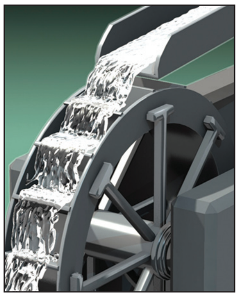

Design by Units

Abstractions for Human and Compute Resources for Elastic Systems

Stefan Tai • *Karlsruhe Institute of Technology*

Philipp Leitner and Schahram Dustdar • *Technical University of Vienna*

Units make the usage and properties of diverse resources, including infrastructure and human resources, explicit early in system design, and allow for reasoning about complex system qualities, such as elasticity. They advance the measurability and management of systems whose quality depends largely on the resources that the system uses.

A service system's quality is multifaceted, including functional and nonfunctional technical aspects, as well as nontechnical aspects ranging from business to social characteristics. Elasticity[1] is a good example of this multifaceted quality: it describes a system's capacity to add or remove various resources as needed to efficiently operate in a fast and convenient manner and without severe impact on other qualities, such as system availability or performance. Resources can include infrastructure such as compute power, storage space, and bandwidth, but also nontechnical resources such as the financial budget available or the human (expert) manpower needed to skillfully operate the system, make decisions, or perform human-based computing tasks. The elasticity of a system through virtualized resources[2] is thus a fundamental requirement of Web-scale systems; in system design, those resources must receive careful consideration.

Design by contract[3] is a well-known software design approach that extends conventional software component definitions with pre- and post-conditions and invariants; these specifications are referred to as *contracts*. In analogy to design by contract, we introduce *design by units*, which extends software service definitions with a resource model to better address human and compute resource requirements in system design. *Units* are abstractions to model diverse resources in the cloud that are required to operate a system and to guarantee nontrivial system requirements, such as elasticity.

The Emergency Hub

Consider an emergency hub system as an example (see Figure 1). The system provides a Web service interface that lets various clients, including smartphones, signal an emergency situation. Under normal circumstances, the system's request load might be low. In an emergency situation that involves large numbers of people, however — as might occur during a sporting event, demonstration, or other large gathering in a metropolitan region — the system will need to scale elastically on demand. Additional infrastructure resources as well as social resources (such as emergency personnel) will be needed. Once the emergency situation is resolved, the diverse resources will no longer be required, and the system can scale down.

In recent years, systems similar to the emergency hub have arisen in a variety of application domains. Designing, deploying, and operating such systems introduces several challenges. As we might assume from this example, various infrastructure resources are required, including storage systems, servers, and virtual machines. However, the example also illustrates the need for other types of resources and their complex configurations. In the case of a large-scale emergency, for instance, not only one (physical) hospital will be populated but most, if not all, hospitals in the city might need to serve as resources. Thus, an emergency hub system must manage and coordinate all available respective resources in a "virtual hospital." The specific requirements and management needs associated with such situations and peak loads are known only at system runtime and must be provisioned on the fly.

A critical resource commonly neglected in conventional software system design is humans. Required human resources in our emergency hub example include police, firemen, ambulance personnel, medical doctors, nurses, conflict resolution specialists, SWAT teams, and so on. All these people must be "composed" on-demand based on their skills, availabilities, and costs.

Both infrastructure resources and human resources have various relevant properties depending on the case at hand. These measureable properties constitute the building blocks for determining the potential degree of elasticity. We argue for dedicated modeling abstractions that system designers can use to express these properties during design and to deploy and operate the system accordingly. In other words, the overall system consists of both infrastructure resources, and people and social teams. Both resource types constitute building blocks that are critical for the system to fulfill its purpose. We argue that it's critical to define early on in software system design the measureable resources needed and how these relate to different system parts.

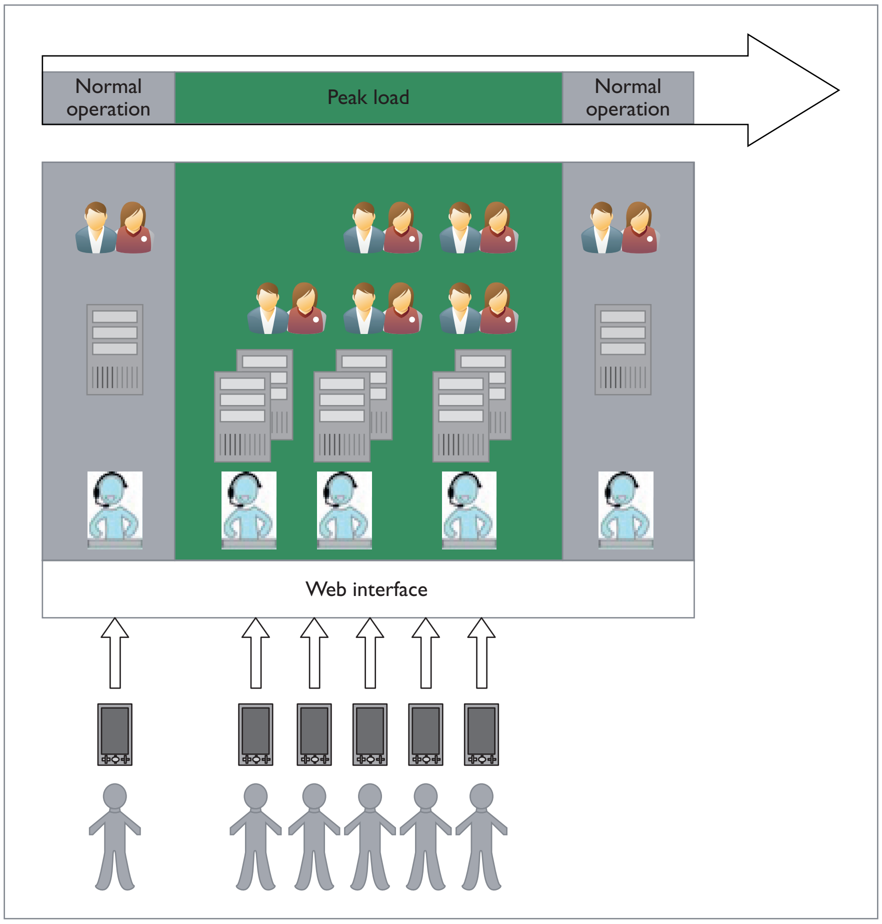

Figure 1. Resource elasticity. The human and Web resources needed to respond to, for example, an emergency situation, must scale elastically based on demand.

Units as Abstractions

The emergency hub scenario illustrates that conventional service-centric application design and development models at best address only half the picture. Service programming models and architectural styles such as SOAP and REST focus on representing application logic. Multifaceted system qualities such as elasticity, however, critically depend on resources in the application environment, such as the infrastructure to which the application is deployed, and the people behind the system. Figure 2 illustrates this concept.

Appropriate first-class abstractions are needed in system design to model such resources. Unlike the design and programming abstractions used to model application logic, we propose units as a complementary abstraction to manifest resources. We can model a variety of resources using typed units. The unit type defines properties specific to the resource that it manifests, such as costs associated with an infrastructure resource or skills associated with a pool of human experts. Examples of concrete unit types are units for modeling compute power (similar to Amazon's Elastic Compute Cloud [EC2] units) or *social compute units* (SCUs)[4] for modeling human expert pools. A system designer using design tools can then link units with elements of the application logic, such as conventional service definitions.

Our objective and claim is to better reason about system qualities such as elasticity when using units in design. Explicitly modeling units lets us handle them automatically and uniformly – for instance, with regard to monitoring, provisioning, and de-provisioning resources.

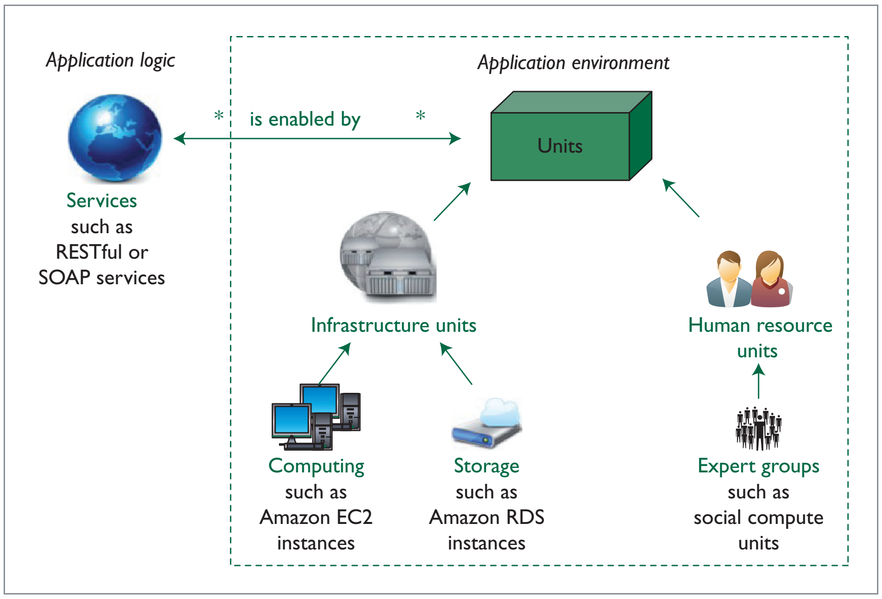

Figure 2. Elastic units. Such units enable the implementation of software services. They can represent both human and compute resources.

The increased attention devoted to specifying units is necessary to assess qualities such as elasticity in Web-scale systems. Measured and assessed qualities are important in a variety of application scenarios and systems, such as the emergency hub example.

Units should exhibit the following common traits:

- *Impact*. Each unit provides a resource to the application, something that's required for it to function or reach its desired quality levels. In our emergency hub, compute units, for example, have an impact on the availability of the call center, storage units on data management, and medical personnel units on the workflows related to treating injured people.
- *Impact measurability*. We should be able to measure the impact on, for example, a metric scale. That is, for each unit, we can measure the relative impact of different configurations; we must be able to decide, for two given unit configurations, which one will provide the "better" service, and by how much. Note that this doesn't need to be an exact science. In reality, we expect that we will often have to make do with estimations and approximations of impact. We can measure call center availability via average response times, for example, whereas we might use different skill sets and levels of medical personnel to determine the number and severity of medical situations that can be dealt with.
- *Cost measurability*. As with impact, units also have defined consumption costs. That is, for each unit, the usage costs must be measurable. In combination with impact measurability, this lets us preemptively reason over the benefits and value of different unit configurations.
- *Dynamicity*. A system must be able to acquire and release new units in a timely, on-demand fashion. Together with impact and cost measurability, this lets us model runtime reactions to changes in the application's environment. Resource management might be automated (as with compute units) or be partly automated or involve human interactions (as with scheduling medical personnel).

Here, we focus on two types of units (infrastructure and human resource) to illustrate the wide array of those possible. Clearly, different applications will depend on different types of units.

Units in Practice

Infrastructure units are currently widely discussed under the umbrella term *infrastructure as a service* (IaaS).[5] Different types of offerings (storage services, elastic computing services, and so on) provide different resources as IaaS, so they implement different types of units. Generally, many of the current IaaS offerings fulfill the requirements we've outlined. As one example, Amazon's EC2 elastic computing services allow for on-demand provisioning of virtual machines, fulfilling the dynamicity requirement. Furthermore, virtual machines have a defined amount of processing power, depending on the so-called instance type. Similarly, the costs of EC2 instances are defined based on the instance type.

Just like most applications, our emergency hub isn't entirely automated. Human resources are necessary for day-to-day operations, and in cases of peak load, more people are required. Traditionally, the costs of human staff are predictable, but impact measurability (of additional or less staff) is problematic. Additionally, human resources are hardly dynamic, given that practical and legislative reasons render labor as one of the most static resources in the application environment. Note that unlike crowdsourcing, which involves masses of people, we focus on the need to model (groups of) skilled experts.

Recently, approaches such as the SCU[4] have started to appear; these treat experts as elastic resources that can scale up and down on demand.

Research and Practice in Elastic Operations

The elasticity of resources is a major driving factor behind the current state of cloud computing. As such, it isn't particularly surprising that many services and tools in the periphery of cloud computing are either inherently elastic to the user, such as *platform-as-a-service* (PaaS) environments (for example, the Google AppEngine; https://developers.google.com/appengine/), or provide means to implement elasticity at the client side (as with the Amazon Elastic Compute Cloud; http://aws.amazon.com/ec2/). For resources other than infrastructure and software, elasticity is currently a neglected research topic. For instance, although the elasticity of human resources has always been relevant in the context of workflow management,[1] little knowledge exists about how to properly scale the pool of staff associated with human activities in a workflow.

On the research side, many ideas appear to increase the elasticity and autonomy of applications, mostly in the context of cloud computing. Recent research has proposed the idea of elastic processes,[2] which started to make explicit different dimensions of elasticity prevalent in dynamic processes. To this end, we can view elastic processes as a first step toward a (domain-specific) abstraction of explicitly modeled units of elasticity, as we describe in the main text. In the cloud world, researchers are working on approaches for modeling functional and nonfunctional cloud requirements.[3] Although these requirements don't make explicit the link between services and resources, as we expect units to do, they could let us establish the quality level that's expected of cloud applications with regard to nonfunctional properties.

What current approaches don't deliver is a common abstraction dealing with different types of resources – such as infrastructure and human staff – in a homogenous way. Currently, no means for modeling these types of resources are available, and no framework exists for automatically managing the runtime provisioning and de-provisioning of different resource types. We argue that units can serve as an entry point toward developing such abstractions, modeling languages, and frameworks.

References

1. D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, 1995, pp. 119–153.
2. S. Dustdar et al., "Principles of Elastic Processes," *IEEE Internet Computing*, vol. 15, no. 5, 2011, pp. 66–71.
3. R. Clarke, "User Requirements for Cloud Computing Architectures," *Proc. Cluster, Cloud, and Grid Computing Conf.* (CCGrid 10), IEEE CS Press, 2010, pp. 625–630.

These expert groups exhibit well-defined competencies, letting systems automatically estimate the impact of additional experts in advance. SCUs help fulfill our basic requirements of measurability and elasticity. However, industrial state of practice will take time to reach the same level of maturity for human resources as is common with regard to many infrastructure units.

Challenges Ahead

To practitioners, it's evident that the variety of units limits the practical elasticity of the emergency hub application illustrated in this article. However, in current development models, resources are typically an afterthought to questions of application and service development. Going forward, we propose units as the abstraction on which to model diverse resources critical to Web-scale service applications. Units make resource usage and resource properties explicit early in system design, thereby allowing for better reasoning about system qualities. The overarching objective is to make complex system qualities such as elasticity measurable and manageable. Measurability is a prerequisite to addressing larger challenges such as quality certification or accountability.

A variety of resources matter, and a variety of units (unit types) for manifesting these resources can correspondingly serve as an appropriate abstraction and handle. In addition to modeling infrastructure resources and social (human expert) resources with units, the following resources (with potential examples of unit abstractions) are the subject of future research:

- *Social resources*, in the sense of masses of people — for example, using majority votes. Units can serve as a structuring mechanism for crowds.
- *Middleware platform resources.* Units can represent application containers or user workspaces.
- *Legal framework resources.* Units might model compliance policies or contracted license keys.
- *Financial resources.* We can use units to represent funding sources within an organization or from external sources.

In summary, units serve as a generic abstraction and handle that can make a variety of resources — which are offered as services in the cloud — measurable and manageable. Units are a first-class abstraction in service system design, making resource needs, usage, and properties more explicit.

References

1. S. Dustdar et al., "Principles of Elastic Processes," *IEEE Internet Computing*, vol. 15, no. 5, 2011, pp. 66–71.

2. S. Dustdar and H. Truong, "Virtualizing Software and Humans for Elastic Processes in Multiple Clouds – A Service Management Perspective," *Int'l J. Next Generation Computing*, to appear, 2012.
3. J.-M. Jazequel and B. Meyer, "Design by Contract," *Computer*, vol. 30, no. 1, 1997, pp. 129–130.
4. S. Dustdar and K. Bhattacharya, "The Social Compute Unit," *IEEE Internet Computing*, vol. 15, no. 3, 2011, pp. 64–69.
5. A. Lenk et al., "What's Inside the Cloud? An Architectural Map of the Cloud Landscape," *Proc. ICSE Workshop Software Eng. Challenges of Cloud Computing* (CLOUD 09), IEEE Press, 2009, pp. 23–31.

Stefan Tai is a full professor of applied informatics at the Karlsruhe Institute of Technology, and director at the FZI Research Center for Information Technology. Contact him at tai@kit.edu.

Philipp Leitner is a postdoctoral researcher in computer science at the Distributed Systems Group, Institute of Information Systems, at the Vienna University of Technology. Contact him at leitner@infosys.tuwien.ac.at; www.infosys.tuwien.ac.at/staff/leitner/.

Schahram Dustdar is a full professor of computer science and head of the Distributed Systems Group, Institute of Information Systems, at the Vienna University of Technology. Dustdar is an ACM Distinguished Scientist. Contact him at dustdar@infosys.tuwien.ac.at; www.infosys.tuwien.ac.at/staff/sd.