Microservices as CSCW components

Case Study: Pool^2 of Viva con Agua de St. Pauli e.V.

Viva con Agua de St. Pauli e.V. (https://www.vivaconagua.org/) is a German non-governmental organization that aims to collect donations and raise awareness for *Water, Sanitation and Hygine* projects worldwide. Viva con Agua can be classified as an evolutionary-teal organization (Laloux 2014) and thus, it consists of about 50 *crews* (regional, decentralized and loosely coupled groups) of volunteers in Germany, Austria and Swiss.

In 2011 the increased number of volunteers required a technical support tool and the organization implemented the first version of the *Pool*. Managing the different activities of the organization, but also communication and handling of finances become core functionalities of the *Pool*. Thus, the social system shapes the functions and also is shaped by the functions of the *Pool*.

Viva con Agua is successful, because many people participate with joy and creativity. It is not about having larger, but more and outstanding events. Thus, the decentralization of the organization guarantees the success of Viva con Agua and is directly supported by the *Pool*.

At the same time, the decentralization becomes a challenge for the further development of the *Pool* and its adjustment to the changes of the social system. How to bring the different developments of various crews and their required technical support into one tool? So, for example, one crew needs a chat, while the next crew explicitly refuse to use a chat to ensure the human contacts between the supporters.

Implementing such different requirements is complex and needs a well coordinated bigger team of software developers. Many organizations, as the non-governmental Viva con Agua, are not able to maintain the required software development projects.

IT project culture

Thus, the Heureka! architecture allows to move the social concept of Viva con Agua to IT projects. Developers should become intrinsically motivated and should have fun in realizing the project. The new version of the *Pool* bases on the Heureka! architecture and is named Pool^2. Thus, it is a composition of several small, decentrally organized and loosely coupled software artefacts.

Software developers involved in the project are aligned to the symbiotical relation between social and technical system. Thus, structures similar to open source communities may evolve, but also student works and support of other organizations may implement some change. Furthermore, the loosely coupling between the participants has to be ensured. Therefore, it should require only less effort to learn about the *Pool^2* and the Heureka! achitecture. Additionally, the independence of technologies should be assured.

Microservice architecture

The previously mentioned challenges will be addressed by a loosely coupled architecture. Nowadays, microservices are typically used (Newman, 2015). A microservice is a stand alone application running in one process and implements a strong cohesion regarding a *bounded context*. Furthermore, such a service constantly communicates with other services using leightweight technologies, like RESTful webservices. A set of such microservices is named a microservice architecture (Newman, 2015; Dragoni et al., 2017).

Since microservices are loosely coupled to each other, a free choice of technology is possible, considering the constraints of section concepts. Also coordinative dependencies are restricted and thus, the architecture allows to integrate many different and not or loosely coupled software developers.

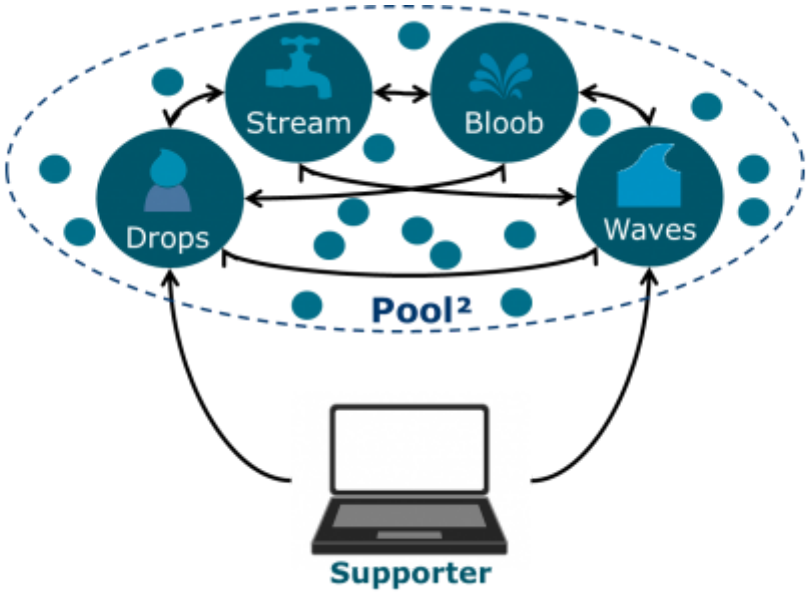

Figure 1 sketches the functional requirements of the *Pool*[2] projected to a microservice architecture. Drops handles the user profiles, while stream holds the finances and waves the activities and events. Bloob implements communication functions.

Concepts describes the non-functional requirements as well as some of the microservices implementing these needs. Furthermore, some non-functional requirements are still open, like an Awareness component.

A microservice architecture implicits some challenges not occuring in classic monolithical architectures, like implementing a *Shared Session* between the microservices. How to implement the communication between microservices? How to ensure the usage of a common corporate Design (CD) without implementing duplicate code?

Concepts

Several challenges have to be considered for implementing a microservice architecture. This section draws an overview about the basic concepts addressing the technical challenges. The list is a first orientation for implementing a microservice.

Name	Beschreibung	Status
Dynamic UI Fragment Composition	In a decentralized microservice architecture the implementation of a user interface becomes a challenge. Since a central service implements strong dependencies to other microservices, such an implementation is **no** solution for the required loosely coupling. Thus, every service has to implement an own user interface, considering a corporate design and should avoid code duplication. The present concept describes how these aims can be reached.	IN USE
Business Object Exchange	One microservice is responsible for (multiple) business objects (BO). These BO may have to be transfered to other microservices, to use the described information for other requirements. The present concept describes the BO exchange using RESTful webservices.	IN USE

Name	Beschreibung	Status
Shared Session	Microservices need to identify users. Avoidung the implementation of authentification functions per microservice, the present concept explains the implementation of a shared session using a OAuth2 handshake.	IN USE

References

(Dragoni et al., 2017) N. Dragoni et al., “Microservices: yesterday, today, and tomorrow.” Cornell University, 2017.

(Laloux 2014) F. Laloux, Reinventing Organizations, 1st ed. Brussels: Nelson Parker, 2014.

(Newman, 2015) S. Newman, Building Microservices, 1st ed. O’Reilly Media, 2015.