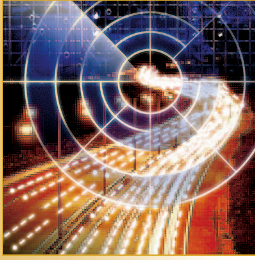

Editors: Michael Huhns • *huhns@sc.edu*
Munindar P. Singh • *singh@ncsu.edu*

Approaches for Service Deployment

Traditional IT service-deployment technologies are based on scripts and configuration files, but their limited ability to express dependencies and verify configurations results in hard-to-use and erroneous system configurations. Emerging language- and model-based tools promise to address these deployment challenges, but their benefits aren't yet clearly established. The authors compare manual, script-, language-, and model-based deployment solutions in terms of scale, complexity, expressiveness, and barriers to first use.

Vanish Talwar and Dejan Milojicic
Hewlett-Packard Laboratories

Qinyi Wu, Calton Pu, Wenchang Yan, and Gueyoung Jung
Georgia Tech

Services — standalone software components that encapsulate and present useful functionalities — can be composed into overall computing systems or single applications. In a broad sense, they can include business services as well as modules such as transaction services or databases; moreover, they can be realized as Web or Grid services, or even as component services in an operating system.[1] This shift points to a general view of service-oriented computing.

In SOC, changes to a service component must be propagated or contained so that the services using that component continue to function correctly. Unplanned changes, such as those caused by failures, must also accommodate dependencies — services that depend on a failed service, for example, might need to be restarted. A concrete and serious challenge in SOC is the long-lived and evolving nature of large-scale services. A system update at even a moderately sized data center can require changes to 1,000 machines, some of which might have interdependencies among their services. A typical Web-based e-commerce application, for instance, consists of a three-tier system — the database, application, and Web server tiers — and each tier has its own interdependencies.

The scale and complexity of today's IT systems make them increasingly difficult and expensive to administer and deploy; as SOC becomes more prevalent, the question of which deployment approach is best gains importance. New computing models offer some answers,[2–4] but recent studies show that most IT service companies have the same requirements: software-deployment management dominates system administration costs,[5] and configuration is a major source of errors in system deployment.[6]

Today's deployment tools provide varying levels of automation, typically classified as manual, script-, language-, or model-based approaches. Automation

 Published by the IEEE Computer Society 1089-7801/05/$20.00 © 2005 IEEE

of service deployment is beneficial for improved correctness, speed, and documentation, but as Figure 1 shows, automation comes at an increased cost in development time and administrators' learning curves. This initial overhead might be acceptable if overall gains are significant and worthwhile, but IT managers face a more general question: which of these approaches should they adopt (and when)?

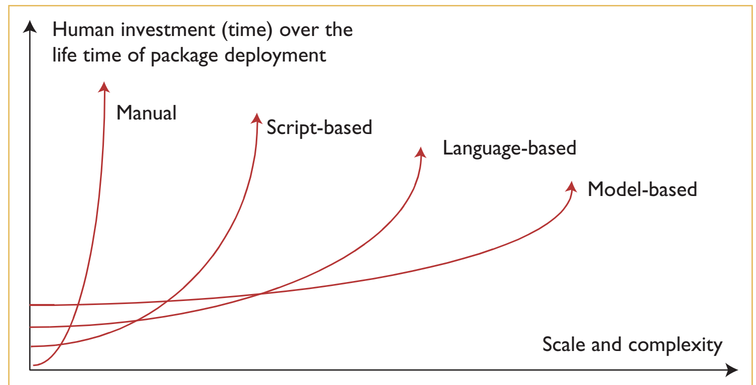

Figure 1. Trade-offs between initial cost (development and learning) and repeated use cost. The level of a tool's automation pushes costs up earlier in the development cycle, but developing tools, learning from them, and creating templates with them pays off as complexity increases.

Use-Case Scenario

Let's look at a real-life scenario that emphasizes the problems with dependencies, failures, and the need to document changes.

Sarah has installed Java PetStore on a three-node Windows-based cluster. She manages it with a remote tool, so she's configured it to be part of a remote domain. It took her a few days to install all the required packages, applications, and tools; because she had specific requirements, she had to make certain changes in several steps of the configuration and deployment. Each part of the installation had its own instructions, so she documented everything in a notebook. Because the application had so many dependencies, she had to manually configure packages with the configuration parameter values from other packages — for example, for node names and IP addresses. She repeatedly had to enter these values in different places, so she occasionally entered them incorrectly. After Sarah used PetStore for several days, an application on the remote system rebooted all her systems because several Windows updates needed to be applied. Unfortunately, this action erroneously reimaged some of her systems, and Sarah had to reinstall everything from scratch.

In this scenario, a more sophisticated deployment tool would have benefited Sarah in many ways. First, she wouldn't have had to do the installation manually each time. A carefully drafted template describing the steps and tools would have helped during the first deployment and even more so during subsequent deployments. Next, the dependencies between certain components could have been instantiated in one spot with variable names used at other areas, reducing the need to make changes and the likelihood of making errors. Moreover, the deployment tool could have evaluated many values at the time the components were started, eliminating the need for manual initiation altogether. The remote configuration tool that caused the incident could also have been part of the configuration, which would have automated changes in interdependent systems. Finally, the system's documentation would have been very coherent and consistent, reduced to a single configuration file, documenting an absolute minimum number of parameters and making subsequent changes easy. (See the sidebars on "Related Work in Service-Oriented Computing" on p. 74 and "Service Deployment Standards" on p. 79 for more information.)

Examples of Deployment Approaches

To better illustrate how to automate Sarah's scenario, let's look at individual technologies. We use Nixes, SmartFrog, and Radia as examples for the script-, language-, and model-based deployment approaches, respectively; each part of Figure 2 (next page) illustrates deployment steps for them. Table 1 (p. 73) presents a breakdown of SOC approaches according to their levels of automation. We will quantitatively compare these steps later in the article. From Table 1, we can see that the increased level of abstraction enables higher levels of automation.

Nixes (www.aqualab.cs.northwestern.edu/nixes.html) is a tool used to install, maintain, control, and monitor applications on PlanetLab (www.planet-lab.org), a globally distributed test bed for experimentation with planetary-scale network services. It consists of a set of bash (Bourne again shell) scripts, a configuration file, and a Web repository, and it can automatically resolve the dependencies among Red Hat Package Manager (RPM) packages. For small-scale systems, Nixes is easy to use: users simply create the con-

Propagate packages to Web repository
Log in to Web server host
Unpackage Apache
Build and install Apache
Edit `httpd.conf`
Start Apache
Inspect Apache process list
Log out of Web server host
(a)

```
WEB_SERVER=poseidon.cc.gatech.edu #Web repository
WEB_PORT=8080
WEB_DIR="/binaries"
APACHE_ARCHIVE=httpd-2.0.49.tar.gz#Binary Archives
DIR=/usr/local #Installation Directory
APACHE_INSTALL_HOME=$DIR/httpd-2.0.49
cd $DIR
if [[ ! -d $APACHE_HOME ]]; then
wget
$WEB_SERVER:$WEB_PORT/$WEB_DIR/$APACHE_ARCHIVE
tar -xzf $APACHE_ARCHIVE
cd $APACHE_INSTALL_HOME
./configure  > /dev/null
make > /dev/null
make install > /dev/null
cd $DIRECTORY
rm -r -f $APACHE_INSTALL_HOME
rm $APACHE_ARCHIVE
fi
```
(b)

```
ApacheInstaller extends GenericTarInstaller {
webServerHost "poseidon.cc.gatech.edu";
tarLocation "/binaries";
file "httpd-2.0.49.tar.gz";
installScript extends vector
   — ATTRIB downloadApache;
   — ATTRIB cdApacheHome;
   — ATTRIB unTarApache
   — ATTRIB configureScript;
   — ATTRIB makeScript;
cdApacheHome extends concat
   — "cd ";
   — ATTRIB installLocation;
untarApache extends concat
   — "tar -xzf ";
   — ATTRIB file;
configureScript "./configure";
makeScript "make install";
actions:downloadApache extends Downloader {
        url extends concat {
            — "http://";
            — ATTRIB webServerHost;
            — ":";
            — ATTRIB webServerPort;
            — ATTRIB tarLocation;}
        toLocalFile — ATTRIB installLocation; }}
sfConfig extends ApacheInstaller;
```
(c)

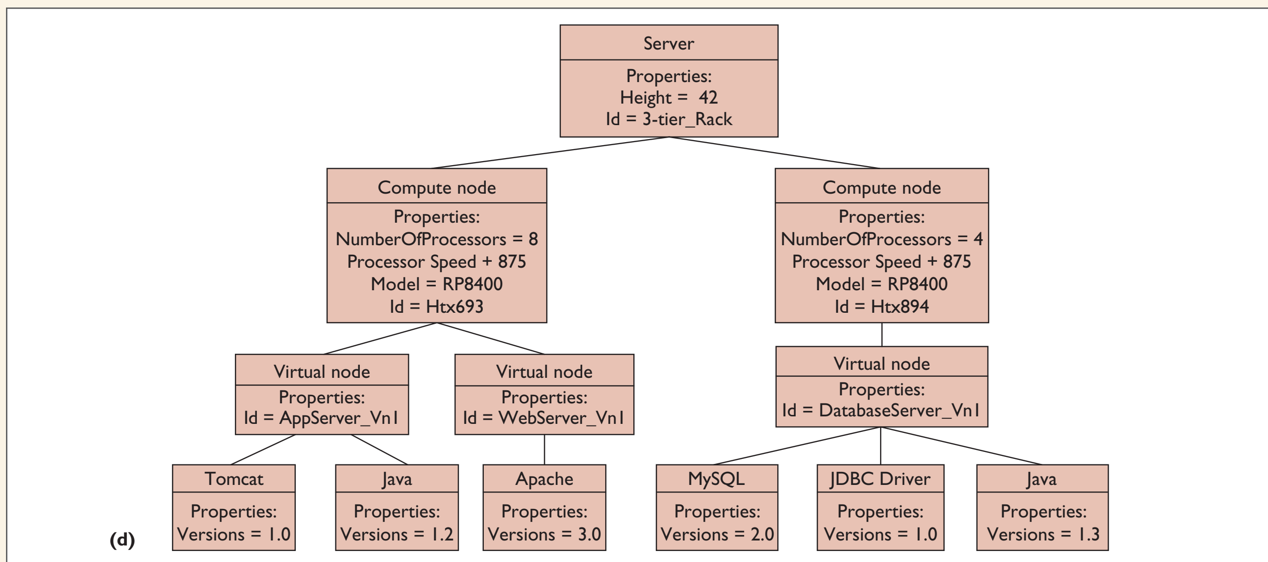

Figure 2. Illustration of deployment steps. (a) For the manual approach, we install Apache manually; (b) for script-based, we run a bash script to install it; (c) for language-based, we use the SmartFrog language to install it; and (d) for model-based, we use a visual form of the system model.

Table 1. Comparison of deployment approaches.*

Deployment phases	Deployment approaches			
	Manual	Script-based	Language-based	Model-based
Development	None	Develop tools and installation and startup script templates	Develop configuration language, parser, tools, and specification templates	Develop schemas for models and tools for lifecycle management, create instances of models, update software dependency model, and create resource models
Design	None	Populate application templates with customer-specific attributes and construct workflow	Populate application templates with customer-specific attributes and construct workflow	**Select package models from best-practice model and perform dependency analysis**
Operational	Distribute packages to repository; log in to each target node; download, configure, and install; activate; and verify	**Invoke distribution module, installation and ignition workflow, and verification scripts**	**Invoke distribution module, installation and ignition workflow, and verify notification events**	**Update unified interoperability model, invoke distribution module and installation and ignition workflow, and verify notification events**
Change	Manually detect and adapt to changes	**Discover and react to changes**	**Discover and react to change, and load predetermined component**	**Automatically react to change, reflect on model, and activate adaptation and execution**

*(darker fonts indicate more automation)

figuration file for each application and modify scripts to deploy on target nodes. Unfortunately, Nixes isn't effective for large and complicated systems because it doesn't provide an automated workflow mechanism.

SmartFrog (SF; www.smartfrog.org) is a framework for service configuration, description, deployment, and lifecycle management.[7,8] It consists of a declarative language, engines that run on remote nodes and execute templates written in that language, and a component model. The SF language supports encapsulation (similar to classes in Python), inheritance, and composition to allow configurations to be customized and combined. It also enables static and dynamic bindings between components to support different ways of connecting components at deployment time. The SF component model enforces lifecycle management by transitioning components through five states: installed, initiated, started, terminated, and failed. This allows the SF engine to automatically redeploy components in case of failure.

Radia (www.novadigm.com), a change-and-configuration management tool, uses a model-based approach. For each managed device, the administrator defines a desired state, which is maintained as a model in a central repository. Clients on the managed device synchronize to this desired state, which triggers deployment actions.

In our experiments and comparison, we also consider a hypothetical model-based deployment solution, based on our experience with Radia, that uses the following models: package (configuration, installation, registry entries, binaries, and such), best practices (matching the needs of specific customers), software dependency (deployment relationship with other software components, operating systems, and hardware), infrastructure (servers, storage, and network elements), a software inventory (currently installed software), and interoperability among management services models.

Evaluation Metrics

Quality of manageability is a measure of the ability to manage a system component. Quantitative QoM measures include:

Related Work in Service-Oriented Computing

We take inspiration for comparing service deployment approaches from software engineering — specifically, the methodologies used for comparing programming languages, domain-specific languages, and software products. Programming languages are typically compared in terms of execution time, ease of use, lines of code, length, amount of commenting, and so on.[1]

Our work relates to domain-specific languages such as the application of compiler extensions to identify errors in systems programming.[2,3] In the same way that these languages enable easier error detection, automated approaches to service deployment and configuration prevent human errors and make the process easier. A comparison between the manageability of Oracle 9i and Oracle 10g motivated us to use number of steps as a metric.[4] Itzfeldt further classifies maintainability as modularity and complexity, testability, understandability, and modifiability, and derives the following quality metrics for software management: size, control structures, data structure, and flow.[5]

Carzaniga et al. characterize product, site, and policy models among software deployment technologies.[6] In our classification, the script-based approach supports the deployed services site (data center) model; the language-based approach supports the product and site models; and the model-based approach supports all three models.

Our work differs from this related research in three ways. First, the previous work hasn't characterized the full spectrum of deployment automation options from manual to model-based approaches. Second, there is no previous quantitative comparison of deployment solutions. And finally, we formulate a set of metrics for comparing deployment approaches.

References

1. L. Prechelt, "An Empirical Comparison of Seven Programming Languages," *Computer*, vol. 33, no. 10, 2000, pp. 23–29.
2. D. Engler et al., "Checking System Rules Using System-Specific Programmer-Written Compiler Extensions," *Proc 4th Usenix Symp. Operating Systems Design and Implementation* (OSDI), Usenix Assoc., 2000, pp. 1–16.
3. F. Mèrillon et al., "Devil: An IDL for Hardware Programming," *Proc. 4th Usenix Symp. Operating Systems Design and Implementation* (OSDI), Usenix Assoc., 2000, pp. 17–30.
4. *Oracle Database 10g and Oracle 9i Database Manageability Comparison*, tech. report, Oracle, Feb. 2004; www.oracle.com/technology/products/manageability/database/pdf/twp03/oracle10g-oracle9i_manageability_comparison.pdf.
5. W.D. Itzfeldt, "Quality Metrics for Software Management and Engineering," *Managing Complexity in Software Eng.*, IEE Computing Series 17, R.J. Mitchell, ed., Short Run Press, 1990, pp. 127–151.
6. A. Carzaniga et al., *A Characterization Framework for Software Deployment Technologies*, tech. report CU-CS-857-98, Dept. of Computer Science, Univ. of Colo., Boulder, Apr. 1998.

- number of lines of configuration code (LOC) for deployment,
- number of steps involved in deployment,
- LOC to express configuration changes, and
- time to develop, deploy, and make a change.

LOC is a relevant metric because of the maintainability of configuration (making changes over the configuration's life time), which is inversely proportional to LOC: the smaller and more expressive a configuration, the easier it is for a system administrator to install, configure, and maintain. Similarly, number of steps is proportional to the time and cost of engaging a human operator.

Qualitative QoM measures include:

- the ability to automate the management process, including its adaptability to changes (such as failures or load);
- robustness, expressed in terms of misconfigurations;
- the ability to express constraints, dependencies, and models; and
- barriers to first use of the deployment tool.

Automation is the most important qualitative metric, because it improves time to deploy and decreases the likelihood of human error.

The Experiments

We conducted two sets of experiments for service deployment to empirically compare a script-based approach, SF, and a model-based deployment approach. The first set studied the deployment of n-tier test beds — specifically, a three-tier test bed that exemplifies a typical Web system administrator's work. The test bed consisted of a Web server, an application server, and a database and was complex enough to have numerous dependencies among various components across the tiers. For this test, we compared a bash script-based approach with the SF language-based approach.

The second set of experiments involved configuration parameters for MySQL (http://dev.mysql.org), a well-known open-source database management system (DBMS) that has a set of tunable configuration parameters for setting up a database. We chose an application with an interesting set of parameters that must be tuned for dif-

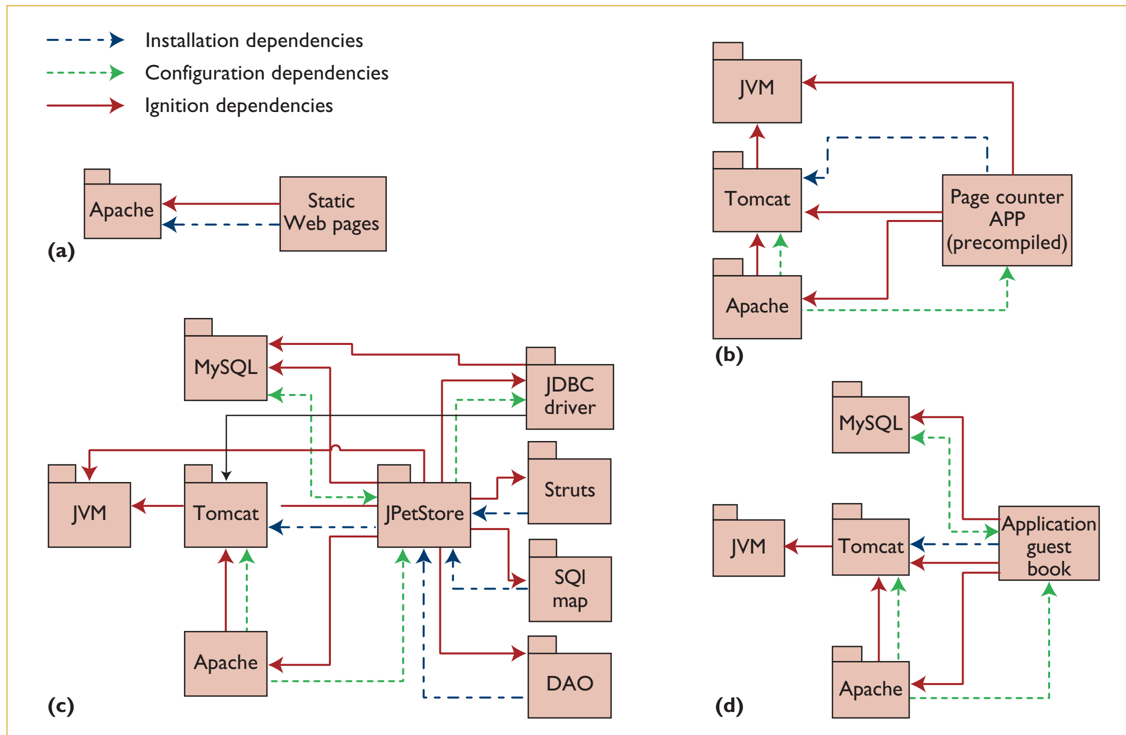

Figure 3. Complexity definitions. The service configuration has four levels of complexity: (a) a simple, one-tier test bed, (b) a medium, two-tier test bed with a simple application, (c) a complex, three-tier test bed with Guest Book application, and (d) a very complex, three-tier application with Java PetStore application.

ferent system setups, and compared native MySQL configuration files with SF.

N-Tier Test Bed Deployment

We conducted our experiments with SF 3.0, a Web server (Apache 2.0.49), an application server (Tomcat 5.0.19), a DBMS (MySQL 4.0.18), and the PetStore (iBATIS JPetStore 4) and Guest Book applications, all running on Linux. We deployed the *n*-tier test bed's components on separate nodes, but each component had native configuration files: `httpd.conf` for Apache, `server.xml` for Tomcat, `web.xml` for Web applications using Tomcat, and `my-*.cnf` for MySQL. We wrote bash scripts and SF components for the components' installation and ignition phases.

Our first experiment measured the number of steps and LOC against the *n*-tier test bed's scale (number of nodes) and complexity (a function of the number of software components and the number of installation, configuration, and ignition dependencies). The system's scale is varied through horizontal scaling of the tiers. For horizontal scaling, the ratios of the Apache Web servers to the Tomcat application servers are 1:2, 2:4, and 4:8. Figure 3 describes the levels of complexity.

We identified a test workload consisting of the set of tasks an administrator would have to perform to deploy the test bed. The workload covers the test bed's installation and ignition, including:

- creating the specifications for the software's configuration, installation, and ignition;
- creating workflow descriptions;
- distributing binaries, specifications, and workflow descriptions;
- executing the installation workflow descriptions to install the test bed;
- executing the ignition workflow descriptions to activate the test bed; and
- verifying that the installation and ignition completed successfully.

Our results represent the deployment effort for an end administrator and reflect the cost of deployment incurred beyond initial development. They don't include development of the script interpreter, language parser and engines, models stores, and so on, but they do include using these tools for developing configurations, deploying them, and configuring and starting services.

Figure 4 (next page) shows that as a system's complexity increases, the difference in the number

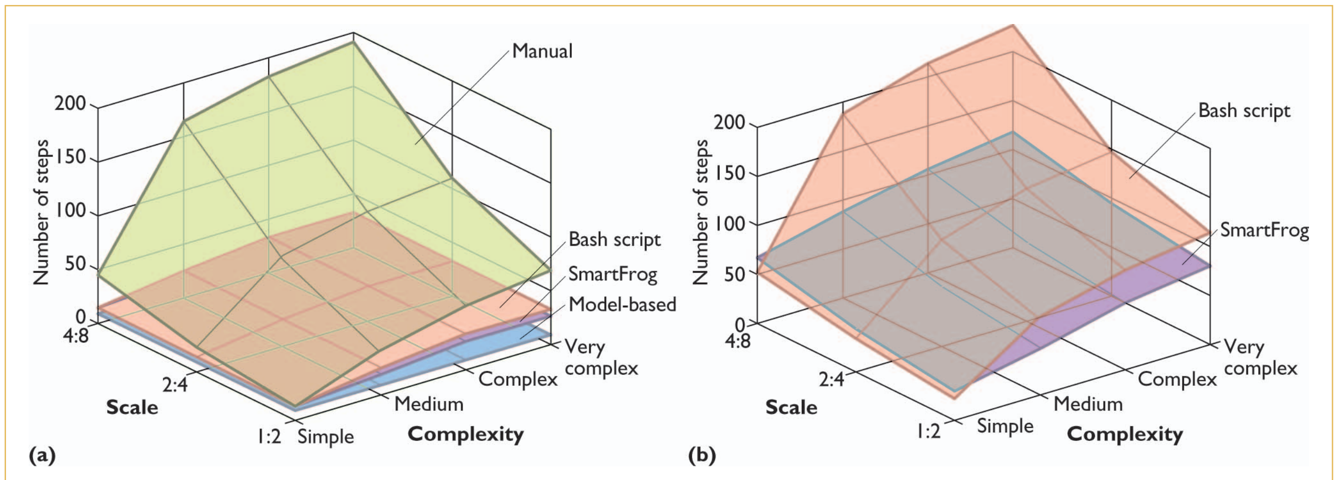

Figure 4. Number of steps as a function of scale and complexity. The graph shown for bash and SmartFrog are from real service configurations described in Figure 3; the results for the model-based approach are estimated and represented as a flat plane in the graph.

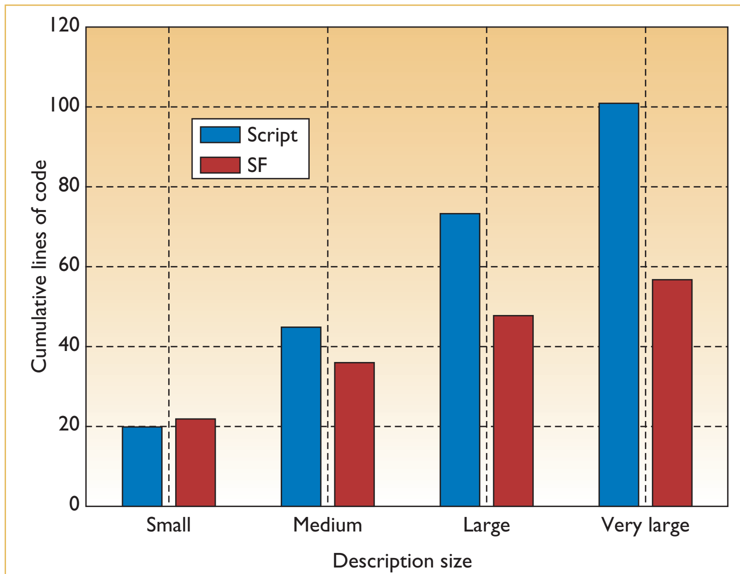

Figure 5. Cumulative lines of configuration code. As the deployed service's complexity increases, the configuration description size grows slower for the language-based approach than for the script-based deployment approach.

of steps to be performed widens between manual and script- or SF-based approaches. For a manual approach, the number of steps is linear in the number of nodes because the administrator must repeat the steps for each node. However, the number of steps for the script and SF cases remains constant because the configuration developer can reuse existing code. In comparison to the manual, script-based, and SF-based approaches, the (hypothetical) model-based approach provides the advantage of a constant number of steps for the administrator to perform with varying scale and complexity.

SF benefits include a reduction in the number of steps and LOC through automation, workflows, and the ability to handle added dependencies via a link-reference feature, which is the ability to link to other configuration descriptions by their name and thereby make changes only at the place of definition. (Anderson and colleagues present more details about the automation of language- and script-based deployment solutions elsewhere.[9]) The SF language also allows specification of the sequencing relationships among different software components through workflows and subworkflows.

MySQL Deployment

We conducted the MySQL experiment with the configuration file for version 4.0.18. We represented the configuration information with the SF language to take advantage of its inheritance and link reference.

This experiment measures the lines of configuration code to maintain and the number of configuration values to be edited in response to system changes. We specifically focused on MySQL's performance-tuning parameters, which include port number parameters, key and sort buffer sizes, and read and write buffer sizes. The experiment varies the setup complexity from small (10 to 100 infrequently used clients that consume few resources) to medium (100 to 1,000 clients used with Web servers) to large (1,000 to 5,000 clients on a dedicated server) to huge (more than

Table 2. Applicability of deployment approaches.*

Characteristics	Deployment approaches			
	Manual	Script-based	Language-based	Model-based
Change	**Simple**	**Configuration**	**Dependency**	**Design**
Repeat/scale	Rare/small	**Many/large**	**Many/large**	**Many/large**
Complexity	Simple	Simple	**Complex**	**Complex**
Documentability	None	Deploy time	+Incremental	**+Runtime**

*(darker fonts represent preferable choices; plus signs represent incremental support from the cell to the left)

5,000 clients on a dedicated server). We tested the SF and MySQL default configuration files as the deployment approaches of choice.

Figure 5 compares the cumulative number of LOC that a MySQL administrator must maintain as the system's complexity increases from simple to huge. An administrator maintaining configurations for only small setups maintains the same LOC with language- and non-language-based approaches, but for a medium setup, the cumulative number of LOC to maintain doubles with non-language-based approaches. For a medium setup, however, a language-based approach such as SF introduces fewer specific LOC for that size of setup or LOC required for tuning the parameters whose values differ from those for a small setup. The medium setup can inherit the rest of the configuration from the small setup. Similar reasoning holds true when we introduce large or very large systems. The difference in the cumulative LOC for maintaining a huge system using a language- versus a non-language-based approach is quite significant. (Experiments with the configuration values to be edited in response to system changes, as well as with the deployment and installation time, appear elsewhere.[10])

MySQL is simple compared to software systems such as those that manage supply chains. We believe that complex services favor even more deployments at a higher level of abstraction (such as with language- and model-based approaches).

Complexity, dependencies, configuration space, and requirements for performance, availability, and scalability of services are all dimensions that bear on our conclusions. Simple service configurations lend themselves to manual or script-based deployment, but more complex services with more requirements are better suited for model-based deployment.

Comparison

Ultimately, no universally optimal solution exists — the best approach is the one that closest matches the deployment need. When the number of deployed systems is small or systems' configurations rarely change, a manual solution is the most reasonable approach. For services with more comprehensive configuration changes, a script-based deployment strategy offers several benefits. In larger environments in which changes involve dependencies, language-based solutions are likely a better choice. If the changes also involve significant perturbations to the underlying service's design, the model-based approach is probably ideal. From the perspective of documentability, manual deployment offers poor support; scripts offer minimal support for the deploy-time changes; language-based approaches support incremental documentability based on inheritance and composition; and model-based approaches add runtime documenting by virtue of capturing all the changes in the deployed service's lifetime. Table 2 shows a high-level comparison of deployment approaches in terms of applicability.

Table 3 (next page) provides a more detailed, qualitative comparison between manual, script-, language-, and model-based deployment solutions in terms of automation, self-management, expressiveness, and barriers to first use.

Automation (Self-Management)

Scripts introduce deployment automation through their ability to repeat a set of steps specified in a file and to form closed control loops through events. Language-based solutions extend this ability by introducing lifecycle management through the use of dependencies (for example, order of deployment or redeployment upon failures). Model-based solutions extend automation to design time by enabling dynamic creation of deployment instantiations.

Self-Healing

As an extension of self-management, self-healing enables a system to react to failures. Scripting has

*Table 3. Comparison of deployment approaches.**

Characteristics	Deployment approaches			
	Manual	Script-based	Language-based	Model-based
Solution based on	Human language	Configuration, files, scripts	Declarative language	Models and policies
Automation	None	Event-based closed loops	+Lifecycle management	+Automated design
Self-healing	None	Minimal	+Redeployment, dependencies	+Change design
Expressiveness	None	Partial: dependencies and constraints	Significant: +inheritance, lazy evaluation	Complete: +reuse, correctness, and maintenance
Barriers to first use	None	Low	High	Very high

*(plus signs indicate incremental support in each row compared to the cells on the left)

some ability to react to events and trigger handlers; language-based solutions build on this ability by exploring dependencies and handling redeployment in a more sophisticated way. Model-based solutions can change the deployed system's design as a reaction to the failure.

Expressiveness (Ease of Use)

Expressiveness is of particular interest when deploying large-scale, complex systems. Language-based approaches introduce inheritance, name scoping, and lazy evaluation (the ability to dynamically resolve link references to actual names at runtime) for easier (re)configuration. The model-based approach introduces model- and policy-based support that better captures runtime state and best practices. As a result, going from a manual to a model-based approach brings an increasing level of reuse, correctness, and maintenance.

Barriers to First Use

Manual deployment usually requires little or no a priori knowledge. Scripts are relatively straightforward and require little effort to get started with, although some script programs are quite sophisticated. Language-based approaches require a certain amount of education before a system administrator can use them — for example, the system administrator should learn the syntax and semantics of the language, component model, and APIs. Model development is the largest barrier to the model-based approach, but front-end tools partially alleviate this obstacle.

From the perspective of our programming-language-inspired methodology, the four deployment approaches differ in nature, yet are synergistic. The manual approach is imperative; the script-based one is automated imperative; the language-based one is declarative; and the model-based one is goal-driven. Ease of use and barriers to first use typically determine the optimal choice, but to define the best deployment method in an SOC environment, our results favor the trend toward using a model-based approach because each successful service composition increases total system complexity as well as scale.

We didn't present as rigorous an evaluation of the model-based approach (prototyping or experimentation) as we did for the other deployment approaches, because model-based deployment tools still aren't widely available for SOC. However, our qualitative comparison is useful because it's based on our practical experience with using models. Similarly, we omitted some other aspects of deployment (such as exceptions), but we didn't lose the generality of our conclusions. Exceptions are an important topic for deployment, but addressing them is either orthogonal to the comparison we made or well aligned with the results — moving to higher levels of abstraction such as language- and model-based, for example, might make it harder to understand some of the errors that occurred at the lower levels of abstraction. That said, though, the deployment system should analyze failures at an appropriate level of abstraction and deal with them accordingly.

Integration with development tools such as Eclipse (www.eclipse.org) should both improve ease of use and decrease barriers to first use because of graphical user interfaces combined with default configuration templates. We also plan further examination of deployment in different

Service Deployment Standards

Major standardization bodies such as the Distributed Management Task Force (DMTF; www.dmtf.org), the Global Grid Forum (GGF; www.gridforum.org), the Organization for the Advancement of Structured Information Standards (OASIS; www.oasis-open.org), and the World Wide Web Consortium (W3C; www.w3.org) continue to address service deployment. The GGF's Configuration Description, Deployment, and Lifecycle Management working group (CDDLM), for example, is actively pursuing Web service deployment based on SmartFrog (SF), but adapted to the Web services (see https://forge.gridforum.org/projects/cddlm-wg). SOAP and WSDL have replaced the remote method invocation (RMI) transport, an XML-based language replaces the SF language, and Web services replace RMI-based distributed engines. The CDDLM deployment is an extension of the OASIS Web Services Distributed Management interfaces, and OASIS is exploring deployment in the larger context of the Web Services Resources Framework (WSRF).

The Installable Unit for Deployment Descriptor (IUDD) group, with representatives from InstallShield Software, IBM, Novell, and Zero G Software, has submitted a deployment schema to W3C (see www.w3.org/Submission/2004/SUBM-InstallableUnit-DD-20040712/). The schema is an XML document describing the relevant characteristics of an installable unit of software for its deployment, configuration, and maintenance. IUDD also describes the aggregation of installable units at all levels of the software stack, including middleware, applications, and the logical topology of deployment targets.

The DMTF Application working group has dealt with model aspects of service deployment (see www.dmtf.org/about/committees/applicationsWGCharter.pdf), developing Common Information Models of services relevant for deployment. Significantly more service deployment research and development is occurring within standardization bodies, which is beyond this article's scope.

underlying environments such as PlanetLab, Grid, and Enterprise.

There is an opportunity to develop more elaborate quantitative comparisons, potentially based on software metrics, such as those in software engineering.[11] We plan to pursue some of these approaches in the future. Although such techniques will increase our evaluation's precision, we expect the general conclusions to remain unchanged. Our future work in the area of deployment is moving in two primary directions. First, we're exploring loosely coupled decentralized architectures for building deployment services in wide area systems. We plan to explore the use of Web services standards and then extend and decentralize them, as appropriate, for large-scale systems. Second, we intend to continue using models to extensively automate the deployment and self-healing within complex systems and to increase scalability and ease of use. We plan to integrate the work on models with the loosely coupled architecture described earlier, to create a scalable deployment service for next-generation computing systems.

Acknowledgments

This article is based on a longer paper that will appear in the *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems* (ICDCS 05). We're indebted to Munindar Singh and Michael Huhns for reviewing the article; their comments significantly improved its contents and presentation. Paul Anderson, Martin Arlitt, Jamie Beckett, Patrick Goldsack, Steve Loughran, Jim Rowson, and Carol Thompson reviewed earlier versions of the article. Julie Symons offered the scenario.

References

1. M.N. Huhns and M.P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, 2005, pp. 75–81.
2. J. Wilkes, J. Mogul, and J. Suermondt, "Utilification," *Proc. ACM European SIGOPS Workshop*, ACM Press, 2004, pp. 34–40.
3. *IBM Systems J.*, special issue on utility computing, vol. 43, no. 1, 2004; www.research.ibm.com/journal/sj43-1.html.
4. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, 2nd ed., Morgan Kaufmann, 2004.
5. J. Humphreys et al., *Service-Centric Computing: An Infrastructure Perspective, Outlook, and Analysis*, tech. report 28934, Int'l Data Corp., Mar. 2003.
6. D.A. Patterson et al., *Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*, tech. report UCB/CSD-02-1175, Dept. of Computer Science, Univ. of Calif., Berkeley, 15 Mar. 2002.
7. A. Carzaniga et al., *A Characterization Framework for Software Deployment Technologies*, tech. report CU-CS-857-98, Dept. of Computer Science, Univ. of Colo., Boulder, Apr. 1998.
8. P. Goldsack et al., "Configuration and Automatic Ignition of Distributed Applications," *2003 HP Openview Univ. Assoc. Workshop*, Hewlett-Packard Labs, 2003; www.hpl.hp.com/research/smartfrog/papers/SmartFrog_Overview_HPOVA03.May.pdf.
9. P. Anderson et al., "SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control," *Proc. Usenix*

Large Installation System Administration Conf. (LISA) Usenix Assoc., 2003, pp. 173–180.
10. V. Talwar et al., "Comparison of Approaches to Service Deployment," to be published in *Proc. 25th Int'l Conf. Distributed Computing Systems*, 2005.
11. W.D. Itzfeldt, "Quality Metrics for Software Management and Engineering," *Managing Complexity in Software Engineering*, IEE Computing Series 17, R.J. Mitchell, ed., Short Run Press, 1990, pp. 127–151.

Vanish Talwar is a member of the Internet Systems and Storage Lab at Hewlett-Packard Laboratories. His technical interests include distributed systems, operating systems, and computer networks, with a current focus on distributed systems management technologies. Talwar has an MS in computer science from the University of Illinois at Urbana Champaign, where he is currently pursuing a PhD in the same subject. Contact him at vata@hpl.hp.com

Qinyi Wu is a PhD student at Georgia Tech, where she is also a member of the Distributed Data Intensive Systems Lab in the university's College of Computing. Her technical interests include workflows, transaction processing, and Web services. Wu has an MS in computer science from the Beijing Institute of Technology, Bejing, China. Contact her at qxw@cc.gatech.edu.

Calton Pu is the John P. Imlay Jr. Chair in Software at Georgia Tech. His research interests include operating systems, transaction processing, and Internet data management. Pu received a PhD in computer science from the University of Washington. He is a member of the ACM, a senior member of the IEEE, and a fellow of the AAAS.

Wenchang Yan is a research scientist in the College of Computing at Georgia Tech. His research interests include distributed systems, quality of service, and adaptive middleware. Yan has an MS in computer science from Florida State University. Contact him at wyan@cc.gatech.edu.

Gueyoung Jung is a PhD student in the computer science department at Georgia Tech. His technical interests include distributed and data-intensive systems, autonomic computing, and general distributed systems, in particular peer-to-peer computing, distributed information monitoring, and data stream processing. Jung has an MS in computer science from Georgia Tech. Contact him at helcyon1@cc.gatech.edu.

Dejan Milojicic is a project manager and a senior researcher at Hewlett-Packard Laboratories. His technical interests include distributed systems, operating systems, systems management, and mobile systems. Milojicic has a PhD in computer science from the University of Kaiserslautern, Germany. He is a member of the ACM, the IEEE, and Usenix. Contact him at dejan@hpl.hp.com.

Visit the IEEE Computer Society's all-new Software Engineering online resource

unbiased and trusted
peer-reviewed
in-depth and topical
practical and timely
free technical content

Go online today

www.computer.org/seonline