Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA AUTOMATYKI

PRACA INŻYNIERSKA

KONRAD MALAWSKI

PROTODOC
IMPLEMENTACJA ODPOWIEDNIKA NARZĘDZIA JAVADOC DLA JĘZKA DEFINICJI INTERFEJSÓW GOOGLE PROTOCOL BUFFERS

PROMOTOR:
dr inż. Jacek Piwowarczyk

Kraków 2011

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

. . . . . .

PODPIS

AGH

University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF AUTOMATICS

BACHELOR OF SCIENCE THESIS

KONRAD MALAWSKI

PROTODOC
DEVELOPMENT OF A JAVADOC TOOL EQUIVALENT FOR THE GOOGLE PROTOCOL BUFFERS INTERFACE DESCRIPTION LANGUAGE

SUPERVISOR:
Jacek Piwowarczyk Ph.D

Krakow 2011

TODO podziękowania

Spis treści

1. Wprowadzenie

1.1. Cel pracy

1.2. Obecnie dostępne narzędzia

2. Szczegóły implementacyjne

Jednym z celów projektu jest umożliwienie wykonania parsowania i wygenerowania dokumentacji bez konieczności instalacji zewnętrznych narzędzi (jakim byłby na przykład *protoc*). Aby sprostać temu wymaganiu konieczna jest implementacja parsera języka Protocol Buffers jako części JVM, nie wołając poza nią. Podjęto decyzję wykorzystania *Scala Parser Combinators* („kombinatorów parserów").

2.1. Parser

2.1.1. Wprowadzenie do kombinatorów parserów

TODO klasyfikacja, opisać że są lewo stronnie rekurencyjne etc.

http://en.wikipedia.org/wiki/Recursive_descent_parser

http://en.wikipedia.org/wiki/Left_recursion http://en.wikipedia.org/wiki/Par

http://stackoverflow.com/questions/17840/how-can-i-learn-about-parser-combi

2.1.2. Fragmenty implementacji

Przedstawić fragmenty parsera. Najlepiej go dać jako dodatek jednak w całości.

Parser w tym projekcie budowany jest za pomocą części języka *Scala* zwaną *„Parser Combinators"*.

```
def messageTypeDef: Parser[ProtoMessageType] = opt(comment)
                                    ~ "message" ~ ID ~ "{"
                                    ~ rep(enumTypeDef | instanceField |
                                        messageTypeDef)
                                    ~ "}" ^^ {
  case maybeDoc ~ m ~ id ~ p1 ~ allFields ~ p2 =>
    // utworzenie instancji ProtoMessageType
}
```

2.2. Verifier

Generalny opis dlaczego musiał powstac

Przedstawić jakie sprawdzania obsługuję. Pokazać że obsługuję importy oraz udowodnić dlaczego konieczny jest dodatkowy krok na to. No bo bez takiego kroku nie wiedziałbym czy przypadkiem gdzies indziej nie zostal zdefiniowany jakis message etc.

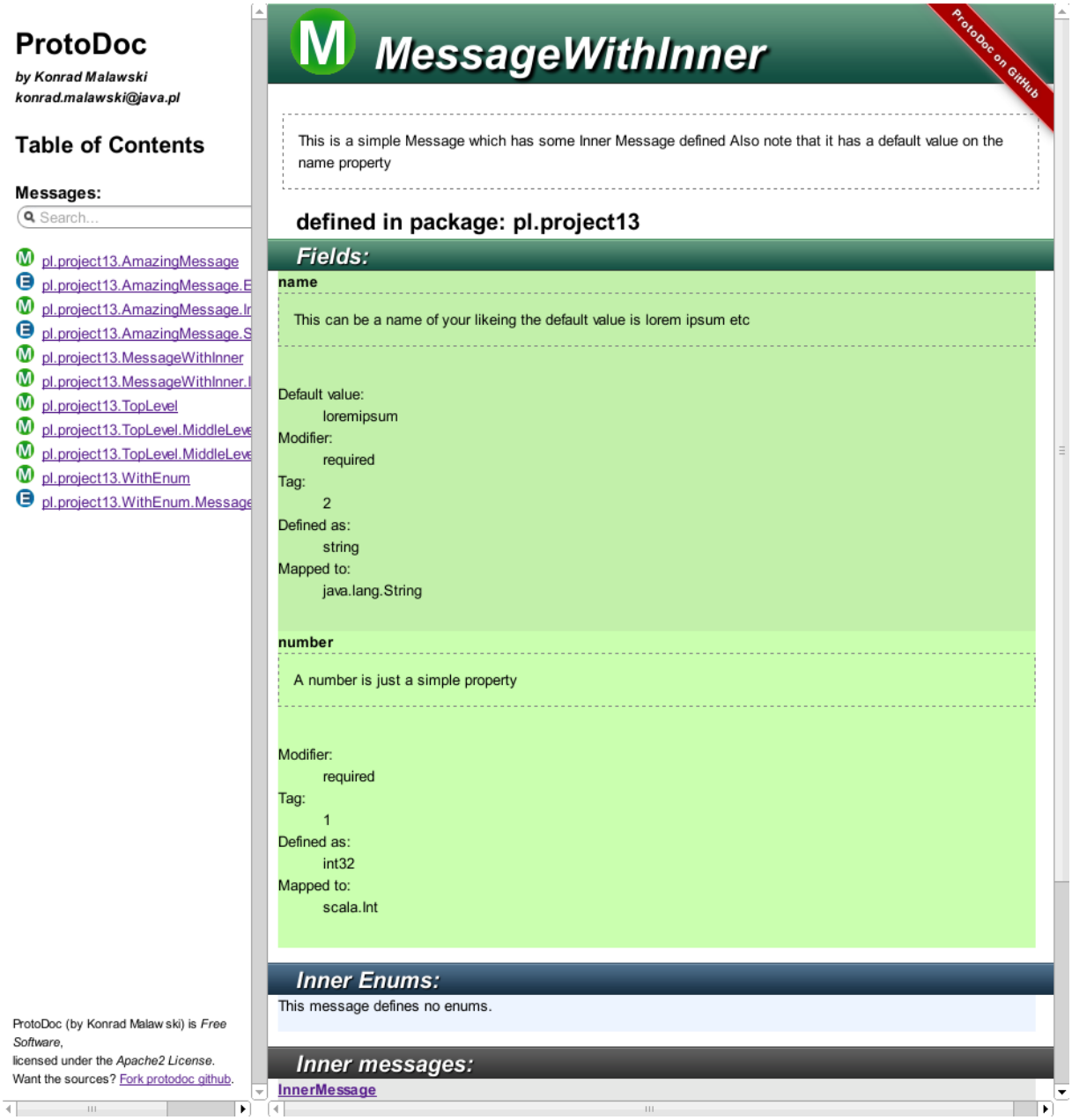
2.2.1. Obsługiwane weryfikacje

Moze sub sectiony o tych checkach oraz konkretne przykłady błędów i jak są komunikowane?

2.3. CodeGenerator

Generator kodu w tym przypadku jest bardzo prostą serią transformacji. Opisać, wspomnieć że korzystam z mustache etc.

2.3.1. Zrzuty ekranu wygenerowanej dokumentacji



3. Zastosowanie ProtoDoc do automatyzacji dokumentacji projektów

4. Przykład

Podajemy takie wejście:

```
message Person {
  required int32 id = 1;
  required string name = 2;
  optional string email = 3;
}
```

Następnie wykonanie:

A ostatecznie otrzymujemy taką stronę: `http://protodoc.project13.pl/sample`.

5. Dodatki

A. Podstawy języka Scala

Celem tego dodatku jest przybliżenie czytelnikowi języka „Scala” aby w wystarczająco płynny sposób mógł czytać przykłady kodu używane w tym dokumencie.

A.1. Krótka historia języka

Język Scala („Scalable Language”) najłatwiej jest przedstawić jako hybrydę dwóch znanych nurtów programowania: programowania obiektowego oraz funkcyjnego, wraz z powiązanymi z nimi językami programowania. Twórca języka Scala, Martin Oderski[Ode07]

Jako konkretnych „rodziców” można by wskazać:

– **Java** - jako reprezentant nurtu obiektowego

– oraz języki: **Haskell**, **SML** oraz pewne elementy języka **Erlang** (głównie *Actor model*).

A.2. Podstawy

Ta sekcja służy przybliżeniu czyletnikowi języka *Scala* na poziomie wystatczającym aby swobodnie czytać przykłady kodu umieszczone w tej pracy. W niektórych przykładach pomijane są przypadki skrajne lub nietypowe, celem szybkiego oraz jasnego przedstawienia minimum wiedzy na temat języka aby móc swobodnie go „czytać”.

Scala jest językiem statycznie typowanym posiadającym lokalne „Type Inferrence”. Pozwala to kompilatorowi *scalac* na „odnajdywanie” typów wszystkich zmiennych oraz typów zwracanych przez metody podczas kompilacji, bez potrzeby definiowania ich wprost. System ten

Użycie nawiasów `()`, średnika `;` oraz kropki `.` jest analogiczne jak w przypadku Javy, jednak w wielu przypadkach opcjonalne gdyż kompilator jest w stanie wydedukować gdzie powinny się znaleźć.

```
val value = Option(42);
val other = value.orElse(0);

// moze zostac zastpione
val value = Option(42)
val other = value orElse 0
```

Jednym z ciekawych przykładów stosowania notacji bez nawiasów i kropek jest *ScalaTest*[1] (przy którego pomocy pisano testy w tym projekcie). Przykładowa *asercja* napisana w *DSL*u definiowanym przez tą bibliotekę wygląda następująco:

```
messages should (contain key ("Has") and not contain value ("NoSuchMsg"))
```

A.3. Scala Parser Combinators

[1] ScalaTest - framework do testowania - http://www.scalatest.org

B. Google Protocol Buffers

W tym dodatku zostanie omówiona idea oraz szczegóły implementacyjne stojące za Google Protocol Buffers.

B.1. Krótka historia języka

B.2. Przykładowe definicje wiadomości

B.3. Dostępne narzędzia

```
message Person {
  required int32 id = 1;
  required string name = 2;
  optional string email = 3;
}
```

C. Bibliografia

Bibliografia

[Ode07] Martin Odersky. *Programming in Scala*. 2007.