HyVar NEWSLETTER

N°3, June 2017

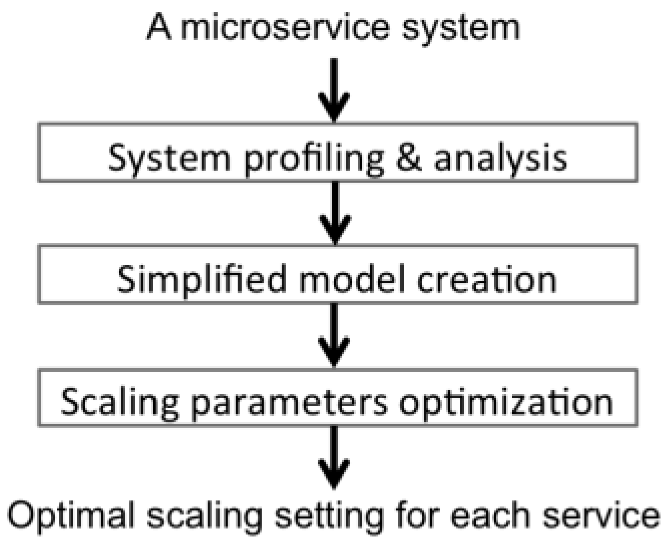
Optimization of HyVar Framework Deployment

Nowadays systems are designed and deployed on the cloud to exploit the cloud elasticity. This allows a system to react to load changes by varying the amount of computational resource used. Deciding the proper scaling settings for a complex architecture is, however, a daunting task: many possible settings exists with big repercussions in terms of the system performance and cost.

In the context of HyVar, we developed a framework composed of 8 different components that are all deployed on the cloud (for more information please see http://www.hyvar-project.eu/hyvar/technical-information/).

Following the paradigm of microservices, the system was developed as a collection of small services, each running its own process and communicating with lightweight mechanisms to obtain a flexible, maintainable, reusable, and compositional framework.

However, each of these micro-services has a lot of deployment parameter to configure, e.g., the metric threshold that if violated triggers the scaling up/down actions, the amount of new virtual machines that are created every scaling up/down action.



A microservice system
System profiling & analysis
Simplified model creation
Scaling parameters optimization
Optimal scaling setting for each service

Figure 1

Clearly, these parameters have a huge impact on the cost and the possibility to handle load spikes in a reasonable amount of times. In order to understand what are the best possible parameters for handling a given traffic we proposed and used a model-based methodology (see Figure 1).
Instead of directly proceed with the optimization of the settings of the entire microservice system, we used a model to simulate the performance of the real system. Compared to running the real system, using a model is cheaper (renting resource in a cloud can be expensive) and also time saving (a simulation may take far less time than running the real system). In particular, as a first step we used worst case analysis (e.g., queuing theory) and profiling techniques to understand what parts of the system can be simplified. This allowed, in the second step, to create a simple model of the entire system with fewer parameters to tune. Finally, in the third and last step of the methodology, we proceeded in the search of good settings by using automatic parameter configurator tools (e.g., SMAC: Sequential Model-based Algorithm Configuration - http://www.cs.ubc.ca/labs/beta/Projects/SMAC/).

In this way, relying on machine learning techniques, we explored in a smarter and more systematic way the possible parameter configuration settings.
In particular, we validated the proposed methodology by running for one day the HyVar framework with the traffic pattern derived from the number of cars registered on the A414 highway.

We use the average latency of every component to understand if it needs to scale up or down with the final goal of having a total average latency of less than 5 minutes. We first verified that the bounds obtained by the worst case analysis (step 1) hold. Then we run the experiment by setting the parameters returned by the SMAC optimizer.

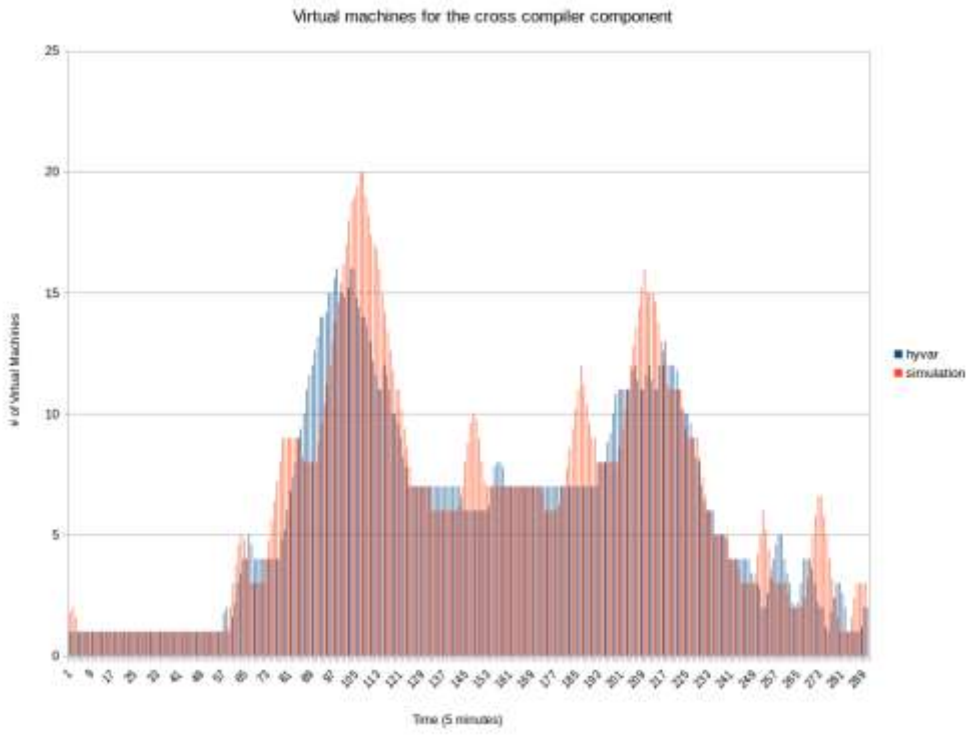


Figure 2

As can be seen from Figure 2 showing the behavior of the real system against the simulation in terms of the number of virtual machines used for the cross compiler component (one of the most computationally intensive in the HyVar framework), the simulation was able to mimic and anticipate the real scaling up/down events.

In conclusion, by following our methodology, we were able to:

- simulate the performance of the real system avoiding to use expensive cloud resources to run the tests,
- optimize in an automatic way the scaling parameters of our elastic application,
- use less cloud resources to run the HyVar framework but still meeting our requirements.

HyVar project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644298

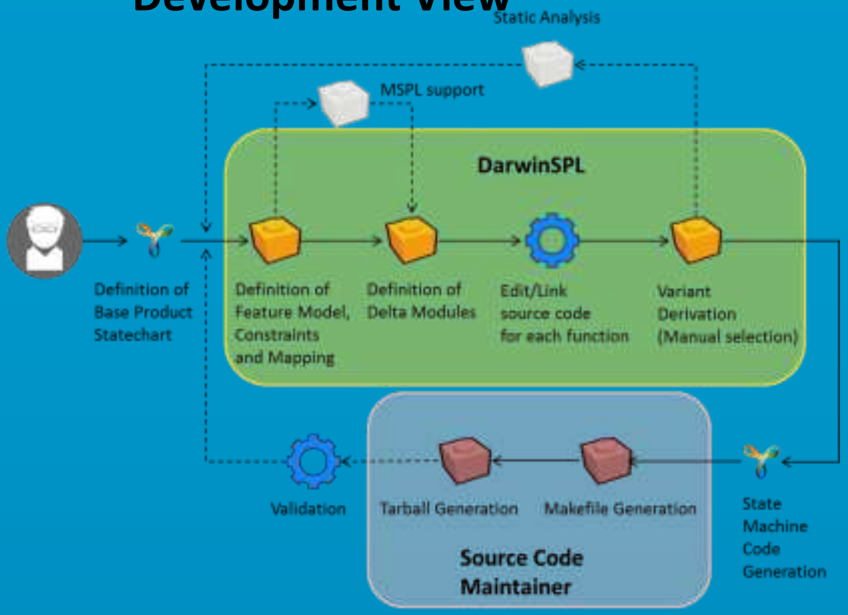
A Tool Chain for efficient Software Reuse

Code reuse is widely adopted by software-intensive companies, such as the ones developing software for automotive, home automation or smart devices, to leverage existing assets and to reduce development cost and time.
As different versions of applications often have similar requirements, rather than create entirely separate solutions, it is more convenient to create a single base solution supporting multiple applications.
Moreover emerging scenarios, such as Internet of Things (IoT) and Cyber-Physical Systems (CPS), are characterized by a huge number of remote devices, each of whom has its own hardware configuration, runs a customizable distributed software application and need to evolve in order to fix or prevent misbehavior, to adapt to environmental changes, accomplish new regulations, satisfy new user requests or meet new market expectations.
Different levels of software reuse exists from completely ad-hoc opportunistic reuse as copy & paste, also referred as cloning, to systematic software reuse as the one promoted by Software Product Line Engineering (SPLE).
In SPLE variability and commonality of a set of product variants regarding their functionality is modeled in what is called a feature model. The feature model captures common and different behavior of software systems in terms of variability and constraints.

Cloning is very flexible and inexpensive in design but extremely costly in later maintenance, as the different codebases are maintained separately.
Formal reuse through Software Product Lines on the other side, allow to increase the productivity and maintainability, but is expensive at design time and is not flexible enough to support continuously evolving distributed software.
HyVar project aims to overcome these issues integrating SPLE principles with existing tools and common used industrial practices, supporting the development and deployment of individualized software adaptations and thus realizing the concept of Hybrid Variability.
In HyVar we consider three dimensions of variability representing sources of software systems to behave differently: configuration as spatial variability, dependence on surroundings as contextual variability and evolution as temporal variability.
The HyVar Tool Chain provides design time support (see Figure 3 – Development View) for integrated modeling of these three dimensions of variability through the DarwinSPL component. The approach adopted is based on delta modeling, a transformational variability realization mechanism.

The HyVar Tool Chain allow to define the underlying architecture of an organization's Software Product Line in terms of base product and variabilities and to derive a specific software product from a SPL, by selecting delta modules, edit/link new required functions, automatically generate the code, validate the resulting executable against requirements.
With the integration of [Yakindu statecharts]() in the DSVL, current industrial practice is supported and industrial users can integrate their existing systems modeled by statecharts. As code in different languages (C, C++ or Java) can be generated out of statecharts, the methodology of HyVar can be adapted to different needs. Moreover, it is sensible to be able to link existing code artifacts to the automatically generated code and to compile the code for different target platform. This goal is realized through the Source Code Maintainer component. After deployment (see Figure 3 - Deployment View), software needs to be customized for the specific user and adapted to the device used as well as to the environmental conditions such as the physical location and context. The sensor data collected from a specific device trigger the automatic generation and deployment of the most appropriate software upgrade. HyVar realizes the automatic reconfiguration of SPLs variants based on current context through HyVarRec component and cloud-based ECU Over the Air Software Update.

Development View


Static Analysis
MSPL support
DarwinSPL
Definition of Base Product Statechart
Definition of Feature Model, Constraints and Mapping
Definition of Delta Modules
Edit/Link source code for each function
Variant Derivation (Manual selection)
Validation
Tarball Generation
Makefile Generation
State Machine Code Generation
Source Code Maintainer

Deployment View

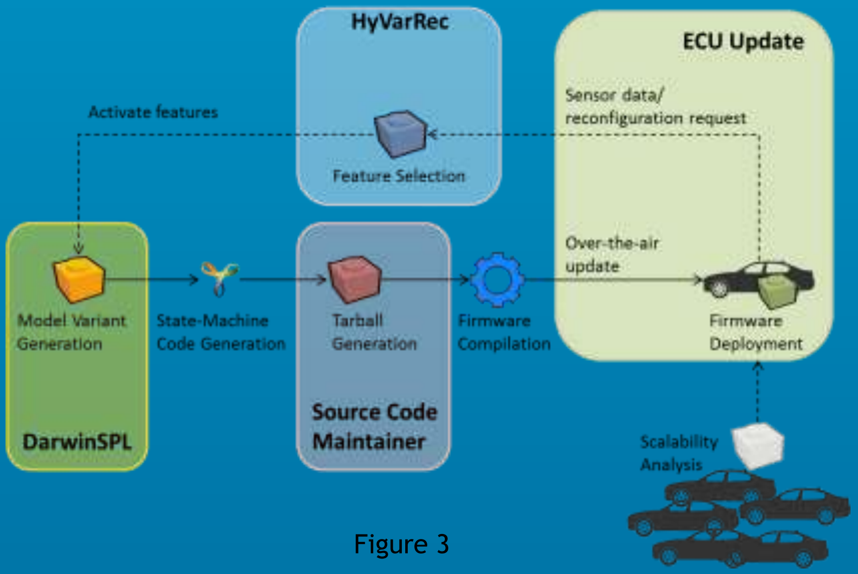


Figure 3

In the continuation of our work we plan to further extend the Tool Chain to support Multi Software Product Lines and Multi-ECUs software updates, in order to enable advanced scenarios.

HyVar Events

- ✓ Integration meeting Torino, February 2017
- ✓ General Meeting in Braunschweig, June 2017
- ✓ NetFutures 2017

Next Events

- ✓ iFM - 13th **International Conference on integrated Formal Methods**
  Torino, September 2017
  http://ifm2017.di.unito.it/
- ✓ ESOCC - 6th **European Conference on Service-Oriented and Cloud Computing**
  Oslo, September 2017
  http://esocc2017.ifi.uio.no/
- ✓ Integration meeting Torino, October/November 2017
- ✓ General Meeting in Norway, January 2018