VehicleFORGE: A Cloud-Based Infrastructure for Collaborative Model-Based Design

Laszlo Juracz, Zsolt Lattmann, Tihamer Levendovszky, Graham Hemingway, Will Gaggioli, Tanner Netterville, Gabor Pap, Kevin Smyth, Larry Howard

Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN

Abstract. Recent increases in industrial adoption of Model-Based Engineering has created demand for more advanced tools, environments, and infrastructures. As a response to the Defense Advanced Research Project Agency's (DARPA) initiative in the Adaptive Vehicle Make (AVM) program, we have designed and built VechicleFORGE, a collaborative environment tightly integrated with the AVM design tools. This paper describes VehicleFORGE concepts and services facilitated by the cloud computing foundation of the infrastructure.

1 Introduction

The VehicleFORGE [5] platform is designed and maintained to host the DARPA Fast, Adaptable, Next-Generation Ground Vehicle (FANG) series of prize-based design competitions as part of the AVM [1] portfolio. In the first FANG challenge, the AVM program set out to apply crowdsourcing practices to involve a larger group of individuals in the design work. After registration, competitors can form teams and gain access to the modeling and analysis tools and modeling components being developed by the program. Although subsequent competitions may be less open in terms of public participation, operating the competition requires a centralized platform where participants can collaborate and which serves as a location for distributing tools, design requirements and components, documentation, sharing results and accessing compute resources.

VehicleFORGE provides the virtual environment and cloud infrastructure which enables the management of the competitions, competitors, and the collaboration of geographically distributed design teams, as well as various cloud-based analysis services and tools for the design work.

We develop and operate VehicleFORGE and the underlying cloud infrastructure using open source technologies. Both the web application and the monitoring tools are designed to enable streamlined deployability and scalability to meet changing utilization profiles and to satisfy security requirements set by DARPA.

2 The Forge Framework

In the past decade, the success of crowdsourcing and the model of distributed problem-solving and software production has been enabled by widely popular open source software forges such as SourceForge.net [4]. After investigating the available technologies, we decided to use the Allura application developed by SourceForge.net as a basis for the development of VehicleFORGE.

2.1 Concepts of the Platform

Although the architecture of the application has greatly evolved, the organization of the fundamental forge concepts in VehicleFORGE is derived from the core Allura application.

Projects embody the collaborative spaces where members of a team of users manage design work, create and share design artifacts and analysis results. Registered users can form new Projects or acquire membership in an existing one. Projects are created based on pre-configured templates but in general, each team controls how it utilizes the Project for its work.

Neighborhoods are collections of projects, usually representing a higher-level real-word organizational concept (eg. competition) with which the teams of the member projects are affiliated. Neighborhoods also offer similar collaboration functionalities to the project spaces: they can have members, customized roles and selected tools installed for neighborhood-level collaborative work.

Tools are provisioned in the project space and house the data and interfaces for engaging in collaborative design work. Privileged project administrators can freely install and administer new tools. Objects created during the collaborative and design work in a tool are referred to as *artifacts*.

Among the various out-of-the-box tools, VehicleFORGE offers *Subversion* [15] (SVN) and *Git* [25] repositories for sharing files created in desktop-based design tools. Through a set of *post-commit hooks*, the forge processes newly added content to update its understanding of a project's designs. Project members can access web-based previews of each other's work, and files and design artifacts recognized this way can be cross-referenced with artifacts created in other VehicleFORGE tools. Thus, repositories work as the bridge between design work done on the desktop and the web-based collaboration environment.

A major extension that VehicleFORGE offers to the basic forge concepts found in software forges is the *Component Exchange*. It offers a globally readable shared space in which users can publish, categorize, share and discover formally characterized reusable design components outside of the scope of an individual project.

VehicleFORGE implements customizable *role-based access control*: each project can create *permissions* and *user groups* to match its requirements. The combination of groups and permissions are used to determine the access to artifacts contained in a tool.

Every project (and neighborhood) space has an *Admin Tool* where team-leaders can provision new project tools and configure the permissions. Each

VehicleFORGE deployment has a dedicated ForgeAdmin project which contains services related to the entire forge. There are various tool-specific aggregation and statistical interfaces available to monitor collaboration and design activities in the team and neighborhood scope.

2.2 Customizability

The basic forge concept developed by software forges was designed primarily for supporting work with source-code, however, VehicleFORGE is easily customizable for collaboration in arbitrary domains, from cyber-physical system design [3] to policy authoring [29]. Beyond the flexible project configuration and access-control administered through web interfaces, VehicleFORGE supports multiple levels of extensibility.

Visualizers provide a means for third party developers to implement new visualization of domain-specific repository content. Visualizers are executed in the user's browser and they are not part of the main application code base, however, they can be deployed along with custom post-commit hooks to do server-side preprocessing of the files containing the information to be displayed.

The forge application and the project tools are written in the Python-based TurboGears framework. Experienced developers can make significant capability extensions by developing new forge tools or modifying parts of the open source forge framework.

3 Services in the cloud

The VehicleFORGE cloud infrastructure facilitates the creation and operation of multiple forge deployments and provides the flexibility to scale deployments to changing loads. Its extensible pool of resources is available for virtualizing various operating environments to extend VehicleFORGE platform capabilities and to offer compute and analysis as a service through the forge to the AVM community.

3.1 Scalability of Forge Deployments

The deployment architecture is designed so that every significant, load-bearing service on the Forge is horizontally scalable. Various strategies are employed on a service-by-service basis to enable this. The web server is run on multiple processes across multiples instances. All requests are initially routed to a server running a fast, lightweight load balancer service that distributes the requests intelligently to the available web servers. A similar strategy is used to scale the compute and repository services. To scale the database, index, and cache services we use replication support built in to the specific software implementations.

The service infrastructure is designed to minimize response time and optimize cloud resource utilization. Figure 1 depicts the service architecture for a VehicleFORGE deployment. Most requests begin and end through a web server

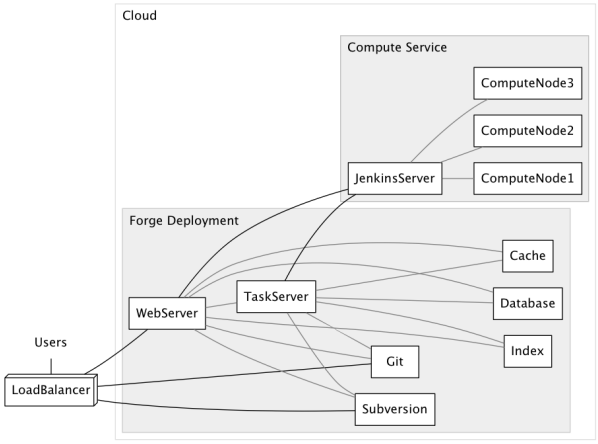

Fig. 1. VF Service Artchitecture.

gated by the load balancer. To minimize latency, the web servers delegate longer-running, more resource-intensive jobs to *Task Servers* that execute the jobs asynchronously. In a similar vein, web servers communicate with the META *Compute Service* to conduct the Testbench analyses. The Compute Service nodes are separated from the main VehicleFORGE deployment to ensure that they have access to the necessary resources for their intensive analysis tasks.

3.2 Test Bench Execution

Test benches, design spaces, components Within the AVM program, a design tool chain (META tools) [23,38] is being developed for exploring design alternatives and analyzing designs under certain conditions. The META tools provide the integration framework for components, designs, design spaces, requirements and test benches. *Components* are atomic building blocks that describe the dynamics behavior and the structural aspect of physical objects. *Designs* and *design spaces* are built up from components, where a design has a fixed architecture (composition of components) and a design space can encode different component alternatives as well as different design architectures.

After a design or design space is created, test cases are defined against the given requirement set. The test cases, which we term *test benches*, are executable versions of the system requirements. From the test bench models, the META tools can compose analysis packages over a design space for different domains such as simulation of DAEs (differential algebraic equations), formal verification, static analysis, and structural analysis. Examples include vehicle model simulation using different drive cycles such as highway speed profile or urban speed profile, cost of the design by aggregating the cost of the components, and structural stress analysis on the composed CAD (3D) model of the design.

Resource considerations Executing the test benches may require different platforms; the execution time varies based on model size and analysis type. Furthermore, the number of test bench executions depends on the number of test benches and the design space size (i.e. number of designs). During this project we used approximately 30 test benches and 400 design variations, which evaluates to about 12k test bench executions. Additionally, we had to take into account the size of the generated analysis results and provide for their storage.

Implementation and cloud usage Initially, all test benches were executed sequentially within the same process, while we had only just 1-2 test benches and 1-5 designs. As we increased the number of test benches and the complexity of the designs, we switched to another solution. We implemented a client side job manager (called META Job Manager) for running the test benches on multiple CPUs using a single machine and limited the number of maximum parallel jobs to the number of CPUs. As the execution time for a single test bench started increasing because the complexity of the design increased, the 1-3 hour simulation times were an unacceptable encumbrance on the user's machine. For this reason, we extended the META Job Manager with a remote execution service that authenticated with VehicleFORGE.

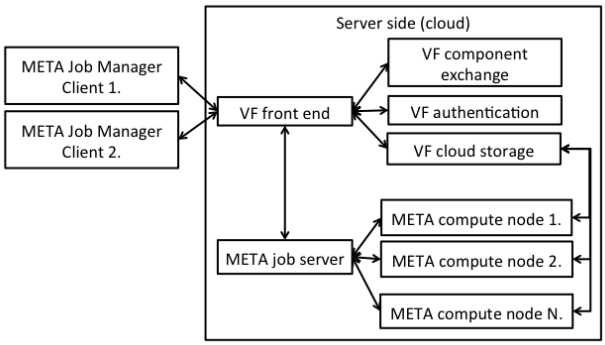

Fig. 2. META VF infrastucture.

Figure 2 depicts the communication path between the client side job manager and the server side services. The META Job Manager can be configured to execute the test benches locally (in parallel) or post them to VF to be executed on compute nodes. If the job manager is configured for remote execution, the user provides the login information and logs in through the VF authentication service. Then, the job manager is ready to accept jobs and post them to VF. When a job is posted to the job manager, it calls a VF service to register the job, uploads the analysis package to a cloud storage server, then indicates the job is

ready to execute. The job gets posted to the META server side job dispatcher and is added to the queue. If there is a free executor on a compute node that has all the tools required to execute the job, the job gets shipped to the compute node and the execution begins. A compute node for a job is chosen based on the version of the desktop tools that the user employed to create his/her design. For example, we had different version of the META tools deployed on the client machines and some of our compute nodes supported one version of the tool chain, while others supported another version of the tools. The compute node downloads the analysis package from the cloud storage, executes the analysis, uploads the results to the cloud storage, and indicates that the job is complete (successfully or with failures). The META Job Manager monitors the pending jobs regularly. When the analysis is done, it downloads the results to the client's machine.

Optimization and cloud benefits Since the component models and their artifacts are stored on VF, which are accessible through the VF component exchange, the data transfer between the server and clients was reduced by sending only links to components rather than the actual content. This significantly improved the turnaround time for the CAD jobs.

Users do not need to install the analysis tools, if they only use remote execution. The remote compute nodes have all the tools set up and configured correctly to perform the test bench executions. As a result of having compute nodes, the load on the users machine was decreased and it requires only communication with the server to send and receive test bench bundles.

Within 3 months the remote compute nodes served 1000 users (200 teams) and ran 50000+ analysis jobs with 92 percent success[1].

3.3 Monitoring the infrastructure

Operating a complex distributed application demands constant monitoring and maintenance. The dynamic nature and virtualized deployment of the VehicleFORGE application further complicate the problem of providing a cohesive understanding of its status. We have deployed a number of monitoring and management tools into both the VehicleFORGE application and its underlying cloud infrastructure in order to automate and simplify the tasks of monitoring and managing operations.

The foremost requirement for application monitoring is simply to understand the current state of the application. This requirement spans from low-level checks, such as disk and memory usage on the virtual machines, to higher-level needs, such as checking database flush times. The VehicleFORGE team selected Nagios3 [33] for status monitoring, though excellent alternatives, such as Munin [32] and Ganglia [28], exist. We chose Nagios because it provides a very large library of built-in checks, is easily extensible for custom checks, and is resource

[1] The job execution largely did *not* fail on server side, but the analysis results may or may not have passed the requirements.

efficient enough to execute all of our checks in very short intervals (less than a minute in our case). A key aspect of status monitoring is the ability to reflect the process topology of the deployment, i.e. which process is running on which machine. As our entire deployment process is based on modeling and automation, it was a natural extension for us to develop a mapping from the deployment blueprint to the Nagios monitoring configuration. As alterations are made to a given deployment a new monitoring configuration can be easily synthesized. In the production deployment of VehicleFORGE, Nagios performs 374 independent status checks every minute. If any of these checks fail, an administrator is notified immediately and can begin to remediate the issue.

While status monitoring helps to ensure the uptime of the application, very frequently administrators and support personnel need to understand the historical context of a specific application operation or event. For example, why was a particular user's registration request rejected, or how many users logged in on a particular day. For these types of operational questions it is best-practice to instrument the application so that it logs all relevant events with some pertinent information. Typically these logs reside on the local filesystem of each machine. Standard Linux processes make extensive use of logging, too. It can be a non-trivial problem in distributed applications to collect all of the desired logging information, centralize it, process and analyze it, and archive it. Similar to status monitoring, several open source alternatives exist for log handling, notably LogStash [37] and Scribe [19]. After evaluating the alternatives, we chose to use LogStash in conjunction with ElasticSearch [22] for log indexing, and Kibana [20] for analysis and visualization. This combination is very easy to deploy and configure and require minimal resources during operation. Every event occurring in both the VehicleFORGE application and the underlying virtual machine it logged, collected, indexed and archived. This provides our operations team with tremendously powerful tools to understand both what is happening at any given moment, and past historical trends. The volume of data generated by our logging approach is non-trivial though. In an average hour of operation, the production VehicleFORGE deployment generates over 4.3 million log records which consume almost 5MB of disk space. In one year of operation, that equates to nearly 38 billion records and 40GB of data. A dedicated cluster of virtual machines is needed to index the data and execute searches across these records.

Finally, both developers and operators need to understand what is happening "inside" of the application. This understanding is at a deeper level than is typically provided by tools such as Nagios, Munin or Ganglia. The need is also more "real-time" than is provided by analysis of historical log information. A typical use of such real-time statistics is a operations dashboard. On this dashboard may be a number of statistics that allow an operator to assess the internal state of the deployment in a glance, for example, a real-time plot of the number of active users. An event such as a DNS failure that sends a large share of traffic away from the site would not be detected by either the monitoring system or log analysis. Another example would be a software update roll-out that causes users to start receiving errors. In both of these cases it necessary to have a deeper and

more immediate view into the state of the application. StatsD [27,26] is just such a tool. The code of an application must be modified in order to support StatsD collection, but once done, as specific events occur within the application a UDP packet is sent to a daemon that receives and aggregates it over time. Within 10 seconds of an event occurring, operators see it appear on a plot that provides significant insight into the internal state of the application. The VehicleFORGE infrastructure makes use of StatsD and in the near future its support will be built into the VehicleFORGE application too.

3.4 Operator tools

The maintenance of a distributed application can be greatly simplified through the development of autonomic tools for configuring and managing a cloud deployment. The complex inter-dependencies between subcomponents of the VehicleFORGE application and the services required to provision the application, as well as the complexities inherent to managing a distributed application of this scale, guided our development of *Da Capo*, a cloud provisioning and orchestration framework that we develop and maintain internally. Da Capo provides an extensible framework into which we have injected a VehicleFORGE-specific component that delineates the particulars of orchestrating our application. We use Da Capo extensively in the development of VehicleFORGE to create and manage both publicly accessible production deployments and short-lived, private sandbox deployments for development and testing.

Da Capo greatly facilitates the creation and configuration of distributed applications. Using its API through a web interface contained in the lightweight VehicleFORGE tool *ForgeCloud*, we are able to configure new VehicleFORGE deployments. Configuration is performed through the definition of app-specific parameters and *services*, persistent/long-running background processes (e.g. the database) upon which the application depends. Through the ForgeCloud configuration interface, we can specify the number of instances in the deployment, the size of those instances, the service(s) contained on each instance, and various configuration parameters that define how the application will function. Da Capo is aware of which services are required, limits to the number of a particular service type that can exist in a functional deployment, and any inter-dependencies between services. In short, it will ensure that a deployment specification will result in a valid deployment before it is initialized.

Once the deployment specification is submitted, Da Capo provisions the necessary resources by communicating with the OpenStack API [31]. When the instances are ready, it installs and configures the application, services, and their dependencies on the designated instances. It handles any requisite deployment-specific and user-specified configuration. Finally, Da Capo initializes the services and the application. Any errors that occur during this process are logged and reported.

Da Capo additionally offers tools for monitoring and managing a running deployment. It can create and restore backups for a deployment by running the appropriate commands on the appropriate services (in our case Solr [35],

MongoDB [13], Swift [6], Git, SVN and Redis [21]). It can execute an operator-inputted command on all instances of a deployment, or only instances running specific services. It can stop, start, and restart specified services, display logs, check the status of all services, detect and run necessary application migration scripts, and perform an operating system upgrade. Further, Da Capo's command infrastructure is easily extensible, ensuring that any future automation needs will be met.

4 Related Work

VehicleFORGE adopts cutting edge FORGE components that have proven themselves in textual language-based projects. Wiki pages [24] offer a convenient way of sharing hypertext-based information, which are aided by other established tools, many based on those of the Allura Project [2], to provide a collaboration hub for distributed design teams.

Although there are several model-based collaboration environments in the field of CAD and general domain-specific languages,[8,34,17,16,18,7]– literature overview with analysis can be found in [11] and [9]–, VehicleFORGE offers a number of novel concepts, including the Component Exchange, among others. The tight integration with the META toolchain and the efficient use of state of the art cloud computing and collaboration technologies also make it a unique infrastructure. The vision of combining model-driven engineering and cloud computing has been proposed in existing publications [10,12,14], but as of yet no publication details the creation of a functioning deployment.

A distributed collaborative design evaluation platform called DiCoDEv [30] uses virtual reality techniques to implement real-time manufacturing design. The focus of VehicleFORGE is broader than manufacturing and it provides several offline services. In [36], the authors describe a collaboration environment with source control management, forum, mailing list web sites, news, and project categorization. As opposed to VehicleFORGE, it is restricted to textual content written in the language R.

5 Conclusions and plans

In this paper, we have introduced VehicleFORGE, a cloud-based application for collaborative model-based development. VehicleFORGE utilizes cutting edge cloud computing technologies in order to maintain scalability for resource-intensive design work. Various domain-specific tools that benefit from high computational power and centralized resources, such as design space explorers, can use the VehicleFORGE cloud to great advantage. VehicleFORGE has been used in United States-wide competitions, which was made possible by the monitoring and operator tools. Development and maintenance on VehicleFORGE deployments is enabled by custom platform as a service software developed in house.

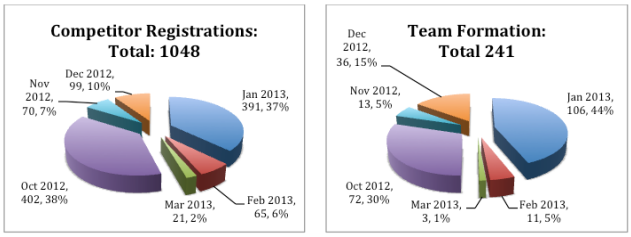

Fig. 3. User and Team registrations throughout FANG-I

There were 1048 competitors and 241 design teams registered on the main VehicleFORGE website which served as the home for the FANG-I Design Challenge (see Figure 3). Besides this deployment, the *VehicleFORGE Production Cloud* hosted the testbench analysis services, a Beta website for staging and testing FANG Challenge resources and an internally used platform for managing the development of the system itself. There will be further forge instances deployed for the AVM program in the upcoming period to the FANG-II Design Challenge.

The *Development Cloud* hosts the Alpha VehicleFORGE website which is maintained for educational purposes and several, on-demand sandbox-deployments created for development and testing purposes.

In the meantime, we are working on the first release of the refactored, new forge framework, which is designed for utilization by a greater open source community–outside of the immediate scope of the AVM program.

6 Acknowledgments

This work was sponsored by DARPA, under its Adaptive Vehicle Make Program. The views and conclusions presented are those of the authors and should not be interpreted as representing official policies or endorsements of DARPA or the US government.

References

1. *Adaptive Vehicle Make*. `http://www.darpa.mil/Our_Work/TTO/Programs/Adaptive_Vehicle_Make__(AVM).aspx`.
2. *Allura*. `http://sourceforge.net/projects/allura/`.
3. *Cyber-physical system*. `http://en.wikipedia.org/wiki/Cyber-physical_system`.
4. *SourceForge*. `http://sourceforge.net`.
5. *VehicleFORGE*. `http://www.vehicleforge.org`.
6. J. Arnold. *Software Defined Storage with OpenStack Swift*. SwiftStack, Inc., April 2013.

7. O. Berger, C. Bac, and B. Hame. Integration of libre software applications to create a collaborative work platform for researchers at get. *International Journal of Information Technology and Web Engineering (IJITWE)*, 1(3):1–16, 2006.
8. R. Bidarra, E. Van Den Berg, and W. F. Bronsvoort. Collaborative modeling with features. In *Proceedings of DET*, volume 1, page 2001, 2001.
9. G. Booch and A. W. Brown. Collaborative development environments. volume 59 of *Advances in Computers*, pages 1 – 27. Elsevier, 2003.
10. H. Bruneliere, J. Cabot, F. Jouault, et al. Combining model-driven engineering and cloud computing. In *Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud'10: Workshop's 4th edition (co-located with the 6th European Conference on Modelling Foundations and Applications-ECMFA 2010)*, 2010.
11. J. Cabot, G. Wilson, et al. Tools for teams: A survey of web-based software project portals. *Dr. Dobbs*, pages 1–14, 2009.
12. J. Castrejón, G. Vargas-Solar, C. Collet, and R. Lozano. Model-driven cloud data storage. *Proceedings of CloudMe*, 2012.
13. K. Chodorow. *MongoDB: the definitive guide*. O'Reilly, 2013.
14. C. Clasen, M. D. Del Fabro, M. Tisi, et al. Transforming very large models in the cloud: a research roadmap. In *First International Workshop on Model-Driven Engineering on and for the Cloud*, 2012.
15. B. Collins-Sussman, B. Fitzpatrick, and M. Pilato. *Version control with subversion*. O'Reilly, 2004.
16. R. Frost. Jazz and the eclipse way of collaboration. *Software, IEEE*, 24(6):114–117, 2007.
17. J. Gallardo, C. Bravo, and M. A. Redondo. A model-driven development method for collaborative modeling tools. *Journal of Network and Computer Applications*, 35(3):1086–1105, 2012.
18. C. Herrmann, T. Kurpick, and B. Rumpe. Sselab: A plug-in-based framework for web-based project portals. In *Developing Tools as Plug-ins (TOPI), 2012 2nd Workshop on*, pages 61–66, 2012.
19. R. Johnson. Facebook's scribe technology. October 2008.
20. R. Khan. *Kibana*. `http://kibana.org/`.
21. J. A. Kreibich. *Redis: The Definitive Guide: Data Modeling, caching, and messaging*. O'Reilly, 2013.
22. R. Kuc and M. Rogozinski. *Elasticsearch Server*. Packt Publishing, 2013.
23. Z. Lattmann, A. Nagel, J. Scott, K. Smyth, J. Ceisel, C. vanBuskirk, J. Porter, T. Bapty, S. Neema, D. Mavris, and J. Sztipanovits. Towards automated evaluation of vehicle dynamics in System-Level designs. In *CIE*, 2012.
24. B. Leuf and W. Cunningham. The wiki way: quick collaboration on the web. 2001.
25. J. Loeliger and M. McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc., 2012.
26. I. Malpass. Measure anything, measure everything. http://codeascraft.com/2011/02/15/measure-anything-measure-everything/, February 2011.
27. I. Malpass. Statsd. StatsD Repository, July 2013.
28. M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.
29. A. Nadas, L. Juracz, J. Sztipanovits, M. E. Frisse, and A. J. Olsen. Policyforge: A collaborative environment for formalizing privacy policies in health care.

30. M. Pappas, V. Karabatsou, D. Mavrikios, and G. Chryssolouris. Development of a web-based collaboration platform for manufacturing product and process design evaluation using virtual reality techniques. *International Journal of Computer Integrated Manufacturing*, 19(8):805–814, 2006.
31. K. Pepple. *Deploying OpenStack*. O'Reilly, 2011.
32. G. Pohl and M. Renner. *Munin - Graphisches Netzwerk- und System-Monitoring*. Open Source Press, April 2008.
33. M. Schubert, D. Bennett, J. Gines, A. Hay, and J. Strand. *Nagios 3 enterprise network monitoring: including plug-ins and hardware devices*. Syngress, 2008.
34. N. Shyamsundar and R. Gadh. Internet-based collaborative product design with assembly features and virtual design spaces. *Computer-aided design*, 33(9):637–651, 2001.
35. D. Smiley. *Solr 1.4 Enterprise Search Server*. Packt Publishing, 2009.
36. S. Theußl and A. Zeileis. Collaborative software development using r-forge. 2008.
37. J. Turnbull. *The LogStash Book*. Amazon, 2013.
38. R. Wrenn, A. Nagel, R. Owens, H. Neema, F. Shi, K. Smyth, D. Yao, J. Ceisel, J. Porter, C. vanBuskirk, S. Neema, T. Bapty, D. Mavris, and J. Sztipanovits. Towards automated exploration and assembly of vehicle design models. In *CIE*, 2012.