Final contribution: the viability of our model is validated with our prototype implementation (Damon), and its experimentation in real large-scale networks like PlanetLab. On the other hand, the applicability is demonstrated with proof-of-concepts: a collaborative wiki system (UniWiki), and a decentralized web platform (SNAP). These projects benefit directly from our proposal, integrating new distributed aspects suitable for large-scale scenarios.

6.1.2 Why Distributed AOP?

After all this research, we conclude that the distributed AOP paradigm presents an excellent solution to deal with distributed concerns in a transparent way. In this dissertation we extend the ideas exposed in previous works to large-scale scenarios, where we need to solve the problems generated by scalability and availability requirements.

However, like every single innovation in computer science, distributed AOP has its own advantages, and its open challenges. Therefore, in the rest of this

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

section we expose the advantages of distributed AOP from our vision, and the challenges that need to be further investigated.

6.1.2.1 Advantages

- **Non-Intrusive**: as we have explained in the first chapters, there exists no other paradigm that permits developers to separate distributed concerns at this high level of cohesion. This advantage allows developers to apply new distributed concerns like scalability, availability, and transparency to existent or new systems. In this work we demonstrate the benefits of combining advanced interception (i.e., AOP) and efficient distribution (i.e., P2P) technologies.

- **Flexibility**: distributed aspects are easily composed, maintained, and supported, because separation among them is clean and understandable. Moreover, developers can apply distributed AOP to transparently decouple the parts of a system. And, for example, they can replace some legacy parts with third-party alternatives. Therefore, these distributed concerns are reusable among other applications, or versions of the same one, in a simple way.

- **Applicability**: a wide variety of middleware architectures and applications can benefit easily from this paradigm. These systems can apply the principle of separation of concerns in a distributed way. As a consequence, we can reuse other concerns easily, and apply them to our design or implementation.

6.1.2.2 Challenges

- **Popularity**: as we have seen in the background of this dissertation, only a few research works are produced in this area. Indeed, a lot more of work is needed to make the Distributed AOP paradigm more popular. Clearly, developers need to know the advantages and applicability of this paradigm. This framework has to provide (i) a powerful remote pointcut language that is preferably extensible, to incorporate domain

specific pointcuts, and (ii) an advanced runtime support for deployment and activation of distributed aspects.

- **Complexity**: AOP languages have associated a non-trivial learning curve, due to its novel syntax and because they are often focused on complex problems. In addition, developers need a specific and detailed knowledge of the application or system in order to intercept its code. This knowledge is used to determine the appropriate hooks for interception and the expected behaviour that these interceptions can produce.

- **Conflicts**: deploying distributed aspects on large-scale scenarios might trigger some conflicts:

  - deploying a distributed aspect might raise conflicts among subsystems and dependencies among running distributed aspects. More difficulties can be found when multiple class loaders are involved.

  - security concerns, specially in enterprise scenarios, need to be controlled, where authority is required for secure deployment and activation of distributed aspects.

6.2 Future Work

In this research work, we have aimed to propose a large-scale middleware approach that is generic enough to be used in any of the decentralization and interception paradigms available. Therefore, we think that our ideas are applicable independently of the underlying infrastructure.

Moreover, we expect this work to be continued in the future, and hope that these ideas can be exploited in different application domains. In particular, we believe that this dissertation opens the way for other lines of future work:

- **Distributed Patterns**

  Design patterns [30] capture successful solutions to recurring problems and are used both to document and to improve the design of software systems. Moreover, in the last year some works like [36] have applied

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

these patterns in a transparent way using AOP. Nevertheless, design patterns maybe have not been widely adopted on the distributed systems arena.

We believe that distributed AOP is a very good area to implement distributed design patterns. As a consequence, new works are emerging, and we believe our model can be extensively used to accomplish new future achievements in this line.

One example is [60], where the authors try to argue that the lack of flexibility of pattern definitions is what they consider the major impediment in distributed environment. For this purpose, they introduce the notion of *invasive patterns* that allow modularization of crosscutting enabling conditions of traditional distributed communication patterns.

- **Autonomic Computing**

  For a system to be considered to be autonomic [43], it must be *self-configurable*, meaning that it must allow for automatic configuration of components; *self-healing*, meaning it should provide automatic discovery, and correction of faults; *self-optimizing*, meaning that resources should be automatically monitored and controlled to ensure the optimal functioning; and, finally, *self-protecting*, meaning that it must allow proactive identification and protection from arbitrary attacks.

  In [33] the authors list a set of properties to implement autonomic systems over a suitable dynamic AOP framework: apply adaptations dynamically and remove easily, encapsulate adaptations, specify relationships, implement fine grained changes, and apply adaptations to various points in a system.

  In this line, we propose new requirements for implementing autonomic systems: apply and remove adaptations remotely, distributed adaptation container, host dynamic linkage (e.g., P2P locator), and several adaptation scopes (e.g., any abstraction). In order to satisfy these requirements in large-scale environments, we believe that these adaptations can be implemented via our distributed AOP middleware proposal.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

- **Cloud Computing**

  The *cloud* [37] concept is a metaphor for the Internet, based on how it is typically depicted in distributed system diagrams, and can be defined as a group of virtualized host resources. Moreover, developers do not need to take care about the underlying infrastructure *in the cloud* that supports them.

  In cloud computing, resources are dynamically scalable and often virtualized. These resources are provided like a service over large-scale networks. In addition, this paradigm allows scalable deployment mechanisms through the quick provisioning of virtual hosts or physical machines (i.e., virtualization).

  Indeed, many cloud computing infrastructures depend on Grid facilities, but cloud computing can be seen as a natural next step from the P2P services model. Moreover, the combination of distributed AOP interception technologies and P2P large-scale networks can contribute to the future scalable and flexible cloud architectures.

  In this sense, the main difference between our interception approach (i.e., distributed AOP) and virtualization, is the scope. While interception is focused in the code at the application level, virtualization will be focused on some resource completely (e.g., virtual machine, or operating system). As a conclusion, we can determinate that our solution uses a fine-grained approach, and the virtualization mechanism is more suitable for general purposes.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Chapter 7

Publications

In this chapter we outline all the publications related to this thesis. Our publications vary from national and international conference proceedings as well as national and international publications in journals and magazines.

1. Gérald Oster, Rubén Mondéjar, Pascal Molli, and Sergiu Dumitriu. Building a Collaborative Peer-to-Peer Wiki System on a Structured Overlay. *The International Journal of Computer and Telecommunications Networking (Computer Networks, Elsevier).* Spring 2010.

2. Oscar Ardaiz, Luis Manuel Diaz de Cerio, Jose Andres del Campo, and Rubén Mondéjar. Sharing Application Sessions for Peer-to-Peer Learning. *International Conference on Social Computing (SocialCom09), Workshop on Social Computing in Education (WSCE 2009).* Vancouver, Canada, August 2009.

3. Gérald Oster, Pascal Molli, Sergiu Dumitriu, and Rubén Mondéjar. UniWiki: A Collaborative P2P System for Distributed Wiki Applications. *19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2009).* Groningen, Netherlands, June-July 2009.

4. Rubén Mondéjar, Pedro García, Carles Pairot, Pascal Urso, and Pascal Molli. Designing a Distributed AOP Runtime Composition Model.

24th Annual ACM Symposium on Applied Computing (SAC), Software Engineering Track. Honolulu, Hawaii (USA). March 2009.

5. Gérald Oster, Pascal Molli, Sergiu Dumitriu, and Rubén Mondéjar. UniWiki: A Reliable and Scalable Peer-to-Peer System for Distributing Wiki Applications. *Research Report RR-6848, LORIA – INRIA Nancy Grand Est (France)*. February 2009.

6. Rubén Mondéjar, Pedro García, Carles Pairot and Antonio F. Gómez Skarmeta. Building a Distributed AOP Middleware for Large Scale Systems. *7th International Conference on Aspect-Oriented Software Development (AOSD 2008), Next Generation Aspect Oriented Middleware Workshop (NAOMI 2008)*. Brussels, Belgium. March - April 2008, pp. 17-22. ISBN: 978-1-60558-148-4/08/04.

7. Rubén Mondéjar, Pedro García, Carles Pairot and Antonio F. Gómez Skarmeta. Adaptive Peer-to-Peer Web Clustering using Distributed Aspect Middleware (Damon). *7th International Conference on Web Engineering (ICWE'07). Workshop on on Adaptation and Evolution in Web Systems Engineering (AEWSE'07)*. Como, Italy, July 2007. CEUR Workshop Proceedings, ISSN 1613-0073.

8. Carles Pairot, Pedro García, and Rubén Mondéjar. Deploying Wide-Area Applications with a SNAP. *IEEE Internet Computing Magazine. Vol. 11, No. 2*. March/April 2007, pp. 72-79. ISSN: 1089-7801.

9. Rubén Mondéjar, Pedro García, Carles Pairot and Antonio F. Gómez Skarmeta. Damon: a Decentralized Aspect Middleware Built on top of a Peer-to-Peer Overlay Network. *14th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 2006). Workshop on Software Engineering and Middleware (SEM 2006)*. Portland (Oregon), USA, November 2006. ISBN 1-59593-585-1.

10. Rubén Mondéjar, Jordi Pujol, Pedro Garcia, Carles Pairot. Sistemas multi-agente en entornos p2p. Departament d'Enginyeria Informàtica i Matemàtiques. *Technical Report DEIM-RR-06-002*. June 2006.

11. Rubén Mondéjar, Pedro García, Carles Pairot and Antonio F. Gómez Skarmeta. Enabling a Wide-Area Service Oriented Architecture through p2pWeb Model. *15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2006).* Manchester, United Kingdom, June 2006. ISSN 1524-4547 ISBN 0-7695-2623-3. *Workshop's Best Paper and Presentation Award.*

12. Rubén Mondéjar, Pedro García, Carles Pairot and Antonio F. Gómez Skarmeta. Tracking the evolution of Collaborative Virtual Environments. *Upgrade Vol. VII, issue no. 2, Visual Environments.* April 2006. ISSN 1684-5285.

13. Rubén Mondéjar, Pedro García, Carles Pairot y Antonio F. Gómez Skarmeta. La evolución de los Entornos Virtuales Colaborativos. *Novatica. No. 180, Monografía Entornos visuales.* Abril 2006. ISSN 0211-2124.

14. Rubén Mondéjar, Pedro García, and Carles Pairot. Bunshin: DHT para aplicaciones distribuidas. *XIII Jornadas de Concurrencia y Sistemas Distribuidos (JCSD). I Congreso Español de Informática (CEDI 2005).* Granada, Spain, September 2005, pp. 111-117. ISBN: 84-609-6891-X.

15. Carles Pairot, Pedro García, Rubén Mondéjar, and Antonio F. Gómez Skarmeta. Building Wide-Area Collaborative Applications on top of Structured Peer-to-Peer Overlays. *14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2005).* Linkping, Sweden, June 2005, pp. 350-355. ISSN: 1524-4547. ISBN: 0-7695-2362-5. *Workshop's Best Paper and Presentation Award.*

16. Carles Pairot, Pedro García, Rubén Mondéjar, and Antonio F. Gómez Skarmeta. p2pCM: A Structured Peer-to-Peer Grid Component Model. *5th International Conference on Computational Science. 2nd International Workshop on Active and Programmable Grid Architectures and Components. Lecture Notes in Computer Science (LNCS), Volume 3516.* Atlanta, USA, May 2005. ISSN: 0302-9743. ISBN: 3-540-26044-7.

17. Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. PlanetSim: A New Overlay Network Simulation Framework. *Lecture Notes in Computer Science (LNCS), Volume 3437. Software Engineering and Middleware, Revised Selected Papers.* SEM 2004, Linz, Austria. March 2005, pp. 123-137. ISSN: 0302-9743. ISBN: 3-540-25328-9.

18. Carles Pairot, Pedro García, Antonio F. Gómez Skarmeta, and Rubén Mondéjar. Towards New Load-balancing Schemes for Structured Peer-to-Peer Grids. *Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Applications, Vol. 21.* January 2005, pp. 125-133. ISSN: 0167-739X.

19. Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol, Helio Tejedor, and Robert Rallo. PlanetSim: A New Overlay Network Simulation Framework. *19th IEEE International Conference on Automated Software Engineering (ASE 2004). Workshop on Software Engineering and Middleware (SEM 2004).* Linz, Austria, September 2004. ISBN: 3-902457-02-3.

20. Carles Pairot, Pedro García, Antonio F. Gómez Skarmeta, and Rubén Mondéjar, Achieving Load Balancing in Structured Peer-to-Peer Grids. Lecture Notes in Computer Science (LNCS) Volume 3038. *4th International Conference on Computational Science (ICCS 2004). 1st International Workshop on Active and Programmable Grid Architectures and Components (APGAC 2004).* Kraków, Poland. June 2004, pp. 98-105. ISSN: 0302-9743. ISBN: 3-540-22116-6.

21. Carles Pairot, Pedro García, Rubén Mondéjar, and Antonio F. Gómez Skarmeta. Towards a Peer-to-Peer Object Middleware for Wide-Area Collaborative Application Development. *Revista Iberoamericana de Inteligencia Artificial. Vol. 8, No. 24.* Winter 2004, pp. 55-65. ISSN: 1137-3601.

22. Carles Pairot, Pedro García, Antonio F. Gómez Skarmeta, Robert Rallo, and Rubén Mondéjar, DERMI: Middleware para aplicaciones de trabajo

en grupo descentralizadas. *Jornadas Técnicas RedIRIS 2003*. Palma de Mallorca, Spain, November 2003, pp. 51-54. ISSN 1139-207X.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

Bibliography

[1] T. Abdellatif, E. Cecchet, and R. Lachaize. Evaluation of a group communication middleware for clustered j2ee application servers. In *CoopIS/DOA/ODBASE (2)*, pages 1571–1589, 2004.

[2] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 2000.

[3] I. Aekaterinidis and P. Triantafillou. Content-based publish-subscribe over structured p2p networks. In *the International Workshop on Distributed Event-Based Systems (DEBS '04)*, May 2004.

[4] Akamai. `http://www.akamai.com`.

[5] N. Ali, I. Ramos, and J. A. Carsi. A conceptual model for distributed aspect-oriented software architectures. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, pages 422–427, Washington, DC, USA, 2005. IEEE Computer Society.

[6] L. Allen, G. Fernandez, K. Kane, D. Leblang, D. Minard, and J. Posner. ClearCase MultiSite: Supporting Geographically-Distributed Software Development. In J. Estublier, editor, *Software Configuration Management: Selected Papers of the ICSE SCM-4 and SCM-5 Workshops*, number 1005 in Lecture Notes in Computer Science, pages 194–214. Springer-Verlag, Oct. 1995.

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, November 2002.

[8] AspectWerkz. `http://aspectwerkz.codehaus.org`.

[9] R. J. Bayardo-Jr., A. Crainiceanu, and R. Agrawal. Peer-to-peer sharing of web applications. In *WWW (Posters)*, 2003.

[10] N. Bencomo, G. Blair, G. Coulson, P. Grace, and A. Rashid. Reflection and aspects meet again: runtime reflective mechanisms for dynamic aspects. In *AOMD '05: Proceedings of the 1st workshop on Aspect oriented middleware development*, New York, NY, USA, 2005. ACM.

[11] M. Bergsma. Wikimedia architecture. `http://www.nedworks.org/~mark/presentations/san/Wikimedia%20architecture.pdf`, 2007.

[12] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[13] K. P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, Berlin, 1. a. edition, May 2005.

[14] G. S. Blair, G. Coulson, and P. Grace. Research directions in reflective middleware: the lancaster experience. In *ARM '04: Proceedings of the 3rd workshop on Adaptive and reflective middleware*, pages 262–267, New York, NY, USA, 2004. ACM.

[15] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, 1998. Springer-Verlag.

[16] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The fractal component model and its support in java. *Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12):1257–1284, 2006.

[17] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC*, 20:2002, 2002.

[18] N. Chomsky. Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124, 1956.

[19] E. Curry, D. Chambers, and G. Lyons. Extending message-oriented middleware using interception. In *3rd International Workshop on Distributed Event-Based Systems (DEBS'04)*, Edinburgh, Scotland, UK, May 2004.

[20] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. *Peer-to-Peer Systems II*, pages 33–44, 2003.

[21] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM.

[22] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *In HotOS VIII*, pages 75–80, 2001.

[23] B. Du and E. A. Brewer. DTwiki: A Disconnection and Intermittency Tolerant Wiki. In *Proceeding of the 17th International Conference on World Wide Web - WWW 2008*, pages 945–952, New York, NY, USA, 2008. ACM Press.

[24] W. Emmerich. Software engineering and middleware: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 117–129, New York, NY, USA, 2000. ACM.

[25] W. Emmerich and N. Kaveh. Component technologies: Java beans, com, corba, rmi, ejb and the corba component model. In *ICSE '02: Proceed-*

ings of the 24th International Conference on Software Engineering, pages 691–692, New York, NY, USA, 2002. ACM Press.

[26] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[27] Facebook: $20 Million a Year on Data Centers. `http://www.datacenterknowledge.com/archives/2009/05/18/facebook-20-million-a-year-on-data-centers/`.

[28] W. Foundation. 2007-2008 annual report. `http://upload.wikimedia.org/wikipedia/foundation/2/2a/WMF_20072008_Annual_report.pdf`, 2008.

[29] M. Freedman and D. Mazires. Sloppy hashing and self-organizing clusters. In *In IPTPS*, pages 45–55, 2003.

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[31] P. Garcia-Lopez, R. Gracia, and M. Espelt. Topology-aware group communication middleware for manets. In *4th International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE 2009)*, Dublin, Ireland, June 2009.

[32] Git – fast version control system. `http://git.or.cz/`.

[33] P. Greenwood and L. Blair. Using dynamic aspect-oriented programming to implement an autonomic system. In *Proceedings of the 2003 Dynamic Aspect Workshop (DAW04 2003), RIACS*, 2003.

[34] P. Greenwood and L. Blair. Policies for an aop based auto-adaptive framework. In R. Hirschfeld, R. Kowalczyk, A. Polze, and M. Weske, editors, *NODe/GSEM*, volume 69 of *LNI*, pages 76–93. GI, 2005.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

[35] W. G. Griswold, K. Sullivan, Y. Song, M. Shonle, N. Tewari, Y. Cai, and H. Rajan. Modular software design with crosscutting interfaces. *IEEE Software*, 23(1):51–60, 2006.

[36] J. Hannemann and G. Kiczales. Design pattern implementation in java and aspectj. *SIGPLAN Not.*, 37(11):161–173, 2002.

[37] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, July 2008.

[38] Jboss AOP. `http://jboss.org/jbossaop`.

[39] JDBC. `http://java.sun.com/products/jdbc`.

[40] Java Message Service. `http://java.sun.com/products/jms/`.

[41] Jspwiki. `http://www.jspwiki.org/`.

[42] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *In ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[43] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[44] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. marc Loingtier, and J. Irwin. Aspect-oriented programming. In *Lecture Notes in Computer Science 1357*, pages 48–3. Springer Verlag, 1998.

[45] G. Kiczales and J. D. Rivieres. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, MA, USA, 1991.

[46] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[47] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, New York, NY, USA, 2001. ACM.

[48] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance evaluation of replication strategies in dhts under churn. In *MUM '07: Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, pages 90–97, New York, NY, USA, 2007. ACM.

[49] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod. Performance Evaluation of Replication Strategies in DHTs under Churn. In *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia - MUM 2007*, pages 90–97, Oulu, Finland, Dec. 2007. ACM Press.

[50] B. Lagaisse and W. Joosen. True and transparent distributed composition of aspect-components. In *Middleware*, pages 42–61, 2006.

[51] P. Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, 1987.

[52] J. McAffer. Meta-level architecture support for distributed objects. In *IWOOOS '95: Proceedings of the 4th International Workshop on Object-Orientation in Operating Systems*, pages 232–241, Washington, DC, USA, 1995. IEEE Computer Society.

[53] P. K. Mckinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[54] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on*, 26(1):70–93, 2000.

[55] R. Mondéjar, P. García, C. Pairot, and A. Skarmeta. Damon: a decentralized aspect middleware built on top of a peer-to-peer overlay network. In *Proceedings of the 6th international workshop on Software engineering and middleware*, pages 23–30, Portland, Oregon, USA, 2006. ACM Press.

[56] J. C. Morris. DistriWiki: A Distributed Peer-to-Peer Wiki Network. In *Proceedings of the 2007 International Symposium on Wikis - WikiSym 2007*, pages 69–74, New York, NY, USA, 2007. ACM Press.

[57] P. Mukherjee, C. Leng, and A. Schürr. Piki - A Peer-to-Peer based Wiki Engine. *Proceedings of the IEEE International Conference on Peer-to-Peer Computing - P2P 2008*, 0:185–186, 2008.

[58] Napster. `http://www.napster.com/`.

[59] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. The interception approach to reliable distributed corba objects. In *COOTS'97: Proceedings of the 3rd conference on USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 20–20, Berkeley, CA, USA, 1997. USENIX Association.

[60] L. D. B. Navarro, M. Sdholt, R. Douence, and J.-M. Menaud. Invasive patterns for distributed applications. In *Proc. of the 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA'07)*, LNCS. Springer Verlag, November 2007.

[61] L. D. B. Navarro, M. Südholt, W. Vanderperren, B. De Fraine, and D. Suvée. Explicitly distributed aop using awed. In *AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 51–62, New York, NY, USA, 2006. ACM.

[62] B. C. Neuman. Scale in distributed systems. In *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, 1994.

[63] M. Nishizawa, S. Chiba, and M. Tatsubori. Remote pointcut: a language construct for distributed aop. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 7–15, New York, NY, USA, 2004. ACM.

[64] G. Oster, P. Urso, P. Molli, and A. Imine. Data Consistency for P2P Collaborative Editing. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*, pages 259–267, Banff, Alberta, Canada, Nov. 2006. ACM Press.

[65] C. Pairot, P. Garcia, and R. Mondejar. Deploying wide-area applications is a snap. *IEEE Internet Computing*, 11(2):72–79, 2007.

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

[66] C. Pairot, P. G. López, and A. F. Gómez-Skarmeta. Dermi: A new distributed hash table-based middleware framework. *IEEE Internet Computing*, 8(3):74–84, 2004.

[67] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[68] R. E. Pattis. Teaching ebnf first in cs 1. In *SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, pages 300–303, New York, NY, USA, 1994. ACM.

[69] R. Pawlak, L. Seinturier, L. Duchien, G. Florin, F. Legond-Aubry, and L. Martelli. Jac: an aspect-based distributed dynamic framework. *Softw. Pract. Exper.*, 34(12):1119–1148, 2004.

[70] R. Pichler and M. Mezini. On aspectualizing component models. *Software and Practice and Experience*, 33:2003, 2003.

[71] M. Pinto, L. Fuentes, and J. M. Troya. A dynamic component and aspect-oriented platform. *Comput. J.*, 48(4):401–420, 2005.

[72] PlanetLab. `http://www.planet-lab.org`.

[73] J. Pujol-Ahulló and P. García-López. Planetsim: An extensible simulation tool for peer-to-peer networks and services. In *Proceedings of 9th International Conference on Peer-to-Peer Computing 2009 (IEEE P2P '09)*, pages 85–86, Piscataway, NJ08855-1331, 2009. IEEE Computer Society.

[74] E. Putrycz and G. Bernard. Using aspect oriented programming to build a portable load balancing service. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 473–480, Washington, DC, USA, 2002. IEEE Computer Society.

[75] I. S. R. Chitchyan. Comparing dynamic ao systems. In *Proceedings of the AOSD'04 Dynamic Aspects Workshop*, pages 23–36, Lancaster UK, 2004.

[76] RepliWiki – A Next Generation Architecture for Wikipedia. `http://isr.uncc.edu/repliwiki/`.

[77] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100, August 2001.

[78] Java Remote Method Invocation. `http://java.sun.com/products/jdk/rmi/`.

[79] S. Robert, A. Radermacher, V. Seignole, S. Gérard, V. Watine, and F. Terrier. The corba connector model. In *SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware*, pages 76–82, New York, NY, USA, 2005. ACM.

[80] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes In Computer Science*, 2218:329–350, Nov. 2001.

[81] R. E. Schantz and D. C. Schmidt. Middleware for distributed systems - evolving the common structure for network-centric applications. *In Encyclopedia of Software Engineering*, pages 801–813, 2001.

[82] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, September 2000.

[83] JSR 53: Java Servlet 2.3 and JavaServer Pages 1.2 Specifications. `http://jcp.org/en/jsr/detail?id=053`.

[84] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, April 1996.

[85] S. Sivasubramanian, G. Pierre, and M. van Steen. Replicating web applications on-demand. *Services Computing, IEEE International Conference on*, 0:227–236, 2004.

[86] S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with aspectj. *SIGPLAN Not.*, 37(11):174–190, 2002.

[87] R. P. Sriganesh, G. Brose, and M. Silverman. *Mastering Enterprise JavaBeans 3.0.* John Wiley & Sons, Inc., New York, NY, USA, 2006.

[88] M. Steiner, T. En-Najjary, and E. W. Biersack. A global view of kad. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, New York, NY, USA, 2007. ACM.

[89] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[90] R. Stroud. Transparency and reflection in distributed systems. In *EW 5: Proceedings of the 5th workshop on ACM SIGOPS European workshop*, pages 1–5, New York, NY, USA, 1992. ACM.

[91] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 3 1998.

[92] B. Surajbali, G. Coulson, P. Greenwood, and P. Grace. Augmenting reflective middleware with an aspect orientation support layer. In *ARM '07: Proceedings of the 6th international workshop on Adaptive and reflective middleware*, pages 1–6, New York, NY, USA, 2007. ACM.

[93] É. Tanter and R. Toledo. A versatile kernel for distributed AOP. In *Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2006)*, volume 4025 of *Lecture Notes in Computer Science*, pages 316–331, Bologna, Italy, June 2006. Springer-Verlag.

[94] A. R. Tripathi and T. Noonan. Design of a remote procedure call system for object-oriented distributed programming. *Softw. Pract. Exper.*, 28(1):23–47, 1998.

[95] E. Truyen, N. Janssens, F. Sanen, and W. Joosen. Support for distributed adaptations in aspect-oriented middleware. In *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*, pages 120–131, New York, NY, USA, 2008. ACM.

[96] WADI. `http://wadi.codehaus.org`.

[97] S. Weiss, P. Urso, and P. Molli. Wooki: a P2P Wiki-based Collaborative Writing Tool. *Lecture Notes In Computer Science*, 4831(1005):503–512, Dec. 2007.

[98] Wikipedia, the online encyclopedia. `http://www.wikipedia.org/`.

[99] Wikipedia Statistics, 2008. `http://meta.wikimedia.org/wiki/Statistics`.

[100] Wikipedia Data, 2008. `http://download.wikimedia.org/enwiki/latest/`.

[101] Xwiki. `http://www.xwiki.org/`.

[102] Youtube. `http://www.youtube.com`.

[103] Code Co-op. `http://www.relisoft.com/co_op/`.

[104] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 301, Washington, DC, USA, 2002. IEEE Computer Society.

[105] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area

UNIVERSITAT ROVIRA I VIRGILI
DISTRIBUTED AOP MIDDLEWARE FOR LARGE-SCALE SCENARIOS
Ruben Mondejar Andreu
ISBN:978-84-693-5426-1/DL:T-1417-2010

data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, NY, USA, 2001. ACM Press.