

Omelette

Raport z projektu

Sławomir Blatkiewicz Jakub Górniak Piotr Piechal
Bartosz Pieńkowski Barnaba Turek Michał Zochniak

4 czerwca 2011

1 co kto zrobić?

1.1 Barnaba Turek

W początkowej fazie projektu zajmowałem się głównie planowaniem architektury i sposobu działania kompilatora. Potem zacząłem implementować klasy, które miały leżeć u podstaw tej architektury (Takie jak **UMLObject**). Na tym etapie projektu znacząca część mojej aktywności polegała na pomaganiu niektórym kolegom w opanowaniu stosowanych technologii (głównie obsługi repozytorium) i czytaniu książki “Dive into Python”. Zajmowałem się też zarządzaniem repozytorium — tworzeniem struktury katalogów, paczek, wybieraniem i sprawdzaniem możliwości frameworków, używanych do testowania.

W dalszych etapach projektu zajmowałem się też modulem *common*, który zawierał generyczne klasy, na których miały bazować konkretne klasy środowiska graficznego reprezentujące obiekty na diagramie. Wprowadzałem też drobne zmiany w modułach *actions*, *code*, *parser*, *layouter*, *logging*. Moduły *common* i *uml* (a także ich testy) wymagały wielokrotnego poprawiania. Dzięki temu zyskałem praktykę w pisaniu dobrych (nie specyficznych) testów jednostkowych.

2 Opis architektury aplikacji

2.1 Opis i uzasadnienie technologii

2.1.1 Python

Jako język programowania wybraliśmy język **Python**.

Python to interpretowany język ogólnego zastosowania, pozwalający programować na wysokim poziomie abstrakcji. Język oparty jest na wielu paradygmatach programowania: programowaniu



obiektywnym, programowaniu funkcjonalnym i programowaniu imperatywnym. Typowanie w **Pythonie** jest dynamiczne, a typy są mocne. Zarządzanie pamięcią odbywa się dynamicznie.

Interpretery **Pythona** dostępne są na wszystkie popularne systemy operacyjne, wiele z nich jest otwartym oprogramowaniem. Sama specyfikacja języka zarządzana jest przez *Python Software Foundation* — niezależną organizację non-profit.

Głównym powodem, dla którego zdecydowaliśmy się na język **Python** to jego popularność. Język ten ma opinię języka o bardzo dobrej dokumentacji. Popularność wpływa także na dostępność dużej ilości otwartych bibliotek, z których wiele jest dojrzałych i wysokiej jakości.

Wielu z nas korzysta na co dzień z systemu GNU/Linux, gdzie wiele aplikacji jest napisanych w języku **Python**. Znajomość języka **Python** pozwoliłaby nam więc robić zmiany w aplikacjach, z których korzystamy na co dzień.

Jednym z powodów, dla którego wybraliśmy język **Python** był także fakt, że nikt z nas go nie znał. Wybranie nieznanego dotąd języka miało sprawić, że projekt będzie bardziej interesujący oraz zwiększyć kompetencje zawodowe członków zespołu ¹.

2.2 pyparsing

Pyparsing to jedna z bibliotek, które skłoniły nas do wybrania języka **Python** do realizacji tego projektu. Biblioteka **Pyparsing** to otwarte oprogramowanie.

Biblioteka pozwala w prosty sposób zbudować rekursywny analizator składniowy zstępujący. Gramatyka, pod kątem której analizowany ma być plik źródłowy, określana jest za pomocą języka **Python** w plikach źródłowych projektu (W naszym przypadku w pliku `lexer.py`). Pyparsing jest używany w takich projektach jak Django, pydot czy Graphite.

Parsowaną gramatykę opisuje się tworząc odpowiednie obiekty. Obiekty te mogą reprezentować symbole terminalne (wyrażenia regularne, zestawy znaków, pojedyncze znaki lub ich ciągi) lub ich produkcje. Każdemu obiektowi można przypisać akcję, która zostanie wykonana, gdy dany symbol zostanie wczytany.

2.3 Qt i PyQt

Qt to zestaw przenośnych bibliotek i narzędzi programistycznych dedykowanych do języków *C++* i *Java*. Pozwala budować graficzne interfejsy użytkownika w sposób zorientowany obiektowo.

Środowisko Qt to otwarte oprogramowanie. Środowisko dostępne jest na platformy *X11*, *Windows*, *Mac OS X*, *Haiku*, oraz na urządzeniach przenośnych opartych na Linuksie, *Windows CE* i *Symbian*.

Biblioteki **Qt** oprócz obsługi interfejsu użytkownika, zawierają także niezależne od platformy systemowej moduły obsługi proce-



¹W razie gdyby projekt nie okazał się hitem na miarę Napstera i musielibyśmy się jeszcze kiedykolwiek starać o pracę.

sów, plików, sieci, grafiki trójwymiarowej (OpenGL), baz danych (SQL), języka XML, lokalizacji, wielowątkowości, zaawansowanej obsługi napisów oraz wtyczek.

Dzięki bibliotece **pyQT** mogliśmy skorzystać ze środowiska **Qt** z poziomu języka **Python**.

Do graficznego projektowania interfejsu użytkownika użyliśmy programu **Qt Designer**.

Zdecydowaliśmy się na środowisko **Qt** ze względu na jego popularność, dostępność na platformy mobilne (wierzymy, że znajomość tego środowiska będzie dla nas cenna w przyszłości), fakt że jest to środowisko w pełni zorientowane obiektowo, oraz dostępność dojrzałej biblioteki do języka **Python**.

2.4 Nostests

Nostests, to biblioteka, której używaliśmy do testowania wytwarzanego przez nas oprogramowania. Biblioteka bazuje na module **unittest** dostarczanym w standardowej bibliotece języka **Python**.

Głównym powodem, dla którego zdecydowaliśmy się na korzystanie z tej biblioteki jest jej zdolność do automatycznego dodawania potrzebnych ścieżek. Ta cecha jest niezbędna do uruchamiania testów bez IDE, a alternatywą jest dopisanie kilku linii, ustawiających ścieżki na prawidłowe w każdym teście.

Ponadto **Nostests** udostępnia wiele rozszerzeń, ułatwiających pisanie testów, takich jak generatory testów.

2.5 Git i Github

Git to otwarty rozproszony system kontroli wersji. Pozwala na łatwe tworzenie i łączenie gałęzi rozwoju projektu, szybkie przemieszczanie się pomiędzy wersjami i sprawdzanie różnic pomiędzy nimi. **Git** jest stosowany w takich projektach, jak jądro systemu Linux i umożliwia bardzo wiele (nawet w stosunku do innych nowoczesnych rozproszonych systemów kontroli wersji). Niestety, powoduje to, że nie jest to system łatwy w nauce.

Zdecydowaliśmy się na system **Git**, ze względu na doskonały serwis *Github*²

Serwis *Github* posłużył nam nie tylko, jako przestrzeń do współdzielenia kodu. Serwis ten udostępnia wiele narzędzi, które były bardzo przydatne w czasie prac nad projektem. Korzystaliśmy z wykresów, które pomagały się zorientować, który członek zespołu co robi. Inna przydatna opcja, z której korzystaliśmy, aby zapewnić wysoką jakość kodu, to komentowanie kodu napisanego przez innych. Od kiedy został wprowadzony ulepszony system zadań (*Issues 2.0*), korzystaliśmy z niego w celu przechowywania wymagań i komunikacji z opiekunem projektu.



²<http://github.com>

Ponadto serwis pozwala na automatyczne powiadamianie o nowych zmianach w projekcie na kanale IRC, co było bardzo pomocne — motywowało do przeglądania kodu innych i usprawniało integrację.

2.6 Inne narzędzia

2.6.1 Zintegrowane środowisko programistyczne

W zespole nie zostało ustalone żadne środowisko programistyczne. Część członków zespołu używała środowiska *Eclipse*, część *Netbeans*.

Niektórzy członkowie zespołu tworzyli kod za pomocą zwykłych edytorów tekstowych.

2.6.2 Kanały komunikacji

Głównymi kanałami komunikacji wewnątrz zespołu były rozmowy w rzeczywistości i na kanale IRC.

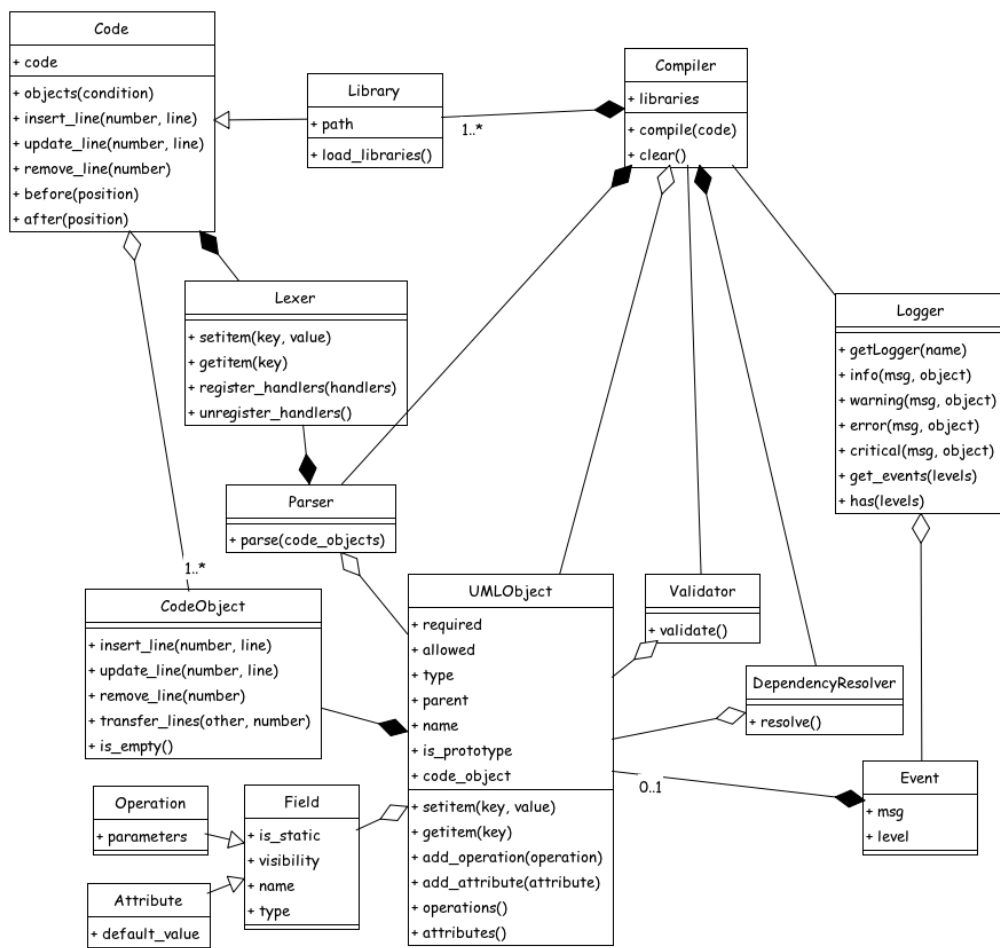
Ponadto, przez pewien czas korzystaliśmy z serwisu *Scrumd*³, służącego do zarządzania projektami prowadzonymi w metodyce Scrum. Głównym zastosowaniem tego narzędzia było przechowywanie wymagań. Wprowadzenie systemu *issues 2.0* w serwisie *GitHub* sprawiło, że serwis przestał być przydatny.

Korzystaliśmy także z serwisu *piratepad*⁴. Serwis ten pozwala na jednoczesną edycję plików tekstowych i łatwe dzielenie się nimi. Z serwisu korzystaliśmy we wczesnych wersjach projektu, kiedy ustalaliśmy ogólne cele i założenia.

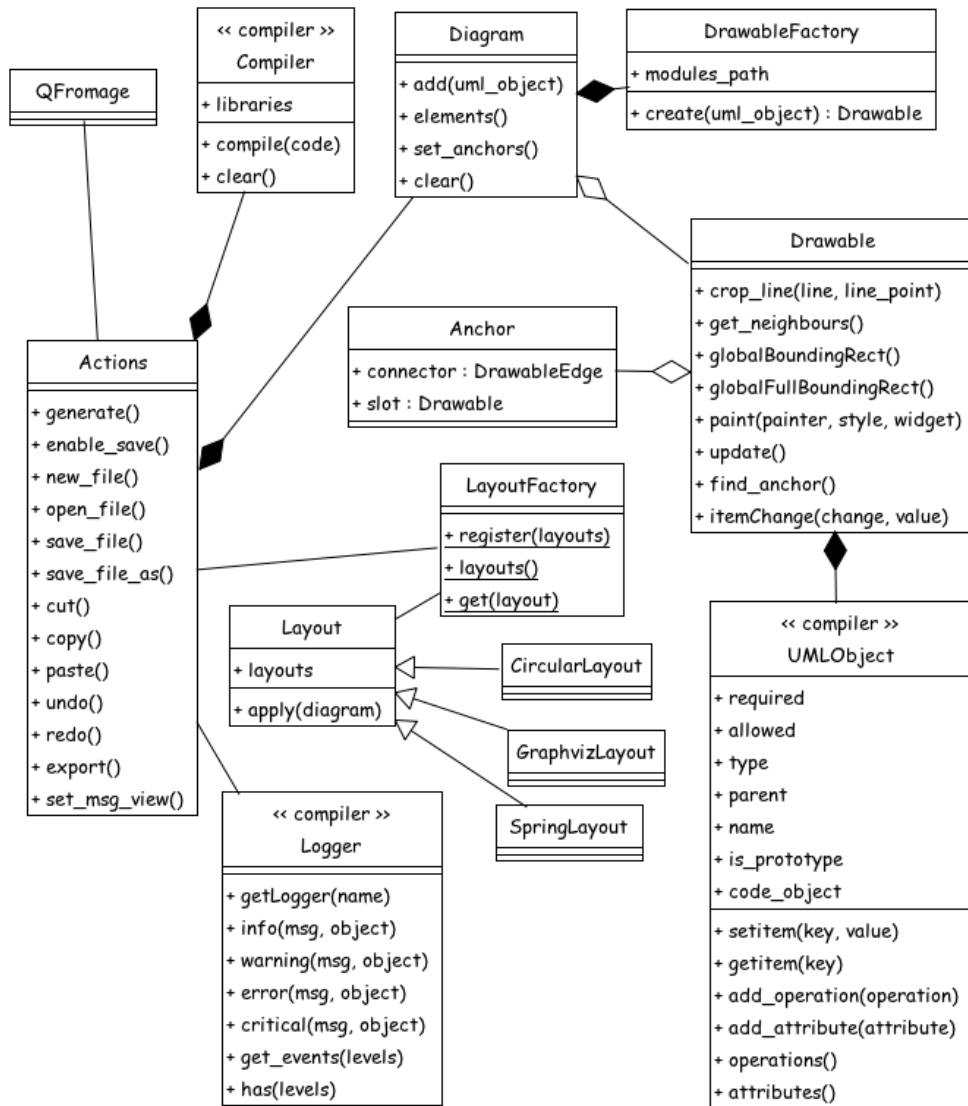
2.7 Diagramy Klas

³<http://scrumd.com>

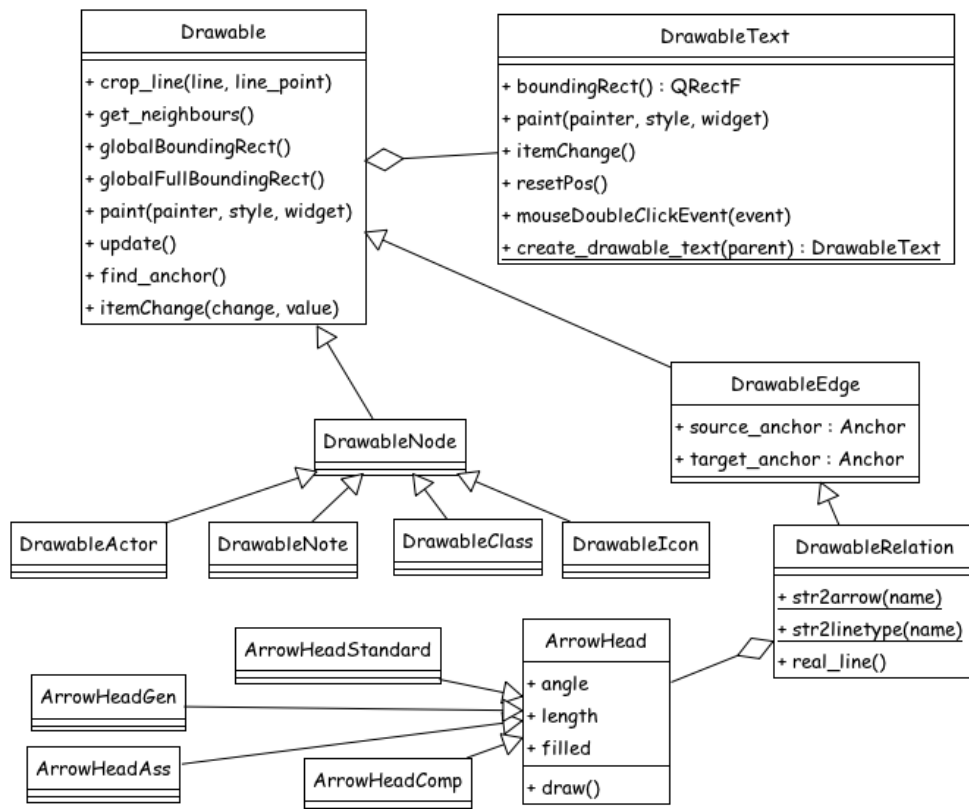
⁴<http://piratepad.net>



Rysunek 1: Diagram klas modułu compiler



Rysunek 2: Diagram klas modułu fromage



Rysunek 3: Diagram klas modułu modules