



# The Adobe® Digital Enterprise Platform: Architectural principles and choices

**Table of contents**

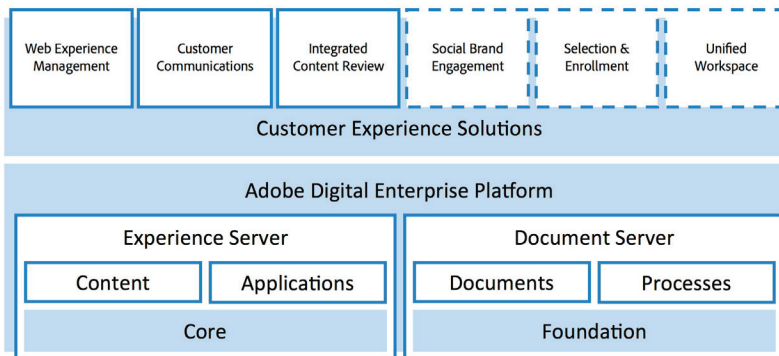
- 3: Implications for existing Adobe enterprise customers
- 4: ADEP architecture principles
- 6: ADEP architecture choices
- 10: Appendix A: A brief OSGi primer

The Adobe Digital Enterprise Platform (ADEP) provides the necessary consistent foundation to meet the customer experience management (CEM) opportunities being realized by today’s leading organizations. The ADEP comprises two servers that provide capabilities through software services, specifically Experience Services for content and applications and Document Services for documents and processes. This paper describes the architectural principles of the ADEP and the choices for implementing those principles.

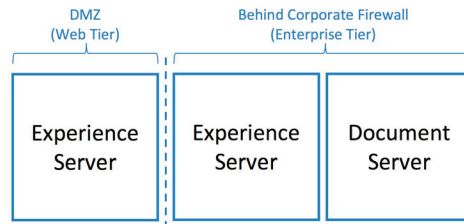
Adobe’s Customer Experience Solutions, which are built on top of the ADEP, leverage the ADEP services and its full range of capabilities to implement specific patterns that create, manage, and deliver end customer experiences:

- Web Experience Management—Offers marketing agility to attract customers, capture insight, and deliver personalized interactions.
- Customer Communications—Provides the ability to create multichannel, customized correspondence and interactive statements.
- Integrated Content Review—Helps manage the review of digital, targeted content across the customer lifecycle.
- (Future) Social Brand Engagement—Enables customers to engage with companies on-site and in the customer’s context.
- (Future) Selection & Enrollment—Helps increase direct and influenced acquisition rates by guiding and enrolling customers.
- (Future) Unified Workspace—Delivers outstanding service with unified insight across enterprise silos.

Customer Experience Solution patterns both from Adobe and its partners will drive the future evolution of the platform.



The Experience Server is based on the rich Internet application (RIA) technology from Adobe LiveCycle® Enterprise Suite (ES) and Day Software CRX (Adobe acquired Day Software in October 2010). The Document Server is based on LiveCycle document and business process management technology.



You can deploy the ADEP Experience Server in both the enterprise tier, for example, in service of authoring content and applications, and the web tier, such as in service of publishing content and applications. Typically, Experience Server instances in the web tier scale at a rate greater than instances in the enterprise tier to meet the much higher consumption requirements of users accessing content as compared to the consumption requirements of users authoring content. The ADEP Experience Server provides its own servlet container (web server) and default persistence management. Therefore, it requires neither a Java™ Enterprise Edition (JEE) application server nor a relational database management system (RDBMS).

The ADEP Document Server is deployed behind the corporate DMZ, closer to other systems of record, such as customer relationship management, enterprise resource planning systems, and enterprise content management. The Document Server architecture is modeled after an enterprise service bus, and it represents a reference implementation of a service-oriented architecture (SOA). The Document Server requires a JEE application server and RDBMS.

Some ADEP implementation patterns, including Adobe Customer Experience Solutions like Web Experience Management and Customer Communications, require integrated enterprise tier and web tier capabilities, and this is precisely what the ADEP provides.

Currently, communication between Experience Services and Document Services occurs via the Simple Object Access Protocol (SOAP). The Document Services client software development kit (SDK) uses a new type of message dispatcher to make outbound calls to Document Services. This SDK is bundled within the Experience Server. When an Experience Service needs to invoke a Document Service, the entire invocation request is serialized, dispatched in a SOAP body, and handled by the SOAP interceptor in the Document Server for service invocation.

The Experience Server and the Document Server both support user management integration with an LDAP source as well as single sign-on (SSO) integration with a third-party SSO provider. The Experience Server can also be integrated with the Document Server to rely on the Document Server as the single provider of user identity.

The ability to use the Document Server as the identity provider for authentication and group membership of users who authenticate to the Experience Server is achieved by enabling a login module supplied with the Experience Server Core. This module performs authentication using a separately deployed Document Server, which can manage locally administered users or leverage an LDAP-based enterprise domain. After authentication, the Experience Server is then able to leverage Document Services invoked under the identity and permissions of individual users as defined in the Document Server.

## Implications for existing Adobe enterprise customers

Both LiveCycle and CRX have enjoyed commercial success prior to the formation of the ADEP; therefore, it is important to relate how the capabilities of LiveCycle and CRX exist in the ADEP. If you are an existing LiveCycle or CRX customer, the ADEP builds on the current infrastructure of powerful capabilities you've become accustomed to. The ADEP provides new capabilities and deployment options to simplify and extend the delivery of differentiated experiences to your customers.

### LiveCycle ES customers

Prior to the ADEP, LiveCycle ES has been predominantly focused in the enterprise tier, deployed behind the corporate DMZ, delivering industry-leading process management, forms, and document services capabilities. With the ADEP, Document Server continues to support the component model based on Document Service Components (DSCs), orchestration of services via Foundation, user-facing workflow via Process Management, and a familiar application packaging and deployment model, previously known as LiveCycle applications (LCAs) and now referred to as Document Services applications.

If you are coming to the ADEP as an existing LiveCycle ES customer, it is important to understand the following:

- The standard upgrade policy that exists between major versions of LiveCycle also exists between LiveCycle and ADEP Document Services.
- Custom DSCs developed against ES2 or ES2.5 will continue to work with the ADEP Document Server.
- LCAs will continue to work with the ADEP Document Server.
- The ADEP offers new capabilities around content and applications via the Experience Server.

### LiveCycle Data Services and LiveCycle Mosaic customers

LiveCycle ES also featured data integration and modeling capability known as Data Services. Data Services was deployed standalone in your custom web application. That option still exists with the ADEP (that is, the consumption of Data Services JAR files in your WAR files), and a new option allows you to leverage Data Services..

LiveCycle Mosaic was introduced in the ES2 release as a framework to compose broader application experiences from smaller, tile-based applications. In the ADEP, this capability is called the Composite Application Framework. It is based on the Experience Server, making consistent context available to both content and applications.

If you are coming to the ADEP as an existing Data Services or Mosaic customer, it is important to understand the following:

- The Experience Server Data Services is a complement to the standalone ADEP Data Services—that is, the Experience Server Data Services is a subset of the standalone ADEP Data Services.
- Data Services can still be deployed in standalone form (as JAR files within custom a web application) or leveraged in a more integrated form via the ADEP Experience Server.
- Experience Services in the Composite Application Framework—for example, the Catalog service—replace LiveCycle Mosaic services.
- Runtime modularity on the client is publicly supported in the ADEP via the Client Component Framework, previously known by the codename Gravity.
- Composite Application Framework is directly integrated with the Experience Server and Data Services. It provides a more productive application composition canvas in terms of data integration, context sensitivity, measurement and optimization, versioning, access control, and so on.
- The Experience Server embraces the quick-start paradigm from CQ5 and CRX.
- The Experience Server embraces the deployment system of Package Manager and Package Share from CQ5 and CRX, which will serve as a primary delivery mechanism for subsequent updates.
- The ADEP offers new capabilities around user context, content management, resource-first request processing, and dynamic (runtime) composition.

## CRX customers

Prior to the ADEP, CRX has been predominantly focused in the web tier, delivering industry-leading composite content application and content management capabilities to customers, such as via CQ5 Web Content Management. In the ADEP, CRX is now referred to as the Experience Server Core, and it is the consistent foundation for context-sensitive delivery of content and applications.

If you are coming to the ADEP as an existing CQ5 or CRX customer, it is important to understand the following:

- The Experience Server Core correlates to CRX—it is equivalent or a superset. It provides resource-first request processing and Open Services Gateway initiative (OSGi) based runtime modularity, an open standards-compliant content repository that supports JCR, CMIS, WebDAV, CIFS, and so on, and a persistence layer for user-generated content that advocates NoSQL (not only SQL).
- The Experience Server supports Spring as a developer pattern for dependency injection (DI) used during customization.
- The Experience Server embraces the quick-start paradigm from CQ5 and CRX.
- The Experience Server embraces the deployment system of Package Manager and Package Share from CQ5 and CRX, which serve as a primary delivery mechanism for subsequent updates.
- The Experience Server supports both the web developer (HTML5, JavaScript) and the application developer (Flex, Adobe Flash®, Adobe AIR®, Java, .NET).
- The Experience Server offers new data integration and modeling capabilities via Data Services.
- The Experience Server offers new application composition capabilities via Composite Application Framework.
- The Experience Server can easily integrate with the Document Server to leverage new capabilities around documents.

## ADEP architecture principles

Architecture is the result of a collective set of design choices, and these choices are informed by principles. So, let's first look at the ADEP architectural aspirations before discussing their realization and the architecture implementation choices.

### **Modularity is critical to industrialize differentiated experience**

Modularity is the most essential architecture principle applied across the ADEP. The modularization enables you to manage the complexity of your CEM solutions by separating them into independent components that can be worked on by different development teams across different times, and all tested in isolation. When deployed, these solutions consume a minimal footprint that is associated with specific workload requirements. Resource usage is kept to what is essential for delivering business value.

The very nature of differentiated customer experience is modular—my experience today should differ from my experience tomorrow, from my experience yesterday, and from what another customer may experience. The ADEP anticipates, is tuned for, and can accelerate composition and reuse. In an enterprise context, supporting such modularity means that you can draw upon the breadth of assets (content, applications, documents, processes, and services) available, including those from Adobe, Adobe's partners, systems integrators, agencies, and other teams within your enterprise. These assets can be brought to bear on business problems in a manner that can be partitioned and rolled out incrementally with minimal disruption, such as hot deployment.

By embracing modularity as a core architecture principle, ADEP software modules are self-contained (local wholes), highly cohesive (fulfill single purposes), and loosely coupled (well isolated from other modules). To support all three properties, modules must have a well-defined interface for interaction with other modules. A stable interface enforces logical boundaries between modules and prevents access to internal implementation details. Ideally, the interface should be defined in terms of what each module offers to other modules, and what each module requires from other modules.

Modularity is driven across the ADEP architecture by the following principles:

- Interface-based programming
- Externalization of cross-cutting concerns (aspect-oriented programming)
- Late binding of implementation instances to interfaces (dependency injection, extensibility)

The next section discusses how these modularity techniques are brought to bear through Experience Server application-based experience composition and delivery.

### **Everything is content**

In the ADEP, everything is content, and content resides in a repository. There are no loose files somewhere else to manage. Source code, dynamic modules, configuration, and even the state of an application reside side-by-side with marketing collateral and digital assets, such as images, audio and video. The content repository recognizes that meta is in the eye of the beholder. Consequently, there is no justification to treat content (that is, the file stream) and metadata differently.

Because the content repository consistently manages this diversity, the rich set of content services above the repository is uniformly available. For example, the resource-first request processing of the Experience Server Core is equally available to traditional content, such as web pages, and to applications such as a product configurator. By managing to a wide definition of content, the ADEP can reduce the amount of code and effort required to deliver a solution.

Because the ADEP provides a virtual content repository that easily connects with existing content silos in an enterprise, "everything is content" also means that any existing content is free to participate in serving customer experience, for example, for marketing campaigns, customer communication, and so on.

### **Context is king**

Given the previous principle, you might be surprised to read that context, not content, is king. Actually it's more like a king and his kingdom. Great content is critical to customer experience, and customers experience content in context.

Adobe's approach to building CEM solutions aims to empower and delight customers by giving web visitors exactly the information they need, in the right form, at the right time. Doing this reliably and in real time can be a challenge. It requires software that can aggregate relevant user information from a variety of sources to drive intelligent provisioning of content on a page according to predetermined strategies.

Adobe's Customer Experience Solution for Web Experience Management features core technology for customizing and optimizing user experience called the Context Cloud (previously known as the Clickstream Cloud). The Context Cloud represents a dynamically assembled collection of user data that can be used to determine exactly what content should be shown in a given situation.

Several things make Adobe's implementation of the Context Cloud unique:

- Much of the information, such as about the user's viewing environment, is derived on-the-fly in real time; it is not persisted anywhere.
- Marketers can experiment with different user data values to see changes to a page in real time, for example, to try different campaign strategies before going live.
- The Context Cloud is extensible. You can add a new (custom) session-store object whose contents can fully participate in campaign "what if" scenarios.
- Nonvolatile information shown in the Context Cloud viewer is persisted on the client side in a cookie, relieving the server of having to maintain and transport back and forth large amounts of user data.

Because user information is persisted on the client, concerns over the privacy and control of sensitive user data are easily allayed: The user has ultimate control over the data.

## Cloud first, mobile first, social first

The Experience Server supports WAN clustering, which is important in high-latency situations and a distributed infrastructure) and hot-cluster join, which allows you to expand infrastructure on-the-fly. It also runs in a very small memory and CPU footprint, making it suitable for deployment in the cloud, whether actual deployments are done in the cloud or on premise.

In pursuing interaction patterns, the ADEP starts with mobile devices, particularly tablets, and then expands to consider other environments. The ADEP can detect over 17,000 devices, enabling content contributors to understand exactly which experience will be delivered to segmented content consumers via device emulation support. The ADEP presents the concept of device groups to reduce the complexity and managing of the diverse range of devices and device types.

Today's customers increasingly leverage social activities to gain validation of their decisions and to share them with others. The ADEP supports a range of social capabilities, including support for local communities and the ability to glean information from public communities, such as Facebook and Twitter, and use that information to tailor the customer experience. Social capabilities in the platform are much like the public social environment: They surround everything we do and are available for use at any time for any purpose.

With the ADEP:

- You build applications for the cloud with on premise in mind.
- You build applications for mobile with desktop in mind.
- You understand that every user is a contributor and has a social graph.

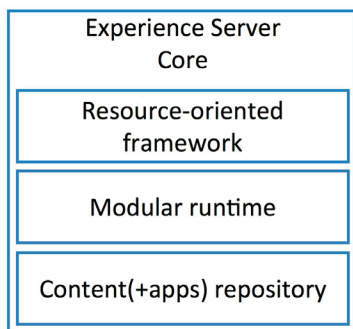
## ADEP architecture choices

Now let's look at how the principles of the ADEP architecture are implemented in the platform for customer experience.

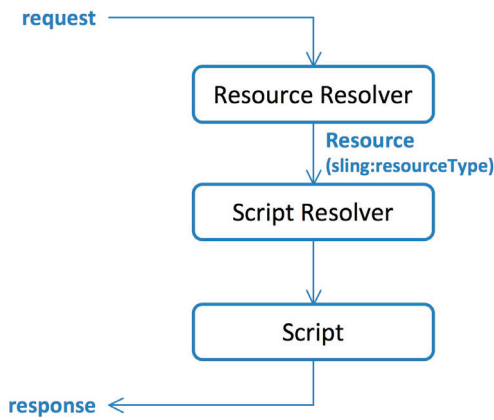
### Architectural choice: CRX and Data Services for Experience Server Core

The heart of the ADEP Experience Server is technology formerly known as CRX and is now referred to as the Experience Server Core. The Experience Server Core is the basis for the content (content management) and application (RIA) capabilities. It supplies three essential elements of the ADEP architecture:

- RESTful application framework (resource-oriented request processing)
- Modular application runtime (OSGi-based modularity)
- Open standards-compliant content repository (such as JCR, CMIS, or WebDAV)

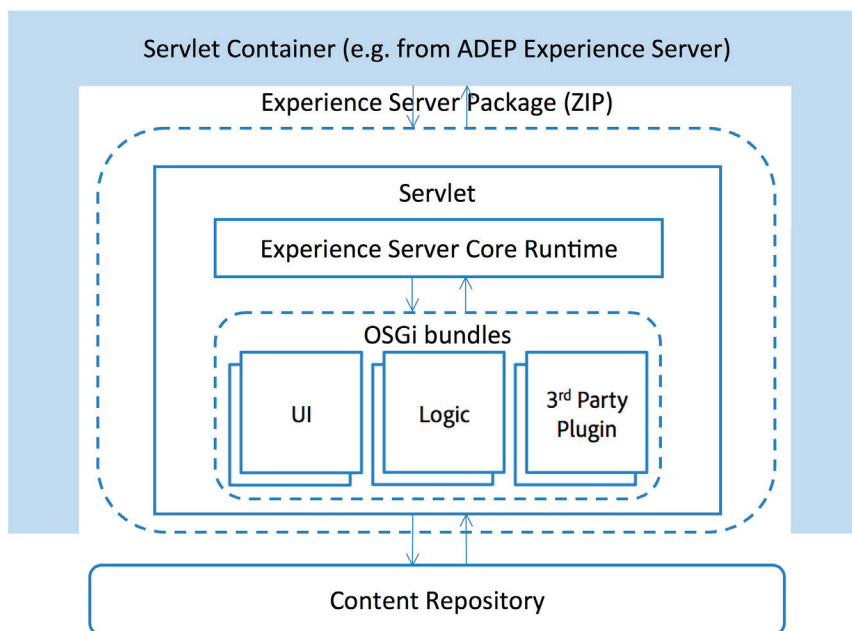


The resource-oriented framework is provided by Apache Sling and features resource-first request processing.



Much like everything is content in the ADEP, everything is a resource in Sling. Request processing in Sling starts by decomposing the input URI to resolve the resource. When the resource is resolved, the resource type of the resolved node determines how to render the resource. Next, Sling creates the rendering chain by assembling the specified filters in the configuration and by honoring user context. Finally, Sling invokes the rendering chain and returns a response containing the requested resource representation.

Sling instantiates the ADEP modular runtime to handle the request. This modular runtime is provided by Apache Felix, which implements the OSGi specification. We'll talk more about OSGi and the importance of the modularity it provides later in this paper.



Because different solutions require different communication styles, the Experience Server Core also features data integration and modeling capabilities in Data Services. These capabilities complement the RESTful integration style provided by Sling.

## Architectural choice: Mosaic for Experience Server application capabilities

Composite Application Framework (previously known as LiveCycle Mosaic) provides the capabilities to compose broader application experiences from smaller, tile-based applications. Composite Application Framework embodies the principles of modularity that we discussed previously.

Composite Application Framework helps you manage the complexity of implementation projects by addressing the multiscreen expectation of today's customer experience, increasing developer productivity across large and geographically dispersed teams, and facilitating greater reuse of component and application assets through a shared catalog and an interface-based design.

Composite Application Framework enables business users, not just developers, to compose application experiences by promoting the notion of applications as content. For example, treating application assets related to branding, skinning, and theming as content enables those who understand best how to brand and visualize to accomplish the right personalization, without having to pick up the phone to dialog with IT. Taking a content-oriented approach makes it easier to administer and tweak an application after its initial development and allows more users to directly participate in the realization of subsequent customer experiences.

Composite Application Framework is based on the Experience Server Core, making consistent context available to both content and applications. Applications are stored as hierarchical content, not XML files, which improves server-side performance. Package Manager can be used to manage and deploy application assets across development, QA, staging, and production environments.

Composite Application Framework assembles applications into experiences, taking advantage of the OSGi-based modularity in the Experience Server. Client Component Framework assembles client-side components into applications served by Composite Application Framework. Inspired by OSGi for Java and targeting ActionScript\* development, Client Component Framework provides dynamic loading, versioning, and dependency injection for Flash Platform and AIR applications, independent of the Flex SDK.

Composite Application Framework serves to complete the picture of enterprise development with Flex, Flash, and AIR technology, and it complements other HTML5 capabilities in the Experience Server.

ADEP applications use a component model that is oriented toward reuse across the spectrum of applications. At one end of that spectrum are static applications that consist of individual, statically linked components. At the other end of the spectrum, individual components are placed in a catalog on a server and dynamically injected into an application at runtime. These user experience components are called UX components.

A UX component is completely independent of its data source. Simply by injecting a data source at runtime and providing a different skin, a different application experience can be delivered. Because components are inheritable, skins are replaceable, and services are injectable, you can achieve a high-level of reuse while providing richness and consistency in the experience.

## Architectural mindset: JEE from an Experience Services perspective and OSGi from a Document Services perspective

It's important to contrast the JEE patterns found in the Document Server with the OSGi patterns found in the Experience Server. You need to understand the following differences, whether you are implementing a Document Service coming from a CRX background or implementing an Experience Service coming from a LiveCycle background.

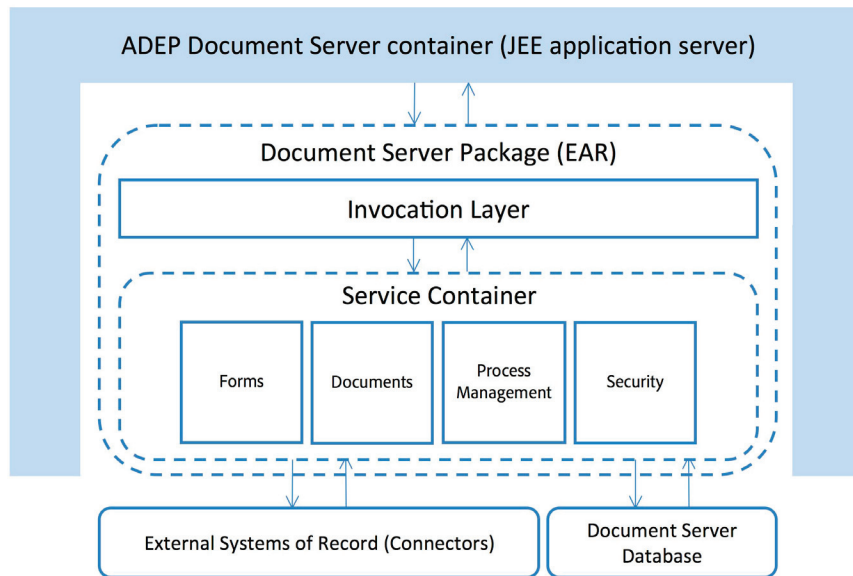
- JEE doesn't promote dependency injection intrinsically
  - Frameworks like Spring do support DI and can be integrated with a JEE container
- In JEE, libraries (JAR files) must be bundled with your EAR or WAR file
  - OSGi solves library dependencies using bundles, which can be installed on an as-needed basis
- In JEE, sharing classes requires placing JAR files in common classloaders
  - OSGi solves class sharing via explicit module metadata—Export-Package to expose internal bundle classes and Import-Package to make external classes visible to internal bundle classes)
- In JEE, packaging multiple versions of the same library is not for the faint of heart
  - OSGi greatly simplifies packaging via versioned packages and so on.

A DSC is analogous to an OSGi bundle. A DSC can be deployed hot, and it can both import and export packages. This is a good example of common principles across the ADEP that can be implemented differently.

### Architectural choice: LiveCycle for Document Server and document capabilities

A JEE-based architecture previously known as LiveCycle Foundation remains the basis for the ADEP document-oriented capabilities and is now referred to as the ADEP Document Server. The Document Server is a mature offering with a significant installed base that is relevant for many types of customer solutions. Adobe considers the Document Server and its Document Services as viable products going into the future and has no plans to end-of-life them. Adobe will add features and improve it, focusing on stability and the goal of enabling ADEP customers to get more value out of their existing LiveCycle investments.

The ADEP is about unlocking new value from existing IT; it is not disruptive or displacing toward existing systems of record. The ADEP is focused on transforming IT into a system of customer engagement. Therefore, it is consistent to treat the Document Server as an existing system of record and to integrate it with the Experience Server as an important part of the ADEP system of engagement.



The Document Server provides models for components and services. A component in the Document Server is referred to as a DSC (Document Service Component). It is essentially a JAR file plus unique metadata specified in a manifest file called component.xml. Service endpoints are cleanly separated from underlying components by the Document Server, managed by a service registry whose APIs for both components and services do not change in the ADEP, and orchestrated by Process Management.

Document Service applications (LCAs) provide an application packaging and deployment model that is served by the Configuration Manager.

### Architectural choice: OSGi as the basis for content and application capabilities

The ADEP leverages OSGi—see Appendix A about the business value of OSGi—to provide the following customer value:

- Familiar pursuit of Java-based application development
  - The ADEP provides tools to help you deploy your Java logic as “bundles.”
- Concise application packaging and deployment
  - You can package applications modularly as logically independent concerns.
  - You can deploy only those pieces actually needed for a given installation (solution).
  - Applications can use different versions of the same resource, without conflicts or collisions.
  - Multiple versions of the same application can be deployed concurrently.
- Easy, reliable component updating and patching (application administration and maintenance)
  - Components are loosely coupled via service interfaces and well-defined dependency contracts.

It's important to distinguish platform internals and implementation details from customer focus, solution building, and business value. The ADEP embodies a common set of architecture principles that can be implemented differently depending on the use case and business context, for example, content and applications versus documents. As far as the ADEP is concerned, OSGi is really just an internal implementation detail.

### **Architectural choice: OSGi as the future basis for document capabilities**

The ADEP Document Server is a mature, proven architecture. In fact, the DSC container in the Document Server is based on an early implementation of OSGi. Therefore, porting Document Services to run as OSGi bundles within the Experience Server is a natural architectural progression and allows for unified core architecture across the ADEP. However, Adobe believes that the nature of the ADEP customer is changing to require capabilities for content, applications, and documents and to deliver differentiated experience in an efficient manner.

In addition to supporting the existing Document Server for customers who do not want to migrate custom DSCs and solution implementations, Adobe will release Document Services on the Experience Server architecture. This migration will be driven by customer use cases and requirements, not for technical reasons.

The ADEP will remain a solutions-oriented platform that is focused on better serving customer experience solutions from Adobe, its partners, and customers. This focus will drive any future Document Service migration.

## **Appendix A: A brief OSGi primer**

In his forward to OSGi in Action, Peter Kriens said, "[OSGi] treats an application as a collaboration of peer modules: modules that can adapt themselves to the environment instead of assuming that the environment is adapted to them."

The OSGi framework defines a dynamic module system for Java. It gives you better control over the structure of your code, the ability to dynamically manage your code's lifecycle, and a loosely coupled approach for code collaboration. OSGi has a positive impact on multiple stages of an application's lifecycle:

- During development, you can clearly and explicitly partition development into independent pieces.
- During deployment, you can easily analyze, understand, and resolve requirements imposed by the independently developed pieces composing a complete system.
- During execution, you can manage and evolve the constituent pieces of a running system, and minimize the impact of doing so.

OSGi presents a layered architecture:

- Module layer—Concerned with packaging and sharing code
- Lifecycle layer—Concerned with providing execution-time module management and access to the underlying OSGi framework
- Service layer—Concerned with interaction and communication among modules, specifically the components contained in them

When it comes to modularity, the central idea of OSGi is in fact rather simple. The source of many problems in traditional Java is the global, flat classpath, so OSGi takes a completely different approach. Each module has its own classpath, separate from the classpath of all other modules. Of course, modules do still need to work together, and OSGi has very specific and well-defined rules about how classes can be shared across modules, using a mechanism of explicit imports and exports. This combination of principles forms the basis for modularity in OSGi that is both logical and physical in nature.

An OSGi module is called a bundle and exists as a physical unit of modularity in the form of a JAR file containing code, resources, and metadata. The boundary of the JAR file also serves as the encapsulation boundary for logical modularity at execution time.

To support dynamic modules, OSGi has a fully developed lifecycle layer, along with a programming model that allows "services" to be dynamically published and consumed across bundle boundaries. Because the OSGi specification provides an explicit lifecycle API, you can take any bundle providing the functionality you need and let it worry about how to manage its internal functions. In essence, it's a matter of compose versus control.

OSGi's service layer builds on and interacts with its module and lifecycle layers. Services encourage a relaxed, pluggable, interface-based approach to programming. Service contracts define responsibilities and match consumers with providers. You don't need to care where a service comes from as long as it meets the contract.

Services aren't limited to distributed or remote applications. There's a huge benefit to applying a service-oriented design to a purely local, single-JVM application, and this benefit is available to customer experience solutions powered by the ADEP.

By adopting OSGi at its core, the ADEP enables you to package applications modularly as logically independent concerns and to deploy only those pieces needed for a given installation (solution). Applications are able to use different versions of the same resource, without conflicts or collisions, and multiple versions of the same application can be concurrently deployed.

Within the ADEP, OSGi is not "in your face" unless you want it to be. You are fully enabled to leverage your expertise with OSGi, such as via Declarative Services, or avoid dealing with OSGi-specific details, for example, by writing Plain Old Java Objects (POJOs) and leveraging IDE gestures to bundle, package and deploy.

OSGi has been under development for over 10 years and has begun to realize mainstream adoption. Adobe (via its Day Software acquisition) has invested more than three calendar years hardening OSGi for use in a web-scale, enterprise software context.

Adobe is not only strategically committed to OSGi in the ADEP, but also works actively in the OSGi Alliance and in the Apache Software Foundation through projects such as Felix, Sling, and Jackrabbit to ensure that the OSGi specification well serves the needs of CEM solutions. This commitment to open community benefits Adobe's customers with higher quality, more functional software.

