

SYLAGEN – An Extendable Tool Environment for Generating Load

Michael Striewe, Moritz Balz, Michael Goedicke

Specification of Software Systems
Institute of Computer Science and Business Information Systems
University of Duisburg-Essen, Campus Essen, Germany
{michael.striewe,moritz.balz,michael.goedicke}@s3.uni-due.de

1 Introduction

Measuring run time behaviour of systems under load can cause the need for complex workload definitions, measurement strategies and integration of load generation techniques. In this contribution we present SYLAGEN (“Synthetic Load Generator”), a load generation environment that focuses on extendability with respect to four different aspects: First, a system under test may offer different interfaces for handling external requests, thus a load generator must be able to handle different protocols randomly and in parallel. Second, load generation for a client-server system may require complex client behaviour that cannot be formulated in a simple descriptive way, but instead with non-trivial algorithms that have to be implemented programmatically. Third, more than simple atomic measurements may be required in complex environments, so that strategies applying sequences of measurements to a system should be configured. Finally, comprehensive requirements engineering may result in complex use cases that cannot be modelled as linear scripts.

The extendability of SYLAGEN considering these four aspects is realized by an architecture providing a platform with load generation functionality upon which modules can be created descriptively or programmatically. They are integrated in the system by a simple provision of libraries. Thus, SYLAGEN can be easily supplemented with project-specific complex functionality. SYLAGEN has been in productive use with this concept since 2002.

To generate load on different independent computers, SYLAGEN has a distributed architecture. The *Master* component controls the measurement process and directs *Clients* that run on different hardware nodes in a network. On each client several *Worker* threads are running that generate the actual load. Master and clients communicate by means of a socket-based protocol that can be used in a variety of programming languages. This architecture is not unusual and can be found in other load generator tools, too. However, we instrument the distribution for our purpose in special ways, as we will now describe.

2 Load Models

Load being generated is often modeled after the behaviour of human users. In many cases no linear or otherwise exactly predictable user behaviour can be assumed. In addition, a realistic scenario may require generation of moderate load during the whole measurement as well as load peaks in some rare moments. Thus the synthetic load generation faces the challenge to represent probabilistic paths being taken between singular actions. SYLAGEN employs probabilistic load models which also exist as additions to other tools like JMeter [1]. Based upon this, actual workloads can be defined for different use cases measuring a system under test.

In SYLAGEN workload is described in terms of *flows* which are mainly a transition system $F = \langle S, T, W \rangle$. $S = \{S_1 \dots S_n\}$ is a set of *states* that contain load generating operations, i.e. requests to the system under test. $T = S \times S$ is a set of transitions connecting states. $W = \{W_{S_1} \dots W_{S_n}\}$ is a set of integer weights assigned to each transition. From all weights the relative probability for each path in the transition system can be calculated. A workload may consist of several flows, again each with a weight, so different flows can be performed with different probability. Flows can also be invoked in states of other flows, providing even more flexibility. While the transition system defines possible sequences of load generating actions and their probability, expectations about the system performance are formulated in terms of *turnaround times*: SYLAGEN considers the whole time needed by the system under test to respond to a request, including the complete stack of underlying platforms, e.g. the network. For each flow, two times can be defined: The *mean turnaround time* is mandatory and defines which turnaround time for the related flow is expected in the average case. The *maximum turnaround time* is optional and may define an upper bound for turnaround times that are acceptable. These time requirements are interpreted by load generation strategies (see section 4) for different purposes.

3 Adapters

Modularity with respect to different load generation technologies is realized with an *adapter* concept: SYLAGEN is not limited to access a fixed set of protocols or platforms, but allows to plug in individual libraries. These libraries can be specific for different protocols, platforms, systems under test or even single use cases. Adapters for protocols exist e.g. for web services (SOAP), Java RMI, file servers (SMB), and simple web applications (HTTP). Rich Internet Applications with AJAX [2] are an example for non-trivial platforms that are supported by existing adapters. Among others, a use-case-specific adapter exists that controls a graphical user interface using macros accessing its visual components.

Although adapters are specialized, they are encapsulated by a common abstraction level allowing to address them in a consistent way in load models. The abstraction is described by *adapter methods*: Each adapter publishes the names of provided load generating operations and optionally a list of typed parameters

as well as a typed result value. A global storage for return values allows to use them as parameters in other invocations, thus enabling to realize complex load scenarios and the related data flow.

During measurement, adapters are distributed in a centralized way by the Master that transfers the libraries to all participating clients. In the clients, each worker creates a dedicated instance of the adapter. Master and clients are decoupled with a simple protocol, so that libraries containing adapters can be created using different languages and platforms. So far, clients and adapters for Java and C# exist. In the case of Java, the libraries are Java Archives (JAR) including class files, other libraries and arbitrary binary resources. The Java client is run using the Java runtime environment. The same client can also be run with the IKVM virtual machine [3] allowing to integrate libraries for adapters written in C#.

4 Load Generation Strategies

Another point of extendability in SYLAGEN is the support of different load generation strategies by providing an abstraction layer: On the one hand, the platform performs the measurement with given parameters like number of workers and computes the results afterwards. On the other hand, modules for load generation can be created on top of this that decide about the measurements to perform and the parameters to use. Currently, the following strategies are used:

The *single* measurement strategy performs exactly one measurement with the given workload and number of workers. The mean turnaround time defined in the workload is interpreted as user behaviour: If the system under test responds faster, SYLAGEN takes random pauses before triggering the next request so that the request frequency meets the expected mean turnaround time. In contrast, the *stress* strategy does not follow any restrictions, but makes as many requests as possible to generate the maximum possible load. In both cases the results of interest are the turnaround times for the requests sent during measurement.

In the *exploration* strategy the number of workers is increased or decreased stepwise depending on the desired turnaround times. A system is considered overloaded if the mean turnaround time of all workers exceeds the desired mean turnaround time *or* the maximum turnaround time is violated by at least one adapter call. The result of this strategy is the number of workers that the system under test can serve with the given limitations regarding turnaround times.

The abstraction layer implies that load generation strategies work with a limited number of variables and describe different well-defined states in the load generation process to fulfill their purpose. They are thus candidate for modeling with state machines, but are also integrated in arbitrary program code and contain complex business logic like calculations based on measurement results. Thus their code cannot be generated from abstract model specifications. Instead, the decision was made to employ *embedded models* [4] that represent model specifications in a well-defined program code pattern. Since the approach bridges the gap between different abstraction levels, it is appropriate for modeling the load

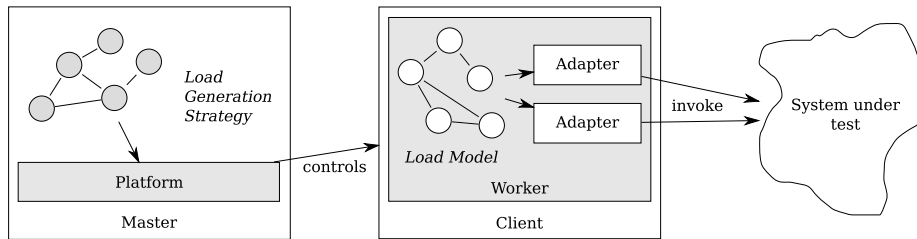


Fig. 1. Schematic view on the SyLaGen components.

generation strategies. However, this is not mandatory since any Java component using the abstraction layer can serve as a load generation strategy.

5 Conclusion

In this contribution we presented SYLAGEN, a load generation environment that focuses on extendability to realize complex scenarios. This concerns different interfaces of systems under test, representation of complex client activities, individual load generation strategies and non-linear client behaviour. The resulting architecture is illustrated in figure 1: The Master controlling the measurement delegates control to load generation modules. The underlying platform transfers measurement data to clients where worker threads execute probabilistic load scripts. The nodes in the scripts access adapters that provide a unified interface, but can invoke a system under test with arbitrary technologies and protocols.

Since 2002, this architecture has proven to be a dependable solution for load generation with complex use cases, unusual protocols, and problem-specific implementations of requirements. However, since SYLAGEN does mainly provide the environment and not ready solutions, simple use cases require the same effort when adapters or strategies must be created. Future work will thus focus on instrumenting the adapter concept to provide out-of-the-box solutions for different technologies and protocols.

References

1. van Hoorn, A., Rohr, M., Hasselbring, W.: Generating Probabilistic and Intensity-Varying Workload for Web-Based Software Systems. In Kounev, S., Gorton, I., Sachs, K., eds.: Performance Evaluation – Metrics, Models and Benchmarks: Proceedings of the SPEC International Performance Evaluation Workshop 2008 (SIPEW '08). Volume 5119 of LNCS., Heidelberg, Springer (2008) 124–143
2. Paulson, L.D.: Building Rich Web Applications with Ajax. *Computer* **38**(10) (2005) 14–17
3. IKVM: IKVM Website <http://www.ikvm.net/>.
4. Balz, M., Striwe, M., Goedicke, M.: Embedding State Machine Models in Object-Oriented Source Code. In: Proceedings of the 3rd Workshop on Models@run.time at MODELS 2008. (2008) 6–15