

CloudScale Environment User Guide



Funded by the European Commission's
Framework Programme Seven (FP7).



20th of January, 2016



Contents

Contents	2
Introduction.....	4
What is CloudScale Environment?	4
Overview	4
Further information	4
Reporting issues	6
Getting started with CSE.....	7
Installation	7
Requirements.....	7
Download	7
Installation & Run.....	7
Development	7
Source code and license.....	7
Maven (Tycho)	8
Eclipse IDE	8
CloudScale Environment.....	9
Quick intro	9
Main perspective	12
CloudScale Project	14
Editors	16
Workflow.....	18
Tools perspectives	22
Examples.....	23
ScaleDL.....	25
Extended Palladio Component Model (Extended PCM)	25
Relation to Workflow	25
Features	26
Input & Output.....	26
Minimal Example.....	26
Architectural Templates (ATs)	30
Relation to Workflow	30
Features	30
Input & Output.....	31
Walkthrough	31
Usage Evolution (UE)	33
Relation to Workflow	33
Input & Output.....	34
Walkthrough	34

Overview	37
Relation to Workflow	38
Input & Output.....	38
Minimal Example.....	39
Extractor	42
Introduction	42
Relation to Workflow	42
Problem.....	42
Features	43
Usage.....	43
Walkthrough	45
Analyser	47
Introduction	47
Relation to Workflow	47
Problem.....	48
Features	48
Usage.....	48
Walkthrough	50
Static Spotter	52
Introduction	52
Relation to workflow.....	52
Problem.....	53
Features	53
Usage.....	53
Walkthrough	54
Dynamic Spotter	56
Introduction	56
Reference to workflow.....	56
Problem.....	56
Features	56
Instrumenting target application (SUT).....	57
Usage.....	58
Walkthrough	63

Introduction

What is CloudScale Environment?

The CloudScale Environment (CSE) is an open-source toolset with integrated method support whose aim it to provide an engineering approach for building scalable cloud applications by enabling analysis of scalability of basic and composed services in the cloud. The toolset is used to predict the behaviour of services in the cloud, measuring different scalability metrics and to find exponential problems in the code by running static analysis and/or running experiments on the live system.

Objectives of the CloudScale project are:

- Make cloud systems scalable by design so that they can exploit the elasticity of the cloud, as well as maintaining and also improving scalability during system evolution. At the same time, a minimum amount of computational resources shall be used.
- Enable analysis of scalability of basic and composed services in the cloud.
- Ensure industrial relevance and uptake of the CloudScale results so that scalability becomes less of a problem for cloud systems.

CSE is a desktop application integrating the CloudScale tools, consisting of Dynamic and Static Spotters, the Analyzer and the Extractor, while driving the user through the flow of the CloudScale method. Modelling language that connects the tools between each other and enable analysis is named ScaleDL, and consists of five sub-languages: ScaleDL Usage Evolution, ScaleDL Architectural Templates, ScaleDL Overview and Extended PCM.

Overview

This document contains the user manual for the CloudScale Environment and a walk-through (Minimal Example) that allows the reader to quickly run the integrated tools using a simple and easy to follow example.

Further information

CloudScale Environment has been developed under CloudScale project founded by the European Commission's Framework Programme Seven (FP7). Project details are available on the CloudScale web site

<http://www.cloudscale-project.eu/>

For additional information regarding the CloudScale project one can find useful information here:

- Results: <http://www.cloudscale-project.eu/results/>
- Publications: <http://www.cloudscale-project.eu/publications/academic-publication/>
- Public deliverables: <http://www.cloudscale-project.eu/publications/deliverables/>

Additionally, results and scalability knowledge are also disseminated through CloudScale Knowledge wiki page, providing best practices (how-tos) for designing and analyzing scalable SaaS applications including pitfalls (how-not-tos) that can result in negative consequences and which scalability engineers should try to avoid. The Wiki also provides the CloudScale Glossary and a number of white papers.

<http://wiki.cloudscale-project.eu/>

For further details on using the tools, one can see tools' detailed descriptions and documentation:

Dynamic Spotter

<https://github.com/sopeco/DynamicSpotter-Demo/releases/download/v1.0/DynamicSpotter-UserGuide.pdf>

Static Spotter

https://sdqweb.ipd.kit.edu/mediawiki-sdq-extern/images/4/49/SoMoX_Metric_Documentation.pdf

Extractor

<https://sdqweb.ipd.kit.edu/publications/pdfs/becker2010a.pdf>

Analyser

<https://anonymous:anonymous@svnserver.informatik.kit.edu/i43/svn/code/QualityAnalysisLab/Documentation/trunk/org.palladiosimulator.qual.docs/QualityAnalysisLab.pdf>

<https://anonymous:anonymous@svnserver.informatik.kit.edu/i43/svn/code/QualityAnalysisLab/Documentation/trunk/org.palladiosimulator.qual.docs/QualityAnalysisLab.pdf>

Palladio Component Model

http://www.palladio-simulator.com/fileadmin/user_upload/palladio-simulator/documents/Introduction-Chapter-PCM.pdf

ScaleDL Overview

http://www.cloudscale-project.eu/media/filer_public/2014/11/12/scaledl_overview_2nd_draft.docx

Reporting issues

Official CloudScale repositories are available on GitHub under CloudScale-Project organization available at

<https://github.com/CloudScale-Project>

In order to make CloudScale Environment as useful and reliable as possible, we encourage users to submit problem reports and feature request via the official repository issue tracker available on GitHub:

<https://github.com/CloudScale-Project/Environment/issues>

If you can already provide a fix or extension to CSE we will be happy to review a change and merge it into the code base.



Getting started with CSE

Installation

Requirements

- Java Virtual Machine: <https://www.java.com/en/download/>
- Windows, OSX or Linux operating system (32 or 64 bits)
 - NOTE: if 32-bit Java is used on 64-bit OS, the 32-bit bundle needs to be downloaded

Download

- The CloudScale Environment bundles available at
 - <http://www.cloudscale-project.eu/results/tools/>
- Releases and Nightly builds
 - <http://cloudscale.xlab.si/cse/products/releases/>
 - <http://cloudscale.xlab.si/cse/products/nightly/>

Installation & Run

1. Download bundle
2. Extract/Unzip bundle
3. Open folder and run 'cse' or 'cse.exe'

Development

CloudScale Environment (CSE) is a Java application built on top of Eclipse Rich Client Platform framework (Eclipse RCP version 4.3), using Eclipse Modelling Tools (EMF) and Graphiti framework for modelling and much more ... The CloudScale Toolchain is externally referenced, therefore all contributions to the tools has to be made directly in the co-responding repositories; nightly build will grab the changes and integrate them into the CSE.

Source code and license

The CloudScale Environment source code is licensed under the Eclipse Public License v1.0

<https://www.eclipse.org/legal/epl-v10.html>

Official repositories are located on GitHub under CloudScale-Project organization:

CloudScale Environment: <https://github.com/CloudScale-Project/Environment>

Toolchain build: <https://github.com/CloudScale-Project/Toolchain>

Examples: <https://github.com/CloudScale-Project/Examples>

Maven (Tycho)

1. Clone the repository.
 - o \$ git clone <https://github.com/CloudScale-Project/Environment.git>
2. Build CloudScale Environment.
 - o \$ mvn package
3. Run Linux, MacOS, Windows distribution.
 - o Bundle location: `plugins/eu.cloudscaleproject.env.master/target/products`

Eclipse IDE

1. Download and install Eclipse 4.3 (EMF version)
<http://www.eclipse.org/downloads/packages/eclipse-modelling-tools/mars1>
2. Download and install Eclipse plugin dependencies for CloudScale development
 - a. Go to Eclipse->Help->Install New Software
 - b. Add CloudScale Toolchain update site:
<http://cloudscale.xlab.si/cse/updatesites/toolchain/nightly/>
 - c. Install Toolchain features (Analyser, Extractor, Static Spotter and Dynamic Spotter), and Sources (sometimes Dependencies needs to be installed first and after that everything else)
3. Clone repository
 - a. \$ git clone <https://github.com/CloudScale-Project/Environment.git>
4. Import CloudScale Environment plugins, under "plugins/" directory, into the workbench.
5. Run product (`eu.cloudscale.env.product`)

CloudScale Environment

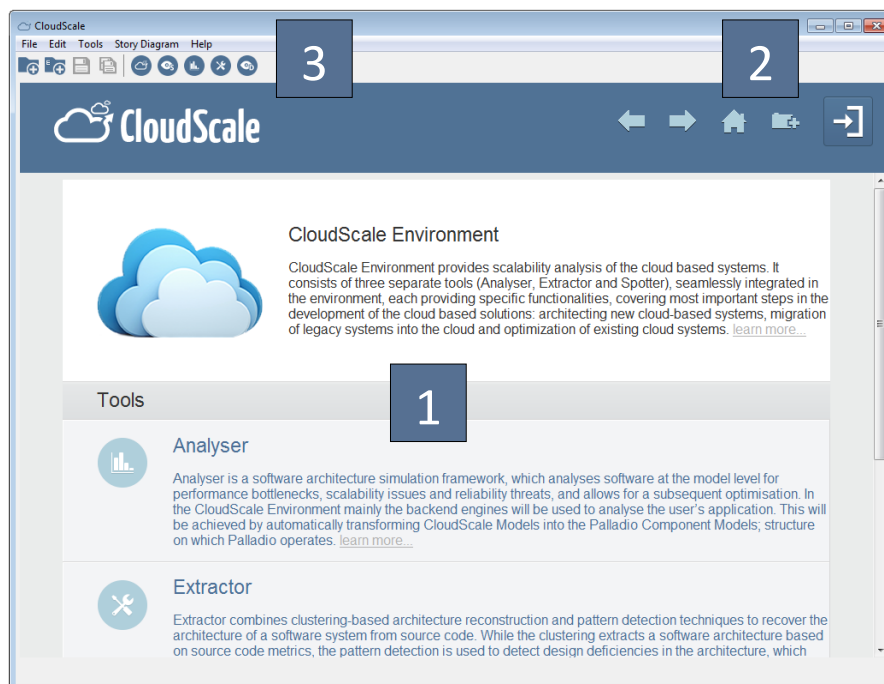
The CloudScale Environment is an OS-Independent desktop application built on top of Eclipse RCP, therefore it should be familiar and easy to use for those who are well acquainted with Eclipse, which we here assume is the case.

This section describes basic functionality of the integration environment, how to use the integrated tools and how one can follow CloudScale Method to analyse a system, to predict its' behaviour in the cloud and to find potential anti-patterns. The section starts with a brief overview of the most important components and follows with detailed descriptions of specific components and instructions how to use them.

Quick intro

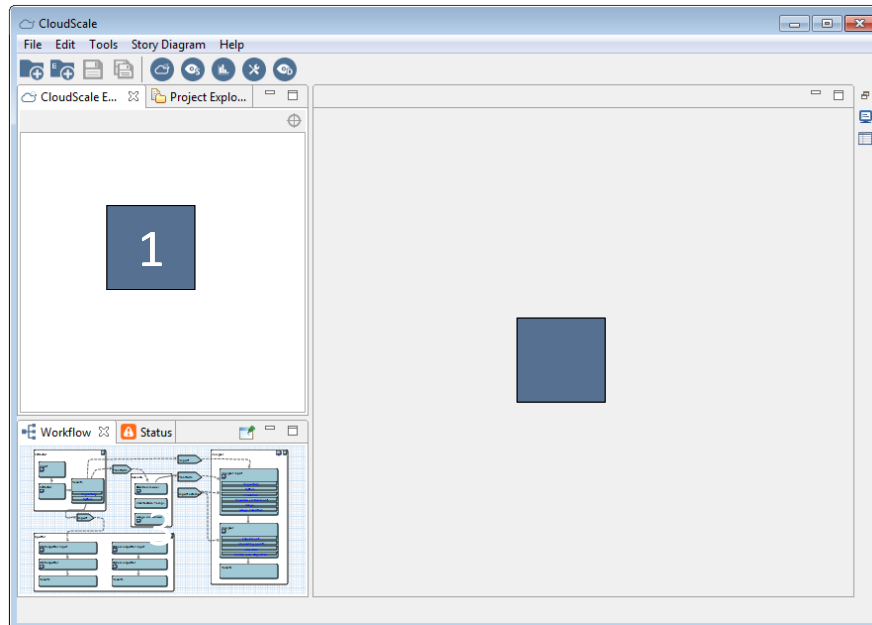
This section shows how to create new project in the CSE and how to navigate through different editors and views. Detailed description and usage is contained in the following sections.

After starting CloudScale Environment user welcomed with introduction screen containing basic information about the project and the tools [1], and allows following links to see more information on tools' home pages.

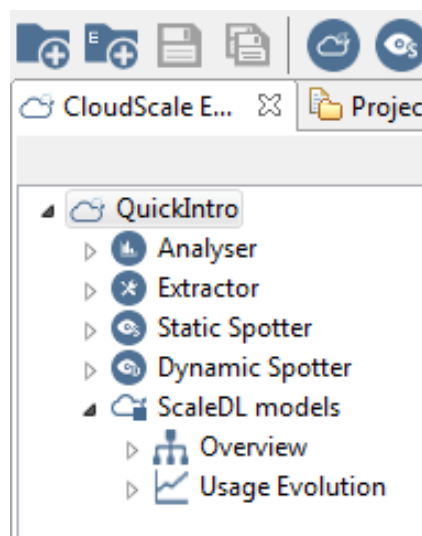


To open main CSE one can close intro screen using button on the right in the intro toolbar [2]. Another option is to use CSE toolbar actions and menus [3] (see **Error! Reference source not found.**).

Main perspective is at the beginning empty. There are three main sections: project [1], editor [2] and status [3] (see **Error! Reference source not found.**).



To create new CloudScale Environment project click first tool item in the toolbar. Furthermore, it is also possible to import integrated examples by clicking second tool item.



After creation, project is displayed in the CloudScale Explorer (Main Perspective > CloudScale Explorer) representing each tool and ScaleDL model that compose functionality of the CSE. User can create new alternatives for each item (tool or ScaleDL model) through

[illegible]

The CSE is continuously validating all alternatives so users are warned when something needs to be done or when something breaks. Image shows errors, since this is initial state of alternative after creation and more work needs to be done by user to actually run it. (see Main Perspective > Workflow). Workflow diagram is context-aware and shows statuses of currently selected items in explorer or currently focused editor. By double-clicking on the nodes users can open editors or wizards (action nodes).

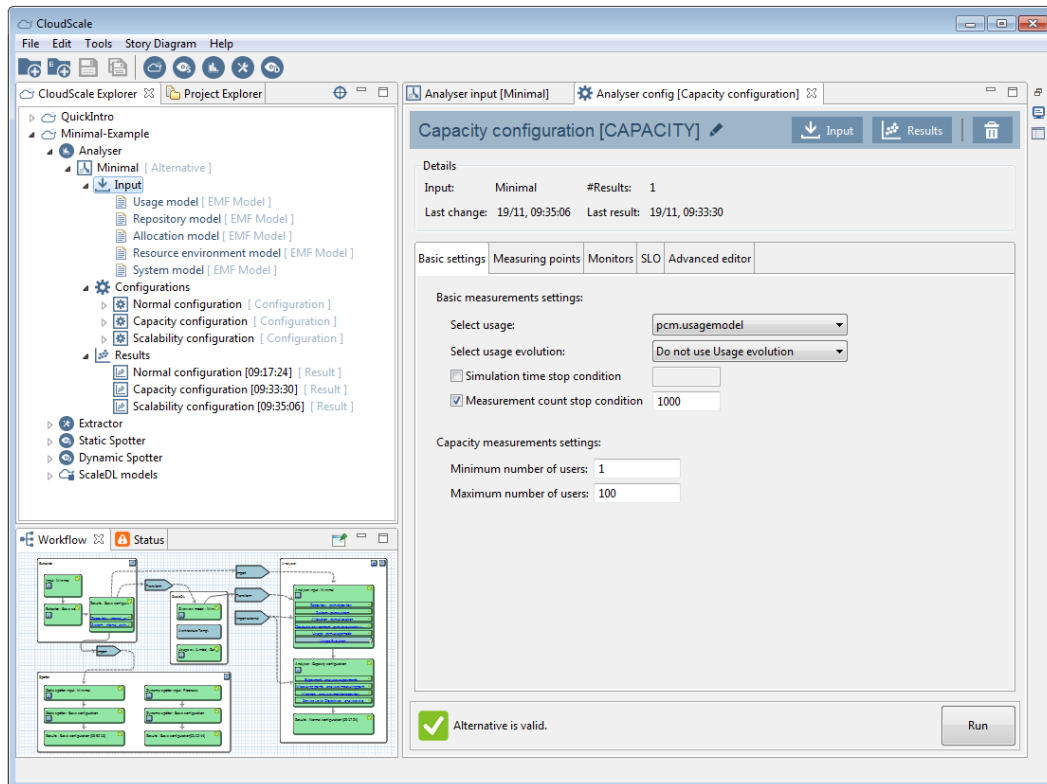
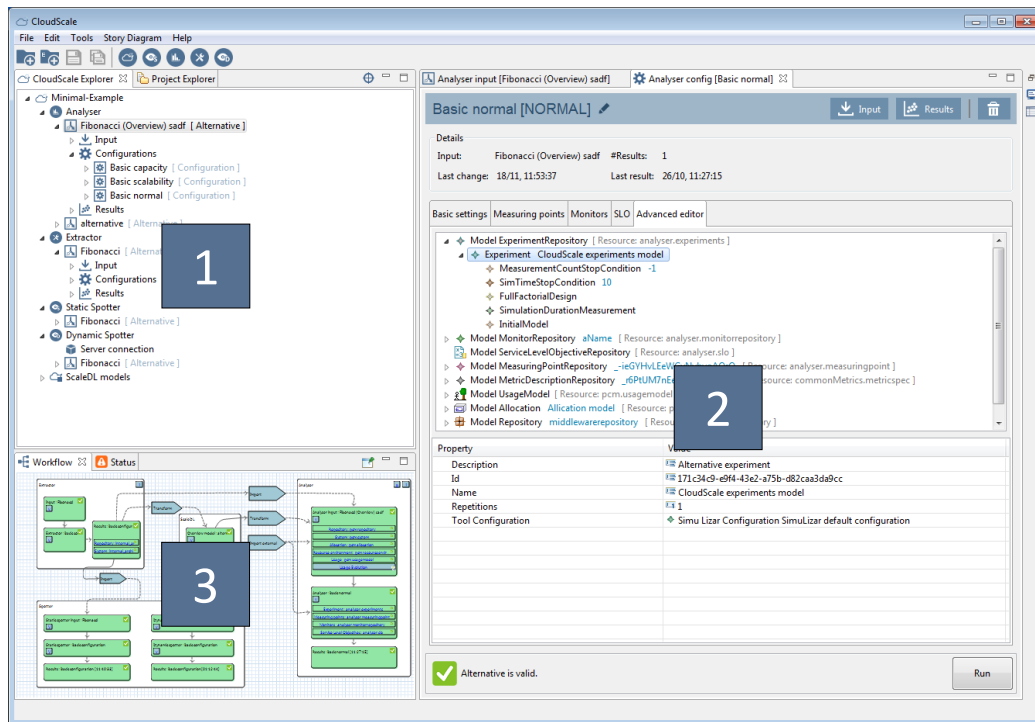


Image shows Minimal example (and Analyser configuration editor), where all alternatives are successfully validated (everything is green). To run alternative one needs to open configuration editor and click on 'Run' button. Examples also contains pre-run results which can be seen by double clicking on them in explorer or clicking on 'Results' in editor.

Main perspective

In the Eclipse RCP a perspective is a visual container for a set of views and editors. The main perspective in the CSE is composed of unified and well defined set of components (views and editors) that exposes most important features of the CloudScale Toolchain and CloudScale Method. The main focus of the integration is to make interconnection between the tools and usage of the tools as simple as possible.



The most important sections in the main perspective are (each is described in the following sections):

- 1) Project section with CloudScale Explorer and standard Project Explorer
- 2) Editor section with tools' editors
- 3) Status section with Workflow diagram and Status view

The main perspective provides toolbar, visible through each perspective and helps user with most common actions.



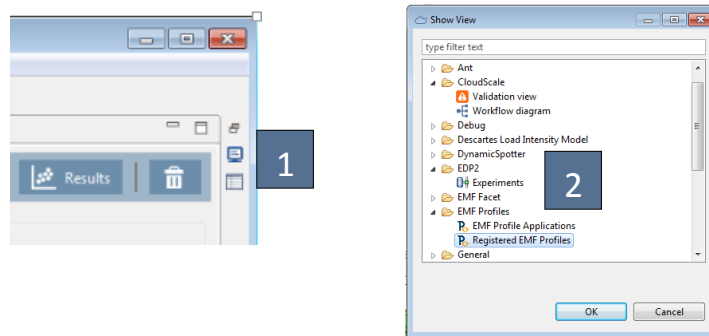
Actions available to users are:

- 1) Creating new CSE project and importing CSE examples
- 2) Saving current editor and saving changes in all editors
- 3) Switching perspectives; Main perspective, followed by the tools' perspectives (Analyser, Extractor, Static and Dynamic Spotters).

Additionally perspective provides 'Tools' menu in the main menu that provides common CSE and tools actions: opening RCP views, opening tools' views, dialogs and perspectives.

The section containing Properties view and console outputs are minimized [1]; it is possible to view it by clicking on the corresponding icons on the right side or by restoring entire

section. To access views that are not available by default in perspective one can open Views dialog [2] through “Main menu > Tools > Show View” action.



CloudScale Project

The core of the CloudScale Environment is CloudScale projects, each containing complete state of the analysed system's inputs, configurations and results. Entire state is persisted and relative to workspace therefor projects can be shared between different CSE instances (e.g. using version control).

The CloudScale project consists of tools' and ScaleDL model alternatives. Each tool alternative is composed of:

- **Input** contains information of the system needed by corresponding tool (e.g. PCM models for Analyser, source-code location for Extractor, etc.)
- **Configuration** contains information how to run the corresponding tool using specified input. The relation between input and configurations are one-to-many, meaning that each input has a set of dedicated configurations.
- **Result** contains results of the tool run. Information of which configuration has been used to produce specific result is persisted and it is possible to see all configuration results. The relation between configuration and result is also one-to-many.

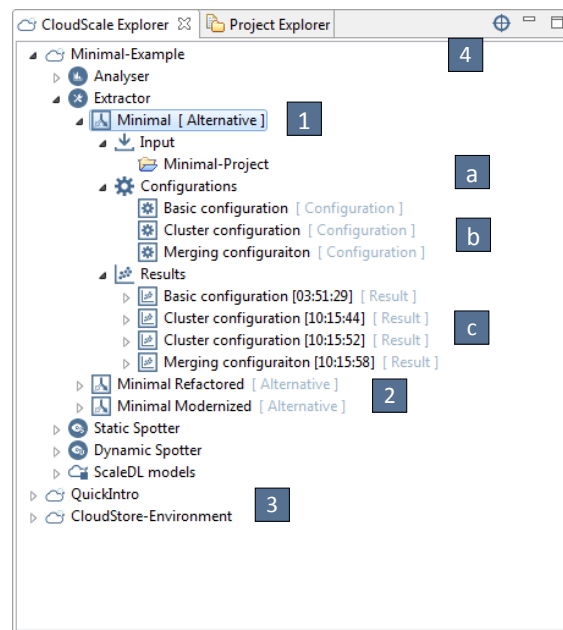
Contrary, each **ScaleDL model** alternative only contains models and diagrams representing an instance of ScaleDL model. Overview item stores also imported PCM models. Note: only Overview and Usage Evolution alternatives are kept here, since Architecture Template instances are integrated into the Environment and are directly linked in Analyser input.

Each item is represented as a folder on the file-system with corresponding properties file with meta information; also contains information regarding relations between alternative items. Folder contains all item's related files (e.g. PCM models, EDP2 results, Simulizar configurations). The CloudScale explorer tries to hide this complexity and show only the most important information and through associated editors provide easy way to manipulate this content. It takes hierarchical properties into account and shows relations between

alternative items, therefore we could say that CloudScale Explorer represents a high-level view of a CloudScale project data/content structure.

CloudScale Explorer

CloudScale Explorer provides hierarchical view of the Tools and ScaleDL models based on alternatives. Alternative [1] is a set of inputs and configurations with corresponding results, which provides findings of the system's scalability or it can be used as an input for linked tools. Each item can have many alternatives [2], which can be created through context menu (New -> Alternative).



There are two types of alternatives: tools and models. Tools alternatives are composed of on input [a], configurations [b] (how to use the input) and a set of results [c] got from configuration runs. On the other hand ScaleDL model alternatives contain one instance of corresponding model instance.

CloudScale Explorer can contain more CloudScale projects [3] and provides easy navigation between them. Each alternative or project can be deleted, renamed or copied through context menu. In addition "Reveal node" action in the view toolbar [4] can be used to reveal node with regards to currently focused editor.

Project explorer

In addition to CloudScale Explorer environment provides Project Explorer view that displays all projects available in workbench [1] using mirrored file system organization (i.e. files and folders).

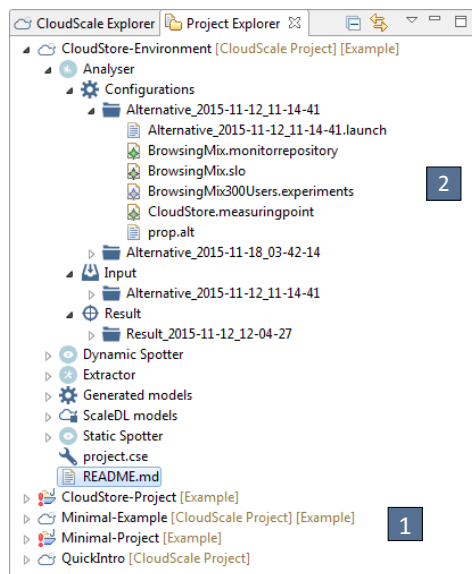


Image shows CloudScale Project representation in Project Explorer. Here one can find all files and folders composing CloudScale Project as they are represented on the file system; all alternatives and their content. For example, Analyser configuration alternative [2] is composed of Experiment model and dependent models (measuringpoint, slo). Properties file which defines common alternative attributes (name, location of files ...) and references to input alternative which is used when running configuration alternative.

Editors

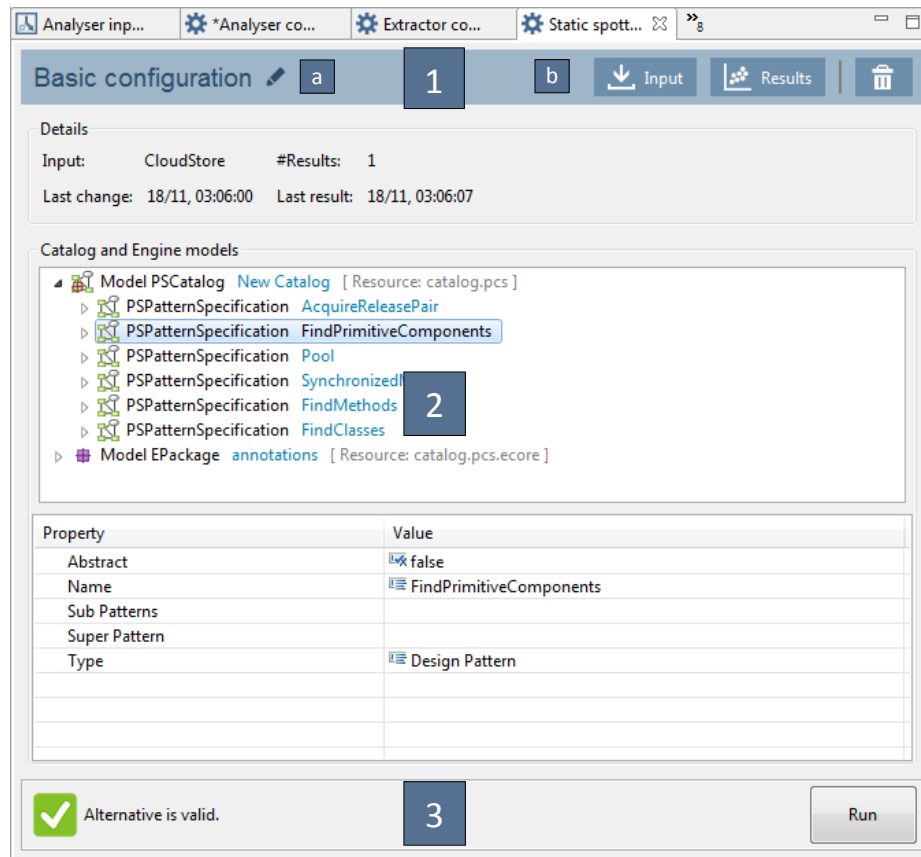
To provide unified and easy access to the integrated tools and models dedicated tools editors are available, allowing most of the actions needed to configure and run integrated tools. There are four different types of editors (input, configuration, result and model) and editors provide dedicated views for each of them.

Input editors define what the input is (e.g. complete PCM model for Analyser) and how to acquire it (e.g. output of the Overview transformation is input into the Analyser). Supporting different CloudScale Method paths, one can use the tool independently (i.e. define manual input) or connect other tool's output to its input.

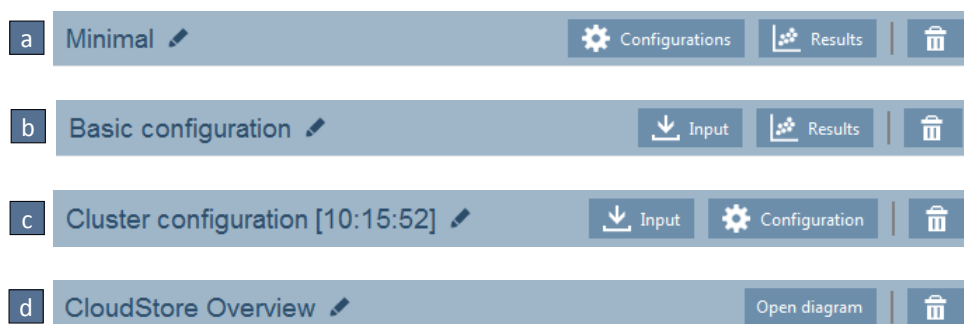
Configuration editors define what configurations are required and provide option to run the tool. Since all integrated tools contain their own run configuration user interfaces, the dashboard does not try to duplicate it, however it tries to provide easier access to the configurations, through pre-configuring what is possible, giving users options to create, delete and modify configurations and run them.

Results editors display all results from the tool runs. Since the tools also provide their own user interfaces the component shows which results are available, status of the runs and provide an option to open specific result in dedicated editor.

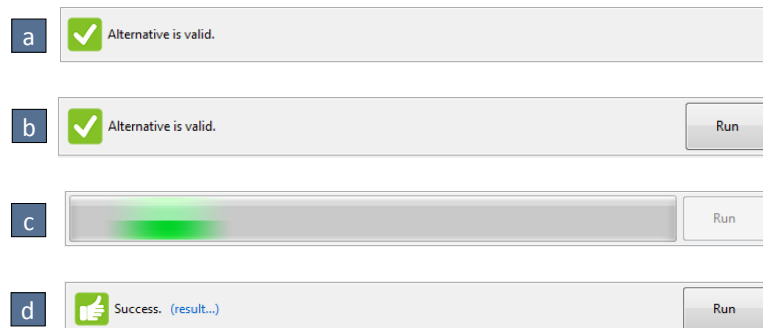
Model editors define ScaleDL model editors and enables easier model manipulation and access to diagrams. In the current version two ScaleDL models: Overview and Usage Evolution, has its own editor, while Architectural Templates are available through Analyser input editor.



Each editor view is composed of header [1], body [2] and footer [3]. Header [1] shows the name of the content displayed (alternative, configuration or result) [a] on the left and context aware actions [b] on the right (selecting depended alternatives, deleting alternative ...). Body provides components to view and edit alternative contents; each alternative has different view (for specific view see Tools sections). Footer [3] is displayed in tools' Input and ScaleDL model types and in tools' configuration types.



There is different header for each editor type: input [a], configuration [b], result [c], model [d]. Context actions allows users to navigate to depended editors (e.g. from input to all of its configurations and results). Name displayed on the right can also be changed by clicking on the edit button next to text or double clicking on the text itself.

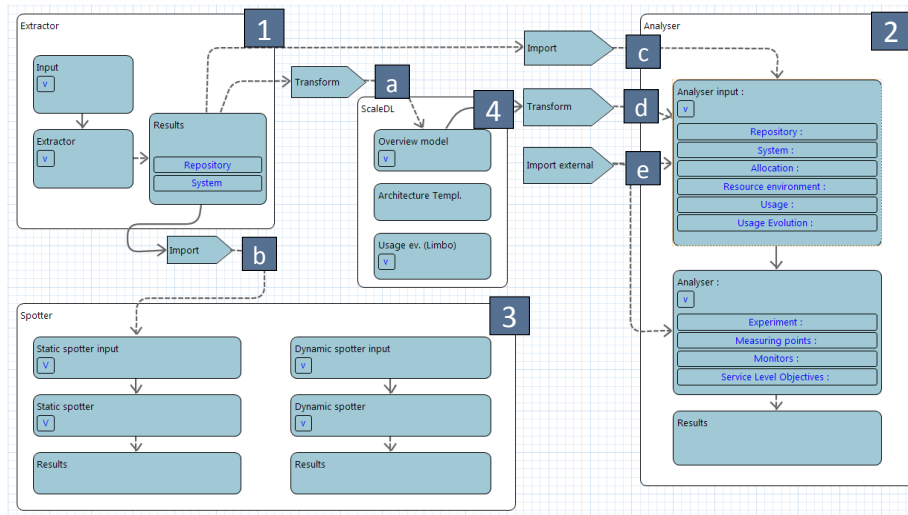


Footer is present in input, model and configuration editor types. Input and model editor type [a] contains validation component with alternative validation status. Clicking on it brings up the dialog listing validation warnings or errors. In addition, configuration editor type also provides functionality to run the alternative by clicking on “Run” button [b]. When running progress indicator is displayed [c] and after the run results component with run status is showed; if there is an error in run, user can bring up the dialog by clicking on the error message otherwise user can jump to corresponding result.

Changes made through the editors are saved only after executing save action (in toolbar, menu or keyboard shortcut) and can be discarded by closing the editor. On the other hand all changes made to the file-system are noticed and reloaded in editor: in case of conflict user is asked to make a decision which changes to keep.

Workflow

The Workflow diagram is a representation of CloudScale Method and helps user to navigate through CloudScale Toolchain and integration components. It represents all possible paths that user can take to analyse scalability properties of the system. Tools are connected using several actions (imports and transformations) which are highlighted only when all dependencies are met.



The diagram is composed of 4 main groups; Extractor [1], Analyser [2], Spotter [3] and ScaleDL [4]. Each group defines a set of states and how they are depended on each other. The tools groups contain three states; input, configuration and results, and these are analogous with the tools' alternatives. In addition these states also contain a toolbar with available actions (e.g. wizards, quick run) and a representation of; required resources in an input state, available run configurations in a run state and available results in a results state.

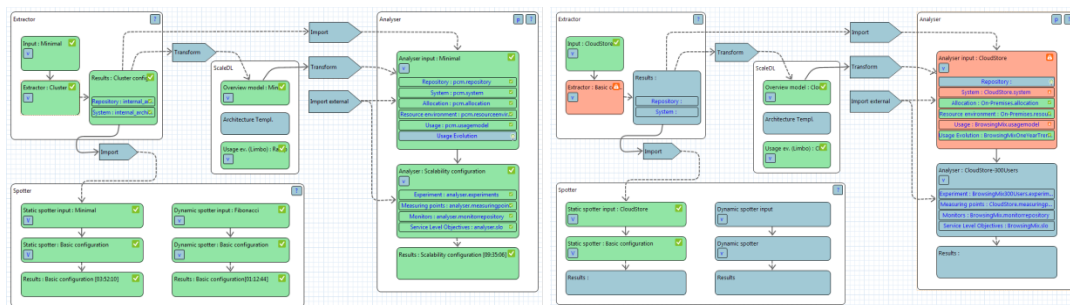
External boxes represents actions (import, transform) that can be executed to connect results of one tool to the input of another or import external resources into the corresponding tools.

- a) Transforming extractor result to new or existing ScaleDL Overview instance
- b) Importing extractor result into the Static Spotter input
- c) Importing extractor result into the Analyser input
- d) Transforming ScaleDL Overview instance to Analyser input (Extended PCM)
- e) Importing external projects into Analyser alternative (input and configuration)

Workflow diagram has two types of connections; required (solid) and optional (dashed). First defines that current state is active only when all required states are validated, while the latter one does not expect the optional state to be validated, however it normally represents tools interoperability and thus producing required resources quicker. Entry points, where user can start at the beginning are all the states that does not require another one to be fulfilled; tools' input states and all ScaleDL models.

Besides basic functionality Overview diagram also shows validation state of each alternative based on the current selection in the CloudScale Explorer or currently focused Tool editor. Each alternative (or resource) can be displayed in three different states:

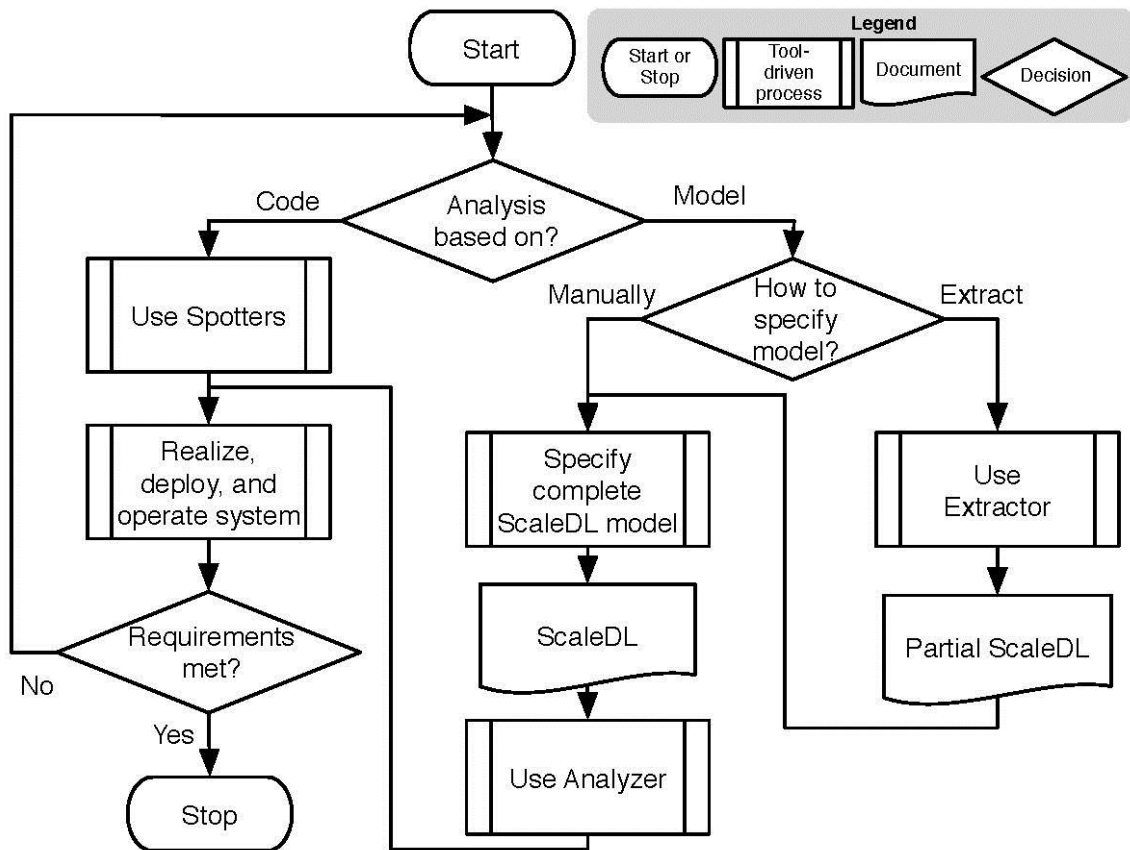
- Default state (blue) when corresponding alternative or resource is not yet available (created).
- Validated state (green) when validation was successful on co-responding alternative or resource.
- Warning state (orange) when validation on alternative or resource has failed; user can see problems by clicking on the warning icon.



Images shows two different states of the CloudScale Projects; left shows complete project with all items validated, while second shows partly completed project with validation errors in some alternatives.

CloudScale Method

CloudScale guides software engineers by the means of an engineering method to develop scalable, elastic, and cost efficient applications. The CloudScale Method defines the tool-supported design and analysis activities needed to engineer a scalable system as well as the order of their execution. It is a flexible method in the sense that it supports various scenarios which can be executed iteratively. Figure below gives a high level overview of the CloudScale method.



The first decision node branches based on the type of available artefacts. If code is already available, we are in a re-engineering scenario. In case no code is available or it should not be used, analyses can be based on an architectural model in a forward engineering scenario.

In reengineering scenarios, the analysis may be based either on static code or on a running application in the Spotters process step. Both re-engineering and forward-engineering scenarios alternatively support an analysis based on modelling the system's architecture, its usage by users, and its need for hardware and software resources. The resulting model is persisted in a special architectural modelling language, called ScaleDL. ScaleDL is comparable to a UML2 model with MARTE quality annotations, e.g., resource demands and workloads. The model can be either a complete new design, a refinement of an earlier system model, or a design which is partially extracted from existing implementations via the Extractor reverse engineering tool. ScaleDL models contain specifications of various application aspects ranging from implementation structure and behaviour, over the system's resource demands, to the system's usage, its change over time, and autonomous elasticity managers.

Independent of the analysis applied, once software engineers are happy with the analysis results, they realize, deploy, and operate their application. If during test cycles or during run-time new requirement violations arise, a new iteration of the CloudScale method needs to be executed.

Tools perspectives

Complete CloudScale tool-chain is available in the CloudScale Environment separately through dedicated tool's perspectives, which can be accessed through Tools menu or directly using toolbar buttons. From these perspectives tools can be used separately as a stand-alone products without the need of using Environment integration functionalities. For instructions how to use integrated tools separately, please see Tools tutorials.

To open other views (UI components) that may be missing in the menus, one can access them through "Tools>Show View" action (Alt+Shift+Q Q).

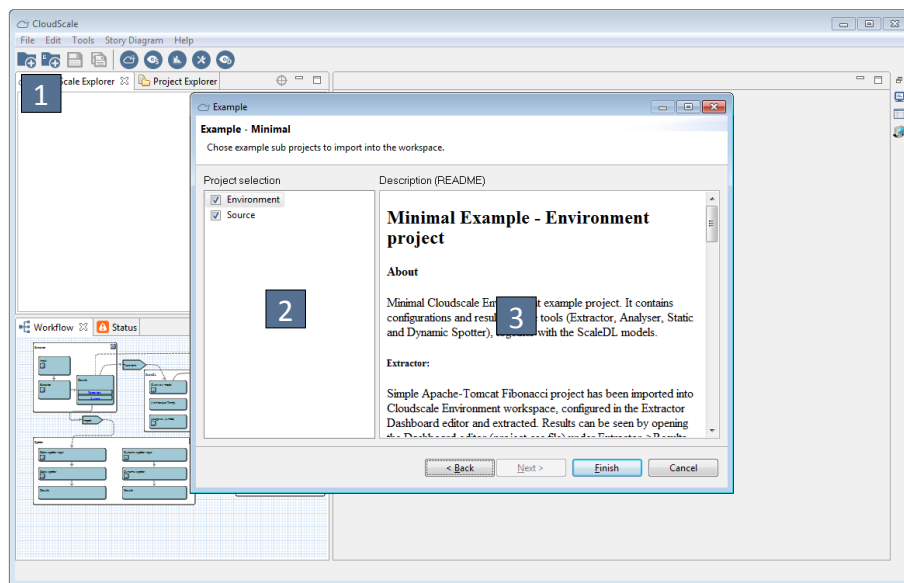
The CloudScale Environment is designed to be fully extensible, giving the possibility to the open-source community of improving and adapting it to its own needs. All the integrated tools (Palladio, SoMox, Reclipse and the SoPeCo) will be available from within the application through separate perspectives; however the main perspective (CloudScale perspective) will try to cover as much functionality as needed through dashboard and workflow components (see 4.2) to achieve a seamless and integrated user experience providing functionalities described before.

- **Analyser (Palladio)** - Palladio is a software architecture simulation framework, which analyses software at the model level for performance bottlenecks, scalability issues and reliability threats, and allows for a subsequent optimization. In the CloudScale Environment mainly the backend engines will be used to analyse the user's application. This will be achieved by automatically transforming ScaledL Overview models into the Palladio Component Models; structure on which Palladio operates.
- **Extractor (SoMoX)** - SoMoX provides clustering-based architecture reconstruction to recover the architecture of a software system from source code. The clustering mechanism extracts a software architecture based on source code metrics and construct PCM models to be used by Analyser.
- **Dynamic and Static Spotter** are tools for identifying ("spotting") scalability problems in Cloud applications. The tool consists of two main components that are provided as two separate programs: Static and Dynamic spotters. The Static Spotter analyses the application code to find scalability anti-patterns. The Dynamic Spotter analyses the execution of the application (through systematic measurements) to find scalability anti-patterns in the application's behaviour. Within this document we will refer to the pair as Spotter, and to their individual components as either Dynamic or Static Spotter.

Examples

CloudScale integrates two examples (Minimal and CloudStore) to show how the toolchain can be used. Each example contains configurations for all the tools along with results.

To import an example open Example wizard using toolbar action [1]. In the first step select an example to be imported (Minimal or CloudStore) and in the second step choose which projects to import (Environment and/or source) [2]. For each selection quick readme is available [3].



Minimal

Minimal example is an open-source project providing a minimalistic web-service that is easy to run and update, created for the purpose of showcasing main functionality of the CloudScale toolchain.

Web-service provides an interface exposing two functions; testOK, testOLB. First is simple and does not contain any anti-pattern, while second produces OLB anti-pattern which can be found by the tools.

To run Minimal example run startup script available in source project dist/ folder. You can also update and re-compile it using maven install command; `mvn install`.

Minimal CloudScale Environment example project contains configurations and results for the complete tool-chain (Extractor, Analyser, Static and Dynamic Spotter), together with the ScaleDL models.

CloudStore

CloudStore is an open-source e-commerce web application developed following the functional requirements defined by the TPC-W standard, and its goal is to be used for the analysis of cloud characteristics of systems, such as capacity, scalability, elasticity and efficiency. It was developed as a Showcase application to validate the CloudScale tools that were developed during the project.

The application was developed in Java using the Spring framework and running on a Tomcat web application server, while using a MySQL database. Static files and images need to be generated with ImgGen tool, but a default database dump and load generation scripts are available in order to start testing a deployment in a very short time.

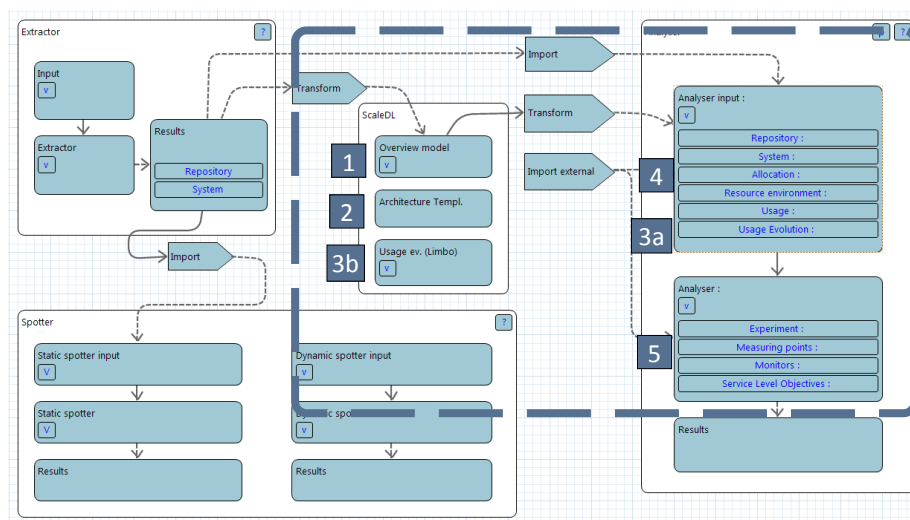
To run CloudStore follow instructions on the GitHub pages. There are deployment scripts available to deploy it into different cloud environments.

The CloudStore example project contains configurations and results for the complete tool-chain (Extractor, Analyser, Static and Dynamic Spotter), together with the ScaleDL models.

ScaleDL

The Scalability Description Language (ScaleDL) is a language to characterize cloud-based systems, with a focus on scalability properties. ScaleDL consists of five sub-languages:

- ScaleDL Overview,
- ScaleDL Architectural Template, and
- ScaleDL Usage Evolution [3a] with Descartes Load Intensity Model [3b]
- Palladio's PCM [4]
- Simulizar's self-adaption language (Experiments model) [5]



In this section each ScaleDL sub-model is described, including usage on the Minimal example.

Extended Palladio Component Model (Extended PCM)

The Extended PCM allows architects to model the internals of services. The Extended PCM consists of the PCM (to model components, components' assembly to a system, hardware resources, components' allocation to these resources, and static usage scenarios) and an extension that additionally allows to model self-adaptations (monitoring specifications, service level objectives, and adaptation rules).

Relation to Workflow

An instance of the Extended Palladio Component Model (Extended PCM) is part of a ScaleDL instance, the input of the Analyzer (see [Workflow](#)).

Features

A PCM instance allows software architects to model the internals of services. The PCM allows to model components, components' assembly to a system, hardware resources, components' allocation to these resources, and static usage scenarios.

The “extended” refers to additional models we added in the CloudScale context. These models cover monitoring specifications, service level objectives, and self-adaptation rules. Monitoring specifications allow one to mark PCM elements, e.g., an operation of a component, to be monitored during analysis using a metric such as response time. Service level objectives specify thresholds for these metrics, allowing software architects to manifest their quality-related requirements. Self-adaptation rules can react on changes of monitored values. For example, when a certain response time threshold is exceeded, an adaptation rule could trigger a scaling out of bottleneck components.

Input & Output

Creating Extended PCM instances is mainly based on the input of the software architects that create these instances. Software architects can, however, be supported by several workflow actions:

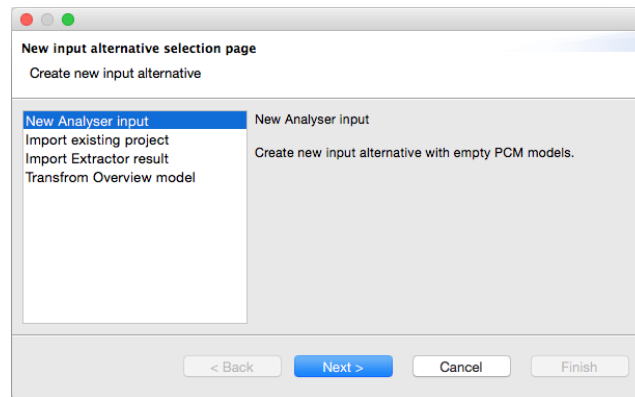
- Import a partial Extended PCM instances from the Extractor, i.e., reverse engineer a partial Extended PCM instances from source code (see [Extractor](#)).
- Import an already existing Analyzer input project or entire experiment project (including configuration alternative)
- Transform a ScaleDL overview model (second input arrow; left). Especially if Architectural Templates (see [Architectural Templates \(ATs\)](#)) are used, such a transformation creates a large part of the Extended PCM.

The main output for specifying an Extended PCM is the Extended PCM instance itself. It can be used for documenting a system's architecture and is part of a ScaleDL instance; the Analyser input (see [Analyser](#)).

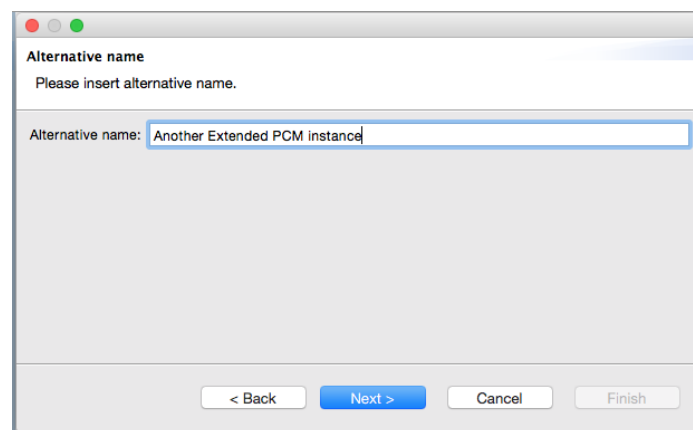
Minimal Example

To create an Extended PCM instance, software architects conduct the following actions:

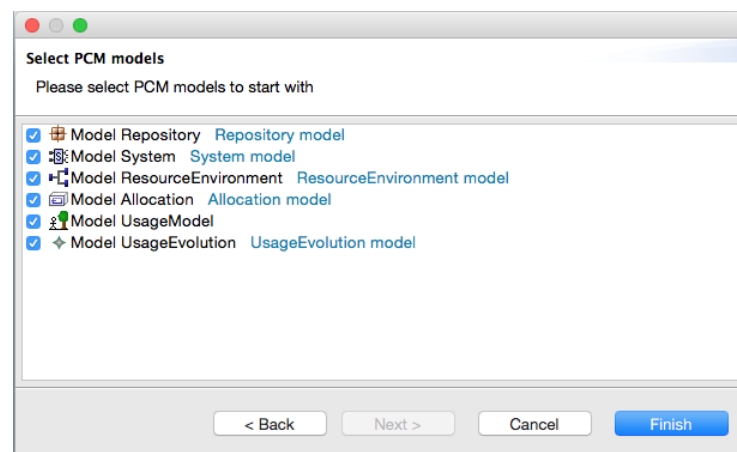
1. Go to the “Input” tab of the “Analyser” category in the dashboard. If you use Analyser's minimal example, you already see pre-specified Extended PCM instances (“Fibonacci from Overview” and “Fibonacci from Extractor”). By pressing “Create”, a wizard opens that guides through various creation options. We select “New Analyser input” to create another Extended PCM from scratch and hit “Next >”.



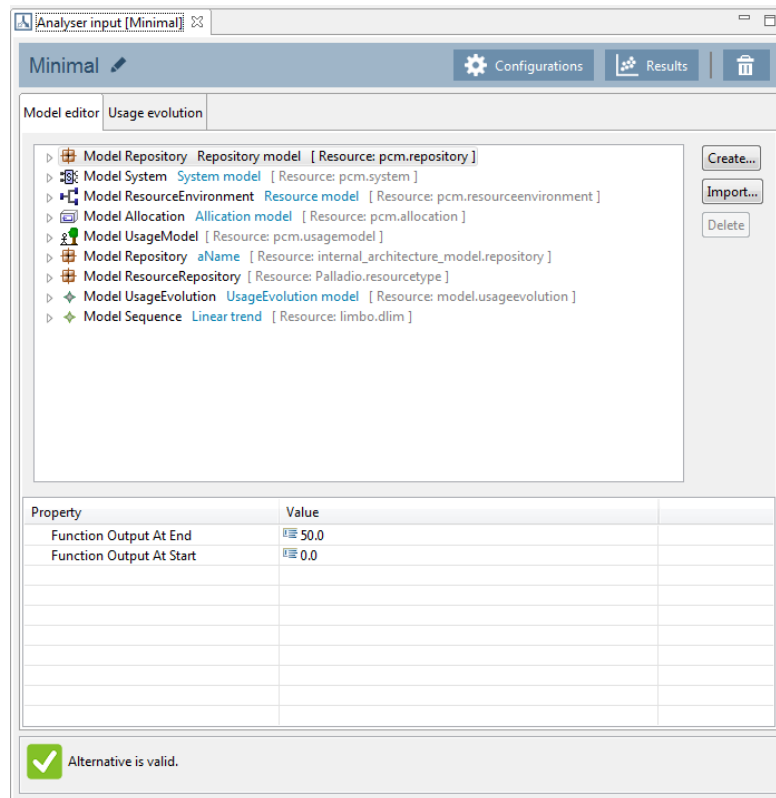
2. On the next wizard page, we enter "Another Extended PCM instance" as alternative name and hit "Next >".



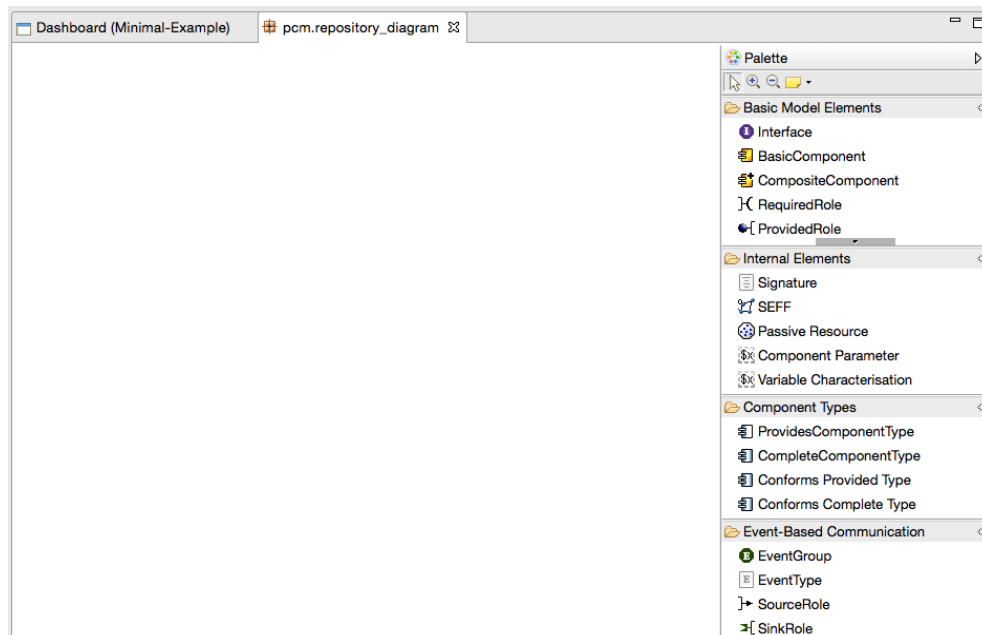
3. We'd like to specify a complete instance, therefore, leave every sub model checked and hit "Finish".



4. We see that we successfully created another instance. Each sub model (e.g., “Model Repository”) is listed in the “Model editor” tab.

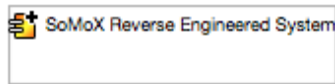


5. Double-clicking any the “Model Repository” sub model. We are provided with a dedicated editor that allows us to edit repository models from scratch.

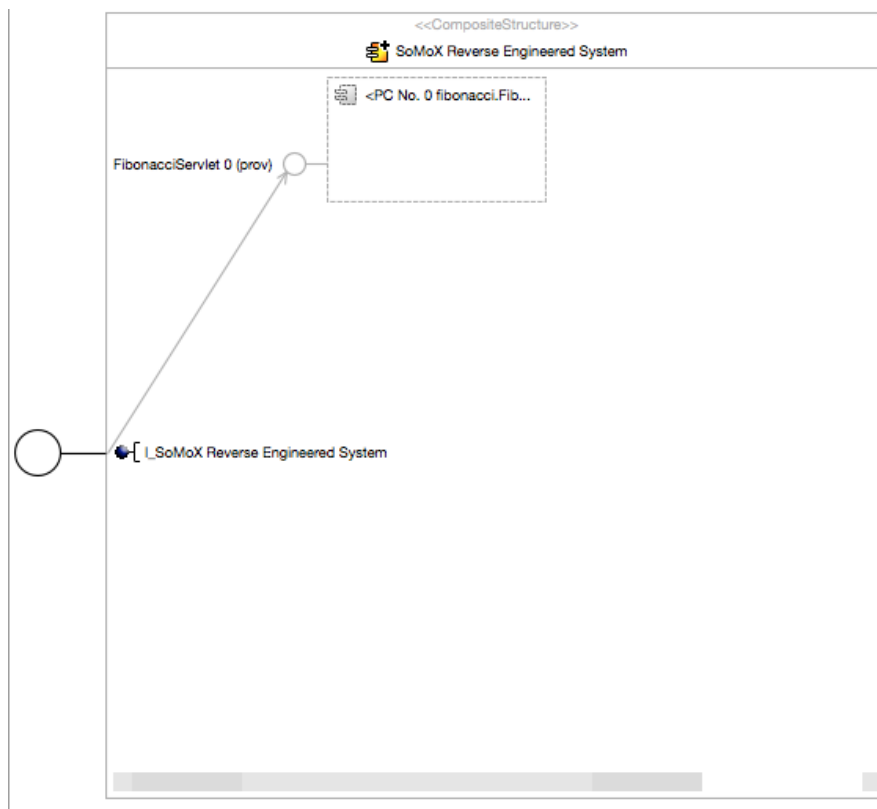


6. Close this editor again (by pressing the “X” of the “pcm.repository_diagram” tab) to come back to the dashboard. Switch to the “Fibonacci from Overview” Extended

PCM instance and double-click the “Model Repository” there. You will see one composite component “SoMoX Reverse Engineered System”.

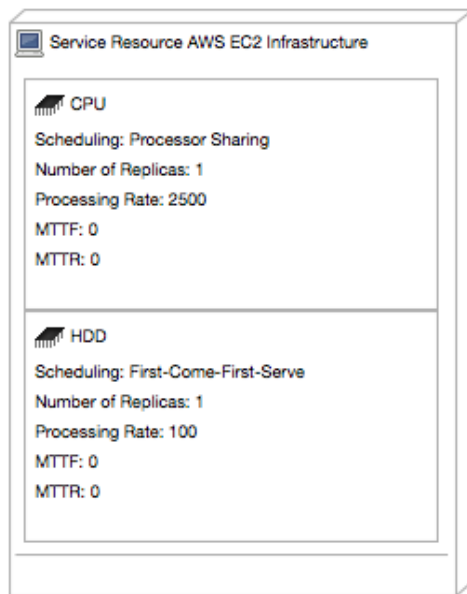


7. Double-click this component to investigate the composed structure of the composite component. You will see that it contains one inner basic component (“<PC No. 0 Fibonacci...”).



8. Close these diagrams again and open the “Model ResourceEnvironment” from the “Fibonacci from Overview” Extended PCM instance. You will see that the resource environment includes one resource container for the “Service Resource AWS EC2

Infrastructure”.



9. Other models can be investigated analogously. The respective editors also provide full editing support. Detailed explanations for editing Extended PCM instances can be found in the [Palladio Workshop white paper](#) and the [Analyser screencast series](#).

Architectural Templates (ATs)

ScaleDL Architectural Template allows architects to model systems based on best practices as well as to reuse scalability, elasticity, and cost-efficiency models specified by architectural template engineers.

Relation to Workflow

Architectural Templates (ATs) are part of ScaleDL, the input of the Analyzer (see [Workflow](#)). Also the transformation from “Overview model” to “Analyzer input” utilizes ATs to simplify the transformation specification.

Features

Architectural Templates (ATs) help software architects to specify Extended PCM instances more efficiently based on reusable ScaleDL templates. We provide the [CloudScale AT catalogue](#) that includes best practice templates for designing and analysing scalable, elastic, and cost-efficient SaaS applications. We based this catalogue on common architectural styles and architectural patterns found in cloud computing environments.

Input & Output

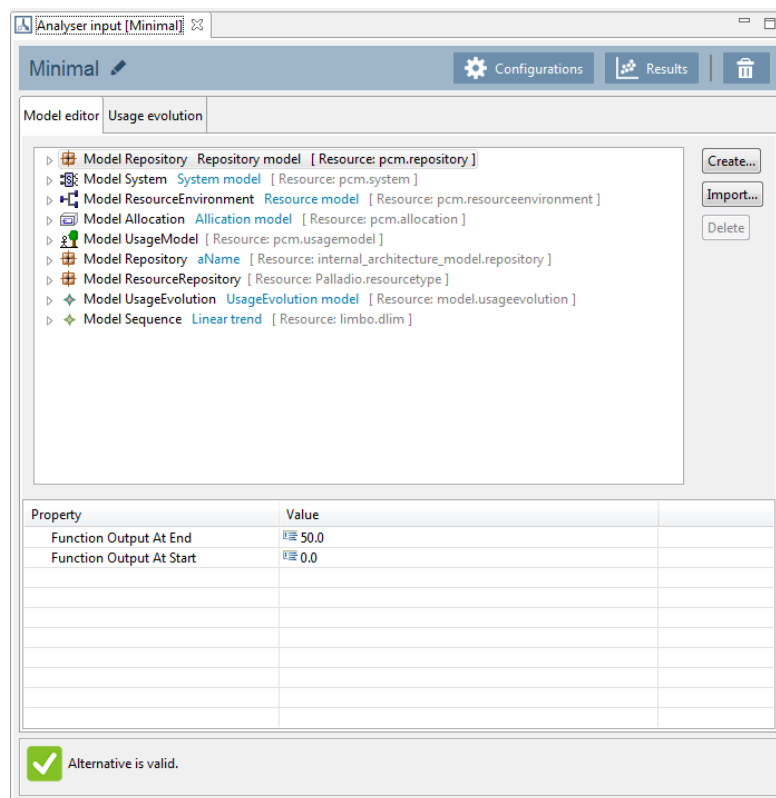
For an AT-based design, software architects need a catalogue of available ATs as an input. For cloud computing applications, we suggest the [CloudScale AT catalogue](#).

The output of an AT-based design is part of a normal ScaleDL instance. The [Analyser](#) fully supports AT-enabled ScaleDL instances.

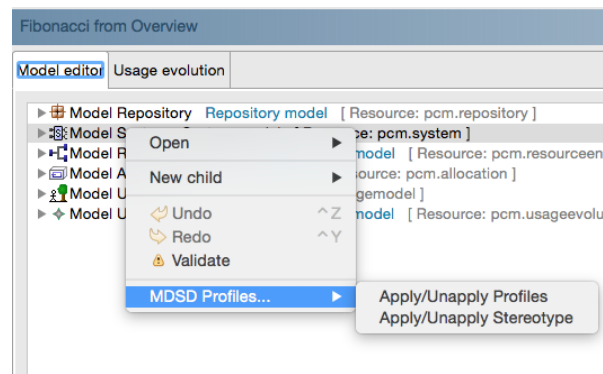
Walkthrough

Software architects conduct the following actions for designing ScaleDL instances with ATs:

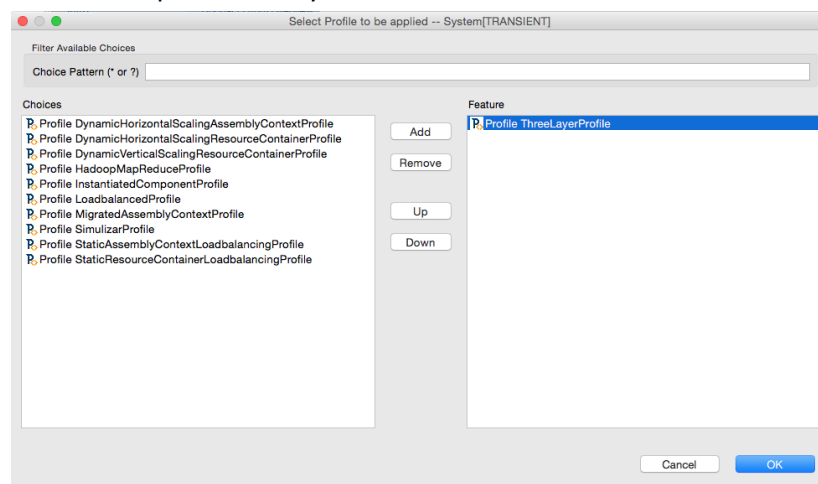
1. Go to the “Input” tab of the “Analyser” category in the dashboard. If you use Analyser’s minimal example, you already see pre-specified Extended PCM instances (“Fibonacci from Overview” and “Fibonacci from Extractor”). We chose the “Fibonacci from Overview” Extended PCM instance.



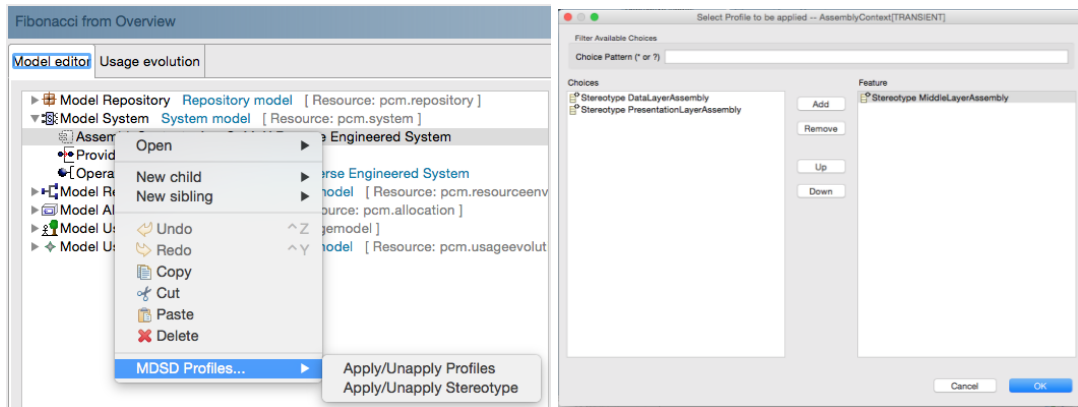
- Right-click on the “Model System” and choose “MDSD Profiles... -> Apply/Unapply Profiles”.



- In the new dialog, select the AT you want to apply and apply it. By default, all ATs of the CloudScale AT catalogue are available. If you want to use third-party AT catalogues, assure that you installed the respective plugins; included ATs will then also be shown in this dialog. We selected the simple “ThreeLayerProfile” AT and confirmed with “OK”.



- Assign all AT roles to respective elements of your Extended PCM instance. For example, if an AT specifies an AT role for PCM’s assembly contexts, right-click this assembly context within the system model tree editor and select “MDSD Profiles... -> Apply Stereotype” and apply the role similar to the previous action. Our [CloudScale AT catalogue](#) describes, per AT, which roles have to be assigned. In the example below, we assign the “MiddleLayerAssembly” to the assembly context of the previously investigated system.



5. As each AT role can have a set parameters, software architects also need to set such parameters when requested. If needed, these parameters can simply be set via the Properties View for elements that have an AT role assigned.

Usage Evolution (UE)

The purpose of the ScaleDL Usage Evolution is to formally specify (in a model) how the load and work of a system evolves over time. This model can subsequently be used by tools that generate load and analyse elasticity and efficiency of cloud computing.

The actual modelling of the evolution (e.g. arrival rates or population in load or average video size in work) is done using the [LIMBO Load Intensity Modelling Tool](#). In addition, Usage Evolution provides a thin modelling layer that couples the model of evolution of a set of variables (as described in LIMBO) with a scenario model expressed in the [Palladio Component Model \(PCM\)](#).

For details of using the Palladio and LIMBO see the [PCM chapter](#) of this document, the [Usage Evolution screencast](#), and the [tutorials](#) and [screencasts](#) on the Palladio home pages.

Relation to Workflow

An instance of the Usage Evolution is part of a ScaleDL instance and is defined under the ScaleDL step in the workflow. It is an input to the Analyzer (see [Workflow](#)).

Input & Output

As input, the Usage Evolution requires LIMBO model(s) and Palladio Component model(s) containing usage scenario(s).

Usage Evolution contains a list of usage elements each referring to a PCM usage scenario. Each usage element can contain a reference to a LIMBO model that describes the evolution of arrival rates or population. Additionally, a usage element can specify a coupling to LIMBO models for a set of work parameters.

The output of the Usage Evolution, that can describe both evolution of work and load, is part of a ScaledDL instance which is used as the input for the Analyser (see [Analyser](#)).

Walkthrough

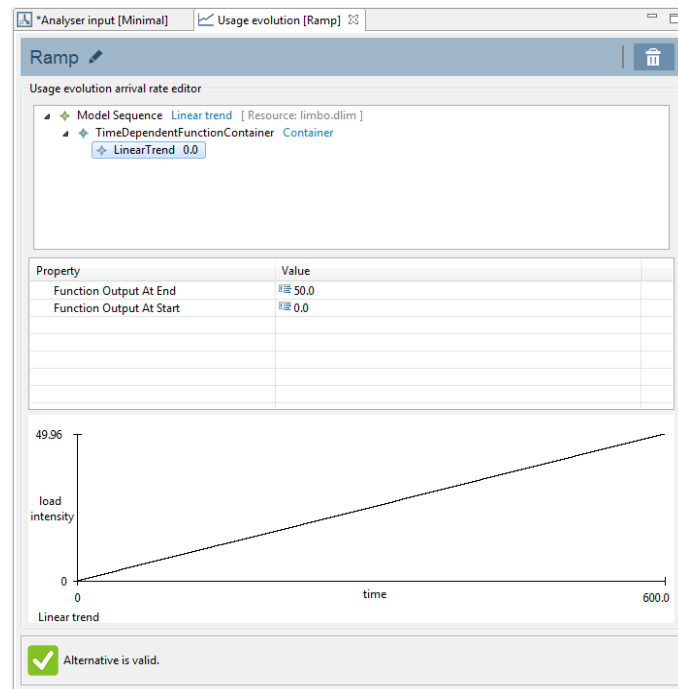
The following is an overview of the steps for creating a usage evolution:

1. Create LIMBO models to specify how load and selected work parameters (e.g. the amount of data to process) evolve over time.
2. Create a Usage Evolution model that couples the LIMBO models to load and work parameters for a selected Palladio usage scenario.

Each of the steps is described in more detail in the sub-sections below. For details on how to create a run configuration and how to execute the simulation, see the [Analyser](#) section.

Creating LIMBO models

The first step is to create LIMBO models of how the load and selected work parameters from the PCM model evolve over time. LIMBO models can be created from the LIMBO tab of the Dashboard. From the dialog that appears when selecting to create a LIMBO model, it is possible to create either an empty LIMBO model, a model based on a template, or to select a more advanced setup through a wizard. (Note that the wizard interface cannot be used to edit the model after it has first been created.)



The LIMBO tool provides a tree-based editor as shown in the figure. At the root of the model is a Sequence which typically contains one or more TimeDependentFunctionContainer that can be regarded as segments of the sequence. Each segment has a duration that adds to the total duration of the Sequence. When using multiple LIMBO models (to describe work parameters as well as load) the total duration of each Sequence must be the same.

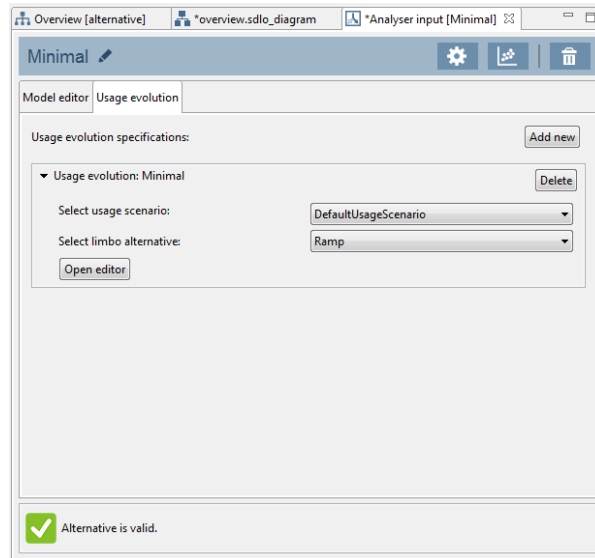
Each segment contains a single function (e.g. linear, exponential, logarithmic or sin trend) describing how the value evolves within that segment, and typically contains a start and end value (see e.g. properties for LinearTrend in the figure).

Note that the LIMBO tool originally was developed with focus on arrival time, and thus uses “arrival time” in some of the labels (e.g. the y-axis in the graph). However the same modelling concept can be used to express the evolution over time of other values such as population in closed workload or work parameters such as average data size (e.g. video or picture size).

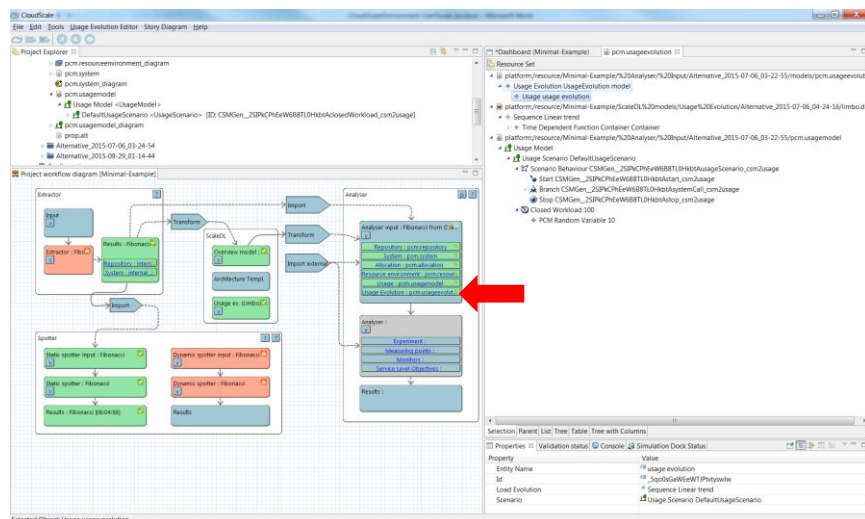
See the [LIMBO web pages](#) for more documentation on the tool.

Creating the Usage Evolution model

Usage Evolution models can be created from the Analyzer tab of the Dashboard by selecting Input from the Main menu. As shown in the figure below, new usage evolution specifications can be added and defined by selecting usage scenario and LIMBO model.



By using the ScaledDL Usage Evolution tree-based structure editor more details of the usage evolution model can be accessed. The usage evolution editor is opened by double clicking on the Usage Evolution link in the workflow diagram as illustrated below.



At the root of the model is a single usage evolution element. To add content to the model, select the root element, and select “New Child / Usage” from the context menu to create a Usage. Then, select the newly created Usage element - its properties will be shown in the Property editor tab at the bottom of the screen.

To reference elements from the PCM usage model and LIMBO models in the usage evolution editor, they must first be imported into the usage evolution editor. This is done by dragging the files into the editing area of the usage evolution editor (or use “Load resource...” from the context menu to

locate them). The PCM usage model file is found under the subfolder of your selected input alternative for the Analyser, while the LIMBO models are found under the LIMBO subfolder of the ScaledDL models.

Once the files have been loaded, make sure the Usage element in the editor is selected. Click on the Load evolution property, and select the sequence from the LIMBO model that appears in the list. Then, click on the Scenario property and select the usage scenario from Palladio that appears in the list. Remember to save the model once the edits have been completed.

If there is any work parameters in your model that you want to describe evolution of, add one Work Parameter Evolution child to the Usage for each of them. From the Evolution property select the LIMBO sequence that describe the work parameter evolution (remember that each LIMBO model must first be imported into your model). Set the Variable Characterization property to point to the variable from the PCM model to evolve.

In some cases it is useful to create more than one usage scenarios and/or usage evolution descriptions to analyse different uses of the systems or how it will evolve in different future scenarios. Such alternative inputs can added from the Input tab for the Analyser tab.

Overview

The ScaledDL Overview is a meta-model that provides a design-oriented modelling language for cloud-based system architectures and deployments. It has been designed with the purpose of modelling cloud based systems from the perspectives of service deployments, dependencies, performance, costs and usage. It provides the possibility of modelling private, public and hybrid cloud solutions, as well as systems running on non-elastic infrastructures.

Steps needed to completely define a system in the ScaledDL Overview are: creating system architecture, defining application view, specifying system services and defining services deployment. Therefore the system model is defined through 4 views:

- **Architecture View** provides a descriptive abstraction of the system's architecture, defined using cloud layers (i.e. infrastructure, platform and software layer), consisting of services as a basic building blocks. Service solution stack (i.e. vertically dependent services) is defined through deployment dependencies; referencing the service needed on the lower layer (i.e. infrastructure layer) for successfully deploying services on an upper layer (i.e. platform layer). The dependencies between service solution stacks are further defined through use of connection elements, representing network connectivity between service stacks. The connectivity with external environment is defined through UsageProxy for exposing services and ExternalSoftwareServices for referencing Software-As-A-Service (SaaS) services.

- **Application View** provides a high-level definition of the system's application, through services exposing operation interfaces (software and support platform services), the dependencies between them (provided and required interfaces). The operation interfaces exposed to external environment are defined through usage proxies, while using operation interfaces provided by external services (i.e. SaaS) through external proxies.
- **Specification View** describes service information, performance (SLOs) and costs. Additionally ScaleDL Overview meta-model contains Cloud Definition, defining services available in specific cloud environment and Platform Service Definition, defining deployable platform services (e.g. databases, application servers, etc.) that are deployable in the environments providing Infrastructure-As-A-Service (IaaS). Both definitions can be defined in advance and used when constructing ScaleDL Overview models.
- **Deployment View** provides information of how the services are deployed, their runtime configuration and also monitoring results. For the infrastructure and provided platform services it defines runtime and generic (i.e. black-box) deployments. First describes the elastic properties of the service deployment while for the later the expected performance and SLOs of external services.

Relation to Workflow

An instance of the Overview model is a part of a ScaleDL instance. It can be generated from the Extractor result, partial PCM model, or directly by creating a new Overview alternative (through CloudScale Explorer). Overview model, when completed, can be transformed to the input model for the Analyser (see [Extended Palladio Component Model](#)).

Input & Output

ScaleDL Overview is modelled using dedicated editor and diagram. The model itself does not depend on other input therefore user can create new empty Overview instance through CloudScale Explorer. In modelling phase one can use all services specified in specification view. These can be improved by providing additional specification plugins which can be easily added to the CSE.

In addition, the CloudScale Environment provides functionality to transform Extractor results (i.e. Partial PCM) into the Overview instance using import wizard. It produces new Overview model instance or creates corresponding services in existing instance.

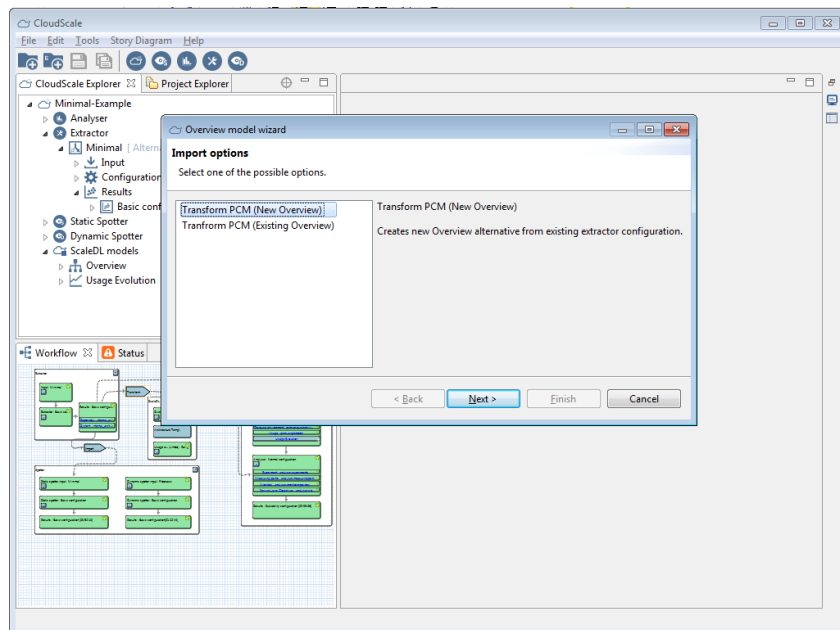
Following the CloudScale Method, the CSE provides functionality to transform Overview instance into the PCM used as a Analyser input alternative. Incorporating Extractor results and service effects specifications this can provide complete PCM model.

Overview provides deployment view which also specifies how services are scaled in the cloud environment. Using this information Architectural Template is also applied to transformed models.

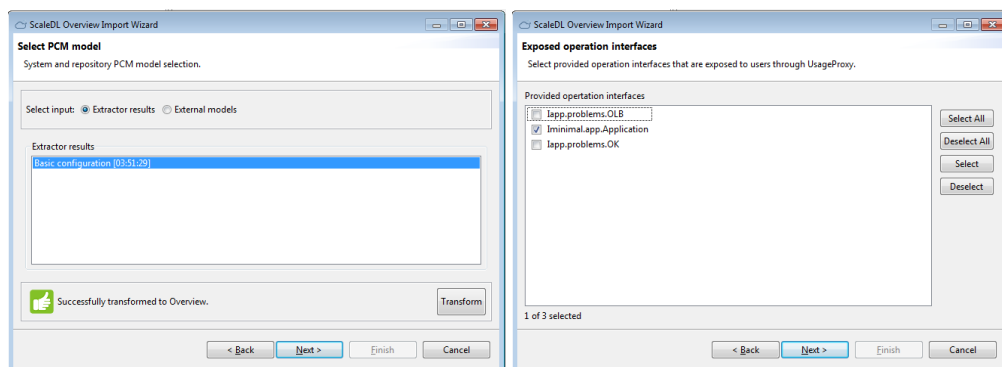
Minimal Example

In Minimal Example we will use Extractor result as a starting point. From it we will create new Overview model using Import Wizard, finalize it. At the end we will transform final model to the PCM forming new Analyser input.

1. Create a new overview alternative using Extractor result. Click on the “Import” action in the Workflow connecting Extractor Result with the Overview node.



2. In the wizard select Extractor result (i.e. partial PCM) to be transformed to Overview software service and click “Transform” button. After successful transformation click “Next” and select exposed interfaces (i.e. Interfaces available externally, to users). In case of Minimal application exposed interface is Iminimal.app.Application.



3. Specify deployment configuration of extracted service

- a) Cloud environment in which it is deployed (e.g. AWS EC2) and provider type (IaaS or PaaS).
- b) Type of deployment: Single, Clustered or Replicable
- c) Computing resource: instance type
- d) Load balancing configuration : initial cluster size and load-balancer type
- e) Scaling configuration: minimum and maximum number of instances

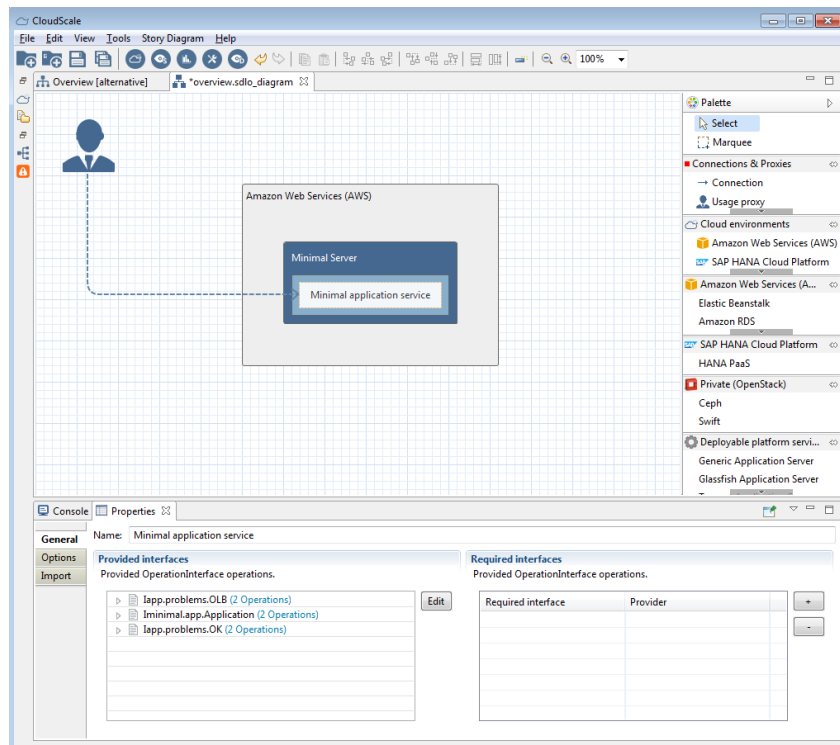
The screenshot shows the 'ScaleDL Overview Import Wizard' window, specifically the 'Basic deployment configuration' step. The window title is 'ScaleDL Overview Import Wizard'. Below the title bar, the section 'Basic deployment configuration' is displayed, followed by the instruction 'Select cloud environment and software deployment.'.

The configuration is organized into several sections:

- Runtime Platform Service:** Radio buttons for 'Create new' (selected) and 'Select existing'.
- Cloud environment:**
 - Providers selection:** Radio buttons for 'IaaS' (selected) and 'PaaS'.
 - Cloud Environment:** A dropdown menu showing 'Amazon Web Services (AWS)'.
 - Platform service:** A dropdown menu showing 'Generic Application Server'.
- Deployment configuration:**
 - Radio buttons for 'Single', 'Clustered', and 'Replicable' (selected).
 - Resource configuration:**
 - Computing Resource:** A dropdown menu showing 'm1.large'.
 - CPU:** 1200 MHz
 - Cores:** 4
 - Memory:** 7500 MB
 - Storage:** 850 GB
 - Loadbalancing configuration:**
 - Initial size:** 2
 - Load balancer:** A dropdown menu showing 'ROUND ROBIN'.
 - Scaling configuration:**
 - Minimum:** 1
 - Maximum:** 4
 - A button labeled 'Create scaling policy ...'.

At the bottom of the window, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

3. After successful import a new ScaleDL Overview alternative has been created containing model instance and diagram. One can modify model further by introducing new services or even cloud environments. In Minimal example we will only change titles of application service and server. You can do this by using property view, which is minimized by default on the right side (click restore button to place property view in the workspace).



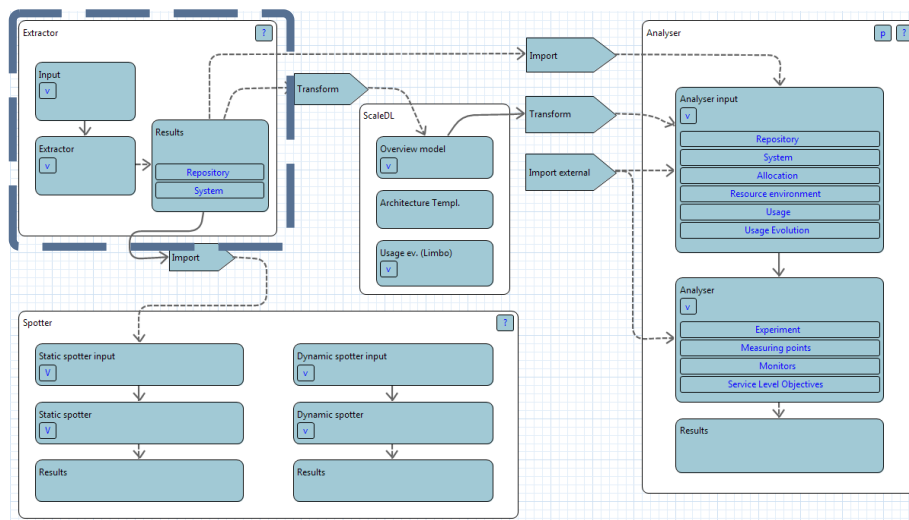
4. Transform Overview instance to the Extended PCM including Repository, System, Allocation, ResourceEnvironment and Usage model. Transformation can be triggered using workflow action “Transform” pointing from Overview node to Analyser Input. Architectural Templates are already applied on the transformed models; transformation is based on the Overview deployment configuration.

Extractor tool

Introduction

The Extractor is a reverse engineering tool for automatic model extraction. It parses source code and generates partial ScaledDL models that are further used by the Analyser and Static Spotter.

Relation to Workflow



The Extractor is an initial tool to be used in reengineering scenarios, when source code is already available and the goal is to improve system's scalability and predict system behavior under different configurations and work-load. Extractor uses Eclipse source project as an input with additional settings which parts of code to include or exclude from the extraction. As a result it produces partial ScaledDL model directly usable by Static Spotter or as an initial point for Analyser input. Additionally, to speed up forward-engineering process, result can be transformed to an Overview model and further transformed into almost complete Analyser input.

Problem

There are situations for a software system that the original architecture has been lost or outdated due to further development. The software architect has no detailed picture of what the software system's architecture looks like but only the source code.

While the software architect would require the architecture when migrating the software system to a new environment such as cloud computing environment.

In CloudScale, the Extractor supports the initiate step of software migration, namely to extract the software architecture from the source code.

Features

The Extractor is a tool-supported re-engineering process that combines different reverse engineering approaches to enable an iterative recovery and re-engineering of component-based software architectures. The Extractor is focused on parsing and architecture reconstruction.

The Extractor parses the source code, extracts and clusters a component-based software architecture based on source code metrics.

Usage

Input

Extractor takes Eclipse Java Projects as an input. Project needs to be imported into the CSE workspace through “Import ...” in main menu or in Project Explorer context menu.

Eclipse project with Java nature can be quickly created by adding default .eclipse file into the root project folder.

NOTE: For best results project must be configured correctly; classpath and build specifications needs to be configured through project properties in the CSE. Use Eclipse IDE documentation to configure these accordingly.

```
<projectDescription>
  <name>My Project</name>
  <comment>...</comment>
  <projects/>
    <buildSpec> </buildSpec>
    <natures>
      <nature>
        org.eclipse.jdt.core.Javanature
      </nature>
    </natures>
</projectDescription>
```

After project has been successfully imported and is available in the Project Explorer, it is possible to use it as an Extractor input. Additionally, Extractor can be further configured how to use this project. Following properties are available:

- `EXCLUDED_ELEMENTS_REGEX` and `INCLUDED_ELEMENTS_REGEX` : allows to include and exclude elements based on fully qualified class names using regex expressions. Eg. `eu.cloudscale.exmples.minimal.app.*`
- `REFERSH_SOURCE_BEFORE` : refresh source tree before executing parsing step.
- `SERIALIZE_TARGET` and `DEEP_ANALYSIS` must be set to true, otherwise second step, model extraction, will not work.

Configuration

For the identification of components, that could be merged, a number of source code metrics are computed. The overall score of these metrics is evaluated for two components at a time where an overall score is influenced by weighting each metric with a probability (an integer value between 0-100). These probabilities can be configured through Extractor configuration editor.

Clustering metrics

To control the merging of components two thresholds (composition, merging) can be given. Only those components, whose score lies in the stated range (given by minimum and maximum threshold) will be considered to be merged.

Additional metrics

Package mapping: Indicator if the components can be mapped to a single package

Directory mapping: Indicator if the components can be mapped to a single directory

Abstractness: This metric measures the object-oriented abstractness of a component.

Instability: Instability characterizes the dependency of a component to others. Components with an instability value of "1" are considered to be instable, because they depend in a very high degree on other components.

Distance from Main Sequence (DMS) : This metric combines abstractness and instability to rate components.

Coupling: Determines the coupling from component A to B. This metric does not influence the overall score directly but is considered for the evaluation of Interface violation and Name resemblance

Interface violation: Measure of interface communication between two components.

Name resemblance: Ratio of same class names of a component and the total number of classes.

Sliced Layer Architecture Quality (SLAQ): The Sliced-Layer-Architecture assumes that the source code is organized in layers (technology orientated) and slices (service orientated). With this structure natural subsystems can be identified.

Subsystem component metric: Checks if all a components can be mapped to a natural system

For additional information see [SoMoX documentation](#).

Result

Extractor result is a partial PCM instance containing System and Repository models, and a source-decorator model that can be used by StaticSpotter.

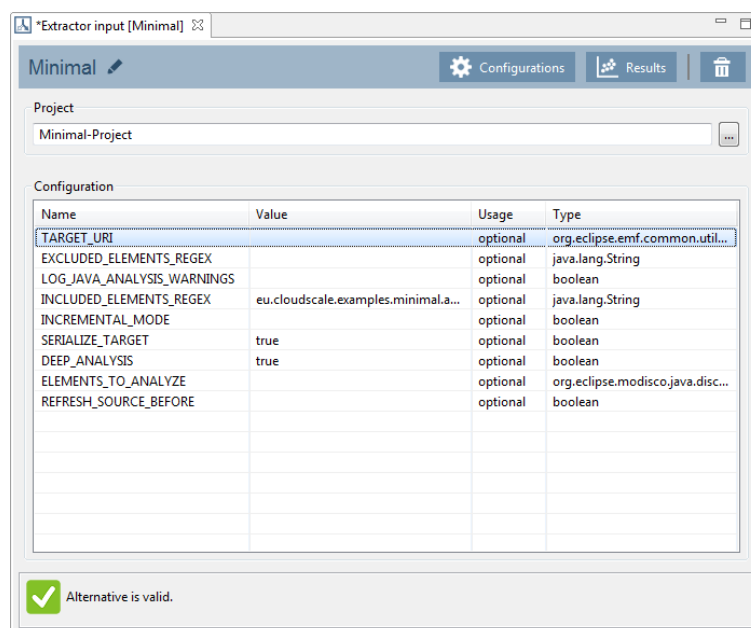
NOTE: Repository model contains initial service effects specifications (SEFFs), which needs to be further measured and adjusted. One needs to measure exact times and resource demands to produce analysable models.

User can browse the produced model using model tree view in the Results editor or can open co-responding diagrams by double clicking on the models. Models are modifiable therefor all changes can be persisted and used before using them further with other tools.

Walkthrough

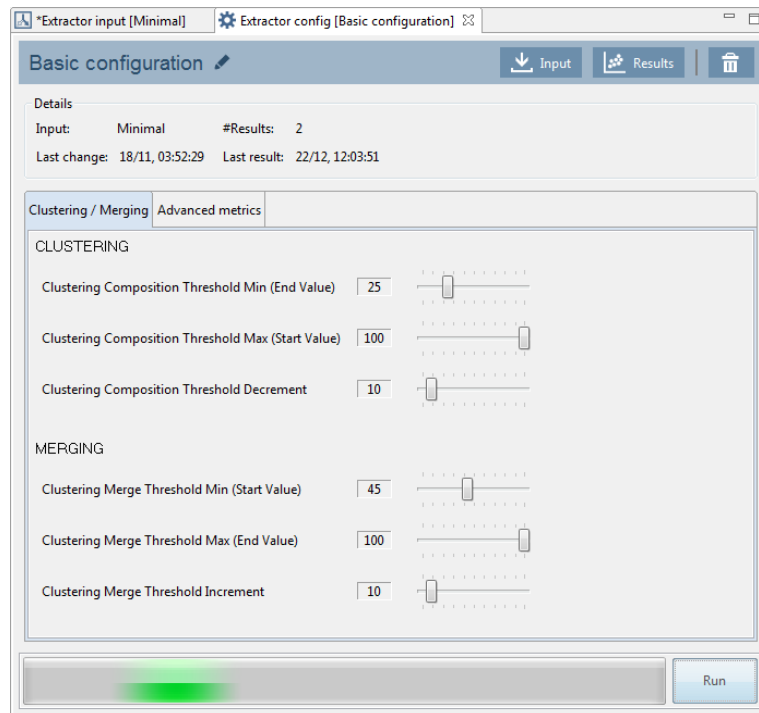
Prerequisite: You already have CloudScale Environment project ready.

1. Import Minimal source project; it can be found under Examples wizard.
2. Create new Extractor alternative; right click on Extractor node in Minimal-Example and New -> Extractor alternative.
3. Open Input editor by double clicking on newly created alternative. Select project to be extracted; previously imported Minimal-Project. Additionally configure Extractor to work only on application classes; set INCLUDED_ELEMENTS_REGEX to eu.cloudscale.exmplaes.minimal.app.*

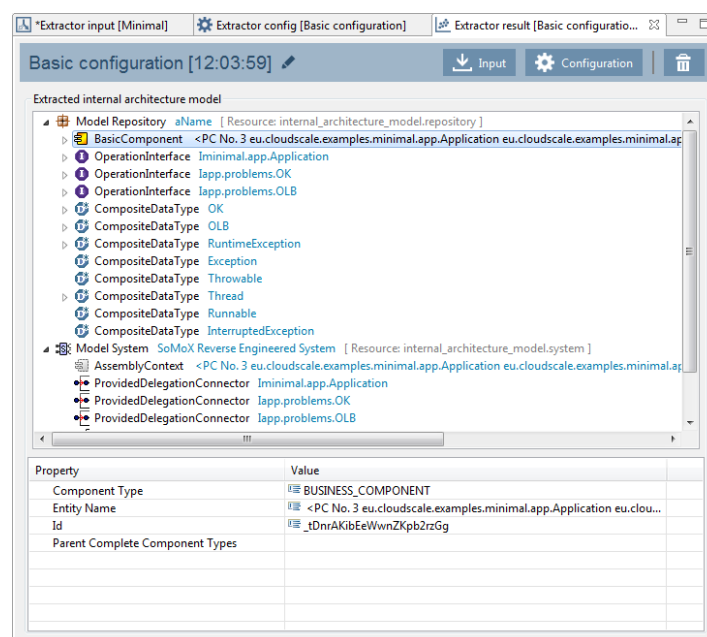


2. Open configuration alternative (through Explorer or context actions in Input Editor). You can update different metrics here (see Configuration) or for the purpose of this walk-through just leave it as it is.

3. Run configuration alternative; using run action in editor.



4. After successful run open newly created result; e.g. click on the '(result...)' in the footer. In Result editor you can view produced models and edit them through properties view or through dedicated diagrams (double-click on model).

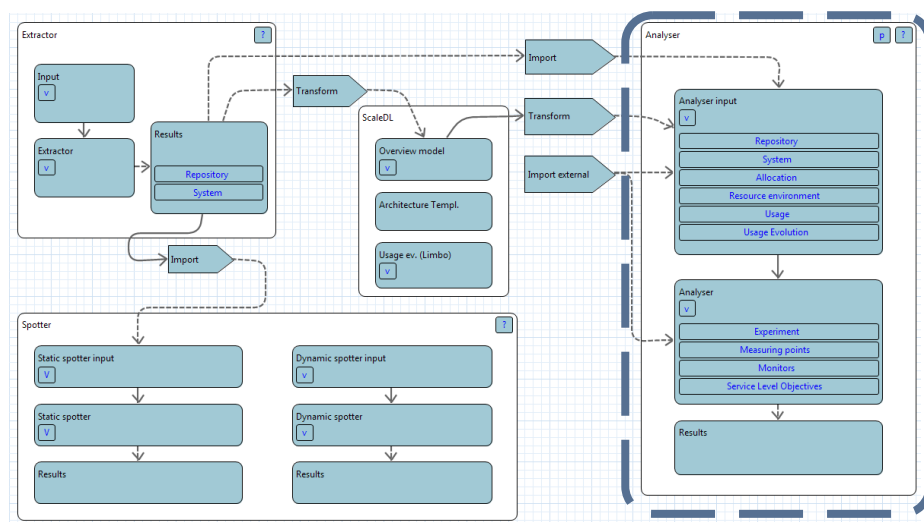


Analyser tool

Introduction

CloudScale's Analyser allows to analyze ScaleDL models regarding scalability, elasticity, and efficiency of cloud computing applications at design time. For these capabilities, CloudScale integrated novel metrics for such properties into Analyzer. Analyses are based on analytical solvers and/or or simulations. Analyzer particularly supports to analyze self-adaptive systems, e.g., systems that can dynamically scale out and in.

Relation to Workflow



The Analyser is important for the right-hand side of the [Workflow](#). It is capable of providing scalability, elasticity, and cost-efficiency analysis results based on a fully specified ScaleDL instance. Such analysis results can already be provided before the system is accordingly implemented, e.g., at early design time or to investigate modification scenarios.

The Analyser input can be specified in several ways. It can be based on a combination of

- 1) an imported partial ScaleDL instance from the Extractor (input arrow; top)
- 2) an already existing Analyser input project (first input arrow; left)
- 3) transforming a ScaleDL overview model (second input arrow; left)
- 4) calibration data gained from Dynamic Spotter measurements (third input arrow; left)

Problem

Scalability, elasticity, and cost-efficiency problems are often detected too late, i.e., during system testing and/or system operation. This approach leads to the risk of SLO-violating systems and expensive re-implementations. Moreover, different design variants remain uninvestigated because it would be too extensive to implement and test all potential design variants.

Features

CloudScale's Analyser measures several scalability, elasticity, and cost-efficiency metrics based on a ScaledL instance as an input. In Analyser's run configuration, software architects can specify properties of an analysis run. Analyser supports three types of experiments:

- Normal - for performance and elasticity experiments.
- Capacity - for capacity metrics that investigate the maximum workload a system can handle.
- Scalability - for scalability metrics that investigate the maximum workload a system can handle when it has virtually unlimited resources like in cloud computing – in such systems semaphores and transactions might become bottlenecks.

Experiment duration can be limited by, time (ie. Maximum time simulation can run) or by number of measurements (ie. how much measurements must be collected before finishing simulation). With measuring points one specifies where to measure and with monitors which metrics are measured at which measuring point. Additionally, service level objectives (i.e. Thresholds) can be specified to observed monitor values.

Usage

Input

As an input, Analyser takes a ScaledL instance: PCM, Usage Evolution, Architectural Templates, where former two are optional.

When creating new alternative user can choose from following options:

- New Analyser input - creating default PCM instance: Repository, System, Allocation and ResourceEnvironment and UsageEvolution instances. It is possible to choose which models should be created.
- Import Extractor result - importing Repository and System models produced by Extractor (extractor results).
- Transform Overview model - transforming ScaledL Overview (including referenced PCM models) into new Analyser input alternative. NOTE: no external references are left, ie. all models are copied into alternative.
- Import external models - importing external PCM models in new alternative. All references between models are fixed. Wizards help with selecting all depended models.

After initially creating alternative user can alter PCM models by creating new or importing external model instances. NOTE: Multiple instances of Repository and Usage model are allowed.

All models can be modified through Input editor, however more user-friendly option is to use dedicated editors. PCM models through diagram editors (accessed by double clicking on the model) and Usage Evolution models through dedicated component in Input editor. In contrary, Architectural Templates (profiles and stereotypes) can be applied to PCM models using “MDSD Profiles” context action (right-click on corresponding model).

Configuration

Analyser is configured via an Experiment Automation instance (reflected in Analyzer’s run configuration). The latter is a dedicated configuration model for the Analyser, e.g., allowing to configure metric measurements and how often and how long Analyser runs are conducted.

When creating new configurations user can choose from following options:

- New normal configuration - creates new experiment of type normal, suitable for performance and elasticity experiments.
- New capacity configuration - creates new experiment of type capacity, suitable for measuring capacity metric.
- New scalability configuration - creates new experiment of type scalability, suitable for measuring scalability metric.
- Import external experiment - imports existing projects into the CSE. Through experiment model selection wizard automatically creates new input and configuration. Input alternative includes all referenced PCM models and Usage Evolution models. Configuration alternative is based on the experiment configurations and all referenced models are copied.

Configuration editor is composed of a set of components, each dedicated to editing specific part of Experiments model and referenced models. These components are:

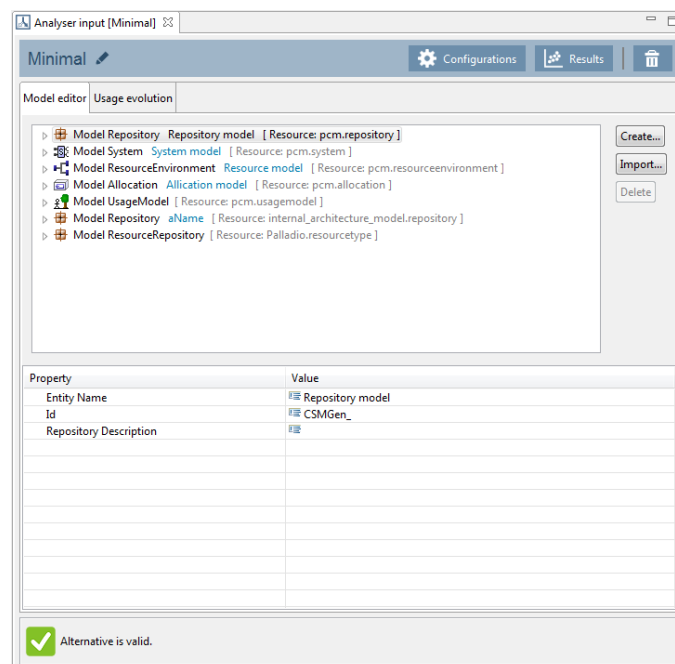
- Basic settings - selection of usage model and usage evolution, configuring stop conditions (simulation time and measurement count) and capacity measurements settings (available in Capacity and Scalability experiments).
- Measuring points - manipulation of measuring points (model instance is automatically created in the alternative).
- Monitors - manipulation of experiments monitors, composing measurement specifications (metric, statistical characterization, intervals) and linking to target measuring points. (model instance is automatically created in the alternative)
- Service Level Objectives - manipulation of SLO used when measuring Capacity and Scalability metrics. Allows user to define upper and lower bounds for each measurement specifications defined. (model instance is automatically created in the alternative)

- ## Result

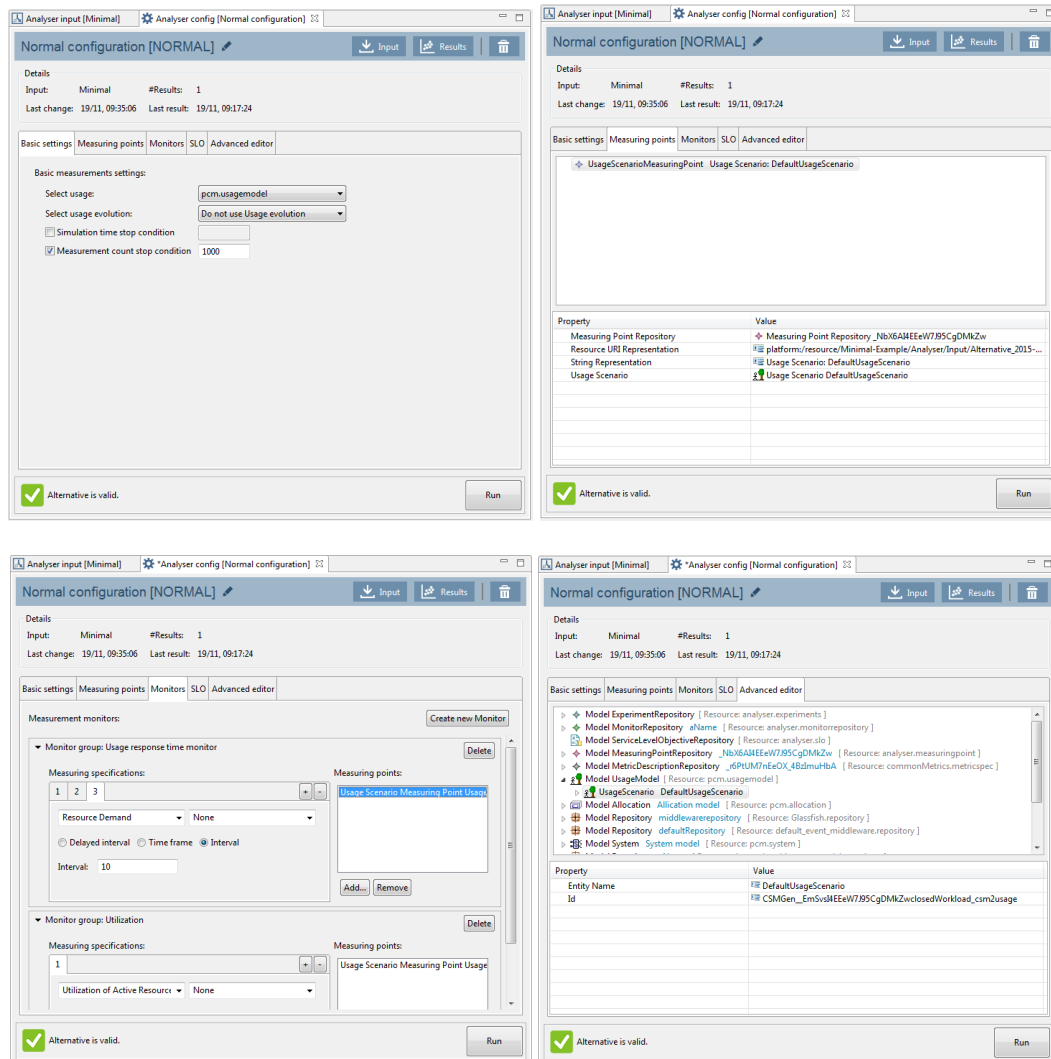
Results editor provides two components: Results view and EDP2 view. First allows to interactive view of measurements gathered for each measuring point in using three different chart visualizations: line, bar and pie. EDP2 view in additional provides advanced functionalities and allows viewing different measurements on the same chart, incorporating SLO specifications by highlighting violations, etc.

Walkthrough

1. (optional) Open Analyser Input editor. Extended PCM is already complete therefore there is nothing to do in this step.



2. Create a new Normal configuration; right click on Configurations in Explorer. Basic configuration can already be run, but you can update it to your needs.



3. Run Analyser configuration. This can take some time (especially for Normal scenarios).

4. (optional) Repeat 2-3 steps for Capacity and Scalability configurations.

5. After successful run open newly created result; e.g. click on the '(result...)' in the footer. Investigate analysis results (normal, capacity and scalability metrics). XY Plots are generally good to observe monitored values (Y-axis) over time (X-axis). Histograms aggregate such values into bins, allowing to inspect the probabilistic distribution of values. Other monitors provide single values only, e.g., the "System capacity" metric. Besides these main monitors, the "Advance result viewer" allows to inspect additional monitors. Simply double-click on "Measurement" nodes and select a suitable diagram type from the dialog.

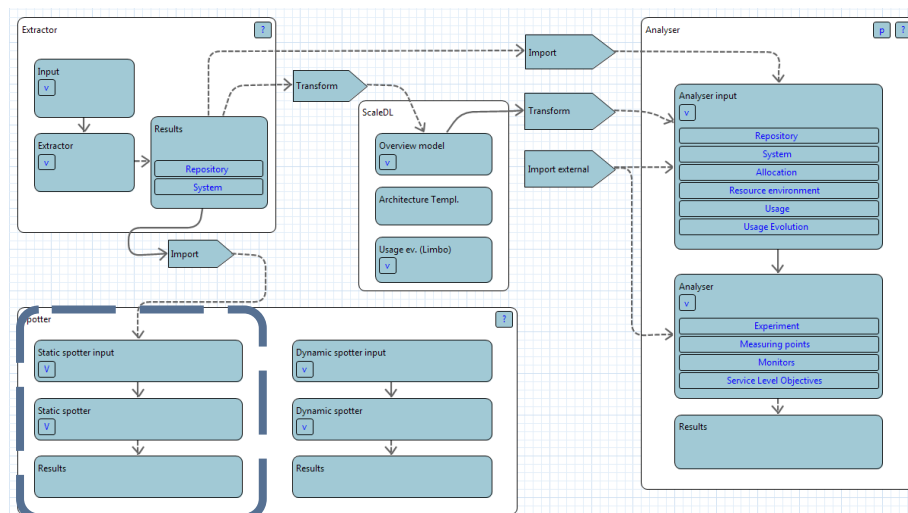


Static Spotter tool

Introduction

The Static Spotter is a reverse engineering tool based on the Reclipse for automatic detection of so called search patterns which then are interpreted as potential scalability anti-patterns. All scalability anti-patterns are defined in the pattern catalogue. Later on, the Static Spotter is searching for the anti-patterns according to the pattern catalogue.

Relation to workflow



The Static Spotter is designed to take the Extractor's output as input. It is able to detect the scalability anti-patterns from the partial ScaleDL model and source code.

Problem

A software system suffers scalability or performance issues while being development without consideration of scalability or performance. This would become even worse when being migrated to the Cloud computing environment. It is usually because there are scalability or performance anti-patterns in the source code.

Features

The Static Spotter exams the ScaleDL model and source code. It uses predefined pattern or anti-pattern catalogues to detect patterns and anti-patterns automatically (catalogues can be extended by the user or community). Currently supported anti-patterns in built-in catalogue are:

- One Line Bridge - occurs, if a passive resource limits the concurrency in an application. Passive resources can be for instance mutexes, connection pools, or database locks.
- The Blob - occurs when one class performs most of the system work relegating other classes to minor, supporting roles.

Additional information on anti-patterns and how to resolve them can be found at

<http://wiki.cloudscale-project.eu/index.php/HowNotTo: Anti-Patterns>

Usage

Input

As an input, Static Spotter takes models produced by Extractor: Somox and Modisco models. First are the PCM models (System and Repository) with additional SourceDecorator model, which connects Modisco AST with the PCM models and which is actually used by the Static Spotter.

Input editor allows user to view all input models and edit them if needed. NOTE: Normally additional modifications are not needed.

Configuration

Static Spotter configuration is a anti-pattern catalogue. Official one is already integrated into the CSE, however user can replace it to use it's own.

Configuration Editor provides a catalogue overview, with pattern specifications and annotations. NOTE: Catalogue can be replaced through Project explorer by replacing catalog.pcs and catalog.pcs.ecore files.

Result

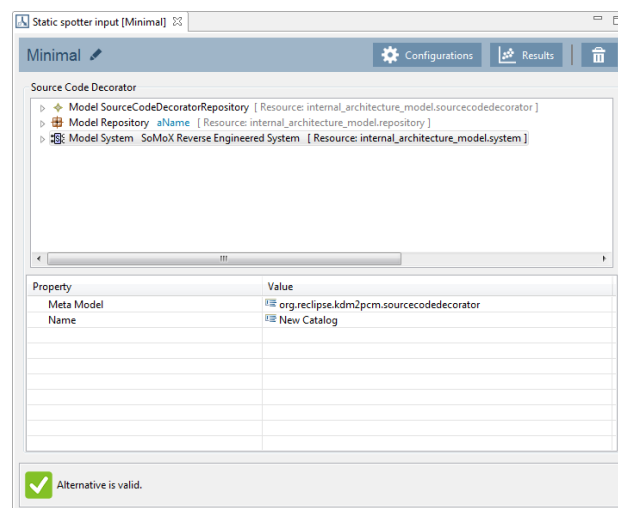
The Static Spotter takes ScaleDL models and source code as input. After searching for predefined patterns or anti-patterns, it generates a list of pattern or anti-pattern candidates as output. The candidates could be methods, classes or components.

Result editor shows an editor with all found annotations based on pattern specifications described in catalogue, with additional information where (source-code location) these problems has been found.

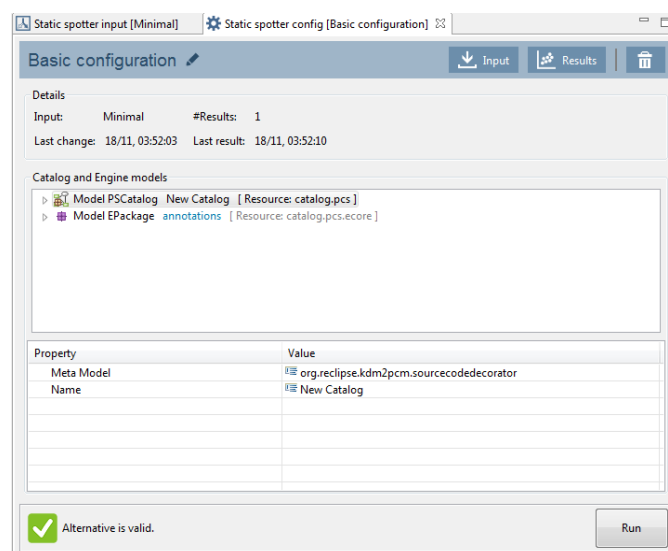
Walkthrough

Prerequisite: Extractor result available

1. Import Extractor result into Static Spotter; this creates new input and configuration alternative.
2. (optional) Open input editor. It contains model view with SrouceCode decorator model and partial PCM model used by Static Spotter. It is possible to update models here, however normally this is not needed.

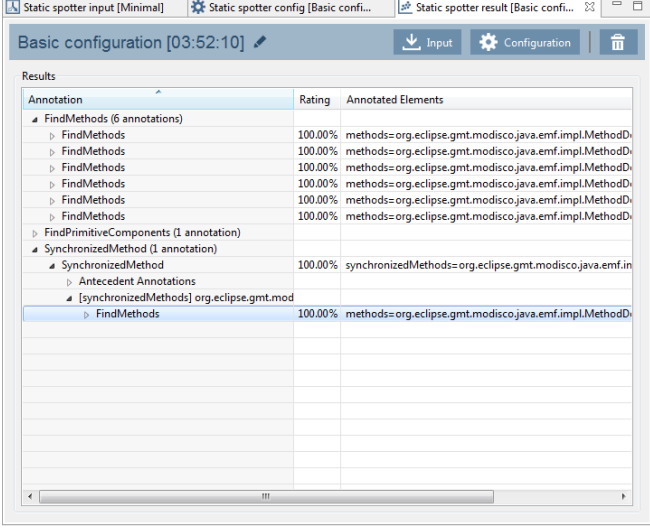


3. Open configuration editor. It contains Static Spotter catalog specifying anti-patterns that Static Spotter will look for.



4. Run static spotter

5. After successful run open newly created result; e.g. click on the '(result...)' in the footer. Result editor contains table with all found annotations. In case of minimal example there is one potential anti-pattern found; synchronize block.



The screenshot shows the 'Static spotter result' window with the 'Basic configuration' tab selected. The window displays a table of results under the 'Results' section. The table has three columns: 'Annotation', 'Rating', and 'Annotated Elements'. The data is organized into a tree structure with expandable/collapsible icons (triangles) next to the annotation names.

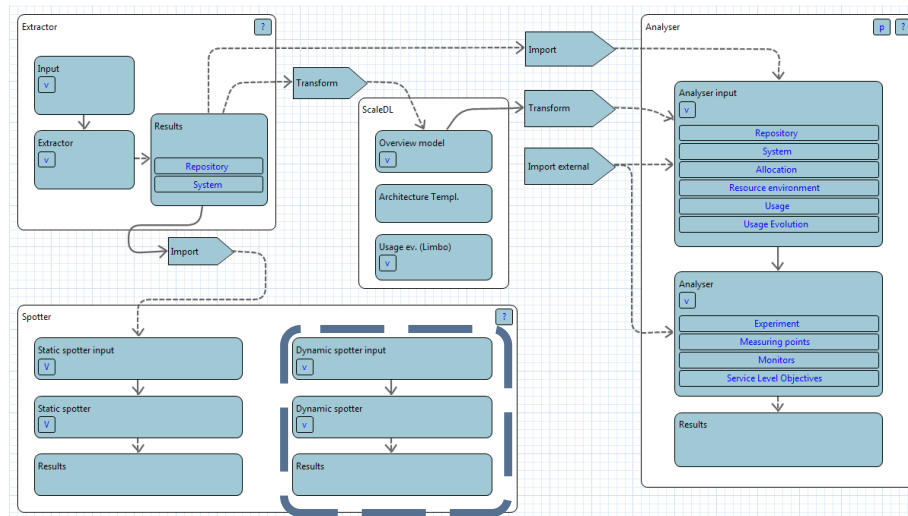
Annotation	Rating	Annotated Elements
FindMethods (6 annotations)		
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD
FindPrimitiveComponents (1 annotation)		
SynchronizedMethod (1 annotation)		
SynchronizedMethod	100.00%	synchronizedMethods=org.eclipse.gmt.modisco.java.emf.in
Antecedent Annotations		
[SynchronizedMethods] org.eclipse.gmt.mod		
FindMethods	100.00%	methods=org.eclipse.gmt.modisco.java.emf.impl.MethodD

Dynamic Spotter tool

Introduction

Dynamic Spotter is a framework for measurement-based, automatic detection of software scalability problems (scalability anti-patterns) in Java-based enterprise software systems.

Reference to workflow



Dynamic Spotter comes as part of the Spotter (lower-left part of the workflow in **Error! reference source not found.**). This part of the workflow becomes relevant if you want to identify and resolve anti-patterns for an existing system based on measurements.

Problem

Scalability issues hinder applications to benefit from additional resources as being available in cloud computing environments. When manually done, the detection of such issues is a complex and effort-intensive task.

On the one hand, our “Static Spotter” helps engineers in detecting potential scalability issue candidates by a static code analysis. However, without putting actual workload of a system in operation, such issues only remain candidates. Using a dynamic analysis – the Dynamic Spotter – such candidates can be identified as true scalability issues.

Features

Dynamic Spotter needs an instrumented application in operation. Dynamic Spotter then can connect to this application to stress it systematically with different workloads. At the same time, Dynamic Spotter monitors scalability-relevant measurements. Based on this systematic experimentation, it can identify scalability issues.

Dynamic Spotter supports detection of following scalability problems:

- One Lane Bridge - occurs, if a passive resource (e.g. mutexes, connection pool, database locks) limits the concurrency in an application.
- Ramp - occurs when processing time increases as the system is used.
- Blob - occurs when one class performs most of the system work relegating other classes to minor, supporting roles.
- Stifle detection - occurs if single database statements changing the data are executed for the same table in a loop instead in a batch.
- TrafficJam detection - represents a scalability problem, either due to software bottlenecks or hardware limitations.
- Expensive Database Calls - are single, long-running database request. They cause high overhead at the database, either due to high locking times, many locks, or high utilization of the database's resources.
- Empty Semi Trucks detection - occurs in software systems where a big amount of requests is needed for a single job.
- Application Hiccups - represents the problem of periodically violated performance requirements.
- Continuously Violated Requirement - checks if performance requirements are violated continuously under high load.

Instrumenting target application (SUT)

Dynamic Spotter works against running applications, and to gather relevant data application must be instrumented with java agent provided by Dynamic Spotter (aim.mainagent). This section describes how to set up the java agent and how to configure JMX to exchange data between Dynamic Spotter service and the agents.

Setting up the javaagent

The Dynamic Spotter needs a connection to a javaagent to do its work. If you use the AIM javaagent (Mainagent), add an argument like this to your application to load the agent on startup:

```
-javaagent:/path_to_agent/aim.mainagent-1.0.4-agent.jar
```

To set up more details of the Mainagent, e.g. for logging, you can add to the javaagent-argument a path to a configuration file where you can set up the agent. Then the complete argument looks like this:

```
-javaagent:/Path_to_Agent/aim.mainagent-1.0.4  
-agent.jar=/path_to_config/
```

Type this into the configuration file (all configurations are optional):

```
logLevel: (DEBUG | INFO | WARN | ERROR)
```

logType: (FILE | STDOUT)

logFile: /path_to_logfile/

Setting up JMX

The AIM javaagent uses JMX to exchange data. So you need to configure JMX to run the Dynamic Spotter successfully. So you need to add some additionally arguments to you application:

```
-Dcom.sun.management.jmxremote  
-Dcom.sun.management.jmxremote.port=9010  
-Dcom.sun.management.jmxremote.local.only=false  
-Dcom.sun.management.jmxremote.authenticate=false  
-Dcom.sun.management.jmxremote.ssl=false
```

The first argument sets up JMX communication at all, the next defines the JMX port number (9010 is default), the next argument allows access to the JMX port from any host instead of just localhost, the final two arguments define to deactivate the authentication and encryption of the channel (this could be activated if required later).

Finally, JMX can be debugged using the jconsole application provided with the JDK. In jconsole, you can see the agent under the MBeans tab after connecting to the JVM that runs the new agent.

Usage

Server

Dynamic Spotter is a client-server solution. Server is responsible to run analysis while client is used to provide configurations. To use the integrated Dynamic Spotter client one must start server beforehand.

The CSE provides two options of how to connect to server; built-in server and external server. First option, using built-in server, is that the CSE automatically extracts server and start it (port is automatically chosen) and needs minimal user interaction. The second option is to connect to external server. NOTE: Different versions of the Dynamic Spotter server can be used this way.

Editors (input, configuration, results) can't be used in case that Dynamic Spotter client is not yet connected, however they provide an option to initialize built-in server automatically (i.e. "Start server" action). After successful initialization and connection editor refreshes itself and shows configuration components.

Input

As an input Dynamic Spotter must be configured to connect to java agents started on the SUT. This accomplished using instrumentation and measurement adapters. NOTE: Though a specific instrumentation adapter and a measurement adapter are often realized within one external tool, they represent conceptually different tasks. In such a case, adapters for both extensions points are required, even if only one tool realizes both tasks.

Instrumentation adapters

The Instrumentation Adapter extension point allows providing adapters for instrumentation tools like DiSL, Kieker, our own instrumentation tool AIM (Adaptable Instrumentation and Monitoring), etc. Thereby, the Dynamic Spotter uses a generic instrumentation description model (IDM) to decouple the instrumentation description from the tool realizing it. Thus, an instrumentation adapter for a specific instrumentation tool has to realize the transformation from the IDM instances to the tool-specific representation of the instrumentation description.

Available instrumentation adapters are:

- `DynamicInstrumentationClient` - The default instrumentation satellite adapter can be used to connect to a instrumentation satellite running in a JVM. This satellite adapter will instrument the JVM. The data collection with the instrumentation must be en-/disabled with a measurement satellite. Hence, you should not forget to configure a corresponding measurement satellite adapter.
- `LoadRunnerInstrumentationClient` - The loadrunner instrumentation satellite adapter can be used to connect to an HP Loadrunner instrumentation satellite. It will only be applicable if you have a Loadrunner as a workload generator.

Measurement adapters

While an instrumentation adapter is solely responsible for instrumenting the SUT, a Measurement Adapter is used to enable and disable data collection, as well as to transform and transfer data from the monitoring tool to Dynamic Spotter in a common data representation format.

Available measurement adapters are:

- `MeasurementClient` - The measurement satellite adapter is used to connect to system where instrumentation is possible. This satellite adapter can enable and disable the collecting of data fetched with instrumentation. This satellite adapter will be mainly used on systems where a instrumentation satellite is running. In addition this satellite adapter comprises the sampling of hardware utilization.
- `LoadRunnerMeasurementClient` - The loadrunner measurement satellite adapter connects to a measurement satellite executed on a Loadrunner system. This satellite adapter is responsible to fetch the result data after a Loadrunner workload execution.
- `DBMSMeasurement` - The DBMS sampling measurement satellite adapter is used to connect to a MySQL DBMS and to query the database status.

- **JMSServerMeasurement** - The jmsserver sampling measurement satellite adapter is used to connect to the special sampling satellites for Java Messaging Service (JMS) server. They sample more than the default sampling satellites.
- **ResourceMonitoringAdapter** - The sampling measurement satellite adapter is used to connect to all sampling satellites.

Configuration

In configuration alternative the Dynamic Spotter is configured with runtime specifications. How to run the experiments, which load driver to use to perform workload on the SUT and the hierarchy of anti-patterns to look for.

Basic run configurations

Basic run configurations specifies how the experiments will run on the SUT. There are set of configuration available to specify scenario duration (including pre warm-up and cool down duration), number of user to simulate in each phase, etc.

- **Duration** - Specifies the duration of the steady state experiment phase in seconds
- **Prewarmup.Duration** - Specifies the duration the pre-warmup phase should be running; to force the JVM to load classes, fill caches, etc.
- **MaxUsers** - The maximal number of users the system under test should be able to handle.
- **Cooldown.IntervalLength** - Defines the cool down phase of load generation. Specifies the length of a single interval in seconds.
- **Cooldown.NumUserPerInterval** - Specifies the number of users which should enter the system in a cool down phase per time interval.
- **Rampup.IntervalLength** - Defines the ramp up phase of load generation. Specifies the length of a single interval in seconds.
- **Rampup.NumUsersPerInterval** - Defines the ramp up phase of load generation. Specifies the number of users which should enter the system per time interval.

Load Driver (Workload) adapters

The Load Driver Adapter extension point provides means to use existing load driver tools like JMeter, Faban or HP LoadRunner for workload generation. In particular, a Load Driver Adapter needs to provide means to control the execution of performance tests programmatically. Furthermore, the adapter realizes a remote communication between DynamicSpotter and the remotely running load generation tool.

Workloads can be generated using three different load drivers:

- Simple workload driver - applicable when load generators are implemented in Java. Driver can use provided Java classes to generate load. It is configured with fully defined class name (workload.simple.userScriptClassName) and absolute path to target build folder (where root package reside) NOTE: path can be made relative to workspace using \${workspace.path} parameter.
- JMeter workload driver - this workload satellite adapter can execute a local JMeter application with a given load script. JMeter home and scenario file must be defined. Additionally thinkTimeMax and thinkTimeMin can be specified to define thinking time.
- LoadRunner workload driver - connects to the workload satellite executed on the Loadrunner system. This satellite adapter provokes the workload satellite to start the workload on Loadrunner. In addition to satellite location (host and port) workload must be specified; LoadRunner scenario file and executable, path to load runner result directory.

Performance Problem Hierarchy

DynamicSpotter uses a performance problem hierarchy and corresponding heuristics to guide the detection process. Using performance problem hierarchy we specify which anti-patterns should be analysed in target application. User can choose from variety of extensions: OLBDetectionController, BlobDetectionController, EDCDetectionController, RampDetectionCtonroller, etc. Each of these can be further configured with different set of parameters. To describe each extension and all available is out of scope for this manual. For further details see <http://sopeco.github.io/DynamicSpotter/>

OLBDetectionCtonroller detect One Lane Bridge anti-pattern. It can be configured using following parameters:

- cpuThreshold - defines the CPU utilization threshold, when a system is considered as over utilized.
- numExperiments - number of experiments to execute with different number of users between 1 and max number of users.
- reuseExperimentsFromParent - indicates whether the experiments form the parent heuristic should be used for this heuristic.
- scope - determines the scope, of the OLB analysis. Possible values are: entry point scope, synchronization scope, database scope. For details see Using scopes section.
- strategy - determines the strategy used to analyze the OLB anti-pattern. Possible values are: queuing theory strategy, t-test-CPU-threshold strategy

RampDetection controller detects ramp which occurs when processing time increases as the system is used. Ramp controller can be configured using following parameters:

- Linear slope threshold - Defines the threshold for linear slope. Growth of response times per time unit of experiment.
- numExperiments - specifies number of experiments to execute with different number of users between 1 and max number of users.

- `numRequiredSignificantSteps` - specifies the number of steps between experiments required to show a significant increase in order to detect a Ramp.
- `requiredSignificanceLevel` - defines the coincidence level to be reached in order to recognize a significant difference when comparing two response time samples with the t-test.
- `reuseExperimentsFromParent` - indicates whether the experiments from the parent heuristic should be used for this heuristic.
- `stimulationPhaseDurationFactor` - specifies the duration of the stimulation phase.
- `strategy` - determines the strategy used to analyse the RAMP anti-pattern. Possible values are: time windows strategy, direct growth strategy, linear regression strategy.

Using scopes (OLB)

The OLB detector of the dynamic spotter uses a so-called scope to determine the set of methods which will be measured and compared against the configured threshold to detect the OLB anti-pattern.

By default, the OLB detector supports two scopes out of the box: the entry level scope and the database scope. The entry level scope instruments all REST-based API calls. As such, it can be used to monitor the interface of the system's business layer in cases where this layer relies on REST to offer its services. The second scope instruments all JDBC database calls. As such, it can be used to instrument the interface to the database from most Java applications.

In situations where the OLB is hidden somewhere inside the layers of the application, the developers have to provide a custom scope. The most easy way to do this is to adapt the code inside of the OLB extension (see class `OLBDetectionController`). There, one can define a custom scope using the builder API of the instrumentation framework.

For example, the following code put inside the `getInstrumentationDescription()` method, instruments all direct methods in the `core.businesslayer.api` package:

```
idBuilder.newMethodScopeEntity("core.businesslayer.api.*")
    .addProbe(ResponsetimeProbe.MODEL_PROBE)
    .entityDone();
```

In cases where you would like to have also all the methods called from the methods defined in a scope to be instrumented, you just add a `enableTrace()` call to the definition like so:

```
idBuilder.newMethodScopeEntity("core.businesslayer.api.*")
    .enableTrace()
    .addProbe(ResponsetimeProbe.MODEL_PROBE)
    .entityDone();
```

In this case, however, you might get a lot of measurements slowing the application under inspection down too much. To reduce the number of measurements, you can add a `setGranularity()` call like so:

```
idBuilder.newMethodScopeEntity("core.businesslayer.api.*")
    .enableTrace()
    .addProbe(ResponsetimeProbe.MODEL_PROBE)
```

```
.entityDone()  
.newGlobalRestriction()  
.setGranularity(0.1)  
.restrictionDone();
```

This roughly instructs the instrumentation framework to just record every tenth measurement at a particular probe. Using these techniques, it should be easy to define the measurement scope for the OLB detection. Using an iterative refinement process, a developer should be able to find a scalability issue's root cause easily.

Result

The output of the Dynamic Spotter are results for each detection controller configured (e.g. OLBDetectionController) along with collected measurements (e.g. response times, utilizations).

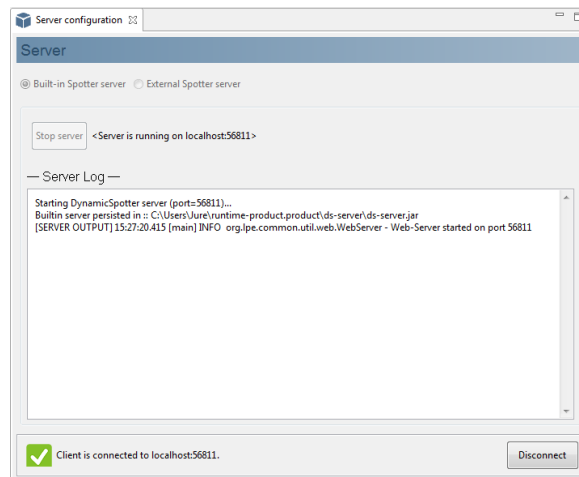
Results editor provides status of each configured detection controller along with message containing where anti-pattern has been detected and some pointers how to resolve it. All collected measurements are visually displayed in line charts along with expected thresholds, ranges, etc.

Additionally, the Dynamic Spotter provides textual report containing basic information regarding experiment (run) and detected anti-patterns with locations in source code.

Walkthrough

1. Run Minimal project application. Project can be imported through Example wizard and run with startup script; startApplication.sh or startApplication.bat (located in dist/ folder). This will locally run application with web-service on port 8080 satellites on port 9010. NOTE: You can update sources and prepare new version with maven - mvn install

2. *In order to run Dynamic Spotter, a Dynamic Spotter server has to be started first. Open dedicated view through CloudScale Explorer; Dynamic Spotter > Server Connection node. Start built-in server and connect Spotter client.*



3. Create Dynamic Spotter alternative; right-click on the DynamicSpotter node and choose 'New > DynamicSpotter alternative'. This will create input and configuration alternative.

4. Open Input editor and configure instrumentation and measurements satellites to target your application.

- a) Create Default Instrumentation Satellite and configure it to target your application's java agent;

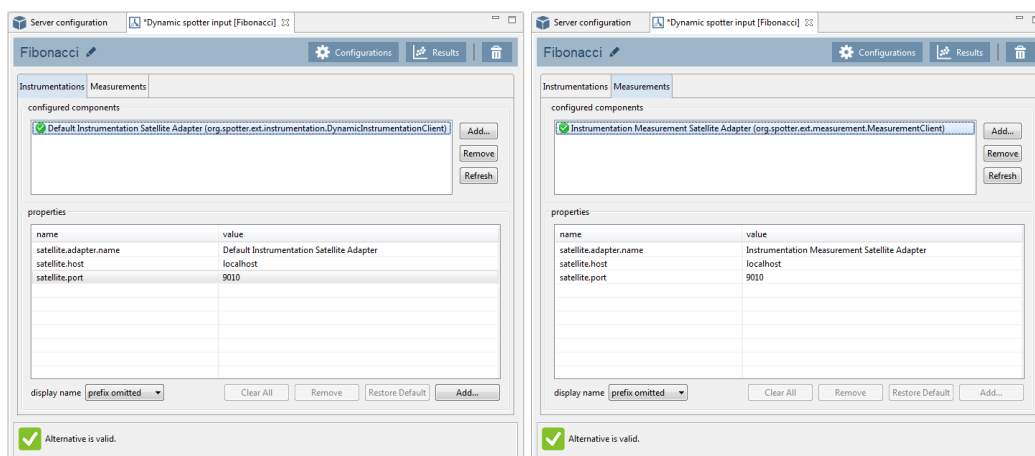
satellite.host = localhost

satellite.port = 9010

- b) Create Default Measurement Client and configure also configure it to target your application's java agent;

host = localhost

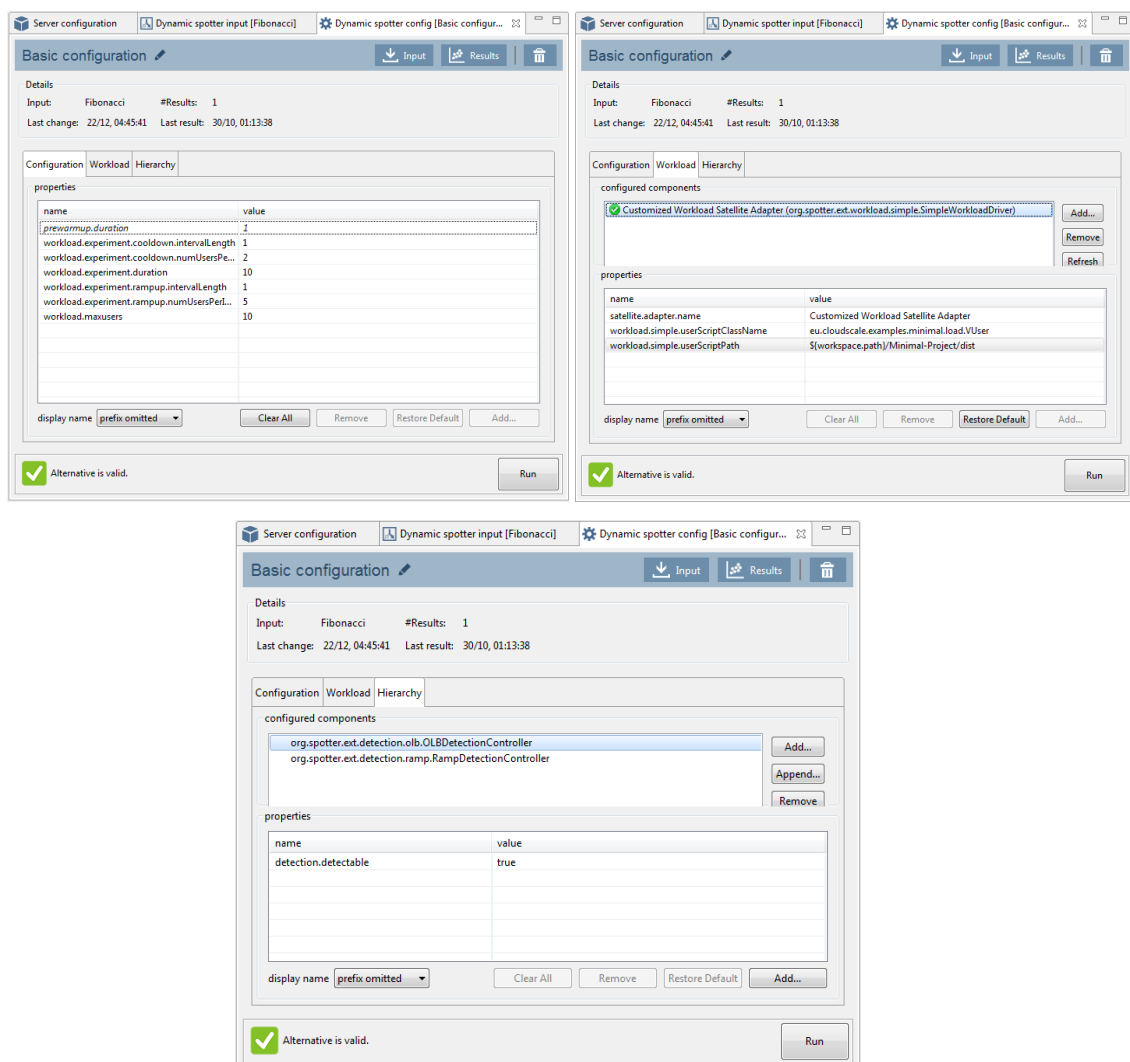
port = 9010



5. Open Configuration editor and configure run properties, workload scenario and anti-pattern hierarchy:

- Default run configuration is already fine. You can change different parameters here (see Usage > Configuration).
- Add Simple Workload Driver to run Java based scenarios. Configure it to use scenario already available in Minimal project.

userScriptClassName = eu.cloudscale.examples.minimal.load.VUser
userScriptPath = \${workspace.path}/Minimal-Project/dist
- Configure anti-pattern hierarchy. Add OLB detection controller and Ramp detection controller. Additional detectors or further configurations are not necessary for Minimal case.



6. After the Dynamic Spotter has finished, a result is created. Open it by clicking on '(result ...)' link beside the successful message in configuration editor. Editor shows report for each detection controller along with resource charts. In the Minimal Example OLB should be detected in `Application.testOLB()` method.

