CTWedge

Combinatorial Testing Web Editor and Generator

Contatti:
Prof. Angelo Gargantini – angelo.gargantini@unibg.it
Dott. Marco Radavelli – marco.radavelli@unibg.it

CTWedge in brief

1. Language for CIT problems with a precise formal semantics and a grammar by XTEXT
2. A textual editor integrated in the eclipse IDE
3. Set of tools for importing/exporting CIT models, and for generating test suites (by using external tools)
4. Framework based on the Eclipse Modeling Framework (EMF)
5. Integrates a collection of benchmarks (Phone, Banking, HealthcareSystem, Concurrency, ...)

Comes in two flavors:

- Web: http://foselab.unibg.it/ctwedge
- Eclipse Plugin: https://fmselab.github.io/ctwedge/ctwedge_update/

CTWedge Features

- **CTWEDGE offers:**
  - **A rich but simple language for combinatorial models,**
  - **a powerful web editor,**
  - **generation of CIT test suites on a (our) server (web version)**
- **The only software needed to use CTWEDGE is a modern web browser (SaaS)**
- **Successor of CITLAB**

Language

- definition of parameters, each with its name and:
- the following parameter types:
  - Boolean
  - Ranges (that are integer intervals): e.g.: [0..5]
  - Enumerative that are a list of possible values
  - values: {900, "hello phone", BUSY}
- Constraints
  - propositional logic (with the usual logical operators) with equality and arithmetic
  - tests that do not satisfy the constraints are considered invalid and do not need to be produced.

Example + Grammar

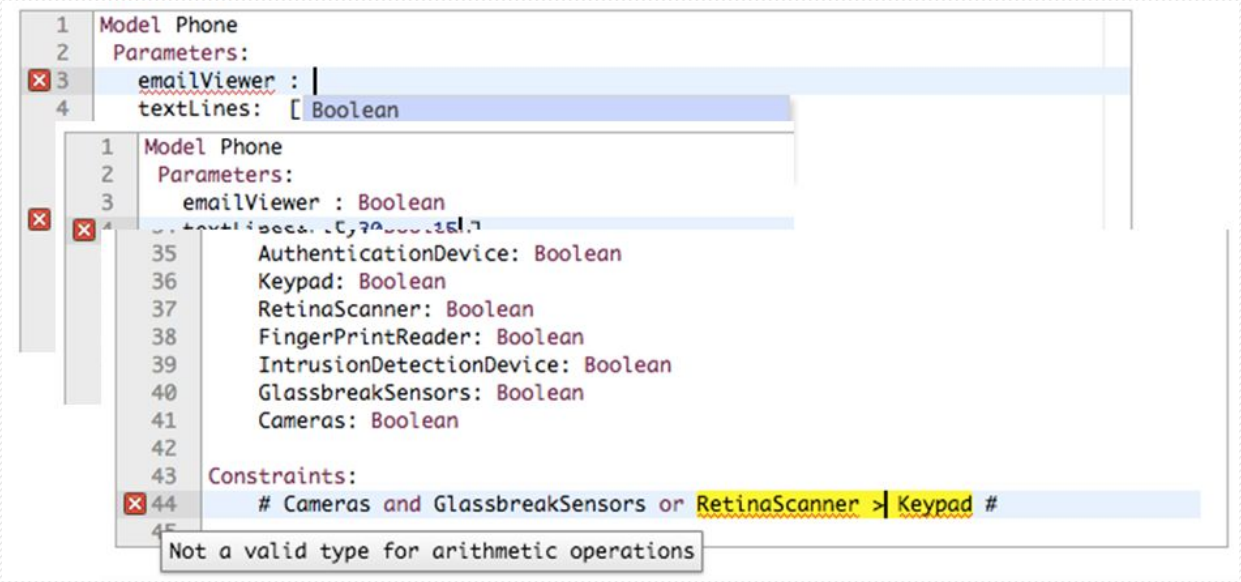
```
/*
 * This is an example model
 */
Model Phone
 Parameters:
   emailViewer : Boolean
   textLines:  [ 25 .. 30 ]
   display : {16MC, 8MC, BW}

Constraints:
  # emailViewer => textLines > 28 #
```

GRAMMAR:

```
CitModel:
  'Model' name=ID
  //list of parameters
  'Parameters' ':'
(parameters+=Parameter)+
  // constraints
  ('Constraints' ':'
(constraints+=Constraint)+)?;
```

Web Editor

- **The CTWEDGE web editor is based on JavaScript Ace (Ajax.org Cloud9 Editor), but other web editors (like Orion and CodeMirror) are supported**
- **Some features: Syntax highlighting, runtime validation, auto completion,…**



Test Suite Visualizer

Test suites are directly shown in the browser

Can be exported in CSV (excel)

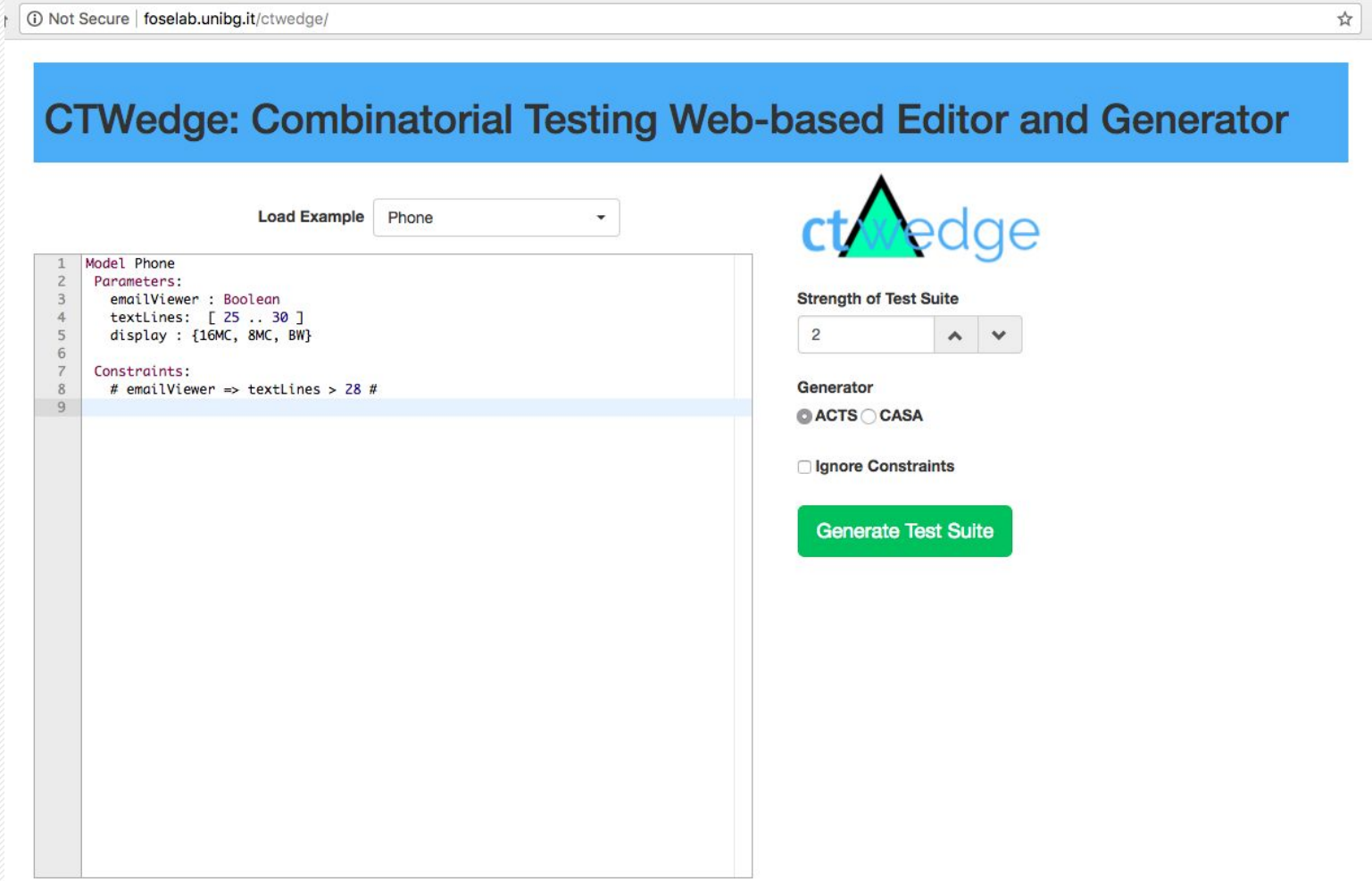
Generated Test Suite

Generated using ACTS strength=2 ignoringConstraints: false

Download CSV Separator: Semicolon (;) - DEFAUl

#	emailViewer	textLines	display
1	false	25	16MC
2	false	25	8MC
3	false	25	BW
4	false	26	16MC
5	false	26	8MC
6	false	26	BW
7	false	27	16MC
8	false	27	8MC
9	false	27	BW
10	false	28	16MC
11	false	28	8MC
12	false	28	BW
13	true	29	16MC

CTWedge DEMO



CTWegde 4 Eclipse

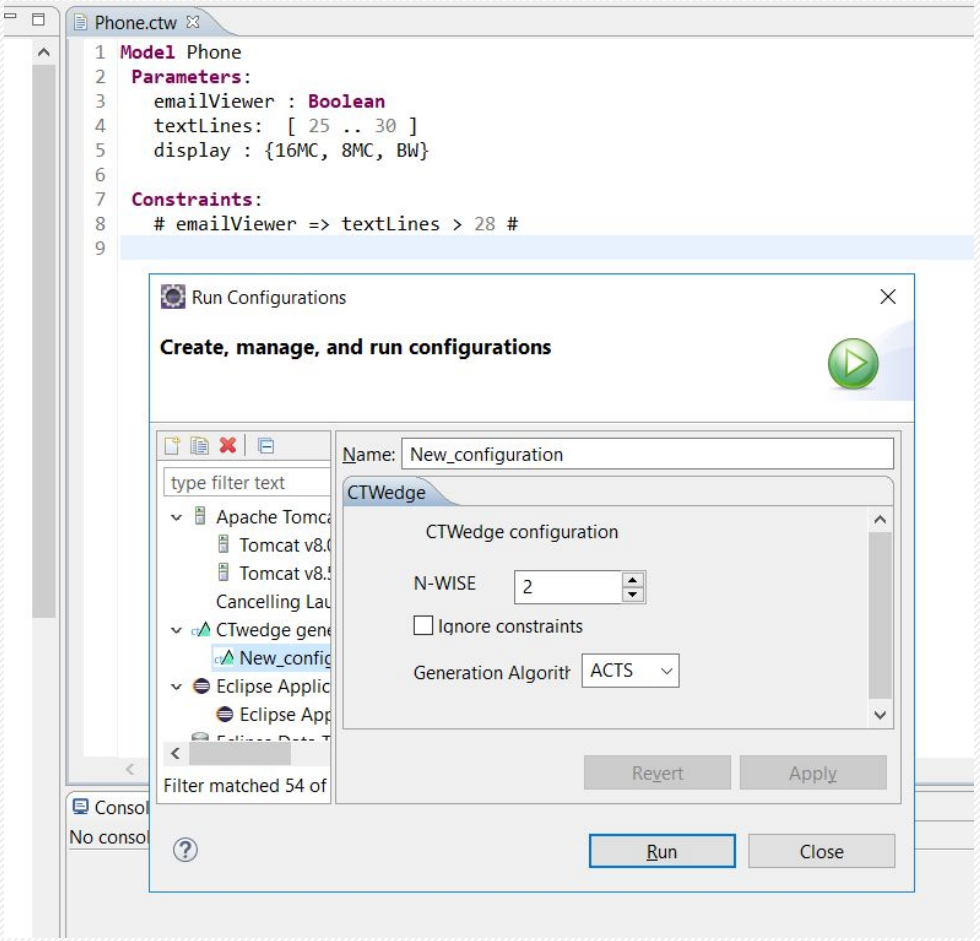
Help->Install New Software **(NB: install Xtext first)**: https://fmselab.github.io/ctwedge/ctwedge_update/

Create file (in a project) with extension .ctw . Eclipse opens it automatically with CTWedge Editor.

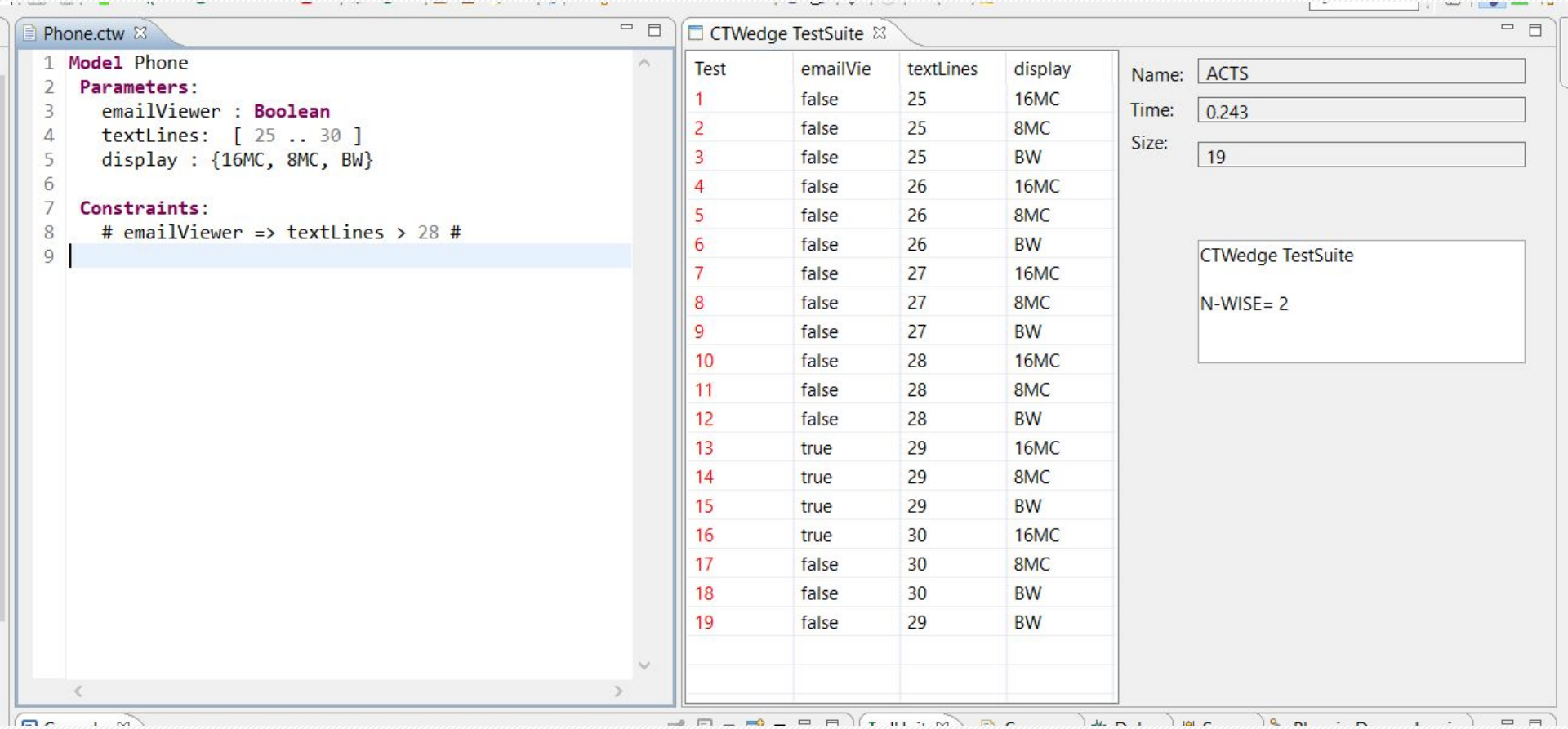
Right click on file or editor view -> Run As -> Run Configurations -> CTWegde generator -> Run

If it doesn't run the first time, right click (again) -> Rus as -> CTWedge

So far, it only works with ACTS (not CASA)



Combinatorial Test Suite in Eclipse



Phone.ctw

```
1  Model Phone
2   Parameters:
3     emailViewer : Boolean
4     textLines:  [ 25 .. 30 ]
5     display : {16MC, 8MC, BW}
6
7   Constraints:
8     # emailViewer => textLines > 28 #
9
```

CTWedge TestSuite

Test	emailVie	textLines	display
1	false	25	16MC
2	false	25	8MC
3	false	25	BW
4	false	26	16MC
5	false	26	8MC
6	false	26	BW
7	false	27	16MC
8	false	27	8MC
9	false	27	BW
10	false	28	16MC
11	false	28	8MC
12	false	28	BW
13	true	29	16MC
14	true	29	8MC
15	true	29	BW
16	true	30	16MC
17	false	30	8MC
18	false	30	BW
19	false	29	BW

Name: ACTS

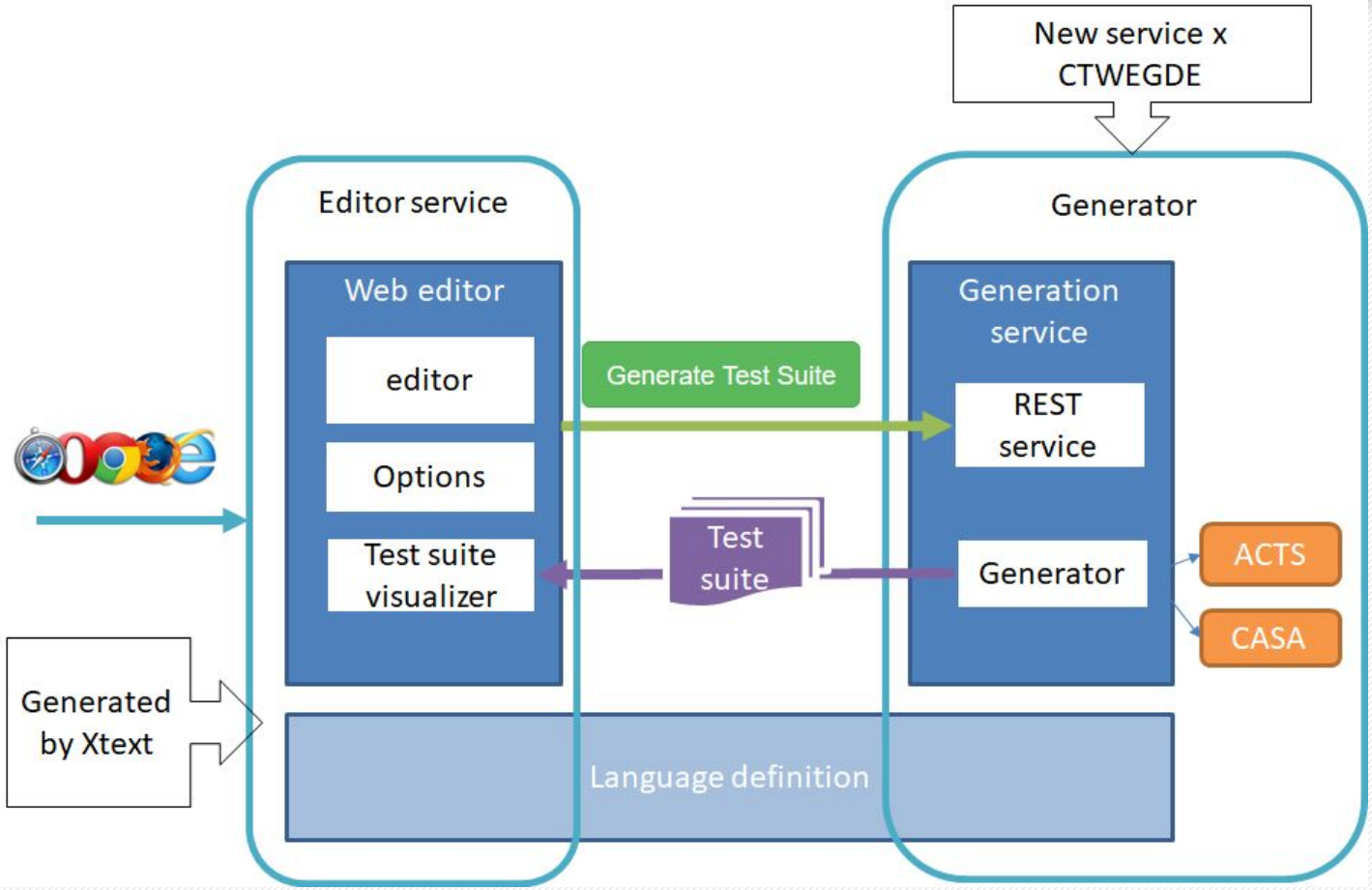
Time: 0.243

Size: 19

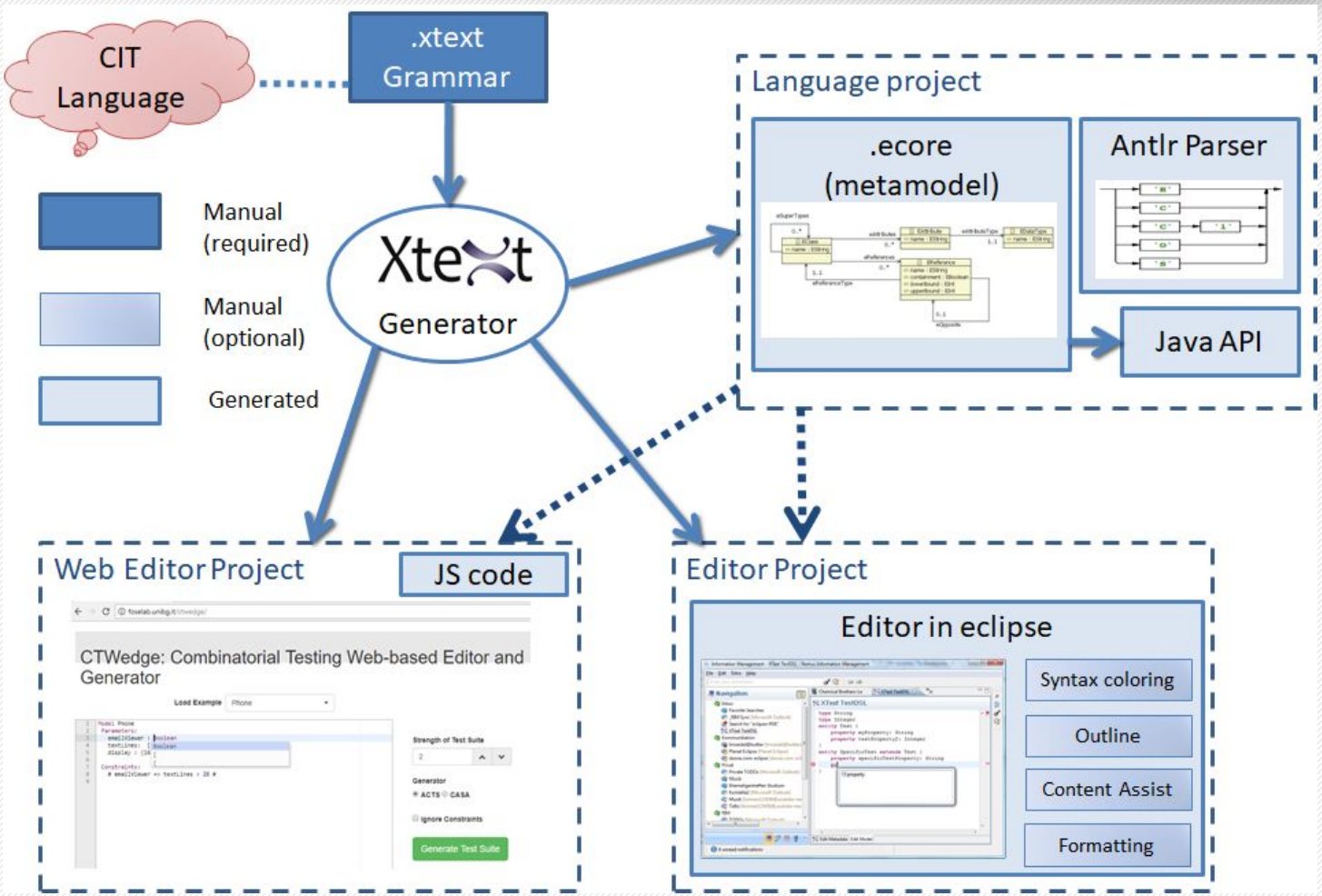
CTWedge TestSuite

N-WISE= 2

CTWedge Architecture



CTWedge Architecture (2)



unibg
UNIVERSITÀ DEGLI STUDI DI BERGAMO

Esercizi

Equazioni di 2° grado, RushHour, Magazzino

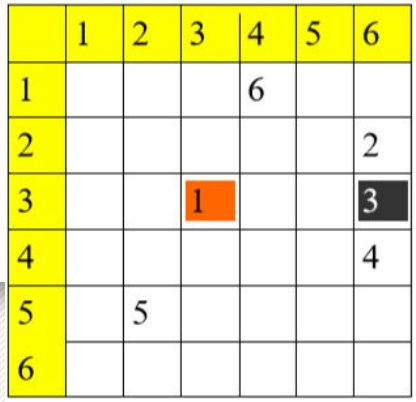
Esercizio 1 - Equazioni di 2° grado

Dato un metodo che restituisce il numero di soluzioni per una equazione di secondo grado:

static public int numSolutions(int a, int b, int c){....}

- implementa il metodo
- Fai input domain modeling: modella l'insieme degli input in modo da coprirli secondo diversi criteri di copertura. Partiziona l'insieme in triple in modo:
  - INTERFACE-BASED: considera una partizione per a, per b e per c e prendi dei valori significativi (però senza considerare il problema) per ognuna delle partizioni.
  - FUNCTIONALITY-BASED: dividi il dominio in partizioni a seconda del risultato che ti aspetti dal metodo che devi testare. Prendi un rappresentate per partizione e scrivi i casi di test in JUnit.
- Applica il combinatorial testing al precedente modello (solo interface-based) usando CTWedge, e sulla base della test suite restituita (provare con strength 2 e 3), scrivi i casi di test in Junit per ogni caso di test.

Esercizio 2 - RushHour



	1	2	3	4	5	6
1				6		
2						2
3			1			3
4						4
5		5				
6						

Nel RushHour una griglia 6x6 contiene 6 macchine numerate da 1 a 6 (ogni macchina per semplicità ha dimensione 1x1). L'obiettivo è portare - tramite spostamenti in celle libere adiacenti - la macchina numero 1 (rossa) all'uscita della griglia, che corrisponde alla cella con indici (3,6) (o 2,5 se le coordinate iniziano da 0) indicata dallo sfondo scuro. La configurazione iniziale del gioco è quella indicata.

Il programma Java ha un metodo **moveCar(int row, int col, int dir)** che muove l'auto sulla griglia (1-Nord, 2-Est, 3-Sud, 4-Ovest) solo se la cella di destinazione è all'interno della griglia e libera da auto. C'è anche il metodo **redCarAtExit** che dice se la macchina rossa ha raggiunto l'uscita.

Consegna: Applica il combinatorial testing al sistema semplificato con solo 3 auto in modo di avere tutte le configurazioni possibili. Prova a generare la copertura pairwise delle posizioni. Includi i necessari constraints. Se riesci aggiungi anche una variabile che indica se l'auto rossa è nella posizione di uscita.

Esercizio 3 - Magazzino

Un magazzino contiene 5 tipi di prodotti, e al massimo 100 unità di ogni prodotto. Si possono aggiungere al massimo 10 unità alla volta per uno stesso prodotto, e solo fino a raggiungere il livello massimo.

In Java scrivi una classe **Magazzino** che implementa (tramite un array di interi) il sistema sopra, con 3 metodi:

boolean insert(int productIndex, int addQuantity) : aggiunge il valore addQuantity (al massimo 10 prodotti) al prodotto identificato da productIndex, e ritorna true se l'aggiunta viene eseguita, false altrimenti. L'aggiunta non viene eseguita se l'indice di prodotto è errato, o se la quantità aggiunta non è corretta (non compresa tra 1 e 10), o se la nuova quantità che si otterrebbe con l'aggiunta supera le 100 unita.

boolean isFull(int productIndex) : dice se un certo prodotto ha raggiunto il massimo

boolean isFull() : restituisce se il magazzino è completamente pieno (tutti i prodotti sono la massimo)

Consegna: Rappresenta in combinatorial testing la chiamata del metodo insert, considerando come variabili del problema: **productIndex** e **addQuantity** (che sono i parametri del metodo), **returnedValue** (che è il valore ritornato), **nproductsOld** (il numero di prodotti prima della chiamata), **nproductsNew** (il numero di prodotti dopo la chiamata). Introduci i domini opportuni e i vincoli tra i paramteri. In questo modo hai anche l'oracolo.

Applica il combinatorial testing. Prova a generare la copertura pairwise. Prova a tradurre una riga dal tuo test in un test Junit per la classe Magazzino.