Software Configuration, Distribution, and Deployment of Web-Services

Rainer Anzböck
D.A.T.A. Corporation
Invalidenstrasse 5-7/10
A-1030 Wien, Austria
++43/1/5955319-2
ar@data.at

Schahram Dustdar Harald Gall
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040 Wien, Austria
++43/1/58801-18414
{dustdar,gall}@tuwien.ac.at

ABSTRACT
Web-Services can be seen as a newly emerging distributed computing model for the Web. They cater for the need to establish business-to-business (B2B) interactions on the Web. Web-Services consider a loosely coupled component model encapsulating business logic and interact with other components using XML protocols. Based on one case study, this paper discusses architectural issues and requirements for software configuration, distribution, and deployment of web-services.

General Terms
Design

Keywords
Software Distribution Environments, Web-Services, Software architecture

1. INTRODUCTION
Software distribution has changed through the Internet evolution from simple e-mail distribution of software to sophisticated distribution and configuration portals for software. Such Web portals are built to provide the newest releases of a vendor's software: product updates, service packages, or complete software packages. Internet users are provided easy and fast access to large collections of software across many different software and hardware platforms.

Many commercial tools and environments have been developed and are in use in such portals: System Management Server [10], Marimba Management Solution [8], Webstart [19], InstallShield [5], Tivoli Software Distribution [21], and many others [1]. We have performed a detailed evaluation of 12 products in the area of software distribution (and configuration) environments (SDEs) that defines the basis for our architectural considerations. Details on this evaluation can be found in [1].

SEKE '02, July 15-19, 2002, Ischia, Italy.

In the early stages configuration management (CM) aspects were not part of SDEs, but nowadays they cover software distribution and configuration management tasks for the development, deployment and maintenance of software systems as described in [2]. Many concepts covering software distribution are also related to configuration management. Producers require software configuration management (SCM) to be integrated in their development environments. Software distribution should be based on SCM information to ship or offer particular releases (or configurations) to customers. For that, many SDEs also cover some software configuration tasks such as management of development artifacts, product and release management, software description (languages), or software packaging.

The contribution of this paper is to discuss a software architecture for a SDE suitable for Web-services. This is achieved by discussing its architectural properties, common components, and relationships across particular tools and products, and quality attributes of a reference architecture for SDEs derived from our previous analysis [1]. Our results are based on the aforementioned product evaluation and three case studies that functioned as means to distill common architectural elements.

The paper is organized as follows: the next section provides a brief overview of Web-services. Section 2 describes a case study for Web-services and discusses its architectural issues and requirements for a SDE. Logical – and Process views for configuration, distribution, and deployment are presented in section 2. Finally section 3 concludes the paper.

1.1 Web-services
Web-services [21, 22] can be seen as a newly emerging distributed computing model for the Web. The standardization process is driven by the growing need to enable business to business (B2B) interactions on the Web. Web-services are self-contained, self-describing modular applications. The web-services model develops a componentized view of web applications and is becoming the emerging platform for distributed computing. The architecture considers a loosely integrated component model, where a web-service (component) encapsulating any type of business logic is described in standardized interface definition language, the Web Services Description Language (WSDL) [21]. Web-service components interact over XML messaging protocol and interoperate with other components using the Simple Object Access Protocol (SOAP) [22]. Recently there have been proposals [3, 20] for a composition model for web-services, which build on the component interaction model defined by the interaction types defined by the WSDL interface.

1.2 Case study “Web-service"

The real-world case study of this paper consists of a complex product family that consists of conventional client/server products as well as Web-based applications. All applications are being developed in one physical location from one producer (software company) with some external software engineers having external access. Further, all applications are under version control. Versions are accessed on a standard file system for the distribution process. A Software Distribution Environment (SDE) should consider these development tasks. The client/server applications consist of Windows client executables and a database server system. The Web-based applications consist of a database tier, an application tier on the Web-server (i.e. application server) that provides a Web-based client interface for end-users, and a Web-service interface for the client/server applications through the SOAP protocol [23]. Therefore a Web-service interface for these central services is also implemented on the database servers of the client/server products. The Web-server based configuration takes place on the company site, where the application servers are hosted. The client/server products need an initial setup and further reconfiguration at the customer site. It is necessary to maintain a network infrastructure to deploy configurations to the customer via ISDN or VPN connections.

The main focus of this case study are the *Internet portal*, the *Web-based applications* provided directly, and the *Web-services* that extend the client/server products. Upgrades, deletions, and re-installations of the server-side components are currently performed manually using the network infrastructure. Most tasks are performed with push semantics. The company decides about *which* customer-site receives *which* product version and *when* upgrades have to be performed.

The IETF WebDAV [4] working group works on extensions to the HTTP protocol to support versioning within software projects. Extending the HTTP protocol has the strength to build on a widely used standard Internet protocol. Prior to the WebDAV protocol, authoring tools only supported read/write instead of check-in/check-out semantics. Existing versioning products have been accessed through proprietary HTTP extensions, which forced clients to support different interfaces for each implementation. The RFC [4] also suggests that the interoperability of clients and servers with or without WebDAV support should be guaranteed and that the client implementation has to be simple. The extensions to the HTTP protocol are new methods (headers, mime types, document properties) and new behaviors. The support for versioning, parallel development, and multi-resource locking results in a Web-based versioning systems. Additionally the usage of URIs and a new data format that allows for distribution makes the WebDAV protocol an interesting alternative for a lightweight end-user configuration client. However the distribution is described for packaged software and not yet useable for software under Configuration Management. For now this approach is a solution for web versioning. Further progress in this area in the future will help Content Management systems and web-application builder tools implement Version Management and some Configuration Management tasks.

Interfaces are used to integrate systems based on common communication techniques and language models. J2EE [15, 16, 17] and DCOM [9] are two RPC (remote procedure call) based communication mechanism for distributed systems. A tight integration between two applications should use one of them. A lightweight interface can also be implemented with the HTTP based SOAP [23] protocol and Web-server integrated Java Beans [18]. Besides a protocol implementation it is also possible to integrate the products by customizing scripts or by scripting against a product’s interface using languages like Java Script, VB Script, CGI, Python or others.

The standard mechanism for integrating application functionality consists of an API that is used by external programs or plug-in components running as a part of the integrated software. Another mechanism for synchronizing activities between two applications is the implementation of conditional execution and the usage of triggers. For example a product can be repackaged based on a trigger from the CM repository registering a new component’s version. A further mechanism for integration covers a command line execution utility, and other tools. It is also possible to provide an interface through an open software description like a standard package definition or wide used operation semantics like check in/check out.

All these different integration techniques should be chosen carefully. The more interfaces the more flexible is the integration. It shouldn’t be necessary to choose a specific technique for its functionality but to choose it, because it best fits for the integration needs.

Supported platforms and protocols extend the product’s integration techniques. One important factor is the support for a variety of operating systems. Beside the widely used Windows platform a distribution system might have to support different Unix and Novell server platforms, Macintosh, and other systems. There are products that are just available for a homogenous platform

2. ARCHITECTURE FOR WEB-SERVICE DISTRIBUTION

In this section a reference-architecture is elaborated, based on the requirements and functionalities presented throughout the previous section, and the web-services case study.

The case study "Web-service" consists of Web-services that are part of an Internet portal. These services are used by server processes at the customers’ site and by Web-clients used by the customer and other portal visitors. The customer also uses a client/server application that is not considered here.

The Configuration Management tasks consist of the selection of the correct Web-service component version by incorporating a Version Management system. Further the customer database has to be instantiated and configured. A database version description file can be stored separately in the Version Management system or a specific product component generates the database and specifies the configuration options. The distribution tasks consist of the installation and registration of Web-service components. Additionally consistent code and content changes have to be managed for the Internet portal as well as integration of Content Management functionalities. Additionally the activation of the components and the changes to the customer configuration, have to be synchronized and the customers’ database has to be setup. An (de-) activation and uninstall of components and databases has to be supported. For these tasks a distribution system is integrated with a Web-server for the portal and an application server for the Web-based components. Further, the customer database and application servers are part of the infrastructure and a Version Management system interface is implemented for the configuration related tasks.

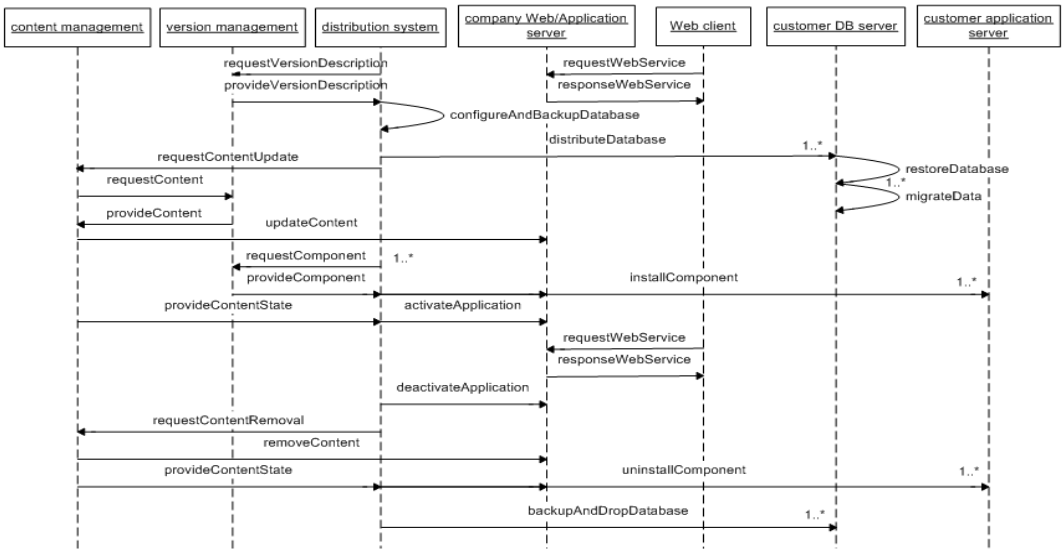


Figure 1: Case study "Web-service" processes

The processes shown in Figure 1 can be described as follows: First the distribution system requests a version description for the customer database. The description covers the database scheme and the configurable application options. In conformance to this information the database is configured and customized. A database backup file is generated and distributed to the database server. The database is restored and optional configurations can be done on-site. By executing scripts against the database from the distribution system site, state for the distribution task can be maintained. Then a content update request is performed corresponding to the distributed database version. The Content Management system can use its own version control system or incorporate an existing version control through an additional interface. The content is then updated on the Web-server and changes to the content state are provided to the distribution system. The distribution system contacts the Version Management for all corresponding components asynchronously. The Web-based components can be installed without any specific distribution system or external packager and installer application, reducing the expense of a distribution infrastructure. Enterprise Java Beans or .NET components are registered by procedures of the application server that can be automated by the distribution system implementing a remote interface. The corresponding mechanisms are described in [15] and [11]. State information can be used for synchronization.

By activating the application, new versions of the web-services become available. Through the use of different entry pages provided by the Content Management, clients can be redirected to the appropriate site implementing the new web-services. In a staged distribution model, a restricted area can be given access first to test for consistency. Next the site is made available to all clients. For Web-services the registration information defined with the WSDL [22] protocol has to be updated. In a test environment a different registration has to be used temporarily. An activation and deactivation procedure has to implement an interface to registration functions provided for example with the Microsoft SOAP toolkit [11]. After activation in the production environment a period of time has to be considered to allow all clients to migrate to the new services. Afterwards a deactivation of the old application is performed and a content removal request is sent to the Content Management. The content is removed from the Web-server and content state information is provided for the distribution system. For the application server components an uninstall procedure is executed by implementing the described interface. Finally a database backup is performed and the database is dropped.

The customer related install und uninstall procedures are repeated for several sites depending on the participation in specific Web-services. Throughout the process, clients asynchronously request services from the Web-server. As long as an application is activated clients can request its services. The active periods have to overlap to provide optimal service availability.

The distribution part can be implemented with an SDE-infrastructure such as Novadigm [12], NET Deploy [6], Tivoli [21], SMS [10] or Marimba [8]. Content Studio [13] might be used for Content Management and ClearCase [1] for Version Management.

2.1 Analysis of the case study

This section covers the functionality required by the Web-Services case study in terms of the criteria discussed above. The degrees of the respective functionalites are visualized using net diagrams, as depicted in Figure 2. The stronger requirements for criteria are, the longer is the corresponding vector.

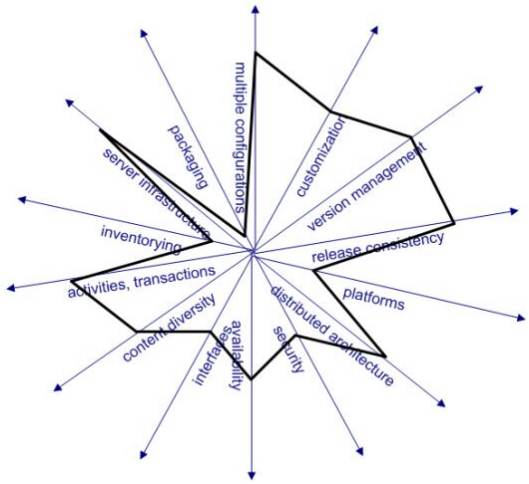
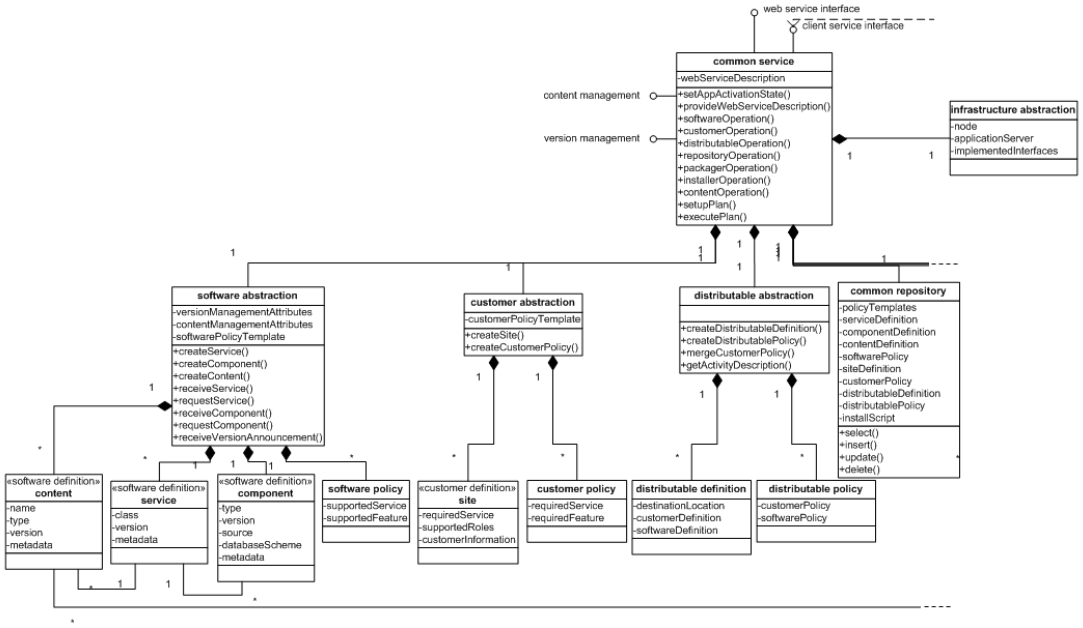


Figure 2: Case study "Web-Services" key factors

The larger the created area, the higher are the overall requirements with respect to the chosen factors. The requirements profile of our case study "Web-service" is shown in Figure 2. The criteria server infrastructure, distributed architecture, activity, and transaction based distribution as well as the support of multiple configurations, are highly relevant. The configurations are complex because the database configuration and setup is not separated from the distribution process. Because the company itself develops the application (business logic), parts of the configuration are highly customized. More frequent software changes due to a high responsibility for customer requirements require an integrated Version Management. Specific to the scenario is the Content Management interface. The database, the customized products, and the content integration extend the content diversity. Two aspects influence the availability requirement. First, the installed products also work without the distribution system, but during staging of a product it is necessary to provide continuous availability for the clients of the Web-services. Secondly, security is already implemented for customer sites to enable remote administration of the products. This part is additionally sensitive because of the exchange of patients' medical data. The Web-services have to be secured to prevent misuse by non-authorized software. Platform support is less relevant in our case study, because Microsoft operating systems, database systems, and Web-servers are used.

2.2 Web-service model logical view

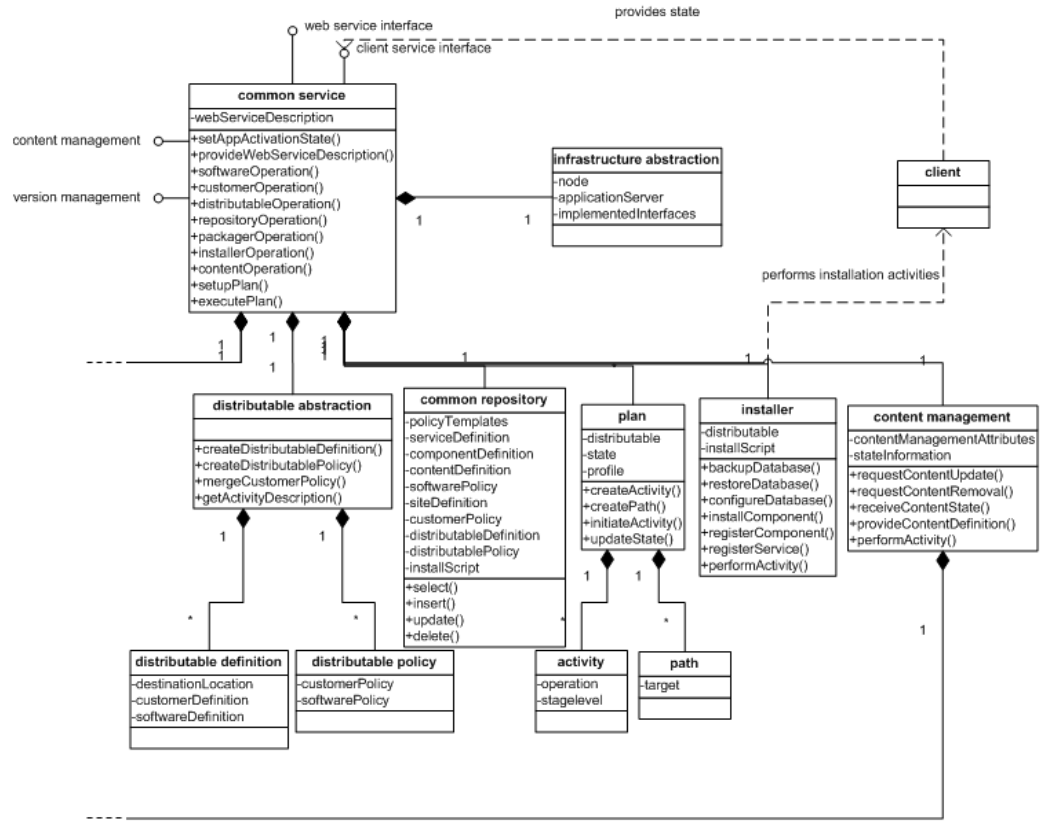
In this section we present the logical view for the model as well as its classes related to the configuration process, as depicted in Figure 3.



The web-service model uses a software- and a customer abstraction and omits the target abstraction. The software abstraction uses a definition of a Web-service and of its components as well as a content definition. All classes contain a name and version attribute. The content and the component classes additionally provide a type attribute

that is used to identify the operating system or application server registration method for components and the way of displaying content for Web-browser. The service additionally contains a class attribute that is used for application negotiation (e.g. print service, file service). All definitions contain a metadata attribute that is a more complex data type and consist of all other attributes used to describe the structure of the service, component, and content classes. Normally this information is used to install and register files, to know about deadlines of content actuality and to avoid the necessity for a target abstraction, an inventorying mechanism or other expensive infrastructure with similar functionality. Further, the software abstraction has a corresponding policy that implements required/support semantics for specific services or features of services. The metadata has to contain information about these restrictions otherwise rules regarding this data cannot be resolved. The software definition can be used to generalize the metadata definition found in different service and component models like COM, CORBA or EJB. The customer abstraction is therefore primarily used to implement and resolve the requirements for services and features. Implementers of this model have implicit knowledge about the customers and can configure and customize the customer definition manually or with support from a directory service. Payment, licensing and other information is irrelevant for this case study. The customer abstraction could also be implemented as a target abstraction, but its policies depend more on organizational than on infrastructural requirements, which should be used to distinguish between those practices.

Figure 4 shows the classes related to the distribution processes. The distribution process can be best described as "staging-like". First, most components distributed are self-describing, perform self-registration, are distributed, fine-grained, and wide-spread. Therefore the model omits a packager class. Second, a Content Management system is integrated that provides an interface to the distribution system for staging content. Content related activities have to be integrated into the process that basically executes registration functions and performs installer-based activities for components that are not able to be registered without an installation function like database-related activities where backup and script files have to be executed on the target machine. Further a history class is omitted because the distribution process doesn't adapt installation activities based on this information. The plan class provides profiles that allow different distribution activities related to the same set of distributables and the same path definitions. This functionality allows a mixture of service and component releases and subsequent content updates to be modeled and executed using a similar infrastructure. The activity class additionally contains database and content related functionality and operates on distributable definitions (single or several files) rather than on packages. The client shown in Figure 4 represents the target platform. The target is a database server or an application server. Nevertheless state information is provided that is used to update the state of the staging process.



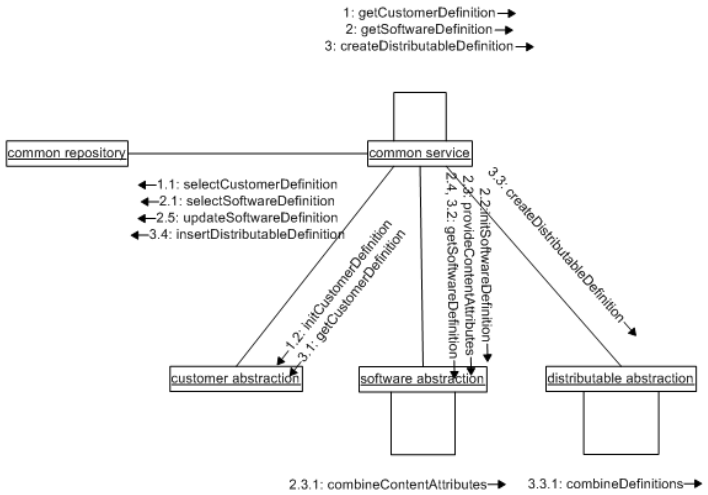


Figure 5: Web-service model process view - configuration

2.3 Web-service model Process view

In this section the process view of the Web-service architecture is shown. First, the distributable definition creation is shown in Figure 5. The customer definition is read from the database and initialized (1). The software definition is read, initialized, and in the third step the content metadata including other attributes are provided through the Content Management interface (2.3) and combined with the existing attributes of the content definition of the software abstraction (2.3.1). Then the software definition is provided (2.4) and the data in the common repository is updated (2.5). The third phase for distributable creation (3) can be compared to the other client/server and standard-software architectures with respect to the different attributes stored. Figure 6 depicts the process of plan creation and execution as required by a Web-service-based model. The Content Management system is integrated and a staging-like approach is chosen to provide the required flexibility in a distributed environment.

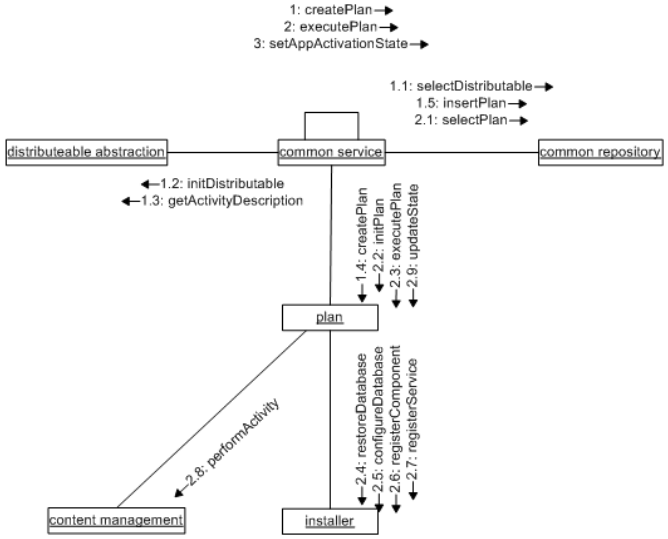


Figure 6: Web-service model process view - distribution

The distribution process contains three phases. In phase one a distribution plan is created (1), which is related to all required distributables as stated in the corresponding policy and contains all activities for component, database, and content installation. First, the distributables are selected from the repository and initialized (1.1, 1.2). Second, a description containing Web-service related activities is provided (1.3) and used in the operation of plan creation (1.4). This step can be further detailed for the path and activity subclasses. Finally the plan is stored in the repository (1.5). In the second phase the created plan is executed (2). The plan is selected from the repository and initialized (2.1, 2.2). The plan is then executed and based on its activity definition one or more of the described service, component, database, and content operations are performed (2.4-2.8). Activities that should be supported cover the restore and configuration of a database, the registration of services and components and the staging of content. Further activities for deregistration or database backups should be considered. An important issue to support staging regards content updates that are independent of the deployed components.

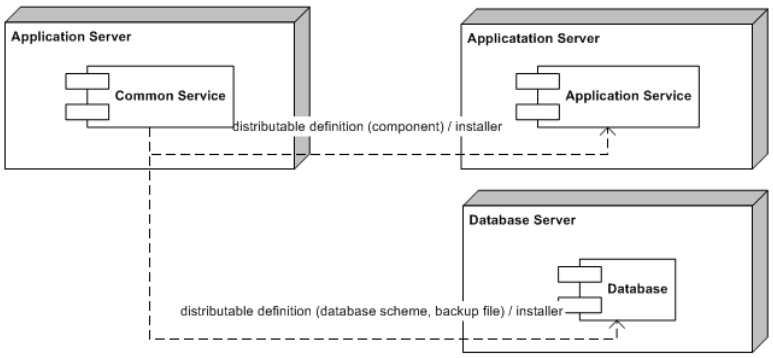


Figure 7: Web-service model deployment view

The plan class therefore supports different profiles that cover the base distribution and subsequent content updates.

Figure 7 depicts the deployment view of the Web-service architecture as well as the system nodes interacting in this model.

An application server hosting the common service is required. It is able to host a Web based application service and integrated HTTP support, object pooling, transaction support and many other advantages. The Web-service-based system consists of several application and database servers hosting content, components, and databases. The distribution process provides distributables and installer applications to the destination nodes for installation activities on the target site. The common service might also be distributed across several sites for geographically distributed application servers. In this case a repository replication is most appropriate to allow on-site installation of components and a distribution consistency of the Web-service functionality. Additionally all nodes can be clustered to provide load balance and fault tolerance.

3. CONCLUSIONS

Software distribution environments have become important parts for software vendors to distribute their latest software versions and updates effectively and efficiently to customers. Configuration management, version management, and software description languages are essential elements for software distribution and should be considered in an integrated fashion.

The contribution of this paper was based on the evaluation of 12 software distribution environments (SDEs) and their software architectures [14]. In this paper we presented a real world case study for web-services distribution. Based on our previous work [1] we discussed those quality attributes that can be used to assess architectural elements for web-services based software distribution. We have presented an architecture for web-service software configuration, distribution, and deployment that is based on a real-world case study. The main criteria required for such an environment is the support for multiple configurations, customization, version management, release consistency, and server infrastructure. Future work will concentrate on quality attributes and reference architectures for software distribution environments.

4. REFERENCES

[1] Anzböck, R. (2002). Architectures for a Software Distribution Environment, Master's thesis. Distributed Systems Group, Technical University of Vienna, Austria 2002

[2] Dart, S. (2000). Configuration Management: The Missing Link in Web Engineering, Artech House, London.

[3] IBM „Web Services Flow Language (WSFL 1.0)", May 2001. http://www-4.ibm.com/software/-solutions/webservices/pdf/WSFL.pdf

[4] IETF Network Working Group, RFC2518, HTTP Extensions for Distributed Authoring - WEBDAV, http://www.ietf.org/rfc/rfc2291.txt, 1998

[5] InstallShield, InstallShield for Windows Installer Whitepaper,http://www.installshield.com/isd/resources/ 2001

[6] ManageSoft, NETDeploy Technical Specification, http://www.managesoft.com/products/technical/index.xml, 2001

[7] ManageSoft, Managing software for mobile and remote users, http://www.managesoft.com/products/mobile.xml, 2001

[8] Marimba, Marimba Management Solution, http://www.marimba.com/products/intro.htm, 1999

[9] Microsoft Corporation, DCOM, http://www.microsoft.com/com/, 2001

[10] Microsoft, System Management Server Reviewer's Guide,http://www.microsoft.com/smsmgmt/ 1998

[11] Microsoft, DotNET Framework http://www.microsoft.com /net, 2002

[12] Novadigm, Radio Software Manager Factsheet, http://www.novadigm.com/products/radia/index.html, 2001

[13] Rational, An overview of Rational Suite ContentStudio, http://www.rational.com/products/-cstudio/whitepapers.jsp, 2001

[14] Shaw, M and Garlan, D., Software Architecture: Perspectives on an emerging discipline, Prentice Hall, 1996

[15] Sun Microsystems, Enterprise Java Beans, http://java.sun.com/products/ejb/, 2001

[16] Sun Microsystems, JAVA, http://java.sun.com, 2001

[17] Sun Microsystems, Java 2 Enterprise Edition, http://java.sun.com/j2ee, 2001

[18] Sun Microsystems, Java Beans, http://java.sun.com/products/javabeans, 2001

[19] Sun Microsystems, Webstart, http://java.sun.com/products/javawebstart/, 2001

[20] Thatte, S. (2001) “XLANG. Web Services for Business Process Design“, May 2001, http://www.gotdotnet.com/team/xml_wsspecs/-default.aspx

[21] Tivoli, Tivoli Software Distribution Factsheet, http://www.tivoli.com/news/press/pressreleases/en/2000/supplement/software_dist_factsheet.html, 2001

[22] World Wide Web Consortium, WSDL Web-service Description Language, http://www.w3.org/TR/wsdl, 2001

[23] World Wide Web Consortium, SOAP (Simple Object Access Protocol), http://www.w3.org/TR/2001/WD-soap12-part1-20011002/, 2001