EvSyS – Evolving System: Challenges and Payoffs

Special Session along with ICONS 2017, April 23 - 27, 2016 - Venice, Italia
http://www.iaria.org/conferences2017/ICONS17.html

Fabrice Mourlin
LACL lab
UPEC University
Creteil, France
fabrice.mourlin@u-pec.fr

***Abstract*— Evolving systems are inspired by the idea of system model evolution in a dynamically changing and evolving environment. They use inheritance and gradual change with the aim of life-long learning and adaptation, self-organization including system structure evolution in order to adapt to the (unknown and unpredictable) environment as structures for information representation with the ability to fully adapt their structure and adjust their parameters.**

Keywords-component; restful architecture, Web service

I. THE CURRENT SITUATION

One of the key achievements of the past decade has been both the wide availability of evolving systems for the masses, especially Cloud based systems, and the super-scalability of some Web applications. Facebook for instance serves 13 million requests per second on average with peaks at 450 M/s. Even with such results, the concepts and architectures behind evolving systems are still rapidly appearing. About four years ago, Ricky Ho, a software architect based in California, had written documents detailing the state of the art of evolving systems.

What exactly does it mean to build and operate a evolving system or application? At a primitive level it's just connecting users with remote resources via the Internet—the part that makes it evolving is that the resources, or access to those resources, are distributed across multiple servers.

Like most things in life, taking the time to plan when building a system can help in the long run; understanding some of the considerations and tradeoffs behind big applications can result in smarter decisions at the creation of smaller applications. Below are some of the key principles that influence the design of large-evolving systems:

The availability is a first key concern: The uptime of a system is critical to the reputation and functionality of many companies. For some of the larger online retail sites, being unavailable for even minutes can result in thousands or millions of dollars in lost revenue, so designing their systems to be constantly available and resilient to failure is both a fundamental business and a technology requirement. High availability in distributed systems requires the careful consideration of redundancy for key components, rapid recovery in the event of partial system failures, and graceful degradation when problems occur.

System performance has become an important consideration for most systems. The speed of an application affects usage and user satisfaction, as well as search engine rankings, a factor that directly correlates to revenue and retention. As a result, creating a system that is optimized for fast responses and low latency is key.

A system needs to be reliable, such that a request for data will consistently return the same data. In the event the data changes or is updated, then that same request should return the new data. Users need to know that if something is written to the system, or stored, it will persist and can be relied on to be in place for future retrieval.

The scalability also a key feature. When it comes to any large distributed system, size is just one aspect of scale that needs to be considered. Just as important is the effort required to increase capacity to handle greater amounts of load, commonly referred to as the scalability of the system. Scalability can refer to many different parameters of the system: how much additional traffic can it handle, how easy is it to add more storage capacity, or even how many more transactions can be processed.

Designing a system that is easy to operate is another important consideration. The manageability of the system equates to the scalability of operations: maintenance and updates. Things to consider for manageability are the ease of diagnosing and understanding problems when they occur, ease of making updates or modifications, and how simple the system is to operate.

Cost is an important factor. This obviously can include hardware and software costs, but it is also important to consider other facets needed to deploy and maintain the system. The amount of developer time the system takes to build, the amount of operational effort required to run the system, and even the amount of training required should all be considered. Cost is the total cost of ownership.

Each of these concerns provides the basis for decisions in designing a distributed evolving architecture. However, they also can be at odds with one another, such that achieving one objective comes at the cost of another. A basic example: choosing to address capacity by simply adding more servers (scalability) can come at the price of manageability. New solutions have to be found and this is the place of EvSys workshop.

II. New Concepts

When it comes to system architecture, there are a few things to consider, what are the right pieces, how these pieces fit together, and what the right tradeoffs are. Investing in scaling before it is needed is generally not a smart business proposition; however, some forethought into the design can save substantial time and resources in the future.

The core factors are central to almost all large systems: services, redundancy, partitions, and handling failure. Each of these factors involves choices and compromises, particularly in the context of the principles described in the previous section.

When considering evolving system design, it helps to decouple functionality and think about each part of the system as its own service with a clearly defined interface. In practice, systems designed in this way are said to have a Service-Oriented Architecture (SOA). For these types of systems, each service has its own distinct functional context, and interaction with anything outside of that context takes place through an abstract interface, typically the public-facing API of another service.

Deconstructing a system into a set of complementary services decouples the operation of those pieces from one another. This abstraction helps establish clear relationships between the service, its underlying environment, and the consumers of that service. Creating these, clear delineations can help isolate problems, but also allows each piece to scale independently of one another. This sort of service-oriented design for systems is very similar to object-oriented design for programming.

In order to handle failure gracefully a system architecture must have redundancy of its services and data. For example, if there is only one copy of a file stored on a single server, then losing that server means losing that file. Losing data is seldom a good thing, and a common way of handling it is to create multiple, or redundant, copies. This same principle also applies to services. If there is a core piece of functionality for an application, ensuring that multiple copies or versions are running simultaneously can secure against the failure of a single node. Creating redundancy in a system can remove single points of failure and provide a backup or spare functionality if needed in a crisis.

There may be very large data sets that are unable to fit on a single server. It may also be the case that an operation requires too many computing resources, diminishing performance and making it necessary to add capacity. In either case, you have two choices: scale vertically or horizontally.

Scaling vertically means adding more resources to an individual server. So for a very large data set, this might mean adding more (or bigger) hard drives so a single server can contain the entire data set. In the case of the compute operation, this could mean moving the computation to a bigger server with a faster CPU or more memory. In each case, vertical scaling is accomplished by making the individual resource capable of handling more on its own.

To scale horizontally, on the other hand, is to add more nodes. In the case of the large data set, this might be a second server to store parts of the data set, and for the computing resource, it would mean splitting the operation or load across some additional nodes. To take full advantage of horizontal scaling, it should be included as an intrinsic design principle of the system architecture, otherwise it can be quite cumbersome to modify and separate out the context to make this possible.

III. The Alternative

There are many tentative around the code mobility. The technologies are evolved and now it is quite common to exchange code between parts of codes into a larger system. Since the use of interpreted programming languages like Java, Haskell [1], etc. the programmers know how to export objects from one component to another. However, their aim is often more technical than the evolution of a system. It is time to define new design architectural patterns for evolving systems. Integration patterns [2] are new steps into the adaptability of software components. They are considered as a reference in SOA architecture and the use of software bus.

In a special session on Evolving system as part of the ICONS 2017 conference in Venice, Italia [3], two papers are presented that discuss the challenges faced by, and payoffs expected from, the tools and methods that will facilitate the management of system at runtime. Ali Esserhir presents his approach of a Restful system [4] suitable for the exchange of mobile agent at runtime. He proposes a three-step methodology based on tools like Swagger and Spring REST. Next F. Mourlin explains how to build a mobile application [1], which exchanges mobile codes with other platforms with the respect of interoperable directives. This approach allows developers to configure and adapt mobile systems in a heterogeneous network. These works can be extended in other domain like IoT (Internet of Things) where the nodes of a low consumption network are very different each other's.

IV. References

[1] R. Y. &. Y. B. Chen, «Java mobile agents on project jxta peer-to-peer platform,» chez *the 36th Annual Hawaii International Conference on*, 2003.

[2] Y. L. X. X. L. S. M. &. Z. Liu, «Composing enterprise mashup components and services using architecture integration patterns,» *Journal of Systems and Software*, vol. 84, n° %19, pp. 1436-1446, 2011.

[3] «ICONS,» chez *The Twelfth International Conference on Systems*, vENICE iTALIA, 2017.

[4] A. Esserhir, «Evolving agent architecture for data collection,» chez *The Twelfth International Conference on Systems*, Venice Italia, 2017.

[5] F. Mourlin, «Design of Mobile Services for Embedded Device,» chez *The Twelfth International Conference on Systems*, Venice Italia, 2017.