Benjamin Wallin
Mikael Andersson
Grupp: IP1-A-1

Systemdokumentation

Version 1.2

Innehåll

Introduktion 2
- Överblick 2
- Vad är Xtat? 2
- Bakgrund 2

Lexikalisk analys (scanning) 3
- Tokenkontroll och semantisk analys 3

Använda tekniker och referenser 6

Källor 7

Bilagor 8
- Teknisk specifikation 8
- Grammatik (BNF) 15

Introduktion

Överblick

Denna rapport är ett resultat från ett projekt på programmet Innovativ programmering (IP) vid institutionen för datavetenskap (IDA) Linköpings tekniska högskola (LiTH). Projektet har utförts under andra terminen inom kursen TDP019 Projekt: Datorspråk.

Detta dokument innehåller information om språket Xtat och dess uppbyggnad. Den är till för dig som är intresserad av att lära dig mer om hur saker och ting fungerar under skalet.

Vad är Xtat?

Xtat som språket heter kommer ursprungligen från att vi ville göra ett språk som enkelt kan hantera Xml-filer. Xml är ett format av datafiler som kan innehålla stora mängder av data på ett strukturerat sätt. Utifrån data kan man med Xtat skapa statistik med diagram och tabeller. Språket körs på en webbserver för att pressentera resultatet i en webbläsare. I namnet så står X för Xml och tat, som ger stat i uttal, vilket är en förkortning av statistik.

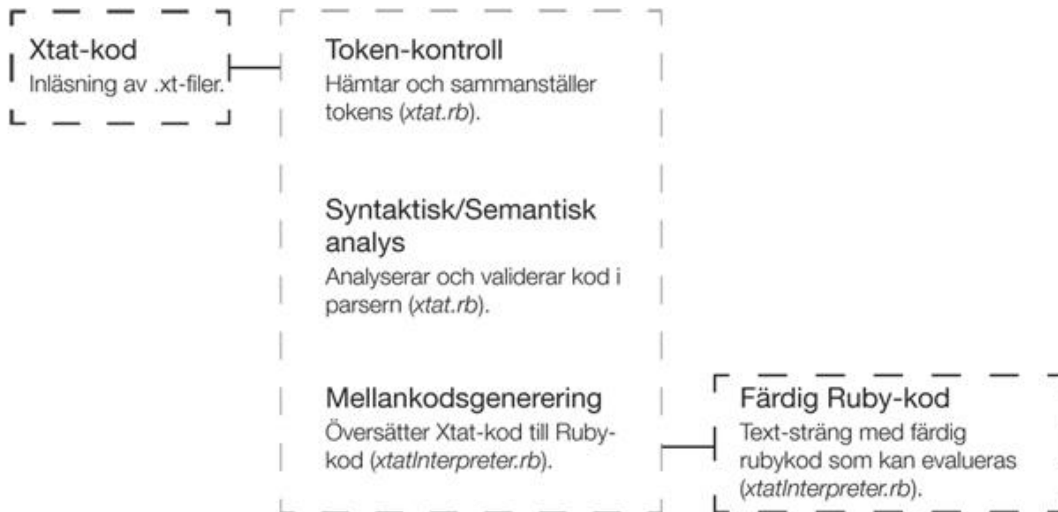
Målet var även att hitta tecken och kombinationer som gör att man får ett flyt vid programmeringen och där av har vi använt oss av: §, eftersom den är lätt att nå och skriva.

Bakgrund

Xtat började växa fram som ett skolprojekt våren 2010 med Mikael Andersson och Benjamin Wallin som utvecklare. Målet var att skapa ett språk som förenklar arbetet för personer som arbetar med statistik och Xml. Då både Mikael och Benjamin är intresserad av webbutveckling så gjordes Xtat för att köras på en webbserver som användarna sedan kan anropa via webbläsaren och där få upp resultatet, likt språken Php, Python med flera.

Då kursen byggde på att man skulle skapa programmeringsspråket i Ruby så konstruerade vi Xtat med hjälp av Ruby-kod i grunden.

Lexikalisk analys (scanning)



Figur 1: Xtat som helhet fungerar enligt illustration ovan.

När ett Xtat-dokument - vilket är ett dokument som består av Xtat-kod som avgränsas med <?xt-taggar i ett Html-dokument - öppnas med Apache genom en webbläsare börjar vår modifierade Apache-modul (mod_ruby) med att leta upp och skicka vidare Xtat-kod till vår parser (xtat.rb).

Parser-funktionen hittar sedan rätt kombinationer av våra nyskapade tokens och matchar dem med givna regler som finns i parsern. När en funktionalitet i Xtat-koden matchas på korrekt sätt skickas uppgifterna från matchningsfunktionen vidare till mellankodsgenereringen där all kod förbereds och lagras tills parsern har gått igenom hela filen.

När hela filen är parsad och genererad körs vår nygenererade Ruby-kod som vanligt med Ruby och presenteras sedan tillsammans med den Html som fanns i filen från början i webbläsaren.

Tokenkontroll och semantisk analys

Kontroll av kod som läses in bygger helt på en Rdparser med match-funktioner.

Tokenizer

Vår tokenizer samt syntatiska analys är skapad på ett sätt där hela processen samarbetar på samma gång.

När filen läses in körs en tokenisering av all data som är uppdelat i ett antal punkter vi anser vara en relevant uppdelning. Tokenizern jobbar med följande tecken:

Logiska operatorer

- Jämförelseoperatorer
- Tilldelningsoperator
- Kommentartaggar
- Alfanumeriska tecken inkl. svenska tecken
- Siffror
- Sträng
- Ny rad
- Mellanslag

Det finns även undantag av där tecken kontrolleras även i semantik-delen.

Exempel på token-funktioner taget från xtat.rb:

```
token(/\/\*/) {|m| m }
token(/\*\//) {|m| m }
```

Semantisk analys

Vår syntaktiska/semantiska analys i Xtat skiljer sig inte mycket ifrån hur parsern skannar av tokens i avsnittet ovan. Skillnaden är att den sätter samman de olika tokens som tonenizern har genererat samt testar sammansättningen i olika kombinationer för att hitta och tolka rätt funktionalitet.

Exempel på matchnings-funktioner taget från xtat.rb:

```
match(:identifier, :digit)
      {|identifier,digit|  [identifier,digit] }
match(:digit) {|digit| digit }
```

Mellankodsgenerering

Efter att parsen har gått igenom och kontrollerat så att programmeraren har skrivit korrekt Xtatkod skapas sedan speciella klasser i xtatInterpreter.rb som omvandlar Xtatkoden till körbar rubykod.

Evaluering

Då mellankodsgeneratorn har körts och all kod är omvandlad till validerad rubykod så körs evalueringen av rubykoden direkt på en server för att sedan visa resultatet i t.ex. en webbläsare.

Använda tekniker och referenser

När vi har utvecklat språket har vi använt oss av utvecklingsmiljön Emacs under Linux (Gentoo och Ubuntu), vi har sedan testkört koden via Ruby IRB. Vi har även testat vår kod på en special konfigurerad Apache-server. För att lösa versionshantering och filhantering så laddas alla filer upp på en server via svn.

Språket är byggt med hjälp av rdParse som är ett färdigbyggt verktyg för att kunna matcha olika regler.

Källor

Diverse källor

Documenting the Ruby Language [www]. James Britt, 2010. Tillgängligt på <http://ruby-doc.org/>. Hämtat 9 maj 2010.
Webbplats med dokumentationen till Ruby.

Formella språk, grammatik, parsing - IDA [www]. Anders Haraldsson, 2008-01-15. Tillgängligt på <http://www.ida.liu.se/~TDP019/material/oh-formella-bredd.pdf>. Hämtat 9 maj 2010.
Föreläsnings-/kursmaterial om formella datorspråk, grammatik och parsing.

mod_ruby-1.3.0 [www]. Shugo Maeda, 2008-11-08. Tillgängligt på <http://www.modruby.net/en/>. Hämtat 9 maj 2010.
Apache-modul för Ruby.

Dave, Thomas with Chad Fowler and Andy Hunt (2009). *Programming Ruby 1.9. The Pragmatic programmers' guide*. Pragmatic Bookshelf.
Inlärnings- och referensmaterial för Ruby 1.9.

Lästips

"Python v2.5.2 documentation" [www]. Python Software Foundation, 2008-12-23. Tillgängligt på < http://www.python.org/doc/2.5.2/ref/grammar.txt>. Hämtat 9 maj 2010.

Bilagor

Teknisk specifikation

Klasser och dess relationer

Program

initialize(stmt_list)	Tar emot en **stmt_list**-klass.
eval	Evaluerar all genererad programkod.

Evaluerar hela programmet. Tar emot **stmt_list** som är all kod som skapats i korrekt ordning.

Array_class

initialize(var, args='')	Tar en array som inparameter där **var** är namnet och **args** är dess värden i form av array.
eval	Returnerar körbar Ruby-kod.

-

Var_class

initialize(var, value)	Tar ett namn och ett värde som inparametrar och skapar av dessa en variabel.
eval	Returnerar körbar Ruby-kod.

När variabeln har skapats läggs även dessa in i en global array så andra klasser kan använda denna nyskapade klass.

Print_class

initialize(string)	Tar en sträng som inparameter.
eval	Returnerar en utskrivbar sträng med rubys puts-funktion.

Klassen skapas med hjälp av en textsträng som skall skrivas ut, när klassen evalueras returneras följande:

```
puts 'detta kommer att skrivas ut'
```

StmtList_class

initialize(stmtlist)	Tar kör-uppgifter som inparameter.
eval	Returnerar ett objekt av matchade stmt-regler.

StmtList_class returnerar ett givet antal rader (t.ex. en funktion eller ett en if-funktion) i programmet som körbar Ruby-kod. Denna klass kan ses som en sammanställning av hela programmet om man lägger ihop alla **StmtList_class**-objekt.

Stat_circle

initialize(args)	Skapar ett cirkeldiagram utifrån den data som kommer där args är av speciella typen **Stat_args**.
eval	Returnerar körbar Ruby-kod.

Skapar ett cirkeldiagram med hjälp av google charts API.

Stat_bar

initialize(args)	Skapar ett stapeldiagram utifrån den data som kommer där args är av speciella typen **Stat_args**.
eval	Returnerar körbar Ruby-kod.

Skapar ett stapeldiagram med hjälp av google charts API.

Stat_args

initialize(var,names=nil, type="%", size=Digit.new(200))	Får in argumenten för de olika diagrammen och ger diverse standardvärden om invärde saknas.
var	Returnerar variabeln som innehåller siffror.
names	Returnerar variabeln som innehåller namnen.
type	Returnerar vilken typ diagrammet ska visa, % eller antal.
size	Returnerar storlek på diagrammet
eval	Returnerar inget.

-

Func_class

initialize(name,args, stmt_list='',ret_stat='')	Skapar en funktion av inparametrarna. **name** är namnet på funktionen, **args** är inparametrarna på funktionen, **stmt_list** är innehållet och **ret_stat** är returvärdet.
eval	Returnerar körbar Ruby-kod.

Skapar en funktion enligt Xtat-syntax och konverterar till valid Ruby-kod.

FuncCall_class

initialize(func_name, args,var='')	Kallar på en befintlig funktion där var är namnet på variabeln som tilldelas funktionens värde, **func_name** är den befintliga funktionens namn och **args** dess inparametrar.
eval	Returnerar körbar Ruby-kod.

Kallar på en tidigare skapad funktion.

For_class

initialize(var,expression, count,stmt_list)	Skapar en for-loop där **var** är räkne-variabelns ingångsvärde, **expression** är villkoret, **count** är det som sker med räkne-variabeln under varje upprepning och **stmt_list** for-loopens innehåll.
eval	Returnerar körbar Ruby-kod.

Count-värdet i inparametern körs med fördel som ett **Increment_class**-objekt.

While_class

initialize(expression, stmt_list)	Uttrycket är **expression** samt innehållet i while-loopen är **stmt_list**.
eval	Returnerar körbar Ruby-kod.

-

Increment_class

initialize(var_name, type)	**var_name** är variabeln som kommer att öka eller minska med ett och **type** bestämmer om det är av decrement-typ eller increment-typ.
eval	Returnerar körbar Ruby-kod.

-

If_class

initialize(expression, stmt_list,else_stmt_list='', else_if_stmt_list='')	Skapar en if-funktion där huvuduttrycket är **expression**, **stmt_list** är dess innehåll, **else_stmt_list** är else-uttrycket samt **else_is_stmt_list** är en array av alla else-if-uttryck.
eval	Returnerar körbar Ruby-kod.

Denna klass kan skapa både enkla if-, else-if- och if med else-funktioner. **else_if_stmt_list** kommer som en array där varje post består av [**expression**, **stmt_list**] (uttryck och värde).

Expression_class

initialize(expr,operator='', expr2=false)	Tar ett villkorsuttryck i inparametrarna.
eval	Returnerar körbar Ruby-kod.

Expression_class returnerar ett Expression_class-objekt om den endast får en inparameter som värde (exempelvis det boleanska uttrycket true, eller talet 11). Är alla inparametrarna använda returnerar klassen en array bestående av tre objekt som är av typerna som angavs i inparametrarna i initialize-funktionen.

Math_class

initialize(expr,operator, expr2)	Tar ett enkelt matematiskt uttryck som invärde (t.ex. 4 + 6 eller 3 * 4).
eval	Returnerar körbar Ruby-kod.

Klarar av de fyra räknesätten (+, -, / och *) samt exponenter (**).

Args_class

initialize(args)	Tar ett argument (eller flera som en array) som invärde.
eval	Returnerar körbar Ruby-kod.

Tar flertalet olika typer som invärde (se BNF-grammatik för mer information).

StringStmt_class

initialize(string)	Tar ett sträng-värde som inparameter.
eval	Returnerar körbar Ruby-kod.

Ett strängvärde omfamnat av ’-tecken returneras.

Comments_class

initialize(string)	Tar ett strängvärde radvis som är omfanmat av kommentarstecken - /*Sträng-värde */ - som inparameter.
eval	Returnerar körbar Ruby-kod.

Kommentarer returneras radvis som Rubykommentarer (# En kommentar). Ex:

```
/* Rad ett
   Rad två  */
```

Blir:

```
# Rad ett
# Rad två
```

Upper_letter

initialize(letter)	Tar ett invärde bestående av versaler.
eval	Returnerar körbar Ruby-kod.

Tar versaler som inparameter och skapar ett objekt av en textsträng.

Lower_letter

initialize(letter)	Tar ett invärde bestående av gemener.
eval	Returnerar körbar Ruby-kod.

Returnerar ett objekt som består av gemener .

Operator

initialize(operator)	Tar ett operatorvärde som inparameter (Math, Comp eller Log).
eval	Returnerar körbar Ruby-kod.

-

Digit

initialize(value)	Tar ett invärde och skapar ett tal-objekt.
eval	Returnerar körbar Ruby-kod.

-

Boolean

initialize(bool)	Skapar ett objekt av ett booleanskt uttryck som invärde.
eval	Returnerar körbar Ruby-kod.

Skapar ett objekt av ett booleanskt uttryck som den tar som invärde i inparametern.

Value_class

initialize(bool)	Skapar ett objekt av ett värde/sträng.
eval	Returnerar en sträng

Används i t.ex. klassen funccall_class för att ha ett objekt som lagras i varabellistan.

Grammatik (BNF)

Grammatiska delen

```
<program>        ::=    “<?xt” <stmt_list> “xt?>”
<stmt_list>      ::=    (<stmt_list>)* (<stat_circle> | <stat_bar> |
                        <function> | <function_call> | <list_iter> |
                        <if_stmt> | <comment> | <print_stmt> |
                        <assignment> | <var_list>)+
<stat_circle>    ::=    “circle(”<stat_args>“)”
<stat_bar>       ::=    “bar(”<stat_args>“)”
<stat_args>      ::=    (<var>){1,2} (“st” | “%”){0,1}
<function>       ::=    “func” <func_name> “(” (<args>)* “)”
                        (<stmt_list>)* (<return_stmt>){1} “end”
<return_stmt>    ::=    “return” <args>
<function_call>::=      <var> <assignment_operator> <func_name> “(”
                        (<args>)* “)”
<list_iter>      ::=     <it_for> | <it_while>
<it_for>         ::=    “for “(” <var_list> “,” <expression> “,”
                        <stmt_list> “)”<stmt_list> “end”
<it_while>       ::=    “while” “(”<expression>“)” <stmt_list>
                        “end”
<func_name>      ::=    <identifier>
<comment>        ::=    “/*” <comment_lines> “/*”
<comment_lines>::=      (<identifier> | (<comment_lines>
                        <identifier>))*
<print_stmt>     ::=    “print” (<var_list> | <string_stmt>){1}
<assignment>     ::=    (<increment>) | ((|<var> | <const_var>)
                        <assignment_operator> (<bool> | <math> |
                        <string_stmt> | <array>))
<increment>      ::=    <var> (“++” | “--”)
<var_list>       ::=    <const_var> | <var>
<array>          ::=    ”array(” (<args>)* ”)”
<var>            ::=    (<var_sign> <lowercase_var>){1}
<const_var>      ::=    (<var_sign> <uppercase_var>){1}
<if_stmt>        ::=    “if” “(” (<if_expression>)+ “)” <stmt_list>
                        (“end” | <else_stmt> | <else_if_stmt>
                        <else_stmt>)
<else_if_stmt> ::=      (<else_if_stmtm>)* “elseif” “(”
                        <if_expression> “)” <stmt_list>
<else_stmt>      ::=    “else” <stmt_list> “end”
<oneliner_if_stmt>::=“(” <expression> “)” “?” <stmt_list> “:”
                        <stmt_list>
```

```
<if_expression>::=    <expression>  |  "("  <if_expression>  ")"  |
                      <if_expression>     <log>_operator>     "("
                      <expression> ")"
<expression>   ::=    (<var_list> | <bool>) (<operator>
                      (<var_list> | <bool>))*
<math>         ::=    (<math> <math_operator>
                      <math> | <math_prio>)
<math_prio>    ::=    (<math_prio> <math_prio_op>
                      <math> | <prio_math>)
<math_pare_prio>::=   ("(" <math> ")" | <digit>)
<args>         ::=    ((<var_list> <identifier> <string_stmt>){1}
                      ",")*
                      (<var_list> | (<digit>)* |
                      <bool> | <string_stmt>)+
<string_stmt>  ::=    "'"<identifier>"'"
<identifier>   ::=    (<identifier>)* (<letter> | <digit>)+
<letter>       ::=    <lowercase> | <uppercase> | "_"
<var_sign>     ::=    "§"
<operator>     ::=    <log_operator> | <comp_operator>
                      | <math_operator> | <assignment_operator>
```

Lexikaliska delen

```
<lowercase>             ::=  "a"..."ö"
<uppercase>             ::=  "A"..."Ö"
<lowercase_var>         ::=  "a"…"z" 0…9 "_"
<uppercase_var>         ::=  "A"…"Z" 0…9 "_"
<digit>                 ::=  "0"..."9"
<bool>                  ::=  "1"|"0"|"true"|"false"|"TRUE" |"FALSE"
<log_operator>          ::=  "|" | "&"
<comp_operator>         ::=  "==" | "!=" | "<=" | ">=" | "<" | ">"
<math_operator>         ::=  "+" | "-"
<math_prio_op>          ::=  "*" | "/" | "**"
<assignment_operator> ::=  "="
```