WSC

v0.5

Relazione di Progetto

Pietro Spinella (pietro.spinella@gmail.com)
Eugenio Siciliano (eugenio.siciliano@gmail.com)

Descrizione

WSC è l'acronimo di **W**eb **S**cripting **C**ompiler. Si tratta di un traduttore da un linguaggio progettato per l'automazione di richieste web, chiamato **WSL** (**W**eb **S**cripting **L**anguage) ad una struttura XML standard 1.0 descritta dalle specifiche **KMS** (**K**ool **M**essaging **S**ervice Specification).

(Breve digressione su KMS)

KMS è l'acronimo di **K**ool **M**essaging **S**ervice Specification. Si tratta di un browser automatizzato in cui le comuni operazioni svolte dall'utente (login, logout, compilazione di form, inserzione di codici ottici, ecc.) vengono effettuate via codice tramite un motore interno che interpreta il formato XML definito dalle specifiche sopra accennate.

Linguaggio sorgente

Il linguaggio sorgente è stato progettato cercando di mantenere la maggiore semplicità possibile senza rinunciare, allo stesso tempo, alla completezza (in quanto adesione alle specifiche) ed alla compattezza.

Non trovano collocazione i tipici costrutti dei linguaggi più evoluti come cicli e salti condizionati (con qualche eccezione per i cicli) in quanto da un lato, la sua finalità non è quella di un linguaggio general-purpose, dall'altro non avrebbe molto senso in quanto il linguaggio destinazione non permette di utilizzare tali costrutti.

Per convenzione chiamiamo *servizio* ogni file generato dal traduttore. Ad ogni servizio potranno essere associate una o più *procedure* (come verrà spiegato in seguito).

La definizione di un servizio avviene attraverso una struttura del tipo:

```
SERVICE
	id		= <string>
	name		= <string>
	icon		= <string>
	description	= <string>
	version		= <string>
	author		= <string>
	enabled		= <string>
```

La keyword *SERVICE* definisce il servizio, ed è possibile definirne uno solo all'inizio del sorgente.

Gli *attributi* del servizio (che corrisponderanno agli attributi del tag <service> nell'XML) sono definiti da un'assegnazione in cui nella parte sinistra c'è sempre un identificatore mentre nella parte destra c'è sempre una stringa.

Gli attributi tra di loro non devono contenere né commenti né ritorni a capo. Se gli attributi vengono ripetuti prevale l'ultimo della sequenza. Il controllo semantico della correttezza delle stringhe non viene effettuato.

Ad ogni servizio è possibile associare 0, 1 o più *INPUT*. Ogni struttura *INPUT* è definita nel seguente modo:

```
INPUT <identifier>
	content		= <string>
	type		= <string>
	regex		= <string>
	name		= <string>
	description	= <string>
	defaultValue	= <string>
```

La keyword *INPUT* definisce un input che verrà chiesto all'utente non appena sarà necessario (questo compito non spetta al traduttore).

Per gli input valgono inoltre le stesse considerazioni sugli attributi fatte per la keyword *SERVICE*.

Da notare che la struttura *INPUT*, a differenza di quella *SERVICE*, permette di assegnare un identificatore all'input in modo da poterlo utilizzare all'interno del codice (source).

Una procedura (di un ipotetico servizio) potrebbe essere definita nel seguente modo:

```
PROCEDURE <identifier>

	<identifier> = <string>
	<identifier> = <string> + <integer> + <identifier>

	CYCLE <integer>
		<identifier> = REQUEST <string> <string>
	BREAK

	MATCHES <identifier> <string>
		onTrue = BREAK
		onFalse = CONTINUE
		onTrueOutput = <string>
		onFalseOutput = <string>

RETURN
```

L'identificatore accanto alla keyword *PROCEDURE* identifica la procedura ed, importante, dà il nome alla procedura stessa nel file generato dal traduttore.

Ogni procedura viene terminata dall'istruzione RETURN.

All'interno di ogni procedura si possono trovare assegnazioni, comandi e cicli.

Un'assegnazione è sempre caratterizzata da una parte sinistra (identificatore), il simbolo di assegnazione = ed il valore assegnato. Tale valore può essere la concatenzazione tra interi, stringhe ed identificatori. Se si sommano due interi il risultato è la somma ritornata come tipo stringa.

L'unico comando attualmente definito è *REQUEST* il quale, se trova una stringa alla sua sinistra crea una richiesta di tipo GET, se trova due stringhe crea una richiesta di tipo POST in cui la seconda stringa corrisponde al POST-DATA. Il risultato può essere opzionalmente assegnato ad un identificatore.

Il costrutto *CYCLE* definisce la ripetizione di una sequenza di comandi per un numero fissato di volte. La keyword deve essere seguita da un intero che specifica quante volte il ciclo và eseguito. Il singolo ciclo termina non appena si incontra la keyword *BREAK*, a questo punto il contatore viene decrementato e si procede fin tanto che non arrivi a zero, in tal caso il ciclo termina.

Questo costrutto non trova un equivalente nel linguaggio destinazione, e viene pertanto tradotto come sequenza di istruzioni dove le espressioni sono sostituite dal loro risultato.

I salti condizionati non sono presenti, un costrutto che vagamente li ricorda fa uso della keyword MATCHES: agli attributi *onTrue* ed *onFalse* della keyword *MATCHES* è possibile assegnare le keyword *CONTINUE* o *BREAK*.

Gli attributi *onTrueOutput* ed *onFalseOutput* servono invece a specificare un testo che verrà stampato (stamparlo non è un compito del traduttore).

L'identificatore e la stringa che si trovano dopo la keyword *MATCHES* servono a specificare rispettivamente la pagina (scaricata da internet) su cui verrà effettuato un controllo di matches, e l'espressione regolare per effettuare tale controllo.

Grammatica

Per stilare le produzioni della grammatica abbiamo utilizzato come generatore di parser GNU Bison, il quale tratta grammatiche LALR(1). La grammatica in questione presenta 28 conflitti shift/reduce opportunamente risolti da Bison.

La grammatica è la seguente:

```
start                : newlines service newlines inputs_or_procedures
                     ;
newlines             :
                     | newlines NEWLINE
                     ;
service              : SERVICE NEWLINE service_attributes
                     ;
service_attributes   :
                     | service_attributes ID          ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes NAME        ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes ICON        ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes DESCRIPTION ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes VERSION     ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes AUTHOR      ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     | service_attributes ENABLED     ASSIGNMENT_OP QUOTED_STRING NEWLINE
                     ;
```

```
inputs_or_procedures  :
                      | inputs_or_procedures input newlines
                      | inputs_or_procedures procedure newlines
                      ;
input                 : input_head input_attributes newlines
                      ;
input_head            : INPUT IDENTIFIER NEWLINE
                      ;
input_attributes      :
                      | input_attributes CONTENT       ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | input_attributes TYPE          ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | input_attributes REGEX         ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | input_attributes NAME          ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | input_attributes DESCRIPTION   ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | input_attributes DEFAULT_VALUE ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      ;
procedure             : procedure_head blocks RETURN NEWLINE
                      ;
procedure_head        : PROCEDURE IDENTIFIER newlines
                      ;
blocks                : newlines
                      | blocks assignment newlines
                      | blocks command newlines
                      | blocks matches newlines
                      | blocks cycle newlines
                      ;
assignment            : IDENTIFIER ASSIGNMENT_OP assignment_value NEWLINE
                      ;
assignment_value      : identifier
                      | assignment_value PLUS_OP identifier
                      ;
identifier            : IDENTIFIER
                      | QUOTED_STRING
                      | INTEGER
                      ;
command               : REQUEST strings NEWLINE
                      | REQUEST strings strings NEWLINE
                      | IDENTIFIER ASSIGNMENT_OP REQUEST strings NEWLINE
                      | IDENTIFIER ASSIGNMENT_OP REQUEST strings strings NEWLINE
                      ;
matches               : matches_head matches_attributes
                      ;
matches_head          : MATCHES IDENTIFIER strings NEWLINE
                      ;
matches_attributes    :
                      | matches_attributes ON_TRUE         ASSIGNMENT_OP BREAK         NEWLINE
                      | matches_attributes ON_TRUE         ASSIGNMENT_OP CONTINUE      NEWLINE
                      | matches_attributes ON_FALSE        ASSIGNMENT_OP BREAK         NEWLINE
                      | matches_attributes ON_FALSE        ASSIGNMENT_OP CONTINUE      NEWLINE
                      | matches_attributes ON_TRUE_OUTPUT  ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      | matches_attributes ON_FALSE_OUTPUT ASSIGNMENT_OP QUOTED_STRING NEWLINE
                      ;
cycle                 : cycle_head newlines blocks BREAK NEWLINE
                      ;
cycle_head            : CYCLE integer_expr NEWLINE
                      | CYCLE NEWLINE
                      ;
strings               : string
                      | strings PLUS_OP string
                      ;
string                : QUOTED_STRING
                      | IDENTIFIER
                      ;
integer_expr          : number
                      | integer_expr PLUS_OP integer_expr
                      | integer_expr MINUS_OP integer_expr
                      | integer_expr MULTIPLICATION_OP integer_expr
                      | integer_expr DIVIDE_OP integer_expr
                      | integer_expr PERCENT_OP integer_expr
                      ;
number                : INTEGER
                      | IDENTIFIER
                      ;
```

Per generare lo scanner è stato utilizzato GNU flex.

Le keyword utilizzate sono le seguenti. Da notare che la convenzione minuscola è stata utilizzata per gli attributi dei vari blocchi (es. SERVICE, INPUT, MATCHES).

```
SERVICE         INPUT           PROCEDURE       CYCLE           MATCHES         REQUEST
CONTINUE        BREAK           RETURN

id              content         type            regex           name            icon
description     defaultValue    version         author          enabled

onTrue          onFalse         onTrueOutput    onFalseOutput

'='             '+'             '-'             '*'             '/'             '%'
```

Commenti

I commenti seguono le convenzioni del C++, quindi è possibile scriverli su singola linea (//) o su più linee (/* ... */). In sintesi sono gestiti correttamente i seguenti caratteri di escape:

\\ tramutato in \
\ tramutato in

Linguaggio destinazione

Il linguaggio destinazione segue le specifiche KMS, descritte nel file *./specifications/kms.pdf*. A titolo di esempio, un'implementazione reale di alcuni servizi che seguono tali specifiche è possibile trovarla in *./kms.xml*.

Compilazione ed esecuzione

L'archivio di riferimento contiene sia i sorgenti in linguaggio C, sia l'eseguibile del traduttore (compilato per un'architettura x86 in ambiente linux).

Per compilare il traduttore, una volta decompresso l'archivio, posizionarsi nel percorso principale ed eseguire il comando make:

```
$ cd ./wsl-0.5
$ make
```

Per eseguire/testare il traduttore:

```
$ ./wsc service.wsl
$ less service.xml
```

It tal modo verrà generato il file *service.xml* (opzionalmente si può passare come argomento un ulteriore parametro con il nome del file destinazione, se questo è omesso verrà generato un file con lo stesso nome del file sorgente, ma con estensione *xml*. Per ulteriori informazioni vedere l'*usage* invocando il comando *./wsc* senza argomenti.

Eventuali errori di sintassi terminano la compilazione segnalando la riga in cui è stato riscontrato l'errore. I caratteri non riconosciuti a livello di analizzatore lessicale vengono segnalati mediante un WARNING non interrompendo tuttavia la traduzione.