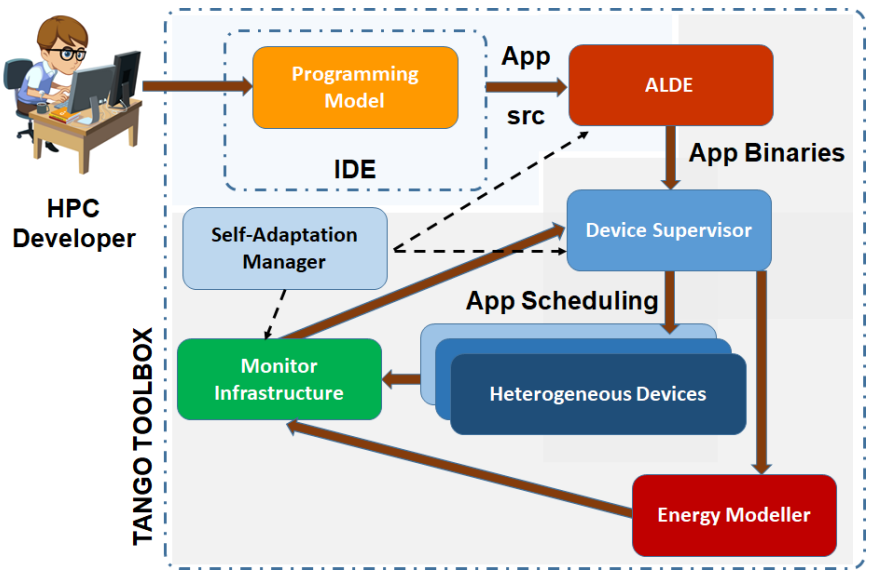
Workflow scenarios description for TANGO project

This document aims at describing the different workflows of usage for TANGO toolbox. It describes the scenarios where to use each of the TANGO components. Each component can be used in one or more scenarios. There can be so many workflows as needed to consider all components at least once, and the list of workflows can be enlarged in the future with new scenarios of usage. A common template is followed to describe the identified workflow scenarios.

Workflow 1: Programming efficiently HPC apps

Scenario storyline: This scenario aims at showing the benefits of TANGO for HPC developers. It will be typically applied to support HPC developers in simplifying their code programing in the most efficient way for taking advantage of all the heterogeneous devices where their code will be deployed and optimizing the energy that their code will consumed in that devices. It will be used to demonstrate how TANGO facilitates the programing, automatic deployment and energy optimization of HPC applications in heterogeneous devices.

Scenario representation:



Involved/target users: HPC developer

TANGO components:

- Programing Model
- Runtime Abstraction Layer
- ALDE
- Device Supervisor
- Monitor Infrastructure
- Energy Modeller
- Self-adaptation Manager

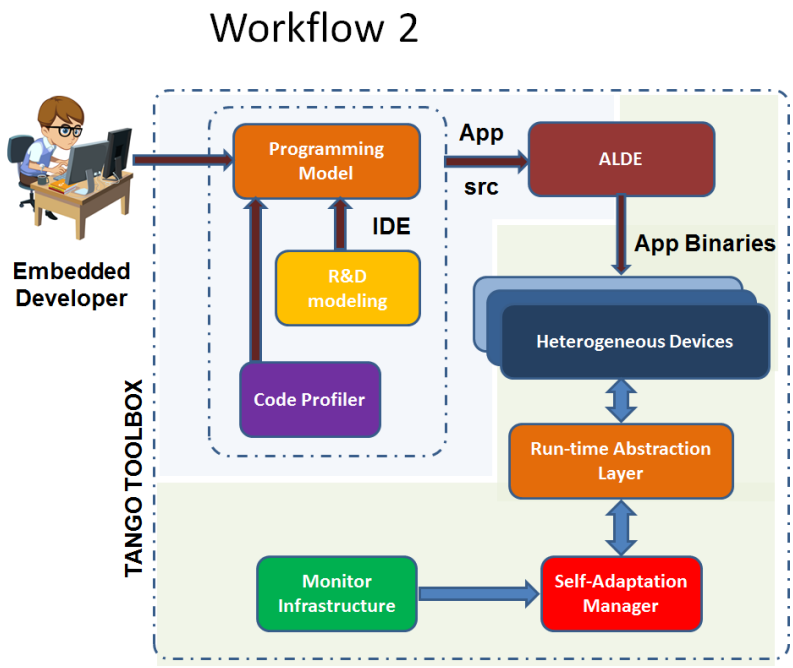
Workflow steps:

- HPC developer uses **Programing Model** for writing their code. Code is compiled once but deployed in two available types of infrastructure without re-compilation. Now PM is only a model, but in future it will be integrated in an IDE. In the final TANGO version, PM should not compile and build, but ALDE.
- Then, **ALDE** takes the output of the IDE (in this case PM) and it prepares the application for deployment. ALDE is able to create several versions of the same application optimized for different heterogeneous architectures. This applications can be built in different package formats/containers: Easybuild, Singularity, Docker… ALDE is a set of scripts that compiles and builds the app to be deployed automatically in the Device Supervisor.
- In the next step, **Device Supervisor** takes the app and some execution requirements (type of resources) from ALDE and is in charge of allocating the different jobs in the available resources. It also supervises the node and collects measures.
- Then, once app is running on heterogeneous devices, the **Monitor Infrastructure** monitors them, especially what concerns energy usage. It uses Grafana (tool for creating graphs) to show the monitoring information and collects diverse measures it takes from the heterogeneous devices where app is running.
- The monitoring infrastructure will be used by the **Self-Adaptation Manager**, to allow it to read event triggers that it can respond to, additionally it will read measurements from the MI so that its own event triggers may be used. These events will be responded to by a customizable set of adaptation rules. These rules will specify forms of actuation that will be permitted. One such adaptation will be to redeploy an application so it can run on an alternative architecture.
- **Energy modeller** is forecasting the energy consumption of an app running in an infrastructure. It generates the model of consumption of the infrastructure to allocate only the needed resources for the app. This information will be used in next phases to optimize the code. This may work either at deployment where power hungry solutions can be discarded or at runtime by the self-adaptation manager to determine how best to adapt an application to maintain its energy efficiency.

Workflow 2: Programing efficiently embedded apps (simplifying code improving performance and energy saving)

Scenario storyline: This scenario aims at showing the benefits of TANGO for embedded systems developers. In this context, embedded systems means autonomous devices dedicated to a specific task. TANGO framework will be typically applied for simplifying the code programming and help improving performance and energy saving on heterogeneous hardware. In embedded world, heterogeneous hardware will often mean one or a few CPU, accelerated with one or a few GPU or FPGA devices.

Scenario representation:



In this scenario, the application performs some data processing either on a regular basis or either on-demand. Programmers develop their code into usual software development tools. Depending on the size of the platform, development may take place directly on the target platform or on a remote cross compiling system. Development system generates an executable file or a set of executable files that will be loaded by the system at power-up from the file system. The application the runs until the target shuts-down. The device is devoted to the application. Therefore, no other application except those being required for the general device operation is running on the system.

Involved/target users: Developers of embedded apps

TANGO components:

- Programming model
- Runtime Abstraction Layer
- Design time characteriser (Poroto)
- Code Profiler
- ALDE

- Self-adaptation manager
- Monitoring Infrastructure

Workflow steps:

- The embedded application developer uses **Programing Model** for writing their code. Code is compiled with tools associated with this PM. Now PM is only a model, but in future it will be integrated in an IDE. In the final TANGO version, **ALDE** should control this compilation and help to create packages that can be loaded on different platform architectures. The application developer can use design time tools, like the **Design time characteriser** and the **Code profiler** to improve their code, selecting the best target architecture or enforcing the execution of part of the code on a specific device.
- Once application is running on the target heterogeneous system, the **Monitor Infrastructure** monitors them, especially what concerns energy usage to provide energy related information to the **Self-Adaptation Manager**, which must interact with the run-time layer of the programming model and/or the application to optimise either energy, quality of service or both.

Workflow 3: Designing and programming heterogeneous platform with configurable hardware (FPGA)

Scenario storyline:
Application programming often relies on existing implementation of simple software components. However, these components may not be targeted to exploit heterogeneous hardware. For instance, an existing standard C implementation of a few software components may already exists but they have been programmed for running on homogeneous distributed platform with multi-core CPU processors.
Thanks to the Design and Development time Tools (notably Poroto, Placer) as well as the C/OmpSs Programming Model, an analyst developer has access to an IDE to craft more easily applications that can exploit heterogeneous hardware platform to achieve efficient time and energy performance.
This scenario will illustrate an iterative development approach where the tools Poroto, Placer, and C/OmpSs plugins are used to develop a simple use case application tailored to an efficient use of an heterogeneous hardware platform with an FPGA.

NOTE: We anticipate that for September review, the application used for the demo will be based on sequences of multiple matrix multiplications with matrices of different sizes (where different deadline requirements may be assigned to different sequences of matrix multiplications)

Involved/target users: Analysts and Developers of apps to be deployed in heterogeneous devices with non-fixed instruction sets (e.g. FPGA)

TANGO components:
- Design Time Characteriser (relying on Poroto for porting code to FPGA - we could augment this scenario with the use of device emulator for characterisation on GPU)
- Placer
- Programing Model
- Monitoring Infrastructure (We will rely on the Monitoring Infrastructure implemented on CETIC's testbed - currently it collects time and energy performance in a dissociated way. And the merging of the monitoring information still requires a manual effort)

Workflow steps:

1. Identify a partially ordered set of matrix multiplications that will be used as Application use case.
2. The developer characterises (measures) the time and energy performance of the execution of different matrix multiplication implementations on CPU for input matrices of different sizes and filled to different degrees. One classical implementation, a tile version, eventually an implementation for sparse matrices, data streaming on inverted-diagonal, etc.
3. The developer annotates a portion of a matrix multiplication C implementation with Poroto pragmas and then generate the resulting C implementation with a call to the

FPGA offloaded portion and the VHDL for the FPGA offloaded portion with data handling included.

4. The developer uses the FPGA vendor low level synthesiser to generate the proper FPGA configuration from Poroto generated VHDL
5. The developer instantiates the FPGA with its configuration
6. The developer characterises the time and energy performance of the execution of the selected matrix multiplication with its FPGA offloading enabled for the same set of input matrices as in Step 1 above.
7. The developer repeats steps 3-6 for various matrix multiplication implementations (e.g. classical, sparse, tile, streamed on inverted-diagonal, etc.)
8. Create the Placer input model for the heterogeneous hardware platform found on CETIC's testbed
9. The developer creates the Placer input model of software tasks for each type of matrix multiplications (from the partial ordered set defined in Step 1) and provide its characteristics on resource usage including the performance characteristics when executed on the different type of hardware (CPU and FPGA)
10. The developers launches Placer with the input specified in steps 8 and 9 to obtain the optimal software task placement and schedule on the specified heterogeneous hardware.
11. The developer provide the code to achieve Placer's proposed schedule. In cases where Placer's schedule show that a certain task can execute in parallel on CPU and FPGA, the developer can achieve this by following the (C)/OmpSs Programming Model for implementing the given task (in this case a type of matrix multiplication on the partially ordered set)
12. The developer can package the final version of the matrix multiplication example with the needed FPGA kernel, the (C)/OmpSs runtime, and the main application annotated with (C)/OmpSs Programming Model pragmas.
13. The developer manually redeploy the packaged application
14. The developer launches the application on various sample input sets to validate that the schedule (time performance/span) and the energy consumption profile identified by Placer are achieved within an acceptable margin of error.